UBIQUITOUS USE OF THE WEB STANDARDS MODEL TO UNIFY WEB APPLICATION AND NATIVE APPLICATION DEVELOPMENT PROCESSES

A Practical Approach

Hoque, Majedul

Bachelor's Thesis
School of Business and Culture
Business Information Technology
Bachelor of Business Administration

2017

Abstract of Thesis

School of Business and Culture
Degree Programme in Business
Information Technology
Bachelor of Business Administration

Author	Majedul Hoque	Year	2017
Supervisor	Johanna Vuokila		
Commissioned by			
Title of Thesis	Ubiquitous Use of the Web Standards Model to Unify Web Application and Native Application Development Processes A Practical Approach		
Number of pages	60 + 15		

Web technology is one of the prominent sector in the field of software engineering. Web development is the most sought career advancement pathway in modern era. The web technology principles i.e. HTML, CSS, and JavaScript usage are not limited to the web platform. However, the needs to share common codebase across platforms has remained unsolved.

The objective of the thesis is to implement the ubiquitous usage of the Web standards model into different platforms. More precisely, the concept of the ubiquitous web standards model is studied. Additionally, a third-party questionnaire and surveys were analysed to determine the suitable utilities and features needed to achieve the objective.

The combination of the exploratory and constructive research methods was used in this research work. The exploratory research was used to understand the concepts of the ubiquitous web standards model. Furthermore, frameworks and tools determined for the project were studied. The constructive research method was used to input the gained knowledge into the practical implementation to develop an efficient prototype. In-depth documentation of the development procedures was included for referencing and expanding of the produced solution.

During the practical implementation, a prototype was built to serve a common codebase for different platforms. Webpack, a build tool providing the core mechanisms of the application development and deployment environment serving common codebase was studied. Finally, demonstration of the application development and deployment environment alias build system was given, as the outcome of the research.

Key words HTML, CSS, JavaScript, NodeJS, Electron, Cordova, Webpack

CONTENTS

ABSTRACT
SYMBOLS AND ABBREVIATIONS
1 INTRODUCTION 6
1.1 Background and Motivation 6
1.2 Research Objectives....7
1.3 Structure of the Thesis....8
2 RESEARCH SCOPE, QUESTIONS AND METHODOLOGY 9
2.1 Research Scope 9
2.2 Research Questions 9
2.3 Research Methodology....10
2.4 Sources....12
3 CONCEPT OF UBIQUITOUS WSM 14
3.1 WSM....14
3.1.1 HTML 14
3.1.2 CSS....16
3.1.3 JS....17
3.2 Ubiquitous usage of WSM 18
4 APP DND UTILITIES IN DIFFERENT PLATFORMS....21
4.1 JS Engine 21
4.2 Node.js....22
4.3 Web 23
4.3.1 Front-End 24
4.3.2 Back-End....26
4.4 Desktop....27
4.5 Mobile 28
5 PROTOTYPE DESIGN AND DEVELOPMENT 31
5.1 Proposed Features 31
5.2 Build System Design....34
5.2.1 Desktop App Development and Deployment Environment....39
5.2.2 Mobile and Web App Development and Deployment Environment 42
5.2.3 Plugins and loaders....44

6 PROTOTYPE BENCHMARK 47
6.1 Features demonstration 47
6.2 Files Comparison 48
7 CONCLUSION 50
BIBLIOGRAPHY 52
APPENDICES 60

SYMBOLS AND ABBREVIATIONS

HTML	HyperText Markup Language
CSS	Cascading Style Sheet VoIP
JS	JavaScript
App	Application
DnD	Development and Deployment
OS	Operating System
WSM	Web Standards Model
W3C	World Wide Web Consortium
CLI	Command Line Interface
UI	User Interface
ES7	ECMAScript2016
Browser	Web Browser
API	Application Programming Interface
SDK	Software Development Kit
NPM	Node Package Manager
Vue	Vue.js
CEM	Chromium Embedded Framework
NMAD	Native Mobile Application Development
WMAD	Web Mobile Application Development
HMAD	Hybrid Mobile Application Development
Cordova	Apache Cordova
PhoneGap	Adobe PhoneGap
HMR	Hot Module Replacement
URL	Uniform Resource Locator

1 INTRODUCTION

The background information and motivation of the research work are discussed initially. Additionally, discussions of the research objectives and thesis structure are conducted in this chapter.

1.1 Background and Motivation

In this age of rapid changes in tools and technologies, there is a wide range of computing platforms available. A computing platform or simply a platform is a hardware and software architecture to develop and/or host an application (hereinafter App) (Bridgwater 2015). Each platform has different software App development and deployment (hereinafter DnD) processes, due to different file system architecture in the underlying operating system (hereinafter OS). Platform agnostic code-boilerplate can help accelerate App DnD process regardless of target platform.

App development is one of the core concepts in the field of software engineering. Gartner Incorporated reports that, 2.3 billion computing device shipments are projected by the end of year 2017 (Forni & van der Meulen 2017). Despite the staggering range of computing device availability, most of the research on App development emphasises a specific device platform. At best, efforts of some research are given into cross OS platforms for one specific type of device. For instance, the same App codebase is applied in mobile platform regardless of OS. The very same applies to desktop and web App development projects. However, the necessity of accessing an App is not restricted to one platform, previous research has neglected App development for multiple platforms. The scarcity of availability in multiple platforms can hinder the success of an App, regardless of its rich functionalities, on the one hand. On the other hand, App development typically requires the usage of multiple tools and dedicated App development frameworks specific to a platform to eliminate code redundancy (Vlăsceanu 2012). Emphasis should, therefore, be placed on determining a common ground to unify App development and deployment

processes for different platforms. This research focuses on streamlining App development process for multiple platforms.

My personal interest and experience in App DnD, and the opportunity of helping App developers alike motivated me to select the topic. App development has undergone a rapid development progress in recent years. There is an estimation of a 17 percent growth in employment in App development industry in the next 7 years (Bureau of Labor Statistics 2015). Experience on tools and technologies to build modularized, scalable Apps are expected from the candidates desiring to step into this industry. Therefore, with this development work, I can gain in depth knowledge of App development principles to ensure a good career perspective. Lastly, the applicability of my university education is relevant to the thesis, which has also had a notable impact on studying this topic.

1.2 Research Objectives

The objectives of this study are to study modern App architecture to unify App DnD processes for multiple platforms and to investigate the impacts of using web standards model (hereinafter WSM) globally in a project. Further, the research is to deliver a prototype consisting of an environment for rapid App DnD for multiple platforms. To achieve the objectives, different programming frameworks are reviewed to determine a suitable approach for this project. Furthermore, multiple third-party tools are investigated and proper ones are implemented based on usage-reviews to achieve effective integration in the research work. The concepts utilized in this development work can be studied further and applied in different App development projects. Hence, individuals who have interests in the similar field will find this work intellectual and informative. A sample DnD app is built on top of outputted prototype. Extracted benchmarks from the app helps to identify and prevent any potential bottlenecks in the prototype.

1.3 Structure of the Thesis

To logically follow the research objectives, the thesis is divided into 7 chapters. In Chapter 2, the research scope is outlined and the research questions are delivered following a brief discussion. Additionally, arguments are given to support the research methodology selection. Chapter 3 focuses on discussing the core mechanisms of WSM to lay the path for discussion of ubiquitous WSM usage in the chapter to follow. Furthermore, tools and/or programming frameworks responsible for interoperability, cross compatibility are identified in Chapter 4. Chapter 5 discussions deliver a prototype to successfully provide an environment for multi-platform App DnD. Subsequently, additional features are included in the prototype for effective and efficient project environment. In Chapter 6, the prototype functionalities are showcased to understand the mechanisms of the prototype. Chapter 7 concludes the research providing results of the work and directions for further improvements in the prototype and suggestions for further research.

2 RESEARCH SCOPE, QUESTIONS AND METHODOLOGY

2.1 Research Scope

The research scope focuses on studying the concepts of App development and deployment processes. In addition, this work concerns the study of WSM. Readers are expected to know the basic usage of the WSM determined by the World Wide Web Consortium (hereinafter W3C) (Lane 2007) to understand the practical implementation of the study. Moreover, the practical implementation consists of the usage of command line interface (hereinafter CLI) to a certain extent and familiarity in it is required. Additionally, the research prioritizes independent individual App development and deployment projects. This research does not promise a delivery of guidelines to use the output of the research in group based projects, assuming the requirements are different from individual App development and deployment projects.

The research includes documentation for debugging and the updates relevant to practical implementation. However, typical programming projects are not fully free from errors. This research only intends to deliver a workable prototype. The research also excludes hardware requirements for the prototype.

2.2 Research Questions

The research questions are presented. Moreover, brief discussions of the research questions are provided.

1. What are the underlying building blocks of ubiquitous WSM?

To answer this question, the applicability of WSM in different platforms is analysed. In order to accomplish the analysis, the definition and concepts of ubiquitous WSM are discussed. Understanding JavaScript (hereinafter JS), one of the building blocks of WSM is crucial for effective integration and interaction to specific platform.

2. What is required to unify App development and deployment processes for different platforms?

The unification of App development and deployment for different platforms is dependent on Node.js, a JS runtime environment. Programming libraries and frameworks responsible for compiling codes are reviewed and the appropriate ones are identified to incorporate into platform specific development and deployment. Following the identification of needed programming utilities, a project structure for the prototype is set up following the Model-View-ViewModel, otherwise known as MVVM pattern. The MVVM pattern is essential to maintain separation of concerns between App logic and the user interface (hereinafter UI). In addition, the MVVM pattern helps to improve App testability, to enable developer-designer workflow (Microsoft 2012.) and above all, to streamline App development and deployment processes for different platforms. However, during the App development and deployment using the prototype, following the MVVM structure is optional, nonetheless encouraged for the reasons specified above.

3. What are the drawbacks of using ubiquitous web standards model?

This research question aims to minimise programming errors, if any, within the prototype. During the App development and deployment processes utilising the outputted prototype, different tools are used to document the project benchmarks in different platforms. On the basis of reviewing the benchmarks, possible hinders are pointed out and the prototype structure is revised.

2.3 Research Methodology

The exploratory research method provides profound understanding and clarity of a concept. The exploratory research method is Appropriate for this research work, due to the fact that, this study is to exploit a thorough understanding of

the tools and technologies used to lay the foundation for subsequent research. (Manerikar & Manerikar 2014.)

The constructive research method is the subsequent selection to achieve the end results of this thesis work. The constructive research method seeks to create innovative construction (Hyötyläinen, Häkkinen & Uusitalo 2014). The construction novelty is determined by the exposure to relevant knowledge beforehand, and realising the possible gaps to provide a sustainable solution. Thereby, the constructive research method is appropriate for this work as the solution seeks to be novel in nature. The field of the research is in software engineering and aims to solve practical problems. (Crnkovic 2010, 2.) In addition, this research approach supports coupling with other methods; thus, it is important for this research to interconnect both the exploratory and constructive research findings. (Lehtiranta, Junnonen, Kärnä & Pekuri 2017.)

The research focuses on the usage of the web standards model across multiple platforms. Figure 1 depicts the percentages of the types of software App developer in different sectors of Information and communication technology.

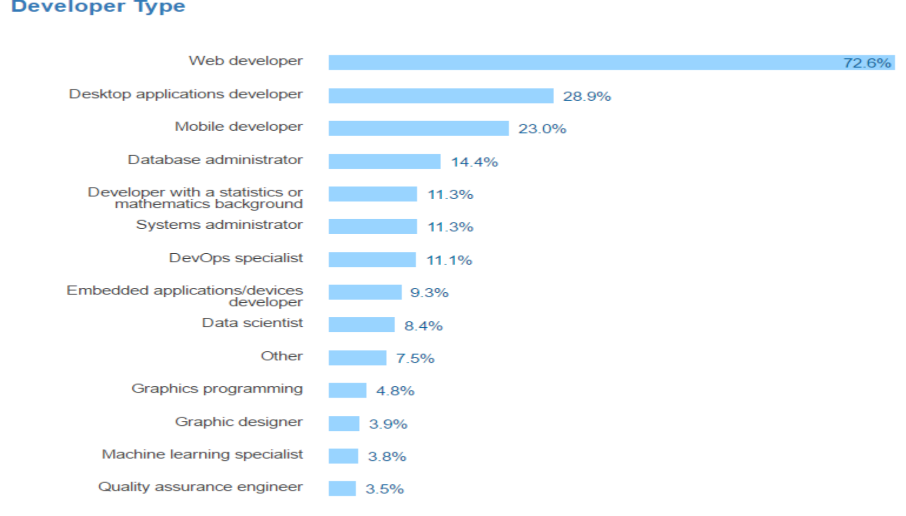

Figure 1. Types of Software Application Developer (Stack Overflow 2017)

As illustrated in the figure above, the top three software App developer positions concern the web, desktop and mobile platforms respectively. The web development professionals occupy 72.6%, astoundingly higher than the number of developers of the two subsequent popular platforms. (Stack Overflow 2017.)

The author of the research provides a solution, that enables existing web development knowledge portability to the desktop, and mobile platforms. Using common knowledge across different platforms ensures a common procedure, and eliminates the gap in expertise across development platforms.

The stream of the constructive development is started by determining the suitable utilities based on third-party questionnaires and statistics report. Based on the in-depth understanding of the utilities determined, an App development and deployment environment is designed and developed. Thereafter, the sustainability of the App development and deployment environment is measured. The constructive development procedure is concluded with the delivery of the prototype to accomplish the motive of using the web standards model in different platforms.

The secondary data analysis method is opted predominantly to extract necessary information by analysing data effectively for the research (Manerikar & Manerikar 2014). The research only utilizes relevant secondary sources, such as books, videos, and journal articles. Both the theoretical and practical part comprises the extensive information gathered from official documentation of tools and programming frameworks, and libraries needed for this work. Primary data extraction is not needed for the research.

2.4 Sources

The research uses ECMAScrip2016 (hereinafter ES7), a relatively new JS standard. The compatibility of the ES7 is not found in established sources during the time of this research. Not all modern web browsers (hereinafter browser) are fully capable to use all the ES7 features. In addition, the results of the compatibility of HTM5, and CSS3 for different browsers across desktop, and mobile platforms are available with the use of an online tool. In chapter 3, the tools used to extract the information of the compatibility of HTML5, CSS3, and ES7 across different browsers are community driven and open-source in nature. The scientific validity of the sources is loosened in order to seek the usage of the HTML5, CSS3, and ES7 in full capacity across different browsers. This

research refers to the authors of the corresponding tools by their nickname provided in the sources, in case of unavailability. The abbreviations of the terms HTML, CSS, and ES7 are given in the next chapter with adequate explanation.

The research excludes the use of primary sources, particularly questionnaires. This is due to the fact that there are comprehensive questionnaire results available showcasing the popular open source JS utilities. This research avoids reinventing the wheel, and proceeds with the available results. This research uses the findings of the ‘2016 JavaScript Rising Stars’ statistics and ‘the state of JavaScript 2016’ surveys to a certain extent, to determine the suitable utilities needed for the prototype. It should be noted that the name of the author of the ‘2016 JavaScript Rising Stars’, and all the utilities used are extracted from the GitHub link provided in the official website.

The research uses official documentation provided for task and platform specific utilities. The official documentations that do not include date of publication, is cited using the release date of the utility as the publication date.

3 CONCEPT OF UBIQUITOUS WSM

This chapter discusses the concepts of ubiquitous WSM. All the basic building blocks of the WSM are briefly discussed in section 1. Upon successful discussions of the individual parts of the WSM, the concept of the ubiquitous WSM is defined in section 2, concluding the chapter.

