



Aalto University
School of Science

Lauri Suihko

**Migrating Windows Sales Configurator Application To Web Based
Application**

Master's Thesis
Espoo, March 28, 2019

Supervisor: Professor Petri Vuorimaa
Advisor: M.Sc. (Tech.) Mikko Halttunen

Author:	Lauri Suihko		
Title:	Migrating Windows Sales Configurator Application To Web Based Application		
Date:	March 28, 2019	Pages:	60
Major:	Software Technology	Code:	T-220
Supervisor:	Professor Petri Vuorimaa		
Advisor:	M.Sc. (Tech.) Mikko Halttunen		
<p>Sales configurator is a key component of a home builder ERP system. In a typical sales meeting the home buyer sits down with a sales representative to select where the home should be build, what kind of home will be build and what extra options will be selected. The configurator needs to be able to calculate correct prices, enable/disable options based on rule sets and store the selections to a permanent storage. Traditionally this meeting has been organized at the home builder's premises using a desktop computer but further flexibility is often needed.</p> <p>It is a general trend that traditional platform dependent applications have been migrated to web applications. There are several benefits for the users to be able to run applications in a web browser on most of the devices they might use without the need to install any application on their devices. The continuous development of different web framework technologies have made this possible and easier than ever before.</p> <p>The goal of this thesis is to migrate an old Windows native sales configurator to a web application and to find out what benefits such migration is expected to yield and what kinds of obstacles need to be overcome. This thesis presents one possible way to create a web application UI using React web development framework and the back-end server access using ODATA web API. This thesis also presents two different ways (code quality measurements and performance evaluation) to measure the success of an application migration from Windows native C# WebForms application to a JavaScript based web application.</p>			
Keywords:	Web-Application Windows-Application Migration Analysis Implementation		
Language:	English		

Tekijä:	Lauri Suihko		
Työn nimi:	Windows-pohjaisen myyntikonfiguraattorin muuntaminen web-pohjaiseksi sovellukseksi		
Päiväys:	28. maaliskuuta 2019	Sivumäärä:	60
Pääaine:	Ohjelmistotekniikka	Koodi:	T-220
Valvoja:	Professori Petri Vuorimaa		
Ohjaaja:	Diplomi-insinööri Mikko Halttunen		
<p>Myyntikonfiguraattori on tärkeässä roolissa omakotitalorakennuttajille suunnatuissa toiminnanohjausjärjestelmissä. Tyypillisesti myyntitapahtumassa kodin ostaja istuu välittäjän kanssa konfiguraattorin äärellä ja valitsee, minkälaisen talon hän haluaa, mihin se tulee rakentaa ja minkälaisia lisävalintoja ostaja mahdollisesti haluaa. Konfiguraattorin tehtävä on laskea talolle oikea hinta, näyttää/piilottaa valinnat niihin liittyvien sääntöjen perusteella ja tallettaa asiakkaan valinnat johonkin pysyvään tietokantaan. Perinteisesti tämä tapaaminen on järjestetty talonrakennuttajan tiloissa pöytäkoneen äärellä, mutta nykyään tapaaminen usein halutaan järjestää joustavammin asiakkaan ehdoilla.</p> <p>Nykysuuntauksen mukaisesti ohjelmistoja kehitetään usein suoraan web-sovelluksiksi tai vanhoja alustariippuvaisia sovelluksia muunnetaan web-sovelluksiksi. Kyvyssä ajaa sovellusta selaimessa on useita hyötyjä kuten se, että web-sovelluksia voi ajaa eri laitteilla eivätkä ne vaadi käyttäjää asentamaan mitään niihin. Erilaisten web-kehitystekniikoiden kehitys on tehnyt monimutkaisten web-sovellusten tekemisestä mahdollista ja helpompaa kuin koskaan aikaisemmin.</p> <p>Tämän diplomityön tavoitteena on muuntaa vanha Windows-pohjainen myyntikonfiguraattori web-sovellukseksi sekä samalla tutkia, mitä hyötyjä muuntamisesta on ja minkälaisia haasteita muuntamiseen sisältyy. Tämä diplomityö esittää yhden mahdollisen toteutustavan käyttäen React web-kehitysympäristöä sekä ODATA web-ohjelmointirajapintaa. Tässä työssä esitetään myös kaksi eri tapaa (koodin laadun estimointi ja suorituskyvyn evaluointi), joilla tämänkaltaisen muunnostyön onnistumista voi arvioida ja mitata.</p>			
Asiasanat:	Web-sovellus, Windows-sovellus, Migraatio, Analyysi, Implementaatio		
Kieli:	Englanti		

Acknowledgements

This project was done along the ongoing process of making the Sapphire Home Building ERP more modern and user-friendly tool for production home builders. I want to thank the initiator of this project and thesis, Jouko Väkiparta, former CEO of Kova Finland OY, who regrettably succumbed to an illness during the writing of this thesis.

I want to thank my supervisor professor Petri Vuorimaa for guidance, feedback and patience during this long project. I also want to thank M.Sc. Mikko Halttunen for being very involved, interested and eager to help with his technical expertise throughout the project. I'm grateful for the support (read: pressure) provided by the closely knit Kova Finland OY development team.

Last, but definitely not least, I want to thank my family and friends for being role models and pushing me to finish this. I owe special gratitude to my fiancée Jenni for providing support, encouragement and gentle pushes all the way from the start to end - without which this thesis wouldn't have been finished.

Helsinki, March 28, 2019

Lauri Suihko

Abbreviations and Acronyms

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DBA	Database Administrator
DOM	Document Object Model
DNS	Domain Name System
ERP	Enterprise Resource Planning
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
MVC	Model-View-Controller
QoS	Quality of Service
REST	Representational State Transfer
RIA	Rich Internet Application
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XSS	Cross Site Scripting

Contents

Abbreviations and Acronyms	iv
1 Introduction	1
1.1 Research Questions	1
1.2 Sapphire Build	2
1.3 Production Home Building	2
1.4 Options	3
1.5 Structure of the Thesis	3
2 Web Applications	4
2.1 Web Application Quality Attributes	4
2.2 Web Application Architecture	6
2.2.1 Tier Architecture	6
2.2.2 Quality Attributes and Tiers	6
2.2.3 MVC Architecture	8
2.2.3.1 Model	9
2.2.3.2 View	9
2.2.3.3 Controller	10
2.2.3.4 Benefits of MVC	10
2.2.3.5 MVC and N-Tiered Web applications	10
2.3 Rich Internet Applications	11
2.3.1 RIA Technologies	11
2.3.2 Adobe Flash	12
2.3.3 Silverlight	12
2.3.4 Java Applets	12
2.3.5 HTML5 And Related Technologies	13
2.3.5.1 HTML5	13
2.3.5.2 JavaScript	13
2.3.5.3 CSS	14
2.3.5.4 React	14

3	Evaluating Code Quality	16
3.1	Code Quality of the Sales Configurator	16
3.2	Software Quality and Code Quality	16
3.3	Code Quality Issues	17
3.4	Code Quality Metrics	18
3.5	Problems with Quality Metrics	20
3.6	Code Quality Evaluation Tools	20
3.6.1	Tool Inspection Scopes	21
3.6.2	C# Analysis with Visual Studio	21
3.6.3	JavaScript Analysis	22
4	Web Application Performance	23
4.1	Application Performance in General	23
4.2	Acceptable Performance	24
4.3	Web Application Performance	25
4.3.1	Load Balancing	26
4.3.2	Web Application Performance Testing	27
5	Current Sales Configurator	28
5.1	Home Option Configurators In General	28
5.2	Origins	28
5.3	How Is Sales Configurator Used	29
5.4	Sapphire Build Option/Option Rule Setup	30
5.4.1	Relevance to the migration project	33
5.5	Sapphire Build Price Setup	33
5.5.1	Relevance to the migration project	34
5.6	Used Technologies	34
5.6.1	Problems	34
6	Methods	35
6.1	Used Web Development Technologies	35
6.2	Technical Challenges	35
6.3	Evaluating success	36
6.3.1	Performance	36
6.3.2	Code Quality	36
7	Implementation	37
7.1	ODATA Web API	37
7.1.1	Authentication and Security	39
7.2	React.JS User Interface	39
7.3	Proof of concept UI	39