3.1 WSM

In order to comprehend the WSM, the concept of the web standards need to be discussed. Web standards are set of rules and guidelines recommended to build web pages. (Schmitt, Blessing, Cherny, Evans, Lawver & Trammell 2008, 6.) The web standards define and describe a range of characteristics of the web. The web standards are important from technical and user perspectives. It provides enriched user experience, and overall usability of the web. (Sikos 2011, 3-4.) The systematic approach to implement the web standards in the user interface is concerned with the distinction between the three layers i.e. the markup layer, the presentation layer, and the behavior layer consisting of hypertext markup language (hereinafter HTML) or extensible hypertext markup language, cascading stylesheets (hereinafter CSS), and JS. (Schmitt, Blessing, Cherny, Evans, Lawver & Trammell 2008, 9.)

W3C does not provide a formal definition of the WSM, rather it headlines and discusses HTML, CSS, and JS to elaborate different aspects of the web standards. Thus, the presumption of the concept of the WSM is the architectural separation between the three user interface layers, i.e. HTML, CSS, and JS. (Lane 2007.) When discussing the concepts of the ubiquitous WSM in section 3.2, the languages compositing each user interface layer of the WSM are the subject of discussion. Each language and their compatibility in different browsers are discussed below.

3.1.1 HTML

Several discussions are found on the usage and applicability of the HTML and HTML5 in different literature. It is important to understand the concepts of

“hypertext” and “markup language” prior to discuss the concepts of ‘HTML’ and ‘HTML5’. The method of storing text in electronic-form with reference to access text on another webpage is known as hypertext. Currently, the scope of referencing is not limited to text only. The scope covers different kinds of web content e.g. image, audio, and video. (Meloni 2012, 2-3.) The concept of “markup language” defined in the dictionary is the systematic structure of documents to accomplish logical representation on electronic medium (Merriam-Webster Dictionary 2017).

The definitions of both the HTML, and HTML5 concepts vary from source to source. Burka (2015) defines HTML as the content provider for the web with the capability to modify contents, and HTML5 as the improved version containing extended functionalities. A purely functional definition of HTML is that it is the language that semantically describes different types of documents (W3C 2016). Pilgrim extended the HTML5 concept by defining the language as a backward compatible cross-platform markup language to define structured documents with added functionalities and refined features (Pilgrim 2010, as cited by Puputti 2012, 5).

Browser is the default platform that runs web documents, and is typically used to access websites via the Internet. Figure 2 depicts the HTML5 features accessible by some modern browsers.

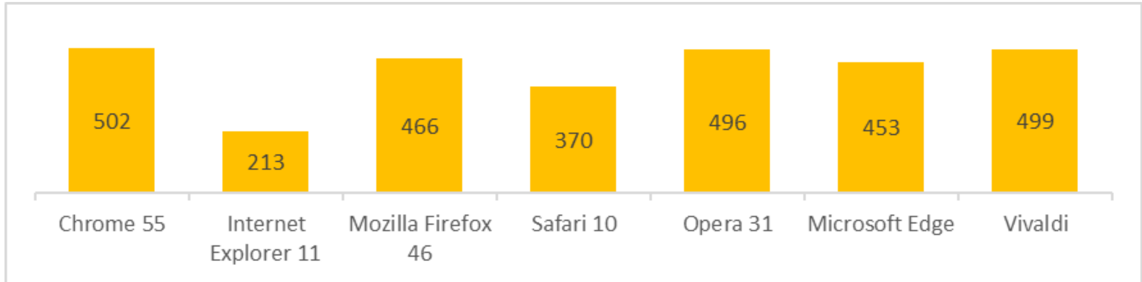

Figure 2. HTML5 Compliance Chart for Major Desktop Browsers (Digital Trends 2017)

As drawn in figure 2, The latest Chrome 55 browser from Google has the maximum HTML5 usage capability, scoring 502 out of 555. Opera 31, and Vivaldi, a relatively new browser, are close to Chrome 55, in terms of HTML5

features accessibility. Vivaldi and Opera 31 scores 499, and 496 respectively. The least HTML5 compliant browser is Microsoft Internet Explorer 11. The support for Internet explorer is terminated, causing the loss to adapt the changes in the web technologies. (Digital Trends 2017.) Mobile browsers HTML5 compatibility is crucial, as mobile usage is significantly higher than the Desktop (comScore 2014). The figure below shows the HTML5 compatibility rate in different browsers.

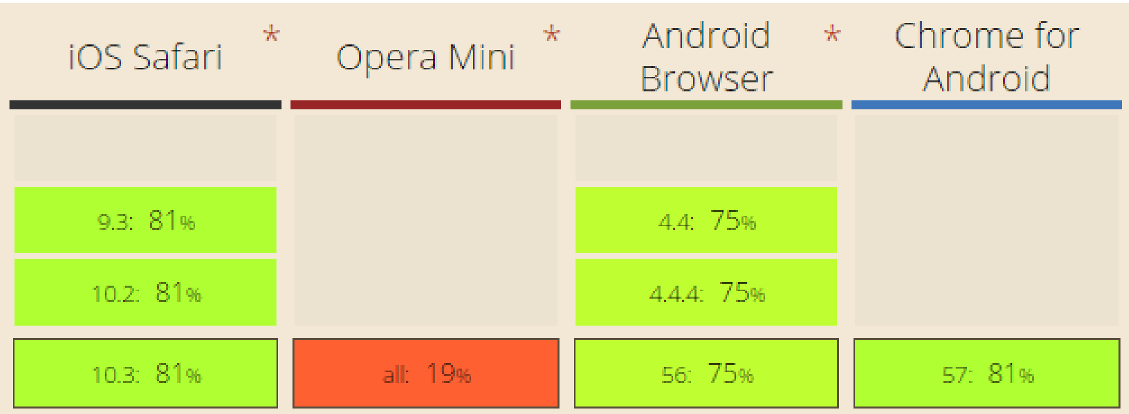

Figure 3. HTML5 Compatibility Rate for Major Mobile Browsers (Deveria 2017a)

In the figure 3 and 4, the blocks surrounded by black border represents current stable release, and the desaturated blocks represents the preview or experimental release of the corresponding browser. The rest of the blocks represents the previous stable release. (Deveria 2017a.) As depicted in the figure 3, Chrome for Android and iOS Safari, are the most HTML5 compliant browsers. Opera mini falls short in terms of HTML 5 compatibility. This is by no mean the full list of available browsers. The tool only shows browsers commonly acknowledged, and are actively maintained and supported.

3.1.2 CSS

According to Bos, CSS is a “simple mechanism for adding style, e.g., fonts, colors, spacing, to Web documents” (2017). CSS rule is a set of instructions that browser uses to alter the presentation of the HTML elements (Schultz & Cook 2007, 22). It can control multiple web pages with a single rule defined (Meloni 2012, 45). It enables to abstraction between the document contents

from the presentation logic, to govern the presentation characteristics, and to improve content accessibility (Aronson 2011, 15). CSS is the only standard to style HTML documents.

CSS3 being the latest standard has backward-compatibility with the previous CSS versions. CSS3 consists of multiple ‘modules’. Each module is responsible for a specific functionality and is standardized by W3C if the module is stable and non-problematic. (W3C 2001.) CSS3 modules are constantly being developed. Many of the modules are not effectively integrated. Figure 4 below represents the CSS3 compatibility in multiple browsers.

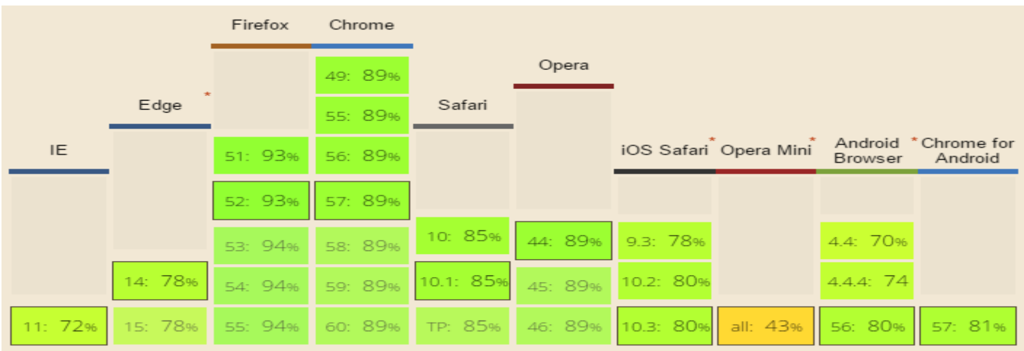

Figure 4. CSS3 Compatibility Chart for Major Browsers (Deveria 2017b)

As given in the figure above, in the desktop platform Firefox has the maximum capability to provide CSS3 features. Microsoft provided Inter explorer 11 and Edge 14 has the least support of CSS3. In the mobile, Opera mini has the lowest rate of CSS3 features accessibility. The other browsers are competent and maintain almost same rate of CSS3 feature integration and access. (Deveria 2017b)

3.1.3 JS

JS is a high-level, dynamic and untyped interpreted programming language. It is known as the language of the web, and used by all modern browsers to specify the behaviour of a webpage. (Flanagan 2011.) JS is object oriented to the core. All the data types are considered as objects. The primitive data types are wrapped within an object data type. It is prototype based, follows a prototype-inheritance model when assigns to an object. (Peschla 2012, 13.) ECMAScript

is the standard for JS defined by European Computer Manufacturer's Association to provide cross-browser compatibility of the language. The terms ECMAScript and JS are used interchangeably (Haverbeke 2014). Typically, the attributes of JS are associated to the use of ECMAScript standard interfacing browser's DOM Application programming interface (hereinafter API) and XMLHttpRequest API, (Peschla 2012, 12). This research only uses the term "JS" when referring the implementation of ECMAScript standard strictly.

ES7, the seventh edition of the ECMAScript language defined in ECMA-262 standard is used in the prototype construction. (Ecma International 2016.) Many of the ES7 features are supported by modern browsers. Chrome 57 and Opera 44 have an 80% accessibility of ES7 features, while Internet Explorer 11 has the least access of 4% of the ES7 standard. (Zaytsev 2017.)

3.2 Ubiquitous usage of WSM

The usage of HTML and CSS is restricted to the browser. JS is used as a server-side programming language and in browsers. This section discusses the applicability of the usage of the WSM components in different platforms, including Desktop OS, specifically Windows OS, Mac OS, Ubuntu, Debian based Linux OS, and CentOS, a Red hat based Linux OS; Mobile phone OS, specifically Android OS, and iOS; and browsers, thus ensuring the applicability of the Ubiquitous WSM.

An App DnD project consists of source codes and resources. The resource contents can be of anything, including icons, images or data files. The combination of source codes and resources is compiled to run as an executable program, or as a website. (Microsoft Developer Network 2017.) However, executable programs architectures are not same in all OS platforms. The common executable file extensions are depicted in the table below.

Table 1. List of Common Executable File Extensions (Fisher 2016)

Extension	OS
.exe	Windows
.app	Mac OS
.deb	All Debian based Linux OS
.rpm	All Redhat based Linux OS
.ipa	iOS
.apk	Android OS

The extensions listed above in Table 1 are the filetype-extensions of the executable files of a platform. The executable files act as the entry points for an App solution for a particular OS. The concept of the ubiquitous WSM derives from the fact that, any App projects using the WSM can be deployed in any platform, in order to ensure that WSM source codes and resources are packaged within OS specific executables natively, as a form of the App.

The ambition of the prototype is to build solutions with indifferent WSM codebase across platforms. Below in the figure, the strategy to accomplish the App deployment using WSM codebase is depicted.

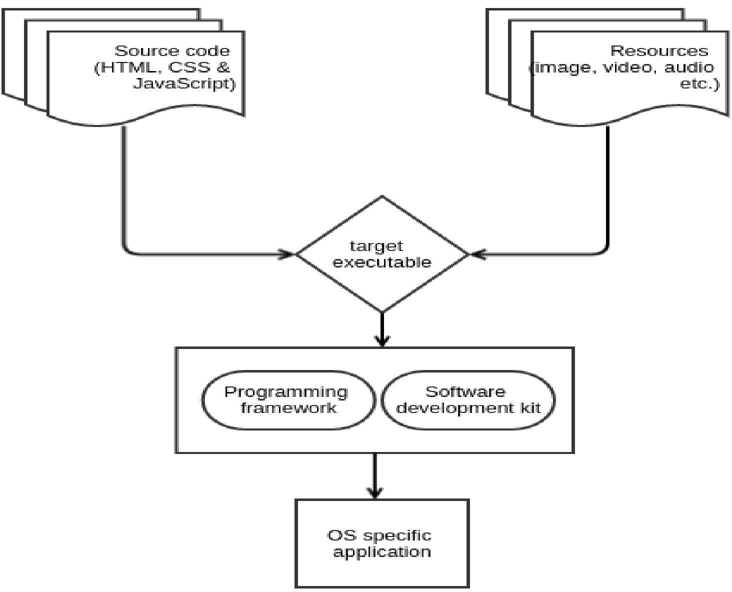

Figure 5. Ubiquitous WSM Deployment

Based on the approach showed in figure 5, developers decision leads to OS specific deployment. Framework is the structural combination of tools, and languages used to build and deploy App (Porebski et al. 2011, as cited by Vlăsceanu 2012, 18). Software development kit (hereinafter SDK) is a toolbox to compile source code and resources interpretable by platforms (Sandoval 2016). The usage of programming framework, and SDK are needed to compile source codes into OS specific Apps.

4 APP DND UTILITIES IN DIFFERENT PLATFORMS

The chapter endeavours to understand the medium of portability of JS in different platforms. section 4.1 covers the concepts of concept of JS engine. section 4.2 details the discussion of Node.js, and the Node package manager. section 4.3 elaborates the selection of web App DnD utilities with the use of the WSM components. In section 4.4, the extended use of the WSM is explored in cross-platform desktop DnD. In section 4.5, a similar approach is taken to seek for cross-platform mobile App DnD utilities.

4.1 JS Engine

Node.js, or simply Node is built on a JS engine, particularly V8 and therefore, the concept of the JS engine is studied primarily. Furthermore, the study covers brief description of different JS engines. Emphasis is given on the V8 to comprehend the concept of Node.js in the section to follow.

The responsibility of JS engine is to compile JS source codes, to allow the browser to interpret and represent JS codes on a web page. JS engine is a browser feature in nature. Different browsers use different JS engines. (White 2009, 12) Table 2 provides a list of major JS engines available for modern browsers.

Table 2. List of JS Engines

Engine	Vendor	Browser
Chakra	Microsoft	Internet Explorer 11, Edge
Spidermonkey	Mozilla	Firefox 52
JavaScriptCore	WebKit	Safari 10.1
V8	Google	Chrome 57

Table 2 only enlists the current stable browsers. The engines are also used in other release versions. During the development stage, BSCF App is tested on the browsers mentioned above.

The core components of Chakra are open sourced under the name "ChakraCore". It has also powered Universal Windows applications, Azure DocumentDB, Cortana, Outlook.com. Chakra has made it possible to use Node.js in Windows 10 IoT Core. (Seth & Foresti 2016.) Spidermonkey is the pioneer of the JS engine. It is written in C++, and used in various Mozilla products in addition to Firefox browser. JavaScriptCore is based on KDE JS engine. It is widely used for scripting in Apple's OSX OS, and JS testing in in-IDE for Adobe Air and Dreameweaver CS4. V8 is developed by Google, written in C++ programming language, and is used in the Google's Chromium, and Chrome browser. (White 2009, 12-13.) V8 is a standalone project on its own, although, it was primarily developed as a part of the Chromium browser. Chromium embeds the V8 by using V8 API. Effective implementation and modification of objects and properties, dynamic generation and optimization of machine code, and the efficient garbage collection process are the main factors for fast data processing in V8. (Peschla 2012, 9.)

Dahl (2010), the creator of Node, stated the reason to use V8 for building Node is the single threaded nature of JS, and V8. Previous solution to use JS on server-side is intended for multi-threaded operation. V8, as a virtual machine was performant than the available virtual machines available during the time of Node development. (Dahl 2010.) In-depth discussion of server-side is provided in the section 4.3.

4.2 Node.js

The reason for choosing Node is apparent, i.e. at the time of conducting the research node ecosystem consists of utilities to support App DnD with the usage of WSM. The official Node website remarks node.as a lightweight and efficient JS runtime environment. A runtime environment is simply the state after program execution, providing the access to different computing services such as processor and main memory access (TechTerms 2017).

The nature of Node is event-driven and performs non-blocking I/O operations. Events, in the context of Node, are registered as soon as a Node App is executed. Node event loop is the background process which is always running to listen to any events to occur, and responds to the event based on instructions, commonly known as event handlers. Event response occurs sequentially via event-queue, because JS is single threaded in nature. (Ihrig 2013, 29.)

The non-blocking I/O operations refer to the asynchronous JS operation and is dependent on the event-driven approach. Node does not wait for a return value to do the next operation; rather the evet-loop waits for an event to occur to perform the operation. Multiple parallel I/O operations are possible, and event-handler is executed to an I/O operation. (Teixeira 2013, 16.)

Node has various built-in modules to perform a wide range of tasks, e.g. accessing file system, creating http, and tcp server (Node.js 2017). Node version 0.6.0 and above ships with Node package manager (hereinafter NPM). NPM enables to use third party Modules, otherwise known as packages in App DnD projects. NPM consists of a package repository for third-party modules. Initializing a ‘package.json’ file is needed to maintain packages in a local computer system (Teixeira 2013, 8-9, 12.) NPM provided command-line tool is needed to initialize a ‘package.json’ file, and to install and modify existing packages interactively. (NPM 2017a.) Platform specific DnD relies on the third-party NPM packages. This section satisfies the partial needs to blueprint the prototype by studying NPM, and the implementation of NPM.

4.3 Web

Web App is client-server app, and this section discusses the use of full-stack JS to adopt the usage ubiquitous WSM in the web app. However, this is partial implementation of the WSM, as JS is integrated to control the behaviour of the App only. The browser serves as the client in this kind of App architecture. (Shklar & Rosen 2003, 201-202.) The browser uses both the rendering engine,

and the JS engine to represent the UI. The layout engine takes the HTML, and CSS source code to compile. (Wright 2013, 29-30.)

The modern web App fits into the term multi-tier architecture, as the server can act as a client while interacting with a database server. Each tier consists of an App layer. The bottom layer is the client that initiate requests to the server or the intermediate layer. The intermediate layer may act as a client, or server, or both concurrently, depending on its responsibility. (Shklar & Rosen 2003, 203.) The client-side, and the server-side are commonly referred as the front-end, and the back-end of the web respectively. This research uses the convention "front-end" and "back-end" from this point forward. The front-end and back-end DnD libraries and/or frameworks chosen for the prototype are discussed below. It should be considered that the in-App utilities are not obligatory for developers to use. The research only tries to clarify the popularity between different libraries, and/or frameworks.

4.3.1 Front-End

The front-end deals with the direct interaction of the user with the web. In this regard, all the parts of the UI layer of the WSM are used in the front-end development. (Long 2012.) The prototype establishes the MVVM architectural pattern for front-end of the App Development. Using plain JS in development is cumbersome to follow MVVM correctly. Besides, using a JS library and/or framework is often needed for improved readability. Figure 6 illustrates the most-starred open source front-end utilities available on GitHub website.

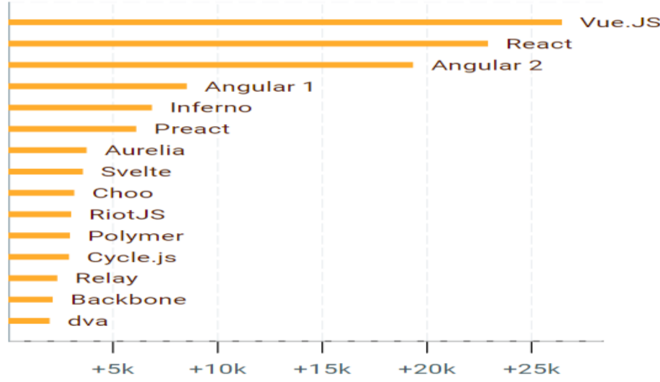

Figure 6. Front-end Projects in 2016 (Rambeau 2017)

As depicted on the figure above, there is a significant popularity in the Vue and React. React has more than 22000 stars in the GitHub, while Vue topped by a margin of approximately 4000 stars. (Rambeau 2017.) In addition to the popularity of front-end utilities, their awareness among App developers are given in the chart below.

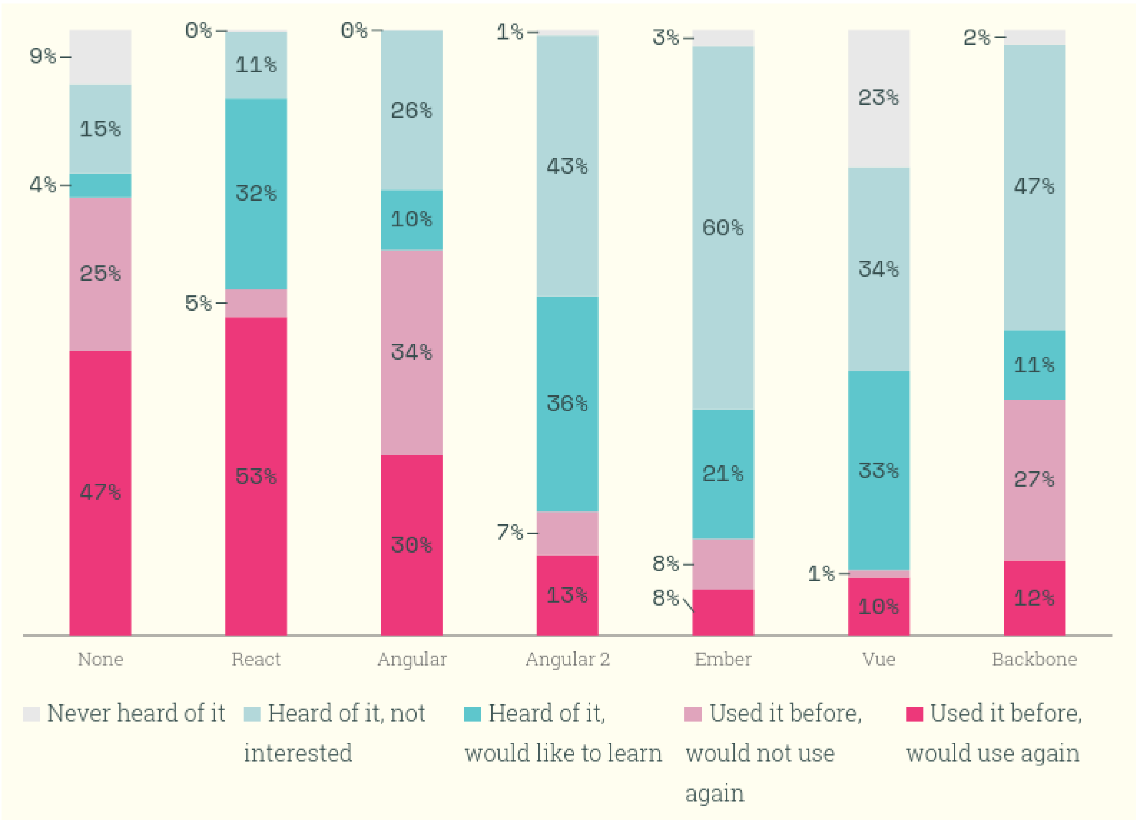

Figure 7 Front-End Frameworks Awareness (Greif 2017a)

There is a coherence between the significant satisfaction rates for React and Vue.js (hereinafter Vue) library, and most-starred open source projects shown in figure 7 and 6 accordingly. Based on figure 7, 92%, and 89% of the user base would like to continue using React, and Vue respectively. (Greif 2017a.)

Vue is lightweight compare to React, providing better performance in execution time than React. In addition, it automatically tracks component's dependencies, giving it edge over react, where the process is manual. The decision to choose the suitable utility for the prototype, a performance benchmark is provided in the table below. (Vue.js 2017a.)

Table 3. Performance Comparison Between Vue and React (Vue.js 2017b)

	Vue	React
Fastest	23ms	63ms
Median	42ms	81ms
Average	51ms	94ms
95th Perc.	73ms	164ms
Slowest	343ms	453ms

Table 3 description boosts the deciding factor between the two popular utilities. Besides, Vue's enriched ecosystem makes the author to choose Vue as front-end development utility for the prototype development. Vue is a progressive framework, that manipulates the view layer of the MVVM (Vue.js 2017b) The concept of the progressive framework is the scaling ability of the front-end app. The minimalistic nature of Vue helps developer to focus on App requirements. Additional functionalities of an App are handled with the use of libraries provided in Vue ecosystem. This helps the customization of the Vue in compliance to App requirements. (You 2016.)

4.3.2 Back-End

The back-end powers the front-end. It "consists of a server, an application, and a database". User provided data from the front-end is accessed by an application to interface with a database in a server. (Long 2012.) There are multiple programming languages available to communicate between the three back-end parts. The Node is developed primarily for back-end development, and uses JS, therefore, is used in the back-end of the prototype. However, there are various frameworks available, built on top of Node. These frameworks provide additional functionalities. The figure below depicts the popularity of different Node frameworks.

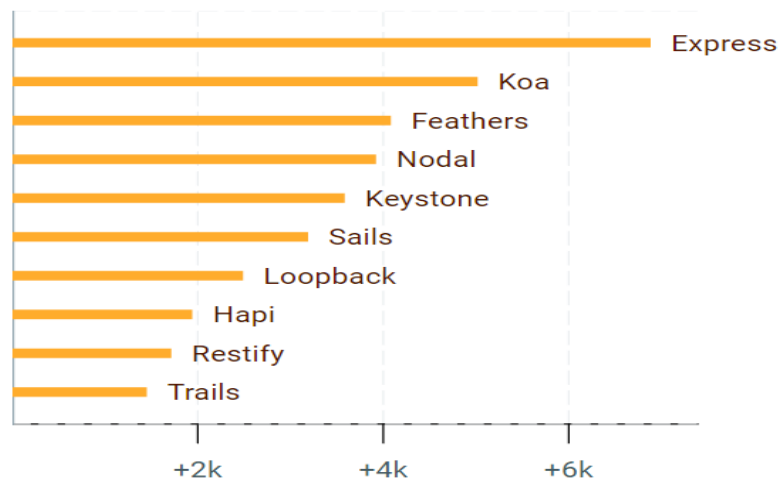

Figure 8. Most-starred Node Frameworks (Rambeau 2017)

As stated in figure 8, express is the most popular back-end framework. Express is often considered as the de-facto Node framework. It provides easy abstraction of codes, server side routing, and enables third-party module injection via middleware. (Rambeau 2017.) This research, implements the Express framework for back-end development.

4.4 Desktop

There are multiple utilities available to develop a cross-platform desktop app. However, there are only four utilities able to compile and execute the WSM to build Desktop app. These utilities are, Adobe air, Chromium Embedded Framework (hereinafter CEF), NWJS, and Electron. (MobiDev 2015.) Figure 9 represents the most popular App development utilities of the year 2016.

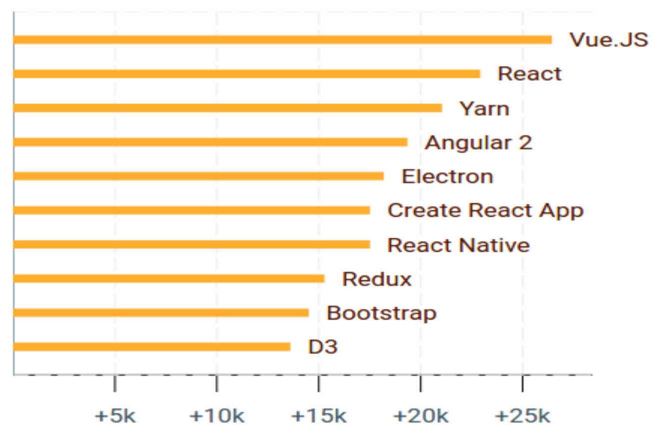

Figure 9. Most Popular Projects in 2016 (Rambeau 2017)

Figure 9 enlists Electron as one of the most popular JS project. Electron and NWJS remains the most popular open source project, on the one hand. On the other hand, Adobe air is closed-source, and it discontinued support for Linux Desktop App development. CEF does not have a stable version released yet. NWJS uses CEF and Node as basis, causing large built size of deployed app. Electron is a stable, features enriched, and actively maintained library. (MobiDev 2015.) This research is to include Electron to provide desktop App DnD environment.

Electron uses the UI layer components of WSM in order to create cross platform desktop application. The combination of Chromium's rendering engine and Node.js API, as a shared environment of V8 is used to provide runtime for Electron. (Lord 2016.)

4.5 Mobile

There are three approaches available to build for mobile App development i.e. native mobile App development (hereinafter NMAD), web mobile App development (hereinafter WMAD), and hybrid mobile App development (hereinafter HMAD). In NMAD, vendor provided tools and technologies are used depending on the OS. Android, and iOS, the current popular platforms use Java, and Object C programming language for App development. WMAD is concerned with website, not application, thus is not discussed further. HMAD allows the usage of ubiquitous WSM languages. Hybrid mobile applications can access the native components to implement device features as native application. (Panhale 2016 ,15, 16, 18.) The figure below illustrates the differences between native and hybrid application.