7.3.1	Customer Panel	41
7.3.2	Main Panel	41
7.3.3	Sales Office Panel	43
7.3.4	Output panel	43
8	Evaluation	45
8.1	Overall success of the migration	45
8.1.1	General	45
8.1.2	User Interface	45
8.2	Performance measurement results	46
8.2.1	Start up performance	46
8.2.2	Changing Community	46
8.2.3	Load Option Data	47
8.2.4	Changing Options	49
8.3	Code quality measurement results	49
8.3.1	Measuring JavaScript Code Quality	50
8.3.2	Measuring C# Code Quality	50
8.3.3	Comparing The Results	51
8.3.4	Results	51
9	Discussion	53
9.1	Migration Insights	53
9.2	Future Work	54
10	Conclusions	55

Chapter 1

Introduction

Goal of this thesis to study and implement the best possible way to migrate Windows native Sales Configurator application to a web based application. Sales Configurator is a part of the Sapphire Build ERP software suite [19]. The main functions of the configurator are to register customer information, select the preferred community and lot, select the desired options, and submit a sales worksheet to a back-end server. In additions to the core functionality the sales configurator can be used to, e.g., handle sales contingencies, custom options, print and digitally sign sales documents and draw redlines, but they are out of the scope of this thesis. The main focus is on the option configurator part since it is the most complicated and performance-wise most difficult piece of the sales configurator.

1.1 Research Questions

This thesis will cover three main research questions. Firstly, this thesis will investigate what kind of fundamental differences there are between Windows native applications and a Web applications and how to tackle the challenges caused by the differences. These differences include but are not limited to, for example, the different ways the state of the option selection process can be stored, or how to make UI interface fluid regardless of Web service calls to a back-end server. Also, it needs to be investigated, how great of a negative impact will this migration cause to the back-end servers. After the migration, a lot of the computing previously done by the native client (e.g., option rule calculation) might need to be moved to the server.

Secondly, this thesis will cover different ways how to evaluate the success of the migration. First is performance. Even though the web application will have to overcome its own performance problems one of the main goals

of the migration is to try to enhance at least the user perceived performance by reducing long load times that plague the configurator currently. The second evaluation method will be code quality. We will investigate what kind of methods are the best for evaluating success through code quality in a situation where part of the code is rewritten completely and also migrated to a fundamentally different technology.

Lastly, in this thesis it will be studied and formulated, what kind of benefits there is to be expected when moving from Windows native client to a Web application.

1.2 Sapphire Build

Sapphire Build is a ERP system originally developed by Finnish software development company Evenflow OY. The complete Sapphire Build ERP package is used mainly by production home builders building 100 to 2000 homes per year. The core ERP system is being developed as a web application mainly using Microsoft .NET technologies. The web application core is supplemented with some Windows native applications (e.g., the sales configurator and a file sync tool) and iOS and Android native applications. [19]

1.3 Production Home Building

Production home builder is a company that builds different kinds of housing (e.g., single-family homes, townhouses, apartment buildings) on a piece of land they have previously acquired. This thesis will focus on production builders that build single family homes on communities containing up to several hundreds of homes. Usually, the builder offers a selection of different kinds of home models from which the customer picks their favorite. Even in large communities with 100+ homes typically only a handful, from 4 to 8, base models are offered. To satisfy the different needs and desires of customers the builders give the customers the ability to customize their homes using options. Production home builder companies vary greatly in size. The usual metric to evaluate the size of the builder is the number of units the company sells in a year. The smallest ones are run by just a few people and can build less than 10 units per year. The biggest ones sell in the tens of thousands per year and their revenue is in the billions (D.R. Horton 41,652 closings and 12,239,900,000 USD revenue in the 2016) [1]. [26]

1.4 Options

Options in the home building industry work in a similar fashion as with the automotive industry. In home building, however the amount of available options and their effect on the end product can be far greater. Home builder can offer, e.g., cabinets or carpets in hundreds of different variations and structural options (e.g., extra rooms, bigger garages, even extra floors) can affect profoundly the layout of the finished home. Option rules control the availability of the options - basement bathroom can be build and sold only if basement is build and sold. Like in the automotive industry the offered options are also an important source of revenue for the home builders and therefore are eagerly marketed and sold for home buyers.

1.5 Structure of the Thesis

The first chapters form the theory part of this thesis. The first part of the theory portion covers the general characteristics, most common technologies and most common architecture designs of web applications. The second part is about common technologies and techniques to evaluate code quality. In the last theory part, this thesis will cover different ways to evaluate web application performance and which aspects of application performance are specifically important and critical for web applications. The last chapters of this thesis cover what techniques were chosen for the implementation as well as how successful the migration was from the perspective of code quality evaluation and performance metrics.

Chapter 2

Web Applications

Web application is a software application, which is built using web development techniques and the user interface is accessed using a web browser. They differ from web sites in that they usually have some functional purpose such as sending emails whereas regular web sites are usually more static and content oriented although the distinction is not completely clear always. E.g., one could argue that a news site that offers the users the possibility to prioritize the content that is primarily offered to them falls somewhere between a web application and a web site.[57][53]. In the recent years, the development of the web application development techniques, browser support, faster Internet connections and mobile devices have made it possible to create more complicated web applications that are responsive, easy to use and are more or less platform independent.

2.1 Web Application Quality Attributes

Web application share a lot of the most important quality attributes as general applications but because of their different nature often are emphasized somewhat differently. Commonly listed quality attributes and their most important aspects with regards to web applications are[32][47]:

1. Scalability
 - User count for a web application can rise up rapidly and it is not uncommon for one to reach millions of concurrent users. Allowing this requires great care when designing the underlying architecture.
2. Performance

- Poor performance is a guaranteed way to ruin the user experience of an application and web applications are no exceptions. Web applications also have to cope with additional performance affecting challenges like poor network conditions and request timeouts.

3. Availability

- Web applications are usually expected to be functional 24/7 merely based on the fact that user can access them from anywhere around the globe.

4. Security

- Since web applications are often accessible through open Internet, they are common targets for malicious hackers. Many of the major data leaks that have made public have been done through some kind of web application.

5. Reliability

- Users expect web applications to function correctly and have low tolerance for failures since they often have other options just by typing a different URL.

6. Maintainability

- The software technologies have been changing and developing rapidly and users expect web applications - arguably more than applications in general - to be able to keep up with the development. Five year old web application can feel outdated already.

7. Usability

- Web applications, more often than regular applications, are accessed by general public with no technical background and need to have intuitive user experience. Similarly to reliability, if users aren't happy they have low threshold to switch to another provider.

This is not a complete list and other attributes can and have been proposed but arguably they are the most relevant ones with regards to web applications.

2.2 Web Application Architecture

Web applications are by nature always client-server applications where a user operated web browser is usually the client[53]. Beyond that however, for a web applications developer there are multiple different architectural patterns and technologies to choose from[37].

Web application architecture defines the structure of the application as well as how the different parts of the web application interact with each other. It's important to design and implement a suitable architecture for a specific application need and the architecture design is critical to be able to score high with regards to the quality attributes.

2.2.1 Tier Architecture

Web applications like all applications can be categorized by how many tiers they utilize. The number of tiers is the number of layers the desired information will have to pass to get from the data source to the presentation tier[35]. 1-tiered application would be a lone program running on a single machine using local data, and therefore web applications by nature are at least two-tier applications, one for back-end server and one for the presentation piece of the applications, usually rendered in a web browser. Usually however, modern web applications are at least 3-tiered applications.

1. Presentation Layer, e.g., HTML5 rendered in a browser
2. Logic Layer, e.g., ASP.NET application running on top of IIS
3. Data Layer, e.g., Microsoft SQL server running on a Windows Server

The distinction between layers isn't always clear and the layers can be further divided down to more fine-grained parts, e.g., logic layer and data layers can be divided to a separate business logic, data access and data layers. Web applications that are divided to more than three or more tiers are also called n-tier web applications. Dividing web applications brings multiple benefits that have to do with the overall quality of the web application[47]. Figure 2.1 below shows an example information flow on a 3-tier architecture.

2.2.2 Quality Attributes and Tiers

Tiered web applications are better equipped to meet most of the quality attributes that can be used to evaluate overall quality of web application. Separate tiers can help tackle scalability and performance by allowing the

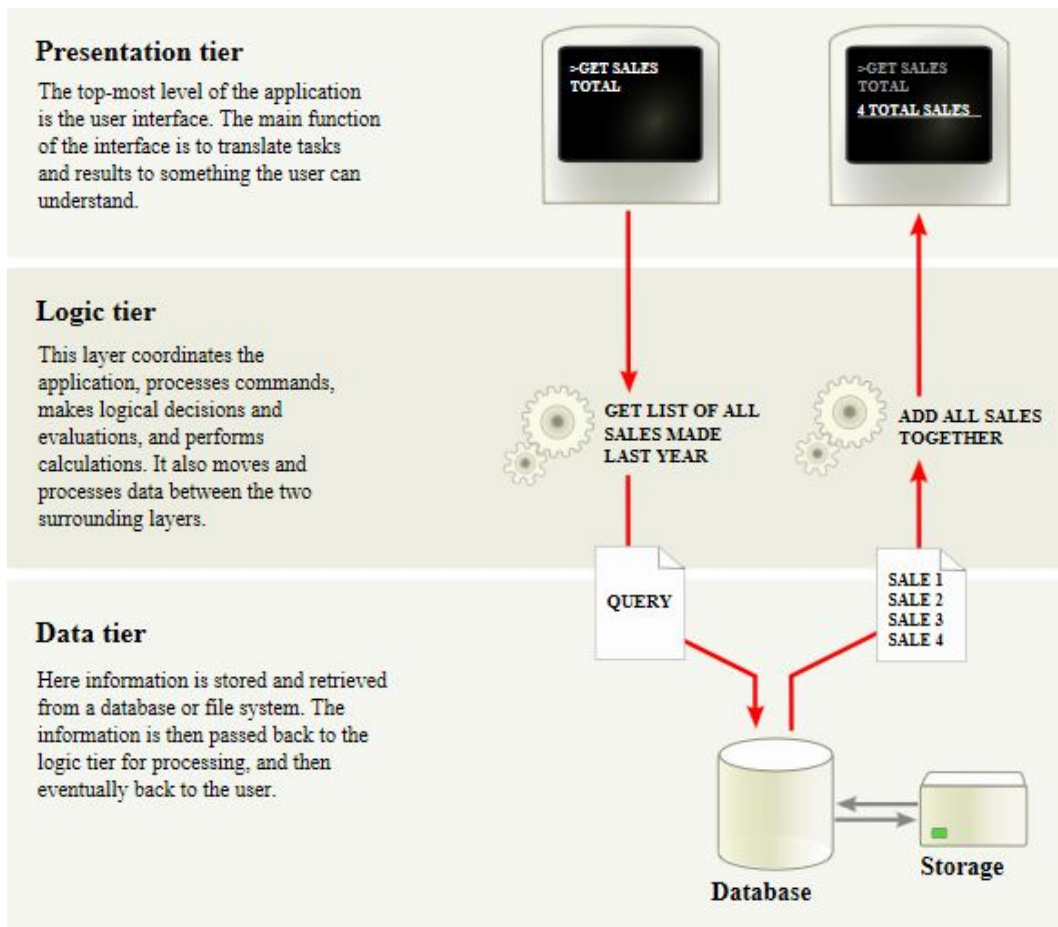


Figure 2.1: Information flow on a 3-tier application.

application to be deployed to separate virtual or physical machines, and thus, e.g., a slow database server will not grind the whole application to a halt. Furthermore, it's easier to replicate logically separate layers to increase the throughput of a web application by, e.g., using load balancing [25] to replicate the logic layer and still use a single database layer. Similarly, just the database layer can be replicated if that is causing problems with performance or scalability or the combination of the two can be used. Similar techniques can be used to ensure the availability of the web application.

Separate tiers are useful also from the perspective of reliability and maintainability. Generally speaking smaller software project are easier to manage from initial planning and development to testing and development. Breaking bigger projects to smaller and logically separate entities is no exception.

When web applications are separated to layers the developers working on a single layer don't need to have expertise on all of the technologies used in the complete application but can focus on the technologies used in their part of the application, e.g., SQL expert DBAs working on the data layer, HTML, CSS and JS expert web designers working on the presentation layer and back-end developers working on the logic layer[47]. The maintainability of the application is improved by the distinction of tiers also because it easier to upgrade, update or even rewrite a single tier versus the whole application. As we will see later in this chapter, the web applications technologies - even ones backed up by big players like Oracle, Adobe and Microsoft - have lost popularity and have been deprecated relatively quickly, and therefore it's important to keep that flexibility in mind when designing web application architecture. Also, tiered approach allows developing different interoperable versions of a specific layer, e.g., the usage of different presentation layers based on the kind of device that the web application is accessed.

Also, security aspects of a web applications can be addressed better using separate layers. The separate layers can be utilized to create separate security layers all of which the attacker should breach to gain full access to the system[47]. The logic layer can be e.g. used to filter and recognize malicious HTTP requests with XSS (cross site scripting) or SQL injection attacks[55]. The client side part of the presentation layer is not a safe place to sanitize user input since it is run on the client's environment and usually is easily modified. The logic layer run on the back-end server, however, is the correct and safe place to do that.

2.2.3 MVC Architecture

MVC or Model-View-Controller architectural paradigm was not designed originally having web applications in mind, but was developed for the programming language Smalltalk in the late 70s in the Xerox Palo Alto Research Laboratory (PARC). Norwegian computer scientists Trygve Reenskaug has often been credited as being one the major innovators behind MVC[50]. Decades later the MVC went through a renaissance when many of the prominent web development frameworks recommended using the MVC paradigm, and therefore it deserves a special mentioning when speaking of web applications. According to HotFrameworks.com - a web site that measures popularity of web frameworks based on how many GitHub project starts there are for specific framework as well as how many times the framework has been tagged on Stack Overflow questions - many of the currently most popular frameworks implement the MVC pattern[9]. Figure 2.2 below visualizes an example information flow on the MVC pattern.

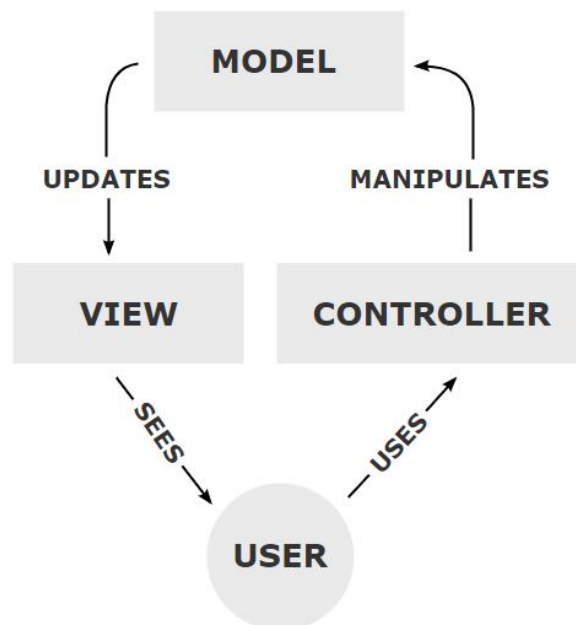


Figure 2.2: Information flow on the MVC pattern.

2.2.3.1 Model

The definition of a model is very flexible. A model represents some data and the means to modify the data[50]. Originally a model could be as simple as a single integer or a string[38] or at least, nowadays more likely, a single object - object as an instance of a class in a object oriented programming language. In a web application, the object would likely represent a database table, a file, session data or cached data[28]. Model is the most independent of the MVC parts. In the purest MVC form, it doesn't need to know anything about the views nor the controllers[41].