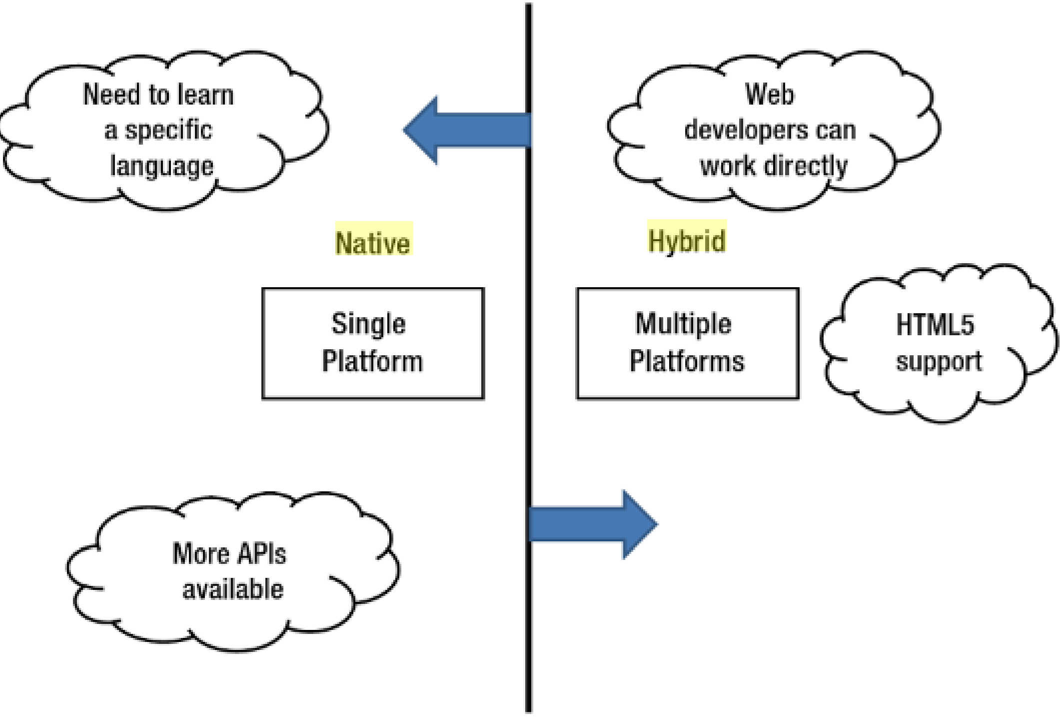

Figure 10. Differences Between Hybrid App and Native App (Panhale 2016, 18)

As seen on the figure 10, App development involves learning device specific language for native development. Traditionally, this language does not support on another mobile OS. However, runtime environment access is significantly large in the Native app. On the contrary, the usage of WSM is possible in the hybrid app. Source codes are cross-platform in nature. (Panhale 2016, 18.) Figure 11 provides a list of renowned mobile App development frameworks.

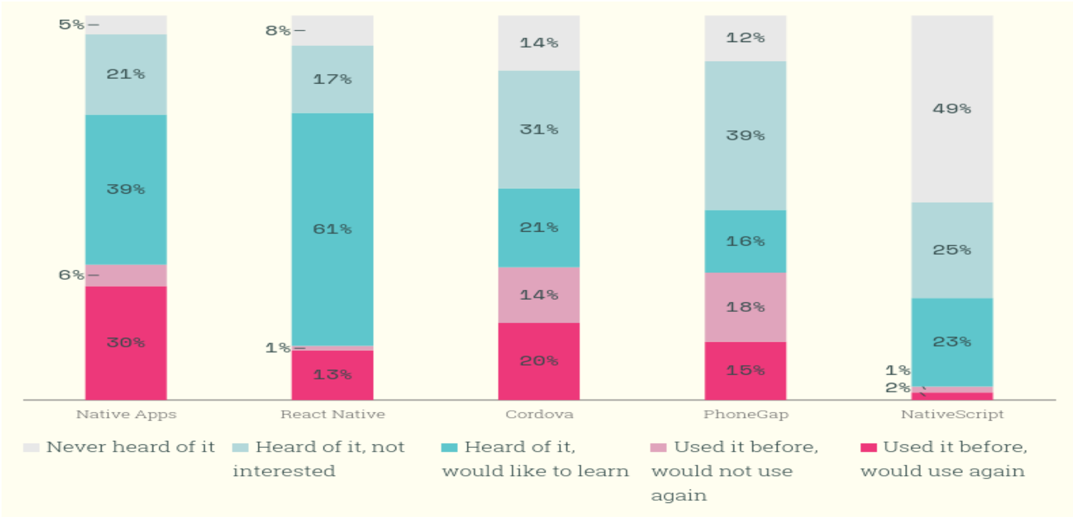

Figure 11. Mobile Frameworks Awareness (Owens 2017)

As indicated in figure 11, 84% mobile App developers are satisfied with using Native App development. React Native is popular in App development, and it uses react, a library for JS. However, the use of the WSM in mobile App DnD is only possible by Apache Cordova (hereinafter Cordova), and Adobe PhoneGap (hereinafter PhoneGap), both utilizes HMAD approach. (Owens 2017.) This research concentrates on the use of WSM and, therefore, Cordova or PhoneGap is suitable for the research.

Cordova is cross-platform in nature. It uses WebView rendering engine to provide app's UI, and set of core plugins to navigate hardware features e.g. camera, accelerometer, geolocation. (Cordova 2015.) On the contrary, PhoneGap is a distribution of Cordova, powered by the Cordova engine (LeRoux 2012). The research uses PhoneGap due to the UI live-reload capability for browser and device simultaneously, in addition to the access of all the Cordova core features availability.

5 PROTOTYPE DESIGN AND DEVELOPMENT

This chapter is dedicated to delivering a workable App DnD environment for multiple platforms. section 5.1 provides a brief explanation of the features to be implemented in the build system. section 5.2 focuses on the actual implementation of the identified utilities and features to the App DnD environment.

5.1 Proposed Features

The App DnD environment seeks for features and functionalities for the efficiency in workflow during the App DnD. However, this research excludes the exploration of features needed for in-App development, since these features are programming framework and library specific, and are not in the scope of this research. Figure 12 depicts the top most important features demanded by App developers.

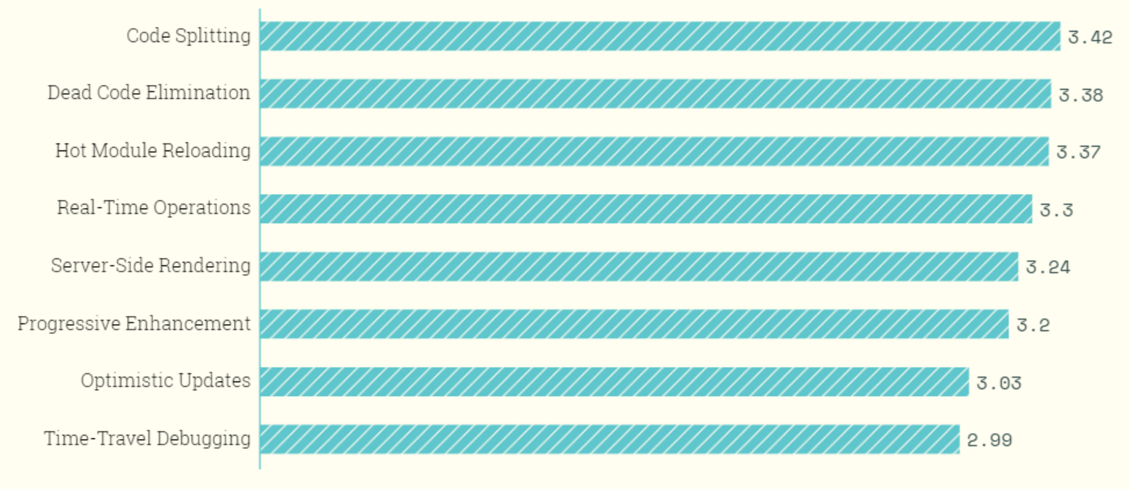

Figure 12. Highest-Rated JS Features (Greif 2017b)

The top 3 highest-rated features depicted in figure 12 are parts of JS build tools for developing a build system. The rest of the features deals with in-App development. (Greif 2017b.) A build system is the automatization process of program compilation. The primary purpose of build systems is to competently map resources into executables. (Williams 2009.) The prototype is a build system in nature, where App DnD is automatized. Specifically, the research

outputs a JS build system. JS build system deals with the automatization of optimized source code translation, error detection, and third-party plug-ins integration. (Adams 2015, 36-38.) In order to setup a build system, this research investigates to find a build tool appropriate for the prototype. Figure 13 depicts the most common build tools for App DnD.

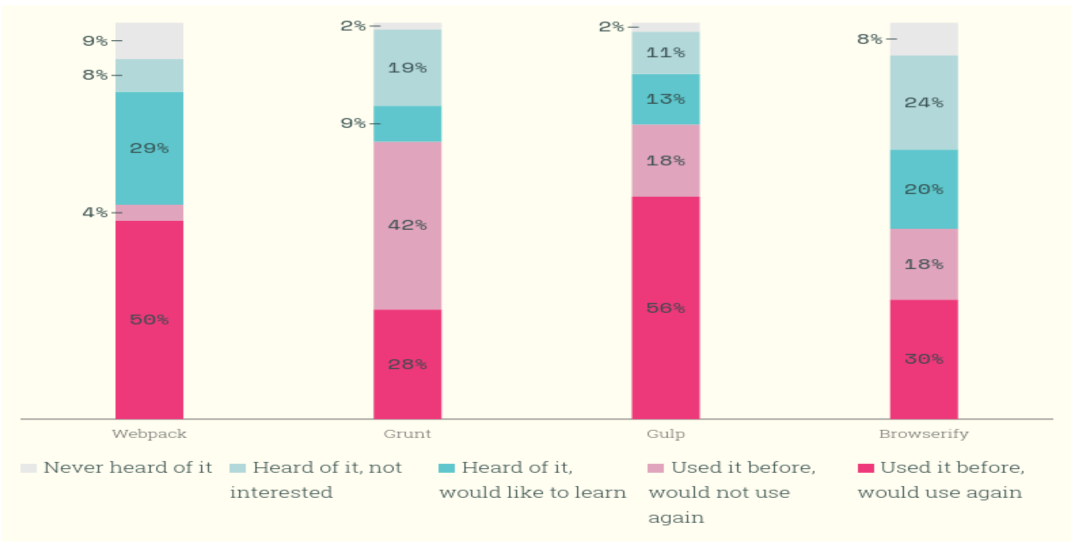

Figure 13. Build Tool Awareness (Wong 2017)

As is shown in the figure above, both Webpack, and Gulp.js have the highest satisfaction rate. Out of all, 93% of the users are satisfied with Webpack, and 75% of them are satisfied using Gulp.js. (Wong 2017.) As mentioned in the Webpack 2 official guide, the latest iteration of Webpack, i.e. Webpack 2 provides the top three features mentioned in figure 12 natively (Webpack 2017a). On the contrary, Gulp.js relies on tools such as Webpack 1, Browserify, or Rollup.js to integrate the additional features mentioned above. Webpack performs a wide variety of tasks such as code minification, and generating source maps out of the box. This research uses NPM, and Webpack 2 to deliver a build system for App DnD environment as a form of prototype. The remainder of the thesis uses ‘Webpack’ as naming convention for ‘Webpack 2’

The App DnD environment automation requires the need of NPM scripts, and Webpack 2. Project dependencies are declared in package.json manifest file. This file includes the ’scripts’ property, which enables to run NPM commands

from CLI (NPM 2017b). NPM script is essential to instruct project specific tasks i.e. executing Webpack script.

Webpack brands itself as a module bundler. It takes all the modules of code and bundles them in a single static file. The concepts of entry, output, loaders, and plugins are core to the implementation of Webpack based build system. Entry is the starting point, including references to dependencies of the application. Output, as the name suggests compiles into an output as a bundle JS file. Static assets such as CSS or HTML file executes with Webpack, not the Browser, with the help of Loaders. Plugins help to provide additional functionalities to Webpack
bundles. (Webpack 2017b.) The features to implement in the build system are briefly discussed below.

The first feature needed for the App DnD environment is the compile of ES7 into ES5 standard, as all the targeted platforms rendering engine support ES5 standards fully (Zaytsev 2017). This compile is done with Babel loader for Webpack, and all the Babel dependencies required. Babel is a tool that enables to use the latest version of JS. (Babel 2017.)

The code splitting feature is next in line to integrate into the App DnD environment. The code splitting concept is the on-demand code compilation of a certain functionality (Webpack 2017c). This approach avoids the monolithic bundles of the app. It accelerates performance due to fact that a chunk of code is downloaded, and read by the JS engine, rather than the whole App codebase. (Greif 2017b.)

The third feature is the Dead code elimination, known as tree shaking in the context of Webpack. The automatic removal of unused codes is known as tree shaking, a crucial feature for App DnD workflow. (Greif 2017b.)

Hot module reloading, the fourth feature to be available in the build system deals with workflow flexibility. The process of keeping the App running, and being able to adapt to changes in the codebase without resetting is known as

Hot module reloading (Bigio 2016). The feature is built on top of Webpack's Hot module replacement (hereinafter HMR) concept. HMR enables the automatic reset only to the manipulated components of the App in real time. The rendering engine, and layout engine does not require a restart to opt to changes. This is performant for browsers engines, because whole App is not needed to be rendered in case of any changes in the JS codebase. (Webpack 2017d.)

The CSS auto prefixing feature is crucial to apply vendor prefixes for CSS to the appropriate browser. Different browsers use different notations to add support for new CSS modules (Cook & Garber 2012, 377-378). The auto prefixing, as the name suggest converts the source code to be usable in different browsers, by automatically prefixing for different browsers. In addition, the automatic code minification feature is implemented for compact build of application.

This research focuses on both the automation, and autonomation of tasks for App DnD. Programming is prone to errors. Error correction is a time-consuming process. The research emphasises automatic detection to prevent in-App errors. (Liker & Meier 2006, as cited by Tommelein 2008.) Code linters helps to write optimised codes by providing indication of any suspicious code that might lead to error in program execution (Spencer & Richards 2015, 74). The build system is to include multiple linters to check both JS and CSS.

5.2 Build System Design

To start with the build system development, a project structure is planned and implemented. The project structure helps to abstract task specific codebase. After the success of the project structure implementation, a platform specific environment is set up. Common utilities and their implementation are done in parallel for the successful implementation of the features discussed previously in this chapter to ensure the success of App DnD environment delivery.

The Webpack is to bundle all the source codes. The source codebase is to be put in a single directory, and the compiled form of the codes are to be put in a separate directory. Figure 14 is to demonstrate the systematic flow of the build system implemented.

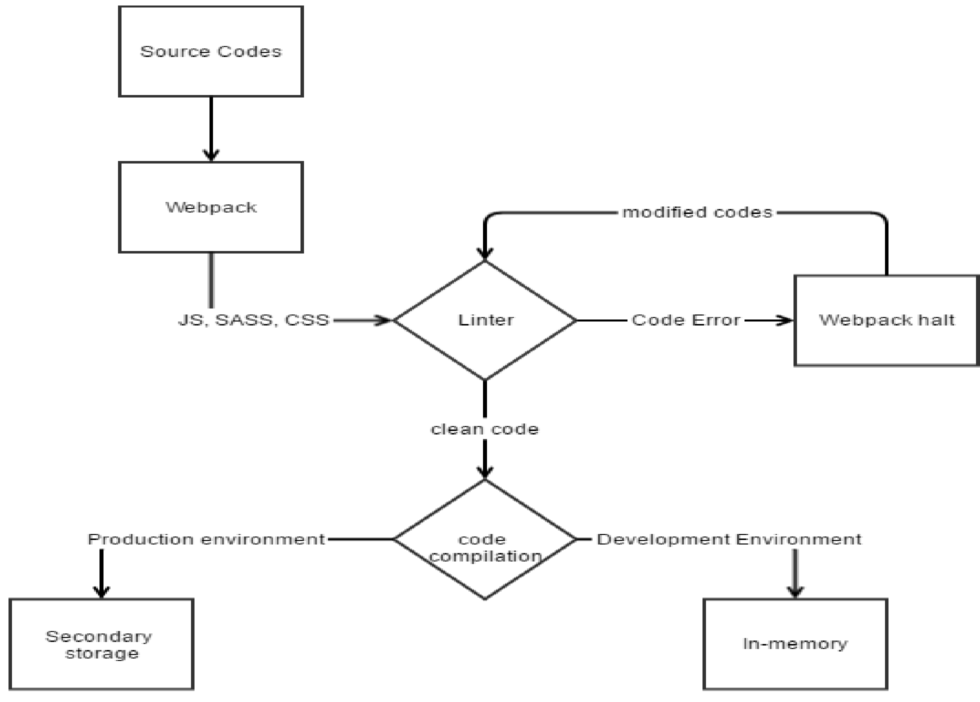

Figure 14. Build System Workflow

As shown in the figure 14, source codes are handled by Webpack. JS, SASS and CSS are to be tested with linter before code compilation. It has to be noted that, HTML is not included, as modern code editors are fully capable of linting it. However, HTML is loaded with Webpack and directly put in a secondary storage for the simplification of all WSM codebase transformation within source directory. In addition, SASS otherwise known as Syntactically awesome stylesheet is included, as modern UI development often requires the presence of programming aspects in the markup e.g. CSS. SASS is a preprocessor for CSS which provides various programming functionalities out of the box (Budd & Björklund 2016, 391). The development and production environment is determined by specifying variables, referring to the environment required. This variable is specified in Node, and evaluated by Webpack. The production environment is responsible for minifying codes, and outputting the source

codebase into physical disks, on the one hand. On the other hand, the development environment includes source mapping for the purpose of debugging, inline styling, and HMR. The development environment is served from the primary memory. Source mapping is the concept of referencing the lines of the code that executes a certain function (Seddon 2012). The mapping is tied to the source code base, not to the bundled code. The bundled codebase is minified for performance, and is not readable by users. The code splitting feature, and the image optimization feature both are common to both the production and development environment.