2.2.3.2 View

Each model has one or more views assigned[50]. Views request data from their models and represent that in some graphical way. Sometimes models also need to have a way to trigger updates on their views if models can be acted upon through separate procedures, e.g., periodically run internal operations[28]. A view allows user to interact with the data, but only through controllers.

2.2.3.3 Controller

Controllers tie the user, models and views together. Users use controllers to manipulate data using some input device, e.g., mouse or a keyboard [38] and calling the data manipulation methods provided by the models.

2.2.3.4 Benefits of MVC

MVC architectural pattern provides similar benefits to what tiered architecture can offer and tackles especially the scalability, reliability and maintainability aspects of the web application quality attributes presented earlier[28][32][47]. More specifically MVC claims to provide the following:

1. Less Coupling
2. Higher cohesion
3. Flexibility
4. Design Clarity
5. Maintainability
6. Greater scalability

2.2.3.5 MVC and N-Tiered Web applications

The n-tier and MVC architectures are fundamentally different. The focus of the n-tier architecture is in describing how information flows behave and how different parts can be separated to different (often physically separated) layers. Also, the n-tier architecture is strictly hierarchical, the presentation layer is never directly in contact with the data layer. The MVC architecture is more flexible in its definitions and the separation of its parts is more conceptual in nature. This flexibility and the multitude of different interpretations makes the MVC also arguably harder to grasp and more difficult to define. Often it is not trivial to determine whether a framework of specific software project implements MVC or not.

MVC and n-tiered architecture styles are not mutually exclusive but often can be used together. There are multiple different ways they can be implemented simultaneously. Sometimes all of the MVC parts are located completely in the presentation layer which is the recommended way to use e.g. the Angular.JS front-end web development framework. The ASP.NET MVC takes a completely different approach to utilize MVC. In the ASP.NET MVC the view would be live partly on the presentation layer and partly on

the logic layer. The models and controllers are represented by classes run on the server, or the logic layer, side.

2.3 Rich Internet Applications

The goal of this thesis is to migrate a fairly complex windows native application to a web application. Such application can be called a RIA or Rich Internet Application [48]. It is critical to decide a suitable RIA platform into which to start the migration process. There are multiple aspects to consider when selecting the web development technology. First of course you have to find out if the potential technology will have the technical capabilities you require. After that is settled, you need to consider the ease of development, browser support, is the technology well documented, does it have a virile developer community, does the technology have a future and so on. It is especially important to consider the future of the potential technology since many potential candidates will eventually fade into obscurity which has happened before even to technologies backed up by big corporations.

2.3.1 RIA Technologies

In the early days of web applications, the standard technologies supported by the browsers of the day were poorly suitable to meet the requirements of a software application. They were more focused on serving static content and couldn't provide, e.g., video playback or a smooth UI experience. JavaScript developed by Netscape in 1995 gave developers some means to make interactive UIs, but to supplement the capabilities multiple additional techniques were developed. Common to these technologies were that they often required installing a 3rd party plug-in to the Internet browser and they run native code, which at least some years ago was performance-wise more efficient[48]. The performance of JavaScript engines has since been enhanced and the gap has narrowed [34]. Also, common for these technologies is that they have each been slowly deprecated and replaced by the HTML5 and the related technologies. The major browser vendors have dropped or are about to drop the support for all of three different RIA technologies described below and the distributors of the technologies have announced the discontinuation of them as well.

2.3.2 Adobe Flash

From the perspective of RIAs, the most important version of the Flash was the fifth version released in the year 2000. Flash 5 introduced the ActionScript language, which helped transition Flash from being merely a playback platform for audio and graphics to a full-blown RIA platform. In 2017, Adobe announced that it is planning to end Flash. They say they will stop updating and distributing Flash in 2020. Adobe is also endorsing HTML5 and recommending the existing developers to migrate existing Flash applications to HTML5 based applications[22].

2.3.3 Silverlight

Silverlight was Microsoft offering to the RIA frameworks. The first version was released in 2007. Like the Flash and Java Applets, Silverlight requires an addition browser plug-in. Early Silverlight versions had the same emphasis on media playback the Flash had. The second version released in 2008 offered a limited support for .NET framework and moved the focus of the Silverlight to full-blown RIA framework[43]. Like Adobe did to Flash, Microsoft has stated the they will not support Silverlight in the future and recommend that existing Silverlight applications will be migrated to other technologies including HTML5 and the related technologies[54].

2.3.4 Java Applets

Java Applet is an application that is run on a JVM (Java Virtual Machine) in a Web browser. Like the Flash applications, Java Applets require an installation of a browser plugin. Java applets are usually written in Java, but can be written in any language that can be compiled to ByteCode. In 2016 Oracle announced they will deprecate the Java browser plugin as part of the JDK 9 update[56]. Unlike the Adobe with Flash and Microsoft with Silverlight, Oracle is not recommending migrating Java applet applications to HTML5 based applications, but recommends developers to migrate existing applets to use Java Web Start technology, which doesn't require a separate browser plugin. Java Web Start is a technology that allows Java applications to be installed and run from a single click on the web browser, but the application itself is run outside the browser[12].

2.3.5 HTML5 And Related Technologies

All of the technologies listed above have been more or less made obsolete by the HTML5 standard and the related tools that moderns browser support out-of-box. The technologies most commonly used with HTML5 to develop RIAs are CSS and JavaScript.

2.3.5.1 HTML5

HTML5 is the latest standard of the HTML. It was published by World Wide Web Consortium on 28.10.2014 [10]. It was long waited since the version before that, HTML 4.01, was published already in 1999. From the start, one of the main goals of HTML5 standard was to facilitate the creation of rich internet applications by adding native browser support for features that previously required an add-on. These included audio/video playback, scalable vector graphics (SVG), drag-and-drop and many more[10].

2.3.5.2 JavaScript

JavaScript is a scripting language developed by Netscape Communication Corporation originally to allow front-end developers to make static HTML pages more dynamic. The first version was published in December 1995 and was supported by the Netscape Navigator 2.0, then a popular browser. In the original press release, the publishers of JavaScript state that the design of JavaScript is to be [15]:

- designed for creating network-centric applications
- complementary to and integrated with Java
- complementary to and integrated with HTML
- open and cross-platform.

Also, easy of use was emphasized since the target users were seen as web page designers with limited software development experience. They aimed to create a simple and easy-to-learn language like Visual Basic. How well they succeeded is arguable. JavaScript has been criticized for getting popular before the language was finished and polished leaving some obvious flaws in the language syntax and design[27]. Also, JavaScript is loosely typed, which on the other hand gives the language flexibility, but on the other hand makes it more error prone especially for inexperienced developers or developers used to writing strongly typed code [30]. TypeScript developed by Microsoft is one

attempt to make writing complex JavaScript applications more manageable. The TypeScript syntax allows, e.g., static types and better support for object-oriented program design.

Despite all this, JavaScript is nowadays extremely popular, and it is estimated that around 94.5 percent of the 10 million most popular web pages use it to some extent. Especially its use along with AJAX reduces the amount of full page reloads needed on a web application, which results in a more dynamic, more desktop application like, user experience. Usually, it is not used as is (so called vanilla JavaScript), but along with some JavaScript libraries or frameworks such as the immensely popular jQuery or AngularJS.

2.3.5.3 CSS

Cascading Style Sheets is language used to define the presentation of a HTML page. Like the JavaScript, CSS is extremely common and used in almost all modern web sites. CSS allows web designers to more easily define a layout and specific look for a web page or application. Previously the web page presentation was defined usually within the HTML components. E.g., each link in the document had to have inline style declarations. CSS allows web designers to create multiple feels of the same page based on the user preferences and perhaps more importantly, allows the same HTML mark-up to be used with different user devices (e.g., desktops, mobile devices or tablets) by only changing the CSS rule declarations. CSS rules consist of a selector and a declaration block[52]. The selector tells the web browser, which document objects are to be affected by a specific rule and the declaration block defines which style declarations are to be applied to said objects. The basic selectors of CSS are used to target specific types of objects (type selectors), objects with specific attribute or class (id, attribute and class selectors). There is large amount of more complicated selectors that can be used to target objects with all kinds of rules.

2.3.5.4 React

React is one of the more modern javascript based UI web framework and is one of the strongest candidates to be used in the front-end part of the migration. In the MVC parading, React would cover the View portion[36]. Jordan Walke, a software developer for Facebook, is credited to be the original author of React. Later it was published as on open source library under the MIT license and is nowadays developed by Facebook, Instagram and a community of developers.

React is at its best when used in Single-Page-Applications, i.e., web applications where the application is loaded in one single full page load and after that is run on the client side. If additional data is needed from the server it is fetched using techniques like AJAX that don't need a full page reload. One of the innovations utilized by react is its Virtual Document Object Model. Browsers handle HTML documents as tree structures and offer a Document Object Model interface for making changes to the HTML structure. Changes in DOM require the browser to re-render the current view which in some scenarios can be slow. Instead of manipulating DOM directly React operations first affect only the virtual DOM and only later are imposed on the actual DOM in a process they call reconciliation[20]. React is often used with a JavaScript extension called JSX. It is used to describe the UI layout but also makes it possible to directly attach functionality to the UI components by writing JavaScript along with the UI declarations.[11]

Chapter 3

Evaluating Code Quality

3.1 Code Quality of the Sales Configurator

One of the most important reasons for this desktop application to web application migration is to at the same time improve the code quality of the sales configurator. During the life span of the configurator, the specification requirements have changed significantly, technologies have been obsoleted and improved, lot of the enhancements and bug fixes have been poorly documented and the original architecture was not flexible enough resulting in a severe code bloat. Developers involved in the development of the current configurator are often frustrated because of these reasons and hence a special focus on the code quality was in demand.

3.2 Software Quality and Code Quality

There are multiple ways to define software quality. Most of the quality attributes defined for web applications in mind on the previous chapter apply to application in general as well so they don't need to be repeated in detail - most important ones being scalability, performance, availability, security, reliability, maintainability and usability.

It's important to note that here poor code quality doesn't mean functional problems in the code (which potentially could be automatically highlighted using various testing methods) but problems in the structure of the code itself. In this sense, code quality indirectly affects all of the quality attributes but in this case we are most interested in its effects on the maintainability aspect.

Research suggests that different code quality issues and design flaws indeed are useful indicators of poor maintainability. Maintainability can be

divided in sub categories[49] - modifiability, testability and understandability - and different code flaws affect each category differently. Also, some code flaws are more useful than other when assessing the overall maintainability of a code base.[58].

It is not always trivial to decide, whether code quality in some specific code fragment or architecture is good or bad. To emphasize the vagueness of code quality Kent Beck and Martin Fowler coined the term code smell[31] that is sometimes used to describe specific aspects of a piece of code that probably are symptoms of low code quality.

3.3 Code Quality Issues

Code quality issues or smells are features of the code which generally are regarded as being bad practice, should be avoided and eventually can have reverse effects on the software quality. Below is a list of some the most common code quality issues[31][45]. The list is divided to two categories or scopes. Line-level issues and design level issues. Line-level issues are issues that occur on a single line (or at most couple of lines) and design issues are issues that are caused by bad design practices either at the method, class or program level.

- Line Level Issues:
 1. Magic numbers and string literals
 - Magic number are numbers embedded in the code where it is not clear where that number is derived from. Magic number can be used, e.g., in arithmetic operations, indexes or condition statements
 2. Fragmented Conditions/Too Complicated Conditions
 - Condition statements that could be combined logical operators (e.g., OR/AND) are written as separate conditions statements, or on the hand, statements are combined to form extremely complex and unreadable long statements
 3. Uninformative variable/method names
 4. Unused method parameters and variables
 5. Long method parameter lists
- Design Level Issues:

6. Interruptions in the code flow
 - E.g., goto statements and return statements that interrupt the natural flow of the code
7. Duplicated Code
 - Code that is exactly the same as well as code that is functionally the same, e.g., only variable names have changed
8. Unused Code - Code that is not executed in any execution path
9. Excessive use of method or operator overloading
10. Large Classes/Methods
 - Large classes (sometimes called god classes) and methods have too much functionality included in single package, which makes their testing, reading and development difficult
11. Small Classes/Methods
 - Classes or methods that have so little functionality they unnecessarily obfuscate code. E.g., classes that don't have any methods - "data classes"
12. Deeply nested loops and condition statements
 - Code containing deep loop or condition statement structures is hard to read in general. Also, deep loop structures can cause performance problems that are non-trivial to recognize from code

Design level issue list could be expanded further to include even more high-level code issues, but it gets increasingly hard to evaluate whether the chosen, e.g., deep class inheritance architecture style is an issue or a justified design decision. We still may want to be aware of such structures in the code so we can evaluate their validity. This is where Code Quality Metrics are useful.

3.4 Code Quality Metrics

Code quality metrics measure some line-level or architecture level feature of the code which is useful in evaluating code quality. Here only metrics that are easily automatically calculated are included, leaving out for example analysis of the usefulness of variable names. Some metrics are specifically designed, e.g., for object-oriented programming languages, some focus on the code quality at a line level, other try to evaluate the higher level design decisions[45].

Quality metrics can measure the prevalence of code issues and hence be easily interpreted (e.g., high value can be regarded as good and low bad), but that is not always the case. Code quality metrics also can measure aspects of the code where careful investigation of the measurements is needed in order to achieve a comprehensive understanding whether some measurement is indication of poor design decisions. They are in any case good starting points to find potential code quality issues. Following is a list of some of the more commonly used code quality metrics. For each of the listed issues there already are multiple tools that can be used to automatically evaluate the existing sales configurator code base[21][23]. The list also references the code quality issues whose prevalence it may reveal (if any).

1. Cyclomatic Complexity

- Measurement of the complexity of the code. Code with multiple execution paths will get a high cyclomatic complexity score. (issues 2/12)

2. Halstead Volume

- Measurement of the complexity of the code. Measures the amount of distinct operators/operands used in the code as well as the overall use count of the distinct operators (issues 10/11)

3. Lines of Code

- Can be used to evaluate whether a class or method is too small/large. Can be counted not from the source code but the compiled code to avoid obfuscating the result by coding styles. (issues 10/11)

4. Depth Of Inheritance

- DoI is a measurement of how far down in the class inheritance tree some class is. High number indicates potential unnecessary complexity in the class hierarchy, low number may indicate underuse of the OO-paradigm

5. Coupling Between Classes

- Classes are considered coupled if a class calls methods of another class. High value of coupling affects negatively the modularity of the design and makes testing harder since changes in one class will affect the behavior of one or more other classes

There are dozens more different metrics that can be used to evaluate code quality and different metrics can be combined to endless amounts of different weighted averages[39]. E.g., Maintainability index is a combination of the Cyclomatic Complexity, Halstead Volume and the number of code lines and Weighted Methods Per Class is the average of the Cyclomatic Complexity values for methods in a class.

3.5 Problems with Quality Metrics

How well these metrics actually reflect code quality and eventually software quality attributes is arguable. Studies have found that different tools that claim they use some specific metric, can still give different results for the same code. Code analysis tools create a model that represents the code in some, from their perspective, useful way. The algorithms the tools most likely are the same, but the way the tools construct the model out of code seems to vary resulting in the conflicting results[39]. This seriously complicates analyzing code quality metrics, e.g., in a migration project where the technologies (programming language, frameworks, etc...) might change and different tools would be needed.

Also, typically using static code analysis (quality metrics and/or finding code issues) is an imperfect indicator of general maintainability. Studies have investigated how well metrics/issues recognize maintainability issues that were pointed out by expert analysis. The results suggested that some metrics are more useful than others and many of the serious issues cannot be reliably recognized by static code analysis. [58]

[39][58]

3.6 Code Quality Evaluation Tools

Some code quality issues or metrics are hard to evaluate using automatic tools (e.g., poor naming conventions), but for most of the items covered in the above lists, automatic tools can be utilized.

Vast amounts of different tools and algorithms have been developed, which can be used to recognize and enhance poor code quality[23]. Different tools target different issues, programming languages and programming paradigms. Common for all of them is that they in some way utilize static code analysis (the code is analyzed without executing it) when calculating different metrics or finding specific code quality issues. In contrast, dynamic code analysis involves executing the code. Static code analysis tools can

evaluate code directly or compiled intermediate language[40].