Figure 15 displays a project structure for the build system representing the overall build system.

Figure 15. Project Structure for App DnD

Conforming to figure 15, in-App development only concerns with src, dist, and www directories. The source directory includes three JS files. The server.js is used as the entry point for back-end development. The electronMain.js powers the Electron to execute, and manage the main process. The index.js file deals with the development for the UI layer for all platforms. In addition, the PhoneGap framework specific codes are included in the index.js file along with common codes. The index.html file serves as the entry point for all three platforms. Further down in this section, in depth discussion of each framework build systems covers the reason for individual HTML files. The dist directory consists of all the generated bundles of source codes for Desktop and Web platform. Cordova reads and executes UI codebase from the www directory. Therefore, bundles are directed to the www directory. Users should not modify anything from both of the dist and www directories. The Webpack-partials directory is the heart of the research, as it contains the mechanisms for App DnD environment. All the files except the package.json file are modules that webpack.config.js depends on. The webpack.config.js is executed with the NPM from a CLI to bundle source codes based on instructions. The package.json included in the project root contains third party packages lists, along with CLI scripts that performs different tasks. The other package.json provided inside the Webpack-partials directory includes electron specific codes, and copied to dist/desktop directory dynamically to deliver desktop app solution. Two lint files include the rules for CSS and JS coding practice. A node_modules directory is automatically generated on the project root directory, during NPM packages installation from CLI. The remaining files and directories are concerned with Cordova platform operability. The research emphasizes on cross-compatibility. However, commands for manipulating files and setting environment variables are different between UNIX and Windows based OS. Therefore, rimraf, copyfiles, and cross-env third party node modules are utilized to use global command for manipulating files and setting environment variables across UNIX and Windows OS's. The CLI commands, listed in the package.json file to set the base to work environment is listed in Appendix 1. All the commands, have to be prefixed by 'npm run' for successful execution. For example, 'npm run mobile:deploy' would result in building an apk solution with source codes. Calling the 'prepare' script scaffolds the outputted codebases for each platform

in dist directory. This is a preliminary requirement before running any other scripts after all the modules are installed by executing npm install command from CLI. The webpack-dev-server command runs a server that stores the files in the main-memory, watches for file changes in the client-side, and provides the HMR functionality (Webpack 2017e). Appendix 1 also includes the third-party modules as devDependencies needed for the App DnD environment. ‘Babel-core’ compiles the ES7 codes into ES5 code (Babel 2017a). All the other modules containing the name ‘Babel’ in them are dependencies for ‘Babel-core’ for Webpack. Babili is also a Babel dependency used for treeshaking, and minification of JS code (Babel 2017b). File-loader is used to load assets i.e. css, images, and fonts to bundle them with Webpack (Webpack-contrib 2017a). Stylelint for Webpack is used for linting CSS, and SASS. PostCSS provides wide ranges of features for CSS i.e. code minification, and autoprefixing (PostCSS 2017). Image-loader for Webpack is used to compress images in order to minimize the bulkiness of App solution (Coopman 2017). The remaining packages’ description is briefed during build system implementation.

There are two environment variables passed into Node when executing Webpack. The first environment variable, BUILD contains either the value of ‘production’ or ‘development’. These are provided to point out specific functionalities needed from each BUILD. The second environment variable, PLATFORM indicates Webpack to bundle codes for either desktop, mobile, or web based on the declaration. The representation of the environments is shown in figure 16.

Env-variables.js

```
const env = {
  production: process.env.BUILD === 'production',
  development: process.env.BUILD === 'development',
  desktop : process.env.PLATFORM === 'desktop',
  mobile : process.env.PLATFORM === 'mobile',
  web : process.env.PLATFORM === 'web'
};

module.exports = {
  production: env.production,
  development: env.development,
  desktop: env.desktop,
  mobile: env.mobile,
  web: env.web,
};
```

Webpack.config.js

```
const {main,renderer} = require('./webpack-partials/desktop');
const cordova = require('./webpack-partials/mobile');
const {back,front} = require('./webpack-partials/web');

let platform =[];
const { desktop, mobile, web } = require('./webpack-partials/env-variables');
if (desktop) {
  platform = [];
  platform.push(main,renderer);
}

if (mobile) {
  platform = [];
  platform.push(cordova);
}

if(web) {
  platform = [];
  platform.push(back, front);
}

module.exports = platform;
```

Figure 16. Environment Variables Relevance in Webpack

In accordance with the figure above, all the Node environment variables are stored in Boolean value. If the variable for a platform exists as true, Webpack bundles the code for that particular platform. The production and development variables are passed in a platform specific build system setup.

5.2.1 Desktop App Development and Deployment Environment

Both of the Electron processes are bundled simultaneously by Webpack. Each of the processes are defined as an object in separate constant variables shown in Appendix 2. The entry property includes the entry file to be bundled along with reference to other files. The HMR is setup and is shown in the figure below.

```
entry: production ?
  [path.resolve(projectRoot, 'src', 'app-front')] :
[
  'webpack-dev-server/client?http://localhost:8080',
  'webpack/hot/only-dev-server',
  path.resolve(projectRoot, 'src', 'app-front')
],
```

Figure 17. HMR in Development Environment

As per figure 17, the ternary operator determines to use HMR in the development environment. The first index value of the array starts a web server with the entry point, and the second index provides hot reloading feature. (Webpack 2017f.) Webpack-dev-server is started with HMR in the development environment. The output is set to dist/desktop. Two Node global variables are disallowed, to avoid bugs in bundle. The target properties in main, and renderer variables guides Electron to bundle codes compatible to Electron. Most of the plugins, and modules are needed for all the platforms. Therefore, plugins and modules are defined in separate files, and referenced in each platform based on the requirements. Deployment to desktop executable App for end users is a straightforward process. Below are the commands to build desktop platform specific executables with electron-packager.

/package.json

```
"desktop:package:linux": "cd dist/desktop && npm run
package-linux",
"desktop:package:mac": "cd dist/desktop && npm run
package-mac",
"desktop:package:win": "cd dist/desktop && npm run
package-win",
```

dist/desktop/package.json

```
"package-linux": "electron-packager . --overwrite
--asar=true --platform=linux --arch=x64 --prune=true
--out=../../Electron-builds",
"package-mac": "electron-packager . --overwrite
--asar=true --platform=darwin --arch=x64  --prune=true
--out=../../Electron-builds",
"package-win": "electron-packager . --overwrite
--asar=true --platform=win32 --arch=ia32  --prune=true
--out=../../Electron-builds
--version-string.CompanyName=CE
--version-string.FileDescription=CE
--version-string.ProductName=\"Electrona-app\""
```

Figure 18. Electron Deployment CLI Commands

As illustrated in figure 18, there are two individual package.json provided in the root directory, and the dist/desktop directory, The reason for providing separate package.json is that electron-package includes all the files in the directory and sub-directories, where package.json is found, when compiling a deployable solution. Since, the electron-packager only concern dist/desktop directory, separate package.json is provided. The package.json file within Webpack-partials directory is copied to dist/desktop, when desktop:prepare is called from CLI. The ‘package.json’ includes electron, and electron-package packages. These two packages are also installed upon invocation of desktop:prepare as listed in the ‘package.json’ file as shown in Appendix 1. Packaging of all the platforms as executables is done from the root of the project, although the command is run within the context of dist/desktop. One caveat is that, executing desktop:package:mac in windows OS requires administration privilege.

The HTML Webpack Plugin is used to compile HTML and bundle with Webpack. JS and CSS does not require to be referenced during the

development or production stage. It automatically writes the appropriate JS, and CSS reference in the output file (Nicklas 2017). However, the electron development environment is operated within Webpack-dev-server. Referencing ‘index.js’ directly results in a static, HMR disabled app. In order to overcome the issue, the steps taken are shown in figure 19.

```
new HtmlWebPackPlugin({
  filename: 'index.html',
  template: './src/index.html',
  js: desktop && production ? '<script type="text/javascript" src="index.js"></script>'
    : '<script type="text/javascript" src="http://localhost:8080/index.js"></script>',
  link: desktop && production ? '<link rel="stylesheet" href="css/style.css">' : '<!---->',
  VueJS: ( mobile || web ) && development ? '<script type="text/javascript"
  src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.2/vue.js"></script>' : '<script
  type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.2/vue.min.js"></script>',
  inject: desktop ? false : true,
  hash: true,
  minify: {
    collapseWhitespace: production ? true : false
  }
}),
```

Figure 19. HTML Webpack Plugin for Desktop Platform

As shown in the figure above, automatic injection of JS, and CSS is disabled. The source of JS, and CSS are determined based on the BUILD environment variable. The development environment server runs on http://localhost:8080, which is why the js property declared in the figure above is prefixed by mentioned Uniform Resource Locator (hereinafter URL). Both the js, and css properties are declared within HTML Webpack Plugin and are used in the index.html shown in the figure below.

```
<!DOCTYPE html>
<html>

<head>
    <!--<meta http-equiv="Content-Security-Policy" content="default-src *; style-src
    'self' http://* 'unsafe-inline'; script-src 'self' http://* 'unsafe-inline'
    'unsafe-eval'"
    />-->

    <meta name="format-detection" content="telephone=no">
    <meta name="msapplication-tap-highlight" content="no">
    <meta name="viewport" content="user-scalable=no, initial-scale=1,
    maximum-scale=1, minimum-scale=1, width=device-width">
    <title>Web everywhere</title>
    <%= htmlWebpackPlugin.options.link %>
</head>
<body>
    <div class="app">
    </div>
    <%if(mobile) { %> <%= htmlWebpackPlugin.options.cordova = `<script
    type="text/javascript" src="cordova.js" />` %><% } %>
    <%if(desktop || mobile ) { %><%= htmlWebpackPlugin.options.js %> <% } %>
    <%if(desktop || mobile ) { %><%= htmlWebpackPlugin.options.VueJS %> <% } %>
</body>

</html>
```

Figure 20. Dynamic Attribute Injection in Index.html

The HTML Webpack Plugin supports embedded JavaScript syntax, in short EJS templates. The js, css, cordova, and VueJS properties from the HTML Webpack Plugin are used in the index.html, as shown in figure 20. This enables dynamically referencing any EJS variable declared in the HTML Webpack Plugin for platform specific code compilation.

The source codes representation for electronMain.js for Electron is shown in the figure below.

```
if (desktop) {
  const electron = require('electron');
  const { app, BrowserWindow } = electron;


  let mainWindow = null;

  app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') {
      app.quit();
    }
  });

  app.on('ready', () => {
    mainWindow = new BrowserWindow();
    mainWindow.loadURL(
      production ? `file://${__dirname}/index.html` : 'http://localhost:8080/'
    );
    mainWindow.webContents.on('new-window', (e) => {
      e.preventDefault();
    });
  });
```

Figure 21. Electron Boot Process in ElectronMain.js

The illustration of the electron boot-up process is depicted in figure 21. This is the minimum barebone to run Electron based Desktop App. Typically, Electron accesses the file protocol to get the HTML file in order to render UI. Since, the author intends to use the HMR capability in development mode, the server URL provided by the Webpack-dev-server is used.

5.2.2 Mobile and Web App Development and Deployment Environment

As discussed in earlier chapter, running the ‘prepare’ CLI command scaffolds a project structure. The ‘prepare’ command scaffolds mobile App DnD with the help of clean command is shown in the figure below.

```
"mobile:prepare": "npm run clean && phonegap create mobile
org.apache.cordova.mobile mobile && node movePhoneGap.js && phonegap platform
add android@6.2.2 browser && rimraf typings mobile README.md CONTRIBUTING.md
www/css www/js www/spec www/spec.html && npm run mobile:build",
```

Figure 22. PhoneGap Scaffolding for Mobile Platform

Based on figure 22, it is understood that if any directory concerning PhoneGap exists, is removed prior to the extraction of a PhoneGap project template. Script in movePhoneGap.js moves all the files and subdirectories of the generated PhoneGap template to the root of the project based on the OS, and is shown in Appendix 3. The whole build system is a scaffold of PhoneGap as seen in the figure below.

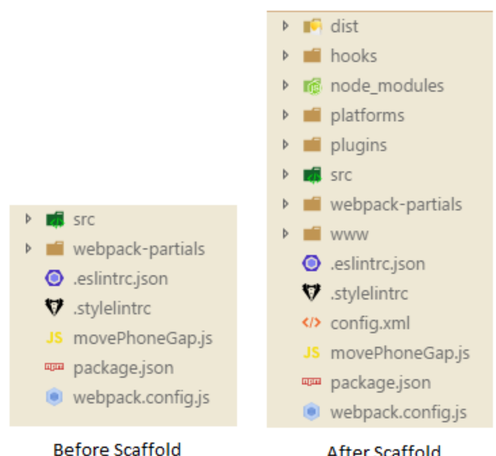

Figure 23. Before and after PhoneGap Template Implementation

The figure 23 shows that the hooks, platforms, plugins directories and the config.xml file comprises of PhoneGap's core mechanism. This research is to ignore discussing the constructs of PhoneGap architecture, and moves on with the implementation phase of PhoneGap only. Currently Android OS for mobile is added, as the author uses both Linux, and Windows based environment where iOS platform is not supported by PhoneGap for development. It should be noted that, Apple's MacOS is the only OS platform for iOS app development. PhoneGap does not have support for the webpack-dev server, and files need to be written in the secondary storage, in order to carry out in-App development. One bug encountered during the mobile platform build system setup is that, running Node server for backend services, and mobile development related scripts concurrently breaks node server to operate. Running Node server after

mobile development related scripts is recommended, although re-executing mobile platform specific commands results in the break of the Node server.

Executing the mobile:deploy command deploys an apk solution in \platforms\android\build\outputs\apk directory. The bundling process of source codes is shown in the Appendix 4. It shares almost identical codebase for web build setup with the exception of the static input, and output sources.

The web App DnD environment is straightforward and almost identical as the Desktop renderer setup process shown in Appendix 5. The only exclusion is the target property that builds a bundle for client side of the web platform. For the back-end, the target property is node, as Node itself is server-side JS environment. The back-end environment does not have the HMR capability, as Webpack-dev-server is meant for client side development. The back-end concentrates on developing a server, and all the server side functionalities. Common plugins and modules are an overkill for back-end development, thereby custom scripts which only bundle JS core files from src/server.js are included in the webpack-partials/web.js file.