3.6.1 Tool Inspection Scopes

Similarly to the way we can categorize code issues based on scope they affect the program, we can categorize tools based on the scope of the issues they aim to recognize. The object management group (OMG) together with the Consortium of IT Software Quality (CISQ) have defined three scopes or levels[33]:

1. Unit Level Analysis
 - Analysis that covers a single unit of code, i.e., method, function or even a program
2. Technology Level Analysis
 - Analysis that covers a collection of related code unit written in the same programming language. Can cover multiple programs within an application
3. System Level Analysis
 - Analysis that covers all code units, different technology layers and programs on the entire integrated application

Tools covered in thesis will all fall under the first two levels.

3.6.2 C# Analysis with Visual Studio

Visual Studio IDE by Microsoft provides various built-in code inspection tools. Out-of-box Visual Studio can calculate five different quality metrics, Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, Class Coupling and Lines of Code. Maintainability index is a number ranging from 0 to 100 and is calculated the following way:

- $\text{MAX}(0, (171 - 5.2 * \ln(\text{HV}) - 0.23 * (\text{CC}) - 16.2 * \ln(\text{LoC})) * 100 / 171)$
 - HV = Halstead Volume
 - CC = Cyclomatic Complexity
 - LoC = Lines Of Code

Microsoft claims that maintainability index of 20-100 is good, 10-19 is average and 0-9 is poor. In addition to quality metrics Visual Studio has a tool for recognizing duplicated code. Visual Studio also has static code analysis tool that uses specific set of rules for identifying vast variety of different design, performance, security etc. issues. The rules are not modifiable, but the users can define which rules they want to use. [21]

3.6.3 JavaScript Analysis

There are multiple tools for analyzing JavaScript with different focuses. ESLint is a open source JavaScript analysis tool created by C. Zakas. ESLint is a tool that statically analyses the JavaScript code and gives warning based on the rule set defined for the ESLint. Rules are configurable, new rules can be added and existing ones can be disabled. The rules mainly target specific code issues but also an upper limit for cyclomatic complexity can be set[2].

Multiple open source tools that calculate different code quality metrics exists as well. E.g., tool called escomplex can calculate multiple metrics from JavaScript code including lines of code, cyclomatic complexity, Halstead metrics, maintainability index [8].

Chapter 4

Web Application Performance

Performance is one of the main software quality attributes described in the earlier chapters. Poor performance degrades user experience and this can be specially devastating for a web application since often there are plenty of options and the switch can be easy. This chapter focuses on different ways performance of a web application can and should be evaluated.

4.1 Application Performance in General

Software application performance is not trivial to define. Performance is a combination of different metrics of which some are subjective in nature. It can be argued that software performance is often a perceived attribute. Software performs well when users aren't irritated by the application and the user experience *feels* smooth and snappy[44]. Of course there are more scientific approaches for defining application performance. The following characteristics can be measured using various testing methods and are often the starting point for any performance measurement setup[44][42]:

- Availability/Uptime
 - Measure of how often the application is available. Application is unavailable when it's completely out of reach (e.g., server failure) or simply when it so overloaded that new requests time out and are never handled
- Concurrency
 - How many concurrent users the application can facilitate without suffering from severe performance degradation. Can be a measure

of maximum number of simultaneous users or the maximum number of users that can use the application during a time period, e.g., an hour

- Throughput
 - Closely tied to concurrency. Instead being of measure of concurrent users throughput is the number of some operations the application can handle in some time period
- Response Time
 - Response time is the time it takes for the application to give a complete reply for some user request. Complete reply here means that the requested operation has been successfully completed, not just, e.g., showing a loading bar

Resource utilization is also often listed as one of the basic metrics and for web application it makes sense to divide it into two parts:

- Network utilization
 - Measure of the strain the application casts on the network. Takes into account the total data transfer as well as the peak transfer rates
- Server utilization
 - The traditional resource utilization metrics. Takes into account the CPU utilization, memory utilization and I/O utilization (typically network and disk utilization)

4.2 Acceptable Performance

End-users typically are only interested in the aspects of performance that directly affect them. Users most likely are not interested in how much network and server resources an application consumes as long as the availability and response time are at an acceptable level. Acceptable availability for a web application is usually 24/7 with some occasional down-time for updates and server maintenance. Acceptable response time is more complicated.

Acceptable response time is very much dependent on the specific use-case, more specifically what kind of operation the user is trying to perform.

Acceptable response time is very different for a complicated building material calculation operation, than it is for the simple operation of viewing the results. The former can take multiple minutes, but viewing the result should take less than couple of seconds.

Studies show that response times more than 15 seconds are only acceptable for complex queries or business operations, but even then users tend to find other things to do while they are waiting for the response. Response time greater than 4 second makes conversation hard and makes data input frustrating. 2-4 seconds is cutoff point, where users start to struggle with concentration intense operations, or operations, where they have to keep something specific in their shot-term memory. Only at one second delays users feel that their flow of thought has not been interrupted. Also at the one second point, users typically no longer require feedback about initiation of the operation. With response times longer than that, users start to initiate processes multiple times wondering whether the first one really started. Still at one second point, users feel the lag and this is unacceptable e.g. in a document editing software. Decisecond response times for humans feel instantaneous. [46][44]

4.3 Web Application Performance

All the general principles of software performance apply to web applications as well. Some characteristics of web application, however, warrant some specific metrics that are especially useful. Typically, these special metrics are some kind of special case of the general performance metrics. The following web application specific metrics have been proposed[24]. Here are the most relevant ones from

- Time To Title
 - Time it takes for browser of the client to show the title of the web application. Important from the user's perspective since it informs the user that the web page is actually responding
- Time to First Byte
 - Time it takes for the first bytes to reach the client's browser. Order of the sent data is important. Generally data containing functionality should precede static contents, e.g., images
- Time to Last Byte

- Time it takes for the last byte of the web application initial screen to arrive to the browser of the client
- Time To Interact
 - Time it takes for the user to be able to start interacting with the web application. Doesn't mean that the web application has necessarily fully loaded
- Overall Weight
 - The overall data that gets sent to the client's browser during a sessions using the web application
- Overall Asset Count
 - The overall number of different assets (typically in different files) the application needs. Assets consists of JavaScript files, CSS files and image files - anything that requires a separate HTTP request to get.

4.3.1 Load Balancing

Since web applications typically need to be available around the globe, need to be up 24/7/365 and can have even millions of simultaneous users, often web applications cannot be run on a single web server. Some method of load balancing is often needed. Load balancing is a method to distribute the load induced by a web application across multiple physical or virtual servers. There are several techniques how load balancing can be utilized, but the main goal is always either increase performance, availability or typically both.

First we need a way to divert the traffic from the client machines to different load balanced servers and there are several ways how this can be achieved. One way is do the balancing on the Domain Name System (DNS) level. DNS load balancing can be achieved by assigning multiple IP addresses to same host name and utilizing some kind of randomizing method (e.g., round-robin) for assigning different requests to different servers. Other method is to utilize DNS delegation to assign different servers to same host name based on different geographical locations[18][25]. The benefit of the DNS approach is that it can be invisible both to the client-side application as well as the back-end servers.

Other way is to handle the load-balancing at the client side. The client side application has information about the back-end server and makes independent decisions on which server to connect to. The decision can be random but also have logic behind it, e.g., selecting the server that has performed best in the recent requests[29][25]. One problem with the client side load-balancing is that typically we want to have more control on the incoming requests due to possible malicious users and also it requires developing special logic to the client-side application.

Finally, we can handle the load balancing at the server side. Typically, we would have a dispatcher server that all incoming requests initially go into and they then get redirected to separate back-end servers by some logic defined at the dispatcher. Again, the logic can be random assignment, round-robin or something more complex.[25]. The benefit here is that this is invisible to the client-side application and the dispatcher can have very detailed information about the currently available servers. They can report the dispatcher their current load, currently active connections and so on allowing more complex and effective algorithms for assigning the incoming connections for suitable servers.

4.3.2 Web Application Performance Testing

There are several different tools for evaluating web application performance with varying focuses. When evaluating or testing web application performance there is always three things to consider. The performance of the client-side application, the performance of the back-end server and the effectiveness of the the network traffic. It doesn't help that the client side and the servers are lightning fast if users using the web application with poor Internet connections suffer from ineffective data transfer between the client and the server. Unless not specifically monitored, unnecessarily large data transfer are easily missed during development when network latencies are minimal.

Simplest performance evaluating tools would be ones that measure parameter of a single web application operation. They measure, time, network traffic, number of HTTP requests for a single operation. Developer tools found in any modern web browser would be an example of such tool.

Typically, more complex performance testing tools allows complicated simulations to be run which try to mimic real life conditions to their best ability. Example of a such tool would be JMeter, a performance evaluating tool with focus on web applications developed by Apache Software Foundation.

Chapter 5

Current Sales Configurator

5.1 Home Option Configurators In General

One of the integral and most complicated parts of the Sales Configurator is the Option Configurotor. Different kinds of option configurators have been developed to ease the option selection process for customers of the home builders. Compared to, e.g., car industry option configurators home option configurators usually have greater variety of options and also need to give support for custom options[51]. If a customer is willing to pay for an extra room, many home builders are willing to give them that option. The focus of the options configurators differs. Some option configurators focus on graphical presentation like 3D graphics[51], others are more data and rule driven. Sapphire Build falls into the latter category. Sapphire Build Option Configurator is designed to handle a fairly complex rule setup and a simple graphical representation of the effects structural options on the floorplan of the home.

5.2 Origins

The development of the Sales Configurator application was started in the 2005. In that time, the developers came in to the conclusion that even though the Sapphire Build was mainly based on web technologies the functionality requirements of the sales configurator couldn't be met with the web development technologies of the time. The sales configurator could be used on the model homes on the communities with poor mobile internet connections.

5.3 How Is Sales Configurator Used

Sales configurator is typically used when a sales representative meets up with a customer who is willing to buy a home in some specific community. This often happens at a local sales office at the community. Later the same customer can meet multiple times with another sales persons to make additional changes to the outcome of the first meet, e.g., contractual changes or changes to the selections made on the initial meet.

Sales configurator has a lot of functionality that will be out of scope of this migration project. It allows, e.g., typing in mortgage information, managing contingencies, creating custom options, redlining floor plans, it visualizes effects on a structural options on the floor plan of a house, facilitates deposit management and supports electronic signatures so the sales worksheets can be ratified on the spot.

Not all sales configurator users use all of the functionality listed above but all use the following functionality. The most important features, that we will call here "core functionality", include the ability pick the community, lot and a model, pick desired options and submit the selections back to the Sapphire Build ERP system. The steps in detail are:

- Pick Community/Lot/Model

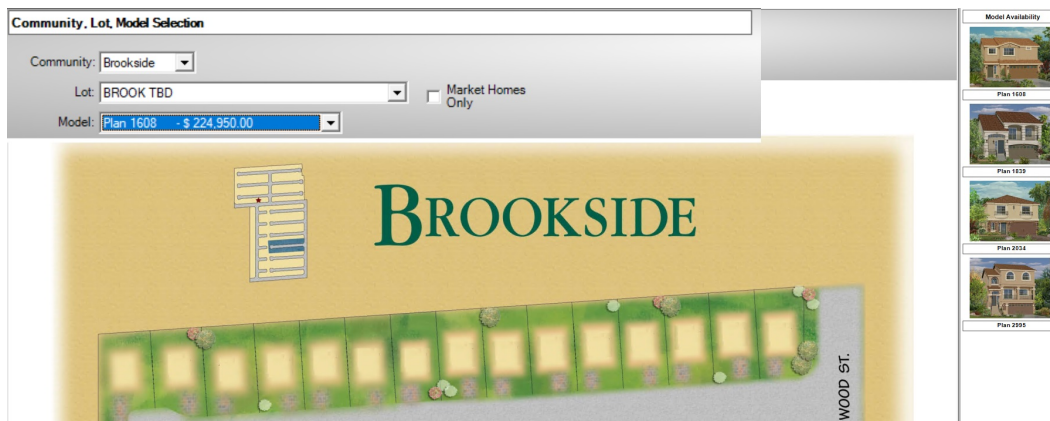


Figure 5.1: Community/Lot/Model Picker

This control allows first selecting the community, then selecting lot and the model. The available lots, as well as the available models, are filtered based on the community selection. Below the dropdown selectors is a "community lot map" where user can pick the desired lot

by clicking the lot location on the screen. To the right is an alternative graphical model picker showing an elevation image for each model.

- Select Options

Figure 5.2: Options with two different visualizations (dropdown and radio button) of multiple choice options and a binary option

Options are grouped either by their categories (e.g., cabinets, plumbing, exterior) or by their locations (e.g., garage, master bedroom, living room). Options can be quantity options (e.g., how many extra sockets), binary options (e.g., window above oven) or multiple choice options (e.g., 1-car garage, 2-car garage or 3-car garage).

Availability of the options and the prices of options are affected by other option selection, the selected community/lot/model (and by other factors as well).

- Print contract and submit to Sapphire build

Finally, when the community/lot/model has been picked, the selections have been made, and other required information has been typed to the sales configurator, a sales worksheet can be printed and submitted to Sapphire Build ERP system.

5.4 Sapphire Build Option/Option Rule Setup

When describing the option rule and option setup here, a simplified version of the actual functionality is used but this is detailed enough for this context.

Sapphire Build support three kinds of options, binary, quantity and attribute (or multi-select) options. Options setup consists of option selection and option values. The option selection is the option itself, e.g., garage configuration and the option values would be the available selection in the option, e.g., 2-car garage or 3-car garage. Binary option and quantity option don't have option values. Available values in a selection can vary based on multiple different attributes. E.g., 3-car garage can be offered only for some models, some communities or even only at some lots. The option setup (available option selections and their values) is calculated by the Sapphire Build back-end when the configurator tells it which model and lot the customer wants to pick. In addition to this static option setup control, option rules allow dynamic control for available options.

One of most important and also more complicated feature of the sales configurator is the option rule engine. That allows builders to setup their options in a way where the sales representatives cannot sell impossible homes (e.g., a house with two floors and three floors) or homes that they for whatever reason don't want to sell because of unnecessary complications and/or profit margin issues (e.g., different type of floor tiles in every room).

Options rules can target option groups (categories or locations), option selections (e.g., garage selection) or option values (available selection values in a option selection, e.g., 2-car or 3-car garage). Option rules consists of a condition statement, condition items, operation and target statement (can contain wildcards).

Here's an example option rule statement from a Sapphire Build customer. The name of the rule is "Disable AutoCourt with Deck Size and Sunroom Bradford".

- Condition Statement: (STR-EXT-EXT:dek-1418 OR STR-EXT-EXT:dek-1420) AND STR-MRM-MRM:mrm-sunr
- Condition Items: STR-EXT-EXT:dek-1418, STR-EXT-EXT:dek-1420, STR-MRM-MRM:mrm-sunr
- Operation: Hide
- TargetStatement: STR-GAR-LOC:gar-aut1

This translates to: If the "14 x 18 Deck" or the "14 x 20 Deck" value is selected on the "Deck/Patio First Floor Arrangement" option and the "Sunroom" value is selected on the "Morning Room Arrangement" option, the system will hide the "3-Car Auto Court" value from the "Garage Location" option. There are several types of option rule operations that can be defined, they are:

- The standard operations: Disable, Enable, Show, Hide, Disable And Hide, Enable And Show
- Add Quantity
 - Affects (adds or subtracts) the value of a quantity option. E.g., adding an extra room can affect the default amount of electrical sockets needed
- Auto Select
 - When the option rule condition is met the target option value is auto selected. E.g., require porch with upgraded elevation
- Auto Select Once
 - Same as above but instead of being a requirement Auto Select Once is more like a suggestion. The auto selected option can be deselected
- Match Condition
 - Force option value at the target to match the value at the condition. E.g., force match the kitchen cabinet style to match the living room cabinet style
- Match Condition Once
 - Same difference between as with Auto Select and Auto Select Once
- Control Values
 - A special case of the Match Condition Rule. Defines that the available option values in the target are defined by the values selected in (multiple) condition options. E.g., require that bathroom cabinets are the same as in the kitchen or in the living room
- Option Cut-Off
 - Can be used to define restrictions on option selections based on the building progress of the house. E.g., if framing has been started structural options (options that affect the structure of the house e.g., extra rooms) are disabled.