5.2.3 Plugins and loaders

All the platforms use multiple plugins. Appendix 6 represents the script of the plugins implementation for all the platforms. All the plugins are included in an array, in the plugins variable. DefinePlugin, a Webpack core plugins is common to all platform builds. DefinePlugin enables to use Node environment variables globally in all the platform app development environments (Webpack 2017g). Platform specific codes that are irrelevant to other platforms are needed to be removed. The use of Node environment variables on in-app development is needed to determine the removal of unnecessary codes. During the production stage, all Sass, and CSS files are bundled and output into secondary storage by extract-text-Webpack-plugin (Webpack-contrib 2017b). One other plugin used in production environment is the babili-Webpack-plugin to provide code minification. HTML Webpack plugin is used in all the platforms, referencing

three different html files discussed in section 5.2.2. The plugin is needed for both production, and development build. Stylelint configuration file is provided to follow a Stylint Sass guidelines of linting Sass, or CSS in the development environment (Jankord 2017). The Webpack-dev-server serves files from in-memory that sits on top of HTML. It does not detect changes of HTML file. The write-file-Webpack-plugin provides bundling in the Webpack-dev-server context, forcing Webpack-dev-server to reload due to restoration of JS entry file. (Kuizinas 2017.) The Friendly-errors-webpack-plugin “recognizes certain classes of webpack errors and cleans, aggregates and prioritizes them to provide a better Developer Experience” (Warin 2017).

The loaders are used under the ‘module.rules’ property and is shown in the desktop.js, mobile.js, and web.js files. The ‘rules’ property is an array that holds all the filetypes to be tasted under one or more loaders. All the loaders are provided in module.js and is shown in Appendix 8. Babel tests and bundles all the JS files for all the platforms. It uses Babel-preset-env for both production and development build to compile ES7 and the other standards. The Babel-preset-env uses the compat-table by Zaytsev (2017) to determine the JS engine of the given environment to compile into correct JS format (Babel 2017). All the Vue files are loaded to vue-loader and passed to Babel to compile into JS codes. The image loader, as discussed optimize and reduces image size. However, if the image is compact, the url-loader is used to include image inline as data URL to avoid calling as external resource (Webpack-contrib 2017c). The url-loader is also used for different kinds of font to be used as data URL. The linting of JS is necessary during development. During the development stage, the eslint is used for linting. All the linting of JS needs to be done before bundling of JS code. The ‘enforce’ property of eslint-loader precompiles the JS with linting, and warns in the CLI if there are any errors. The bundling process is stopped if there are any errors, correcting the errors enables Webpack to continue bundling. Sass is compiled to CSS by sass-loader, the compiled CSS, or raw CSS are bundled by css-loader. The style-loader injects CSS in the head of HTML during development. The css-loader minifies and bundles CSS. During the production stage, PostCSS is used to specifically load the css-next plugin. The css-next plugin is used for auto prefixing CSS for different browsers

specifically. However, a wide range of functionalities are enabled by default with css-next and is available in the official documentation. (Thirouin 2017.) Since, the Sass and CSS files used in Vue templates are needed to compile separately by vue-loader, all the necessary loaders information are passed into Vue option property.

6 PROTOTYPE BENCHMARK

The focus of this chapter is on elaborating the integrity of the prototype delivered. section 1 details a sample in-app workflow based on the prototype to showcase the frameworks implementation, and the features' availability determined in the chapter 5. section 2 completes the chapter by outlining the differences between source codes and outputted bundled codes across platforms. In addition, the known issues developed by the author are pointed out.

6.1 Features demonstration

The code splitting is available by using the 'System.import(module_location)'. Below is sample representation of the features applied.

Figure 24. Code Splitting Sample Codebase

As shown in the figure above, only clicking on a button element will trigger the import of an image file. The bundling is split for runtime efficiency. The image file is only loaded to a platform when the click event is triggered. Knowledge of

the concept of JS Promise is required to implement code splitting. The tree shaking feature is also available on the codebase shown in Figure 24. The code base is true for the web platform. Bundling for other platform results in discarding the above codebase as a whole in the JS. It needs to be noted that all the properties declared in the webpack-partials/env-variables are available on every platform, in every context as global variable. Therefore, the web property is available in the index.js file.

Error detection with ESlint and Stylelint is shown in the figure below.

```
C:\Users\█████Documents\WEB_DEV\wsm\src\index.js
  35:5   error  'web' is not defined    no-undef
  39:46  error  Missing semicolon       semi

✕ 2 problems (2 errors, 0 warnings)
```

ESlint error detection

```
body {
  padding: 5px;
  margin: 2%;
  background: red
}
```

```
src/css/index.scss
 8:3   ×  Expected margin to come before padding
 9:3   ×  Expected background to come before margin
 9:15  ×  Unexpected named color "red"
 9:17  ×  Expected a trailing semicolon
```

Stylelint error detection

Figure 25. Linting Features for JS and CSS/SASS

The error detection feature provides hinting of the type of error, and a reference pointer of the source. However, restriction such as a missing semicolon is optional. Both the eslintrc.json, and .stylelintrc files include the guideline for the JS and CSS filetypes. Restrictions can be customised according to users' needs. ESlint detects 'web' as an undefined variable. ESlint detects it as global by adding web in the ESlint rules as shown in figure 25. The sample codebase to illustrate the above features are excluded from the delivered prototype.

6.2 Files Comparison

The App is a basic 'hello-world' app, where in-app functionality is not given any concern. In brief, the App itself implements Eventful API, a third-party API to list all the events of given location. The location is determined by extracting users' internet protocol address information by using IP-API. The main.js file located in the src/Components directory is the UI entry point of the application and is

referenced in the src/index.js. A RESTful API server is developed to serve Eventful API, and is located in the src/server.js.

The desktop platform does not require bundling of any third-party libraries available in NPM registry, as Electron implements Node in its core. The same rules apply for the back-end platform. The mobile and front-end platforms need to bundle third party libraries along with custom codebase. All the platforms have dedicated port in localhost. The front-end, back-end, desktop, and mobile URL are 80, 5000, 8080, and 3003 accordingly.

The dead-code elimination process is undertaken in all the platforms. However, due to the simplicity of the server-side, and electron main process bundling, the result output does not show significant change due to the lack of extensive codebase. For the sake of demonstration, ‘vue-material’ library is used. The compressed size of the ‘vue-material’ package is 10 MB. Comparison between the src directory files and dist/web is shown in the illustration below.

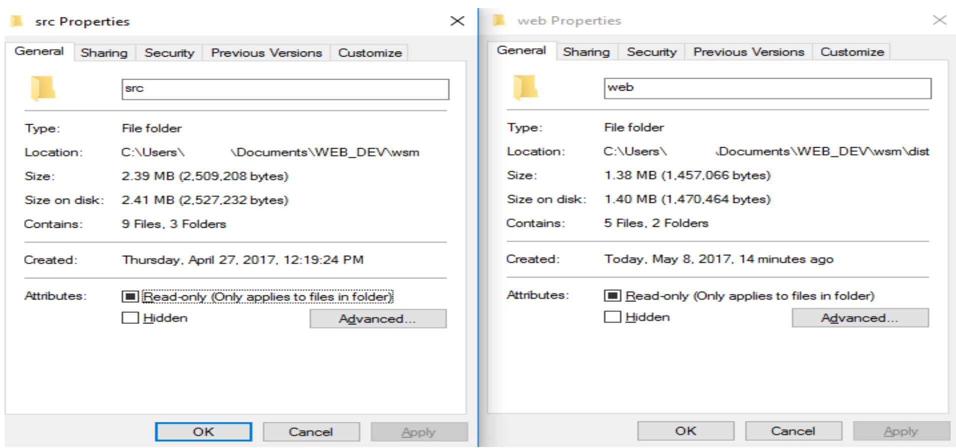

Figure 26. Size Comparison Between src and dist/web

The bundled codebase is significantly less in size compared to the source files inside src directory. However, the confusion may arise after looking into the JS files inside the src directory, where all the file sizes are between the range of 1KB to 2KB. The reason for a significant large bundled file is due to the fact of referencing both the Vue and ‘vue-material’ library. The source image file is 2.38MB, and the outputted compressed format is 1.11MB. It is possible to reference CSS files in the JS files for front-end, and mobile platform.

7 CONCLUSION

The thesis study achieved all the objectives by studying, and implementing the concept of the ubiquitous WSM across different platforms. The implementation process is the follow up of a rigorous study of publicly available questionnaire and survey results. During the practical implementation part of the thesis study, various programming frameworks and utilities are studied and the ones fit for the project are implemented.

The theoretical part widened the understanding of the concept of the ubiquitous WSM. Moreover, JS was studied as a mediator between different platforms, and package distribution system. With the primary emphasis of implementing the WSM model across multiple platforms, Webpack was studied to automatize the in-app DnD work flow and to provide the features agreed upon based on the findings of the questionnaire and surveys. The effective integration of Platform specific frameworks depends on the knowledge of Webpack rather than the frameworks themselves. Thus, the importance of studying Webpack was prioritized during the course of development of the prototype.

The stability of the prototype is given the maximum priority, although the tasks of integrating different tools under the same umbrella remained challenging up until the delivery of the prototype. The wide ranges of tools' available do not always mean quick and easy integration. Lack of documentation, and compatibility issues hardened integration of tools to the prototype. Besides, interoperability was not straightforward. Referencing third-party modules was different among the platforms.

The time frame constrained the optimization of the codebase of the prototype developed, as repetitive codes were found. Besides, bugs for one platform were developed unintentionally when developing solution for another platform. The realization of the bugs' presence was not noticed till the execution of bundling for the bugs affected platform. Debugging process is time-consuming, and do not always point at the core of the problems.

In spite of all the challenges, authors personal endeavour to optimise the projects’ codebase, and decision to make the project available as an open source software will help expand the project. The recent revolution in the JS community, thanks to the Node environment, creates the possibility of expanding platforms’ availability for the WSM model. Future research on integrating this prototype with future platforms i.e. the Internet of things, web virtual reality, and augmented reality has great potential. Nonetheless, Webpack as the heart of the project should be studied further if one is interested in the field of web technology, specifically in the build system for web application.

BIBLIOGRAPHY

Adams C. R. 2015. Mastering JavaScript High Performance. Birmingham, UK: Packt Publishing Ltd. Accessed 31 March 2017 http://pepa.holla.cz/wp-content/uploads/2016/08/Mastering-JavaScript-High-Performance.pdf.

Aronson, L. 2011. HTML Manual of Style. A Clear, Concise Reference for Hypertext Markup Language (Including HTML5). 4th Edition. Crawfordsville, IN: Pearson Education, Incorporated. Ebook. Accessed 29 March 2017 http://3f363218c54d9808ec8.c.it-ebooks.directory/e-books/addison-wesley/Addison.Wesley.HTML.Manual.Of.Style.4th.Edition.Oct.2010.ISBN.0321712080.pdf?l=bfF57TeElTxQZcicqHBquQ&t=1490861145.

Babel 2017. Readme.md hosted in GitHub. Accessed 31 March 2017 https://github.com/Babel/Babel.

Babel 2017a. Babel-core. Accessed 14 April 2017 https://github.com/Babel/Babel/tree/master/packages/Babel-core.

Babel 2017b. Babili. Accessed 14 April 2017 https://github.com/Babel/babili.

Bigio, M. 2016. Introducing Hot reloading. Accessed 2 April 2017 https://facebook.github.io/react-native/blog/2016/03/24/introducing-hot-reloading.html.

Bos, B. 2017. WHAT IS CSS? Cascading Style Sheets. Accessed 13 March 2017
https://www.w3.org/Style/CSS/Overview.en.html.

Bridgwater, A. 2017. What's the Difference Between a Software Product and a Platform? Accessed 22 February 2017 http://www.forbes.com/sites/adrianbridgwater/2015/03/17/whats-the-difference-between-a-software-product-and-a-platform/#5fa01cdc3877.

Budd, A & Björklund, E. 2016. CSS Mastery. Advanced Web Standards Solutions. Third edition. New York, NY: Springer Science+Business Media, LLC. Ebook. Accessed 13 April 2017 https://books.google.fi/books?isbn=1430258640.

Bureau of Labor Statistics 2015, U.S. Department of Labor. Occupational Outlook Handbook, 2016-17 Edition, Software Developers 2015. Accessed 22 February 2017 https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm.

Burka, B. 2015. HTML5 - RESPONSIVE WEB DEVELOPMENT. Bachelor’s Thesis at Turku University of Applied Sciences. Accessed 12 March 2017 http://theseus32-kk.lib.helsinki.fi/bitstream/handle/10024/95710/HTML5%20-%20Responsive%20Web%20Development.pdf?sequence=1.

comScore 2014. The U.S.Mobile App Report. Accessed 29 February 2017 http://www.comscore.com/layout/set/popup/Request/Presentations/2014/The-US-Mobile-App-Report-Whitepaper?req=slides&pre=The+U.S.+Mobile+App+Report.

Cordova 2015. Apache 6.x Documentation, Introduction. Accessed 2 March 2017 https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

Cook, C & Garber, J. 2012. Foundation HTML5 with CSS3. New York, NY: Springer Science+Business Media, LLC. Ebook. Accessed 2 April 2017 https://books.google.fi/books?id=9XUUOdt4R5QC&pg=PA378&dq=vendor+prefixes&hl=en&sa=X&ved=0ahUKEwjOqfeKmobTAhWGJJoKHZ-xAcAQ6AEIIjAB#v=onepage&q=vendor%20prefixes&f=false.

Coopman, T. 2017. Image-Webpack-loader. Accessed 14 April 2017 https://github.com/tcoopman/image-Webpack-loader.

Crnkovic, D. 2010. Constructive Research and Info-Computational Knowledge Generation. Accessed 25 March 2017 http://www.mrtc.mdh.se/~gdc/work/MBR09ConstructiveResearch.pdf.

Dahl, R. 2010. Deep inside Node.js with Ryan Dahl. Interview with Ryan Dahl by Dio Synodinos. InfoQ video. Accessed 31 March 2017 https://www.infoq.com/interviews/node-ryan-dahl.

Deveria, A. 2017a. Can I use? Accessed 29 March 2017 http://caniuse.com/#search=html5.

Deveria, A. 2017a. Can I use? Accessed 29 March 2017 http://caniuse.com/#search=CSS3.

Digital Trends 2017. Battle of the browsers: Edge vs. Chrome vs. Firefox vs. Opera vs. Vivaldi. Accessed 29 March 2017 http://www.digitaltrends.com/computing/best-browser-internet-explorer-vs-chrome-vs-firefox-vs-safari-vs-edge/2/.

Ecma International 2015. Standard ECMA-262 7th Edition / June 2015. ECMAScript® 2016 Language Specification. Accessed 14 March 2017 http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

Fisher, T. 2016. List of Executable File Extensions. Accessed 14 March 2017 https://www.lifewire.com/list-of-executable-file-extensions-2626061.

Flanagan, D. 2011. JavaScript: The Definitive Guide. 6th edition. CA: O’Reilly, 1-2. Ebook. Accessed 14 March 2017 http://www.stilson.net/documentation/javascript.pdf.

Forni, A. & Meulen, R. 2017. Gartner Forecasts Flat Worldwide Device Shipments Until 2018 1/2017. Accessed 21 February 2017 http://www.gartner.com/newsroom/id/3560517.

Greif, S. 2017a. Front-end Frameworks. The state of JavaScript 2016. Accessed 20 March 2017
http://stateofjs.com/2016/frontend/.

Greif, S. 2017b. Features. The state of JavaScript 2016. Accessed 20 March 2017
http://stateofjs.com/2016/features/.

Haverbeke, M. 2014. Eloquent JavaScript. A Modern Introduction to Programming, 7. Ebook. Accessed 14 March 2017
http://eloquentjavascript.net/Eloquent_JavaScript.pdf.

Hyötyläinen, R., Häkkinen, K., & Uusitalo, K. 2014. The Constructive Approach as a Link Between Scientific Research and The Needs of Industry. Accessed 08 March 2017
https://www.researchgate.net/publication/279183260_THE_CONSTRUCTIVE_APPROACH_AS_A_LINK_BETWEEN_SCIENTIFIC_RESEARCH_AND_THE_NEEDS_OF_INDUSTRY.

Ihrig, C. J. 2013, 29. Pro Node.js for Developers, 29. Ebook. Accessed 19 March 2017
https://books.google.fi/books?id=FZcQAwAAQBAJ&pg=PA29&dq=node+event+loop&hl=en&sa=X&ved=0ahUKEwj02Y2e5OLSAhWBhSwKHW_cBI4Q6AEIGDAA#v=onepage&q=node%20event%20loop&f=false.

Jankord, B. 2017. stylelint-config-sass-guidelines. Accessed 16 April 2017
https://github.com/bjankord/stylelint-config-sass-guidelines.

Kuizinas, G. 2017. write-file-Webpack-plugin. Accessed 16 April 2017
https://github.com/gajus/write-file-Webpack-plugin.

Lane, J. 2007. The web standards model - HTML CSS and JavaScript. Accessed 22 February 2017
https://www.w3.org/community/webed/wiki/The_web_standards_model_-_HTML_CSS_and_JavaScript.

LeRoux, B. 2012. PhoneGap, Cordova, and what’s in a name? Accessed 27 April 2017
http://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name.

Lehtiranta, L. & Juha-Matti, J. & Kärnä, S. & Pekuri, L. 2017. The Constructive Research Approach: Problem Solving for Complex Projects. Accessed 08 March 2017
http://www.gpmfirst.com/books/designs-methods-and-practices-research-project-management/constructive-research-approach.

Long, J 2012. I Don’t Speak Your Language: Frontend vs. Backend. Accessed 31 March 2017
http://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend.

Lord, J. 2016. Building cross-platform apps with Electron - GitHub Satellite 2016. YouTube video. Accessed 2 April 2017
https://www.youtube.com/watch?v=WVb2OD49pUA&t.

Meloni, J. C. 2012. Sams Teach Yourself HTML, CSS, and JavaScript All in One. Pearson Education, Inc. Ebook. Accessed 29 March 2017
http://53dcc0239cfababa791.c.it-ebooks.directory/e-books/sams/Sams.Teach.Yourself.HTML.CSS.And.JavaScript.All.In.One.Dec.2011.ISBN.0672333325.pdf?l=dnSbXD6W75Z626ne57qJHw&t=1490860922.

Merriam-Webster Dictionary 2017. Definition of markup language. Accessed 28 February 2017
https://www.merriam-webster.com/dictionary/markup+language.

Microsoft 2012. The MVVM Pattern. Accessed 28 February 2017
https://msdn.microsoft.com/en-us/library/hh848246.aspx.

Microsoft Developer Network 2017. Solutions and Projects in Visual Studio. Visual Studio 2015. Accessed 14 March 2017
https://msdn.microsoft.com/en-us/library/b142f8e7.aspx.

MobiDev 2015. Cross-Platform Development for Desktops: Choosing the Right Technology. Accessed 20 March 2017
https://mobidev.biz/blog/cross-platform_development_for_desktops_choosing_the_right_technology.

Node.js 2017. Node.js v7.8.0 Documentation. Core Modules. Accessed 25 March 2017
https://nodejs.org/dist/latest-v7.x/docs/api/modules.html#modules_core_modules.

Nicklas, J. 2017. HTML Webpack Plugin. Accessed 15 April 2017
https://github.com/jantimon/html-Webpack-plugin.

NPM 2017a. npm-init. Interactively create a package.json file. Accessed 20 March 2017
https://docs.npmjs.com/cli/init.

NPM 2017b. npm-scripts. How npm handles the "scripts" field. Accessed 20 March 2017
https://docs.npmjs.com/misc/scripts.

Owens, J. 2017. Mobile Frameworks. The state of JavaScript 2016. Accessed 20 March 2017
http://stateofjs.com/2016/mobile/.

Panhale, M. 2016. Beginning Hybrid Mobile Application Development. New York, NY: Springer Science+Business Media, LLC. Ebook. Accessed 28 March 2017
https://books.google.fi/books?id=GqtPCwAAQBAJ&pg=PA15&dq=native+vs+hybrid+mobile+app&hl=en&sa=X&ved=0ahUKEwih17ucvoDTAhUsJcAKHSP2CL4Q6AEIGDAA#v=onepage&q=apress&f=false.

Peschla, J. 2012, 9. Information flow tracking for JavaScript in Chromium. Master's Thesis at University of Kaiserslautern. Accessed 19 March 2017 https://kluedo.ub.uni-kl.de/files/3442/thesis.pdf.

PostCSS 2017. PostCSS. Accessed 14 April 2017 https://github.com/postcss/postcss.

Puputti 2012. Mobile HTML5: Implementing a Responsive Cross-Platform Application. Bachelor's Thesis at Aalto University. Accessed 12 March 2017 http://lib.tkk.fi/Dipl/2012/urn100643.pdf.

Rambeau, M. 2017. 2016 JavaScript Rising Stars. Accessed 20 March 2017 https://risingstars2016.js.org/.

Sandoval, K. 2016. What is the Difference Between an API and an SDK? Accessed 29 March 2017 http://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/.

Schmitt, C., Blessing, K., Cherny, R., Evans, M. K., Lawver, K. & Trammell, M. 2008. Adapting to Web Standards: CSS and Ajax for Big Sites. Berkeley, CA: New Riders. Ebook. Accessed 28 March 2017 https://archive.org/download/AdaptingToWebStandardsCSSAndAjaxForBigSites/Adapting%20to%20Web%20Standards%20'CSS%20and%20Ajax%20for%20Big%20Sites'.pdf.

Schultz, D. & Cook, C. 2007. Beginning HTML with CSS and XHTML – Modern Guide and Reference. Berkeley: CA: Apress, a Springer Nature company. Ebook. Accessed 29 March 2017 http://1114668b3938c2bd4ea.d.it-ebooks.directory/Apress.HTML.Jun.2007.ISBN.1590597478.pdf?l=UHhuPY_wVdpHUIffi_G4Zg&t=1490860416.

Seddon, R. 2012. Introduction to JavaScript Source Maps. Accessed 13 April 2017 https://www.html5rocks.com/en/tutorials/developertools/sourcemaps/.

Seth, G. & Foresti, A. 2016. Microsoft Edge's JavaScript engine to go open-source. Accessed 30 March 2017 https://blogs.windows.com/msedgedev/2015/12/05/open-source-chakra-core/#WiRuW5zA20DhQM8Q.97.

Sikos, L. 2011. Web Standards—Mastering HTML5, CSS3, and XML. New York, NY: Springer Science+Business Media, LLC. Ebook. Accessed 28 March 2017 https://doc.lagout.org/programmation/tech_web/Web%20Standards.pdf.

Spencer, L & Richards, S 2015. Reliable JavaScript: How to Code Safely in the World's Most Dangerous Language. Indianapolis, IN: John Wiley & Sons, Inc. Ebook. Accessed 13 April 2017 https://books.google.fi/books?id=thezCQAAQBAJ&pg=PA74&dq=what+is+code+linting&hl=en&sa=X&ved=0ahUKEwjuuYy2-

KHTAhXB1ywKHSdsB6kQ6AEIITAB#v=onepage&q=what%20is%20code%20linting&f=false.

Stack Overflow 2017. Developer Survey Results 2017. Accessed 25 March 2017
https://stackoverflow.com/insights/survey/2017.

Shklar, L. & Rosen, R. 2003, 201-202. Web application architecture: principles, protocols, and practices. Chichester, West Sussex: John Wiley & Sons Ltd. Ebook. Accessed 19 March 2017
http://www.farooka.com/pervez/files/courses/12-WebApplication.pdf.

TechTerms 2017. RTE Definition. Accessed 10 March 2017
https://techterms.com/definition/rte.

Teixeira, P. 2013. PROFESSIONAL Node.js®. BUILDING JAVASCRIPT-BASED SCALABLE SOFTWARE. Indianapolis, IN: John Wiley & Sons, Inc. Ebook. Accessed 19 March 19, 2017
http://htchttp.s3.amazonaws.com/books/professional_node.js.pdf.

Thirouin, M. 2017. postcss-cssnext. Accessed 16 April 2017
https://github.com/MoOx/postcss-cssnext.

Tommelein, I.D. 2008. ‘Poka yoke’ or quality by mistake proofing design and construction systems. Accessed 13 April 2017
http://p2sl.berkeley.edu/wp-content/uploads/2016/04/Tommelein-2008-Poka-Yoke-or-Quality-by-Mistake-Proofing-Design-and-Construction-Systems.pdf.

Vlăsceanu, M. 2012. Zend Framework functionality and web application methodology. Bachelor’s Thesis at Kemi-Tornio University of Applied Sciences. Accessed 14 March 2017
http://www.theseus.fi/bitstream/handle/10024/51741/zend_methodology_vlasceanu_mihai_2012.pdf?sequence=1.

Vue.js 2017a. Introduction. What is Vue.js? Accessed 21 March 2017
https://vuejs.org/v2/guide/.

Vue.js 2017b. Comparison with Other Frameworks. Accessed 21 March 2017
https://vuejs.org/v2/guide/comparison.html.

Warin, G. 2017. Friendly-errors-webpack-plugin. Accessed 6 April 2017
https://github.com/geowarin/friendly-errors-webpack-plugin.

W3C 2001. Introduction to CSS3. W3C Working Draft, 23 May 2001. Accessed 13 March 2017
https://www.w3.org/TR/2001/WD-css3-roadmap-20010523/.

W3C 2016. HTML 5.1. W3C Recommendation, 1 November 2016. Accessed 12 March 2017
https://www.w3.org/TR/html/introduction.html#background.

Webpack 2015. Internal Webpack plugins. Accessed 15 April 2017
https://also.github.io/Webpack/internal-plugins/#NodeTargetPlugin.

Webpack 2017a. Guides. Accessed 31 March 2017
https://Webpack.js.org/guides/.

Webpack 2017b. Guides. Code splitting. Accessed 2 April 2017
https://Webpack.js.org/guides/code-splitting/.

Webpack 2017c. Concepts. Accessed 31 March 2017
https://Webpack.js.org/concepts/.

Webpack 2017d. Concepts. Hot Module Replacement. Accessed 31 March 2017 https://Webpack.js.org/concepts/https://Webpack.js.org/concepts/hot-module-replacement/.

Webpack 2017e. DevServer. Accessed 13 April 2017
https://Webpack.js.org/configuration/dev-server/.

Webpack 2017f. DefinePlugin. Accessed 15 April 2017
https://Webpack.js.org/plugins/define-plugin/.

Webpack 2017g. Concepts. Hot Module Replacement - React. Accessed 14 April 2017
https://Webpack.js.org/guides/hmr-react/#components/sidebar/sidebar.jsx.

Webpack-contrib 2017a. File-loader. Accessed 14 April 2017
https://github.com/Webpack-contrib/file-loader.

Webpack-contrib 2017b. extract-text-Webpack-plugin. Accessed 16 April 2017
https://github.com/Webpack-contrib/extract-text-Webpack-plugin.

Webpack-contrib 2017c. url-loader. Accessed 16 April 2017
https://github.com/Webpack-contrib/url-loader.

White, A. 2009. JavaScript Programmer's Reference, 12-13. Ebook. Accessed 18 March 2017
https://books.google.fi/books?id=XJrXl71TITIC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q=engine&f=false.

Williams, D. 2009. Overview of Build Systems. Accessed 31 March 2017
https://www.cs.virginia.edu/~dww4s/articles/build_systems.html.

Wong, J. 2017. Build Tools. Accessed 31 March 2017
http://stateofjs.com/2016/buildtools/.

Wright, T. 2013, 29-30. Learning JavaScript: a hands-on guide to the fundamentals of modern JavaScript. Upper Saddle River, New Jersey. Pearson Education, Inc. Ebook. Accessed 19 March 2017
http://eureka.com.ve/libros/javascript/Addison.Wesley.Learning.JavaScript.Aug.2012.ISBN.0321832744.pdf.

You, E. 2016. Vue js the Progressive Framework - Evan You. Speech at 2016 UtahJS Conference. YouTube video. Accessed 31 March 2017
https://www.youtube.com/watch?v=pBBSp_iIiVM.

Zaytsev, J. 2017. ECMAScript 2016+ compatibility table. Accessed 14 March 2017
http://kangax.github.io/compat-table/es2016plus/#experimental-flag-note.

APPENDICES

Appendix 1. package.json

Appendix 2. webpack-partials/desktop.js

Appendix 3. movePhoneGap.js

Appendix 4. webpack-partials/mobile.js

Appendix 5. webpack-partials/web.js

Appendix 6. webpack-partials/plugins.js

Appendix 7. webpack-partials/module.js

package.json Appendix 1 1(2)

```
{
  "name": "electron-demo",
  "main": "./dist/desktop/main.js",
  "author": "Majedul Hoque",
  "scripts": { "clean": "rimraf www Electron-builds hooks platforms
plugins typings config.xml mobile npm-debug* README.md typings",
    "desktop": "cd dist/desktop && npm run electron",
    "desktop:deploy": "npm-run-all -s desktop:prepare desktop:package:win
desktop:package:linux desktop:package:mac",
    "desktop:dev": "npm-run-all -p desktop:watch desktop",
    "desktop:package:linux": "cd dist/desktop && npm run package-linux",
    "desktop:package:mac": "cd dist/desktop && npm run package-mac",
    "desktop:package:win": "cd dist/desktop && npm run package-win",
    "desktop:prepare": "rimraf ./dist/desktop && copyfiles -u 1
./webpack-partials/package.json ./dist/desktop/ && cd dist/desktop && npm
install && cd../.. && cross-env PLATFORM=desktop BUILD=production
webpack",
    "desktop:prod": "npm-run-all -s desktop:prepare desktop",
    "desktop:watch": "cross-env PLATFORM=desktop BUILD=development
webpack-dev-server",
    "mobile:build": "cross-env PLATFORM=mobile BUILD=production webpack",
    "mobile:deploy": "phonegap build android",
    "mobile:dev:android": "npm-run-all -p mobile:watch
mobile:run:android",
    "mobile:dev:browser": "npm-run-all -p mobile:watch
mobile:run:browser",
    "mobile:prepare": "npm run clean && phonegap create mobile
org.apache.cordova.mobile mobile && node movePhoneGap.js && phonegap
platform add android@6.2.2 browser && rimraf typings mobile README.md
CONTRIBUTING.md www/css www/js www/spec www/spec.html && npm run
mobile:build",
    "mobile:prod": "npm-run-all -s mobile:prepare mobile:build",
    "mobile:run:android": "phonegap run android --device -- --live-
reload",
    "mobile:run:browser": "phonegap serve -p 3003",
    "mobile:watch": "cross-env PLATFORM=mobile BUILD=development webpack
--watch",
    "prepare": "npm-run-all -s desktop:prepare web:prod mobile:prepare",
    "web:dev:back": "npm-run-all -p web:webpack:build:watch
web:back:server:watch",
    "web:back:server:watch": "supervisor ./dist/web/server.js",
    "web:dev": "npm-run-all -p web:dev:front web:dev:back",
    "web:dev:front": "cross-env PLATFORM=front BUILD=development webpack-
dev-server",
    "web:prod": "rimraf ./dist/web && cross-env PLATFORM=web
BUILD=production webpack",
```

```
"web:webpack:build:watch": "cross-env PLATFORM=back BUILD=production
webpack --watch"
  },
  "devDependencies": {
    "Babel-core": "^6.24.1",
    "Babel-loader": "^7.0.0",
    "Babel-polyfill": "^6.23.0",
    "Babel-preset-babili": "^0.0.12",
    "Babel-preset-env": "^1.4.0",
    "babili-webpack-plugin": "^0.0.11",
    "copyfiles": "^1.2.0",
    "cross-env": "^4.0.0",
    "css-loader": "^0.28.1",
    "eslint": "^3.19.0",
    "eslint-config-google": "^0.7.1",
    "eslint-config-vue": "^2.0.2",
    "eslint-loader": "^1.7.1",
    "eslint-plugin-vue": "^2.0.1",
    "extract-text-webpack-plugin": "^2.1.0",
    "file-loader": "^0.11.1",
    "friendly-errors-webpack-plugin": "^1.6.1",
    "googleapis": "^19.0.0",
    "html-webpack-plugin": "^2.28.0",
    "image-webpack-loader": "^3.3.0",
    "node-sass": "^4.5.2",
    "npm-run-all": "^4.0.2",
    "postcss": "^6.0.1",
    "postcss-cssnext": "^2.10.0",
    "postcss-load-config": "^1.2.0",
    "postcss-loader": "^1.3.3",
    "postcss-reporter": "^3.0.0",
    "purifycss-webpack": "^0.6.1",
    "resolve-url-loader": "^2.0.2",
    "rimraf": "^2.6.1",
    "sass-loader": "^6.0.3",
    "style-loader": "^0.17.0",
    "stylelint-config-sass-guidelines": "^2.0.0",
    "stylelint-processor-html": "^1.0.0",
    "stylelint-webpack-plugin": "^0.7.0",
    "url-loader": "^0.5.8",
    "vue-loader": "^12.0.3",
    "vue-template-compiler": "^2.3.2",
    "webpack": "^2.5.1",
    "webpack-dev-server": "^2.4.5",
    "webpack-node-externals": "^1.5.4",
    "write-file-webpack-plugin": "^4.0.2"
  },
```

Appendix 1 3(3)

```
  "dependencies": {
    "body-parser": "^1.17.1",
    "cors": "^2.8.3",
    "express": "^4.15.2",
    "glob": "^7.1.1",
    "node-fetch": "^1.6.3",
    "onsenui": "^2.2.6",
    "supervisor": "^0.12.0",
    "vue": "^2.3.2",
    "vue-onsenui": "^2.0.0-beta.4"
  }
}
```

webpack-partials/desktop.js Appendix 2 1(2)

```
const path = require('path');
const projectRoot = path.resolve(__dirname).replace('webpack-partials',
'');
const nodeExternals = require('webpack-node-externals');
const plugins = require('./plugins');
const rules = require('./module');
const { production, development } = require('./env-variables');
const {  resolveLoader } = require('./externals');
const renderer = {
  resolveLoader,
  externals: production ?
  [nodeExternals(
    {
      whitelist: [/*'jquery', /^lodash/*/]
    }
    )] : {} ,
  devtool: development ? 'eval' : false,

  entry: production ?
    [path.resolve(projectRoot, 'src', 'index')] :
  [
    'webpack-dev-server/client?http://localhost:8080',
    'webpack/hot/only-dev-server',
    path.resolve(projectRoot, 'src', 'index')
  ],
  devServer: {
    hot: production ? false : true
  },
  output: {
    path: path.resolve(projectRoot, 'dist', 'desktop'),
    publicPath: production ? '' : 'http://localhost:8080/',
    filename: 'index.js'
  },
  target: 'electron-renderer',
  node: {
    __dirname: false,
    __filename: false
  },
  plugins: plugins,

  module: {
    rules: rules
  },
  cache: true
};
```

Appendix 2 2 (2)

```
const main = {
  externals: production ? [nodeExternals(
    {
      whitelist: [/*'jquery', /^lodash/*/]
    }
  )] : {},
  entry: [path.resolve(projectRoot, 'src', 'electronMain')],
  output: {
    path: path.resolve(projectRoot, 'dist', 'desktop'),
    publicPath: '',
    filename: 'main.js'
  },
  target: 'electron-main',
  node: {
    __dirname: false,
    __filename: false
  },
  module: {
    rules: rules
  },
  plugins: plugins
};

module.exports = { renderer, main };
```

movePhoneGap.js Appendix 3

```
'use strict';
/*eslint no-console: 0*/
const exec = require('child_process').exec;

if (process.platform === 'linux' || process.platform === 'darwin') {
  exec(['mv ./mobile/* ./'], (err, out, stdErr) => {
    if (err) console.log(stdErr);
  });
}

if (process.platform.match(/^win/)) {
  exec('xcopy mobile /e /y', (err, out, stdErr) => {
    console.log(out);
    if (err) console.log(stdErr);
  });

}
```

webpack-partials/mobile.js Appendix 4

```
const path = require('path');
const projectRoot = path.resolve(__dirname).replace('webpack-partials',
'');
const plugins = require('./plugins');
const rules = require('./module');
const { externals, resolveLoader } = require('./externals');
const { production } = require('./env-variables');
const phoneGap = {
  externals,
  resolveLoader,
  entry: path.resolve(projectRoot, 'src', 'index.js'),
  devServer: {
    hot: production ? false : true,
    port: 3003
  },
  target: 'web',
  output: {
    path: path.resolve(projectRoot, 'www'),
    filename: 'index.js'
  },
  module: {
    rules
  },
  plugins,
  cache: true
};

module.exports = phoneGap;
```

webpack-partials/web.js Appendix 5 1(2)

```
const path = require('path');
const plugins = require('./plugins');
const rules = require('./module');
const { production, development, desktop, mobile, web } = require('./env-
variables');
const projectRoot = path.resolve(__dirname).replace('webpack-partials',
'');
const webpack = require('webpack');
const BabiliPlugin = require('babili-webpack-plugin');
const nodeExternals = require('webpack-node-externals');
const {externals, resolveLoader} = require('./externals');
const front = {
  externals,
  resolveLoader,
  devtool: development ? '#eval' : false,
  entry: production
    ? [path.resolve(projectRoot, 'src', 'index')]
    : [
      'webpack-dev-server/client?http://localhost:8080',
      'webpack/hot/only-dev-server',
      path.resolve(projectRoot, 'src', 'index')
    ],
  devServer: {
    hot: production ? false : true,
    quiet: true
  },
  output: {
    path: path.resolve(projectRoot, 'dist', 'web'),
    publicPath: '',
    filename: 'index.js'
  },
  target: 'web',
  plugins: plugins,
  module: {
    rules: rules
  },
  cache: true };
const back = {
  externals: [nodeExternals({})],
  entry: [path.resolve(projectRoot, 'src', 'server')],
  output: {
    path: path.resolve(projectRoot, 'dist', 'web'),
    publicPath: '',
    filename: 'server.js'},
  target: 'node',
  node: {
```

```
    __dirname: false,
    __filename: false
  },
  plugins: [
    new webpack.DefinePlugin({
      desktop: JSON.stringify(desktop),
      mobile: JSON.stringify(mobile),
      production: JSON.stringify(production),
      development: JSON.stringify(development),
      web: JSON.stringify(web),
    })
  ],
  module: {
    rules: [{
      test: /\.js$/,
      exclude: /node_modules/,
      use: {
        loader: 'Babel-loader',
        options: {
          presets: development ? ['env'] : ['babili', 'env']
        }
      }
    }]
  }
};
if (production) {
  back.plugins.push(
    new BabiliPlugin({
      deadcode: true,
      mangle: true
    })
  );
}
if (development) {
  back.module.rules.push(
    {
      enforce: 'pre',
      test: /\.js$/,
      exclude: /node_modules/,
      loader: 'eslint-loader',
      options: {
        formatter: require('eslint/lib/formatters/stylish'),
        emitError: true
      }
    });
}
module.exports = { front, back };
```

webpack-partials/plugins.js Appendix 6 1(2)

```
const { desktop, mobile, web, production, development } = require('./env-
variables');
const webpack = require('webpack');
const ExtractTextPlugin = require('extract-text-webpack-plugin');
const BabiliPlugin = require('babili-webpack-plugin');
const HtmlWebPackPlugin = require('html-webpack-plugin');
const StyleLintPlugin = require('stylelint-webpack-plugin');
const WriteFilePlugin = require('write-file-webpack-plugin');
const FriendlyErrorsWebpackPlugin = require('friendly-errors-webpack-
plugin');
const path = require('path');
const glob = require('glob');
const PurifyCSSPlugin = require('purifycss-webpack');
plugins = [
  new webpack.DefinePlugin({
    desktop: JSON.stringify(desktop),
    mobile: JSON.stringify(mobile),
    production: JSON.stringify(production),
    development: JSON.stringify(development),
    web: JSON.stringify(web),
  }),
  new HtmlWebPackPlugin({
    filename: 'index.html',
    template: './src/index.html',
    js: desktop && production ? '<script type="text/javascript"
src="index.js"></script>'
      : '<script type="text/javascript"
src="http://localhost:8080/index.js"></script>',
    link: desktop && production ? '<link rel="stylesheet"
href="css/style.css">' : '<!---->',
    VueJS: ( mobile || web ) && development ? '<script
type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.2/vue.js"></script>'
: '<script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.2/vue.min.js"></scrip
t>',
    inject: desktop ? false : true,
    hash: true,
    minify: {
      collapseWhitespace: production ? true : false
    }
  }),
  new FriendlyErrorsWebpackPlugin()
];
```

Appendix 6 2(2)

```
if (development) {
  plugins.push(
    new webpack.HotModuleReplacementPlugin(),
    new webpack.NamedModulesPlugin(),
    new StyleLintPlugin({
      configFile: './.stylelintrc',
      files: ['src/**/*.vue', 'src/**/*.sass', 'src/**/*.scss']
    }),
    new WriteFilePlugin({
      test: /\.html$/,
      useHashIndex: true,
      log: false
    })
  );
}

if (production) {
  plugins.push(
    new ExtractTextPlugin({
      filename: 'css/style.css',
      allChunks: true
    }),
    new BabiliPlugin({
      deadcode: true,
      mangle: true,
      removeConsole: true,
      booleans: true,
      simplify: true
    })/*,
    new PurifyCSSPlugin({
      // Give paths to parse for rules. These should be absolute!
      paths: glob.sync(path.join(__dirname, '', 'src/*.{js,html,vue}')),
      minimize: true,
      purifyOptions: { info: true }
    })*/
  );
}

module.exports = plugins;
```

webpack-partials/module.js Appendix 7 1(4)

```
const { production, development } = require('./env-variables');
const ExtractTextWebpackPlugin = require('extract-text-webpack-plugin');
const cssMinimized = {
  loader: 'css-loader',
  options: {
    minimize: true
  }
};
let rules = [
  {
    test: /\.vue$/,
    loader: 'vue-loader',
    options: {
      postcss: production ? [require('postcss-cssnext')()] : null,
      loaders: {
        extractCSS: production ? true : false,
        css: production ?
          ExtractTextWebpackPlugin.extract({
            use: cssMinimized,
            fallback: 'vue-style-loader'
          }) :
          'vue-style-loader!css-loader',
        sass: production ?
          ExtractTextWebpackPlugin.extract({
            use: [cssMinimized, 'sass-loader?indentedSyntax'],
            fallback: 'vue-style-loader'
          }) :
          ['vue-style-loader!css-loader!sass-loader?indentedSyntax'],
        scss: production ?
          ExtractTextWebpackPlugin.extract({
            use: [cssMinimized, 'sass-loader'],
            fallback: 'vue-style-loader'
          }) :
          ['vue-style-loader!css-loader!sass-loader']
      }
    }
  },
  {
    test: /\.js$/,
    exclude: /node_modules/,
    use: {
      loader: 'Babel-loader',
      options: {
        presets: development ? ['env'] : ['babili', 'env']
      }
    }
  },
```

Appendix 7 2(4)

```
{
    test: /\.(png|jpg|jpeg|bmp|svg)$/,
    use: [
      'url-loader?limit=10000&name=images/[name].[ext]',
      {
        loader: 'image-webpack-loader',
        options: {
          mozjpeg: {
            quality: 65
          },
          pngquant: {
            quality: '10-20',
            speed: 4
          },
          svgo: {
            plugins: [
              {
                removeViewBox: false
              },
              {
                removeEmptyAttrs: false
              }
            ]
          },
          gifsicle: {
            optimizationLevel: 7,
            interlaced: false
          },
          optipng: {
            optimizationLevel: 7,
            interlaced: false
          }
        }
      }] },
  { test: /\.woff(\?v=\d+\.\d+\.\d+)?$/, use:
['url?limit=10000&mimetype=application/font-woff&name=fonts/[name]-
[hash].[ext]'/*, exclude: /node_modules/*/] },
  { test: /\.woff2(\?v=\d+\.\d+\.\d+)?$/, use:
['url?limit=10000&mimetype=application/font-woff&name=fonts/[name]-
[hash].[ext]'/*, exclude: /node_modules/*/] },
  { test: /\.ttf(\?v=\d+\.\d+\.\d+)?$/, use:
['url?limit=10000&mimetype=application/octet-stream&name=fonts/[name]-
[hash].[ext]'/*, exclude: /node_modules/*/] },
  { test: /\.eot(\?v=\d+\.\d+\.\d+)?$/, use: ['file?name=fonts/[name]-
[hash].[ext]'] }
];
```

```
if (development) {
  rules.push(
    {
      enforce: 'pre',
      test: /\.(js|vue)$/,
      exclude: /node_modules/,
      loader: 'eslint-loader',
      options: {
        formatter: require('eslint/lib/formatters/stylish'),
        emitError: true
      }
    },
    {
      test: /\.css$/,
      use: [
        'style-loader',
        {
          loader: 'css-loader',
          options: {
            sourceMap: true
          }
        }
      ],
      /*exclude: /node_modules/*/
    },
    {
      test: /\.(sass|scss)$/,
      use: [
        'style-loader',
        {
          loader: 'css-loader',
          options: {
            sourceMap: true
          }
        },
        'resolve-url-loader',
        'sass-loader'
      ],
    }
  );
}

if (production) {
  rules.push(
    {
      test: /\.css$/,
      use: ExtractTextWebpackPlugin.extract([
```

```
      {
        loader: 'css-loader',
        options: {
          minimize: true
        }
      },
      {
        loader: 'postcss-loader',
        options: {
          plugins: function () {
            return [
              require('postcss-cssnext')()
            ];
          }
        }
      }]
    ),
    /*exclude: /node_modules/*/
  },
  {
    test: /\.(sass|scss)$/,
    use: ExtractTextWebpackPlugin.extract([
      {
        loader: 'css-loader',
        options: {
          minimize: true
        }
      },
      {
        loader: 'postcss-loader',
        options: {
          plugins: function () {
            return [
              require('postcss-cssnext')()
            ];
          }
        }
      },
      'resolve-url-loader',
      'sass-loader']
    ),
    /*exclude: /node_modules/*/
  }
  );
}
module.exports = rules;
```