5.4.1 Relevance to the migration project

Some of the Sapphire Build users have very complicated option and option rule setup. Typically option visibility can be filtered based on the model and the building site (community or lot). Although not encouraged, the model/community/lot option filtering can be substituted to some degree by model/community specific option rules for controlling the overall visibility of the option or option value. This results in a situation where the option rule engine needs to evaluate possibly thousands of the option rules against thousands of option values every time an option change is made on the configurator. Also, option rules can have cascading effects, e.g., auto selecting some option can trigger the applying of some other option rule. Currently, this logic is run on the Windows native sales configurator application, and it's still cause for some performance concerns with some data setups taking up to several seconds for each option change, but with the web application migration some other approach is needed.

5.5 Sapphire Build Price Setup

Another extremely important responsibility the sales configurator has is the correct calculation of the prices for the selected lot, model and additional options. Again, like with the option setup, the price logic actually is more nuanced than described here but is sufficiently precise to understand the basic functionality.

Lots in a community can have different prices, e.g., cul-de-sac or corner lots can have an extra price called lot premium in the Sapphire Build. Models have a base price, which is the price of the basic model without extra options. This is complicated by the fact that Sapphire allows different base option setups, i.e., different option values can be included in the base model price in, e.g., in different communities. E.g., in a high-end community the base price might be higher but high priced cabinets are included in the base price. If lower level cabinets are selected (if available), a discount is given to the customer. Also, other option selection affect the prices. E.g., upgraded door knobs can be given free-of-charge if upgraded doors are selected. Each option selection or option value is assigned a set of prices that are valid. When multiple prices are valid, the most relevant is selected. This means that community specific price, e.g., for upgraded doors trumps the company wide price, price with option condition trumps the one with no condition and so on. All in all the price of a option is determined by the date when selections are made, the selected community, selected model and other selected options.

5.5.1 Relevance to the migration project

Again, the final processing of prices is done by the sales configurator. The back-end sends the sales configurator the relevant prices based on the lot, date and model selection but the sales configurator is responsible for applying the correct price based on the selected options. This process is not as heavy as the applying of option rules but preferably wouldn't require a round trip to server every time the prices need to be updated.

5.6 Used Technologies

The sales configurator uses the Microsoft .NET technologies. The UI utilizes Windows Forms (or WinForms), which is a GUI library part of the .NET. The source code is written in C#. Sales configurator uses the Microsoft click-once technology, which allows the user to start the Windows native program by just clicking a link on a web page[4]. The problem is that only Internet Explorer and Edge support this out-of-box. Other browsers have to have a 3rd party add-on installed. The click-once smart client communicates with the back-end server using web service calls and gets the initial configuration parameters through query parameters in the click-once download link.

5.6.1 Problems

The main functions of sales configurator were originally planned to work also on an off-line environment. The necessary data was downloaded from a local database and local binary data. In a very simplistic use-case, the user (sales representative) started the configurator with the home buyer present and in the end a sales contract with a specific lot, model, options and prices was printed. This, however, lead to performance problems when the sheer size of the required data grew and the data was moved to a centralized server, but still most of the data was loaded in huge chunks, which then manifested in long load times.

Chapter 6

Methods

6.1 Used Web Development Technologies

It was decided ReactJS will be utilized as the UI framework for the front-end part of the Sales Configurator. ReactJS was already used in some other parts of Sapphire Build ERP system and there is an ongoing effort to rewrite the current complete ERP web UI with ReactJS so it was a natural pick. The ReactJS front-end communicates with the back-end through OData (Open Data Protocol) RESTful web service API. OData is a protocol that standardizes practices how business objects are fetched, modified, added and removed through RESTful APIs originally created by Microsoft[16]. Again, other parts of the Sapphire Build ERP system already offer communication through OData APIs, so it was a natural to utilize and expand current functionality.

6.2 Technical Challenges

Originally it was planned that the price and option rule logic would be run on the server side so that after each option change the changed option set would be sent to server for evaluating the changes in prices and other options. The main reason for this was that the current logic is written in C# and could be run on the server as is. This approach was expected to cause problems with the server load and unwanted delays for the option changes. The time it takes to calculate the option rules and option prices is too high even with the Windows native sales configurator with some more problematic option rule/price setups. The approaches to tackle this problem involved either using a separate server instance just to run the option change calculations (not to kill the main ERP system performance) or somehow port the price/rule

C# code to JavaScript. Unfortunately, the logic is quite complex due to actual complexity of the problem but also due to poor quality of the current C# code that handles that.

6.3 Evaluating success

It was decided that project is successful if the three main functions (pick lot and model, select options, submit contract) of the sales configurator were migrated to a web application so that:

- Sales Configurator can be used on any desktop (not just Windows PCs) and mobile devices with large screens, i.e., tablets
- The Sales Configurator load times are significantly reduced
- The usability is enhanced by more intuitive and modern UI and the overall user experience is "snappier"

Evaluating the last one in a standardized manner is out of the scope of this thesis. One of the reasons why this migration is done is to separate the UI and back-end completely and give our customers who have the expertise to do so, the possibility to modify the UI to their liking.

6.3.1 Performance

The main complains on performance of the current sales configurator usually revolve around its long initial load times and the slowness of the option change operation and hence the performance measurements will focus on these.

6.3.2 Code Quality

The code quality metrics mentioned in the previous chapter will be utilized to get an estimate of the code quality and a small expert review will be conducted. Special interest is given on the correlation between the expert review and code quality metrics since assumably it is difficult to compare code quality metrics between completely different technologies and frameworks (C# WinForms application vs. JavaScript ReactJS application).

Chapter 7

Implementation

As previously stated the technologies selected for the migration projects was a React.JS user interface that communicates on top of a ODATA web API running on a IIS. This chapter that explains the implementation and is divided in two parts. First chapter describes the back-end ODATA API and the second part focuses on the React.JS user interface.

7.1 ODATA Web API

Sapphire Build portal already had an ODATA interface in place to allow some customers to develop their own custom functionality on top of the portal default functionality. The Sapphire Build ODATA interface allows users to fetch any system objects using the default ODATA syntax. To get a single object (e.g., GET webroot/Communities('Community1')) or to get all objects (e.g., GET webroot/Models) or to get some of the objects based on filter statements (e.g., webroot/Lots?\$filter=CommunityRID eq 1). The default ODATA functionality was appended with custom functions to for example fetch option, option rule and option price data on one call. The most important custom functions were:

- POST webroot/Models(1)/SBEntities.GetOptContext
 - Takes community identifier, lot identifier and possibly sales worksheet identifier as parameters
 - Returns a JSON object with following information
 - * Model: name, ID, base price
 - * Lot: name, lot premium price

- * Option Context: option selections, available values, type, name, category/location assignment,
 - * Base Option Values: values of the option selections that are included in the model base price
 - * Default Option Values: Values of the option selections that are selected by default
 - * Rules: option rules translated to JavaScript code that can be evaluated on the client
 - * Floorplans: Information necessary to render a dynamic floorplan on the configurator
- POST `webroot/SlsWsh(1)/SBEntities.SaveOptSelections`
 - Submits the selected options and customer information to the back-end server
 - Takes the following as parameters
 - * List of selected options
 - * The Customer information
 - * The Sales Worksheet information
 - * Selected community, lot and model
 - Returns the following
 - * Flag indicating whether save was successful
 - * Updated (e.g., updated database identifiers) Customer and Sales Worksheet information.
 - POST `webroot/Community(1)/SBEntities.LotsWithModelsForCommunity`
 - Takes community identifier as parameter
 - returns a JSON object with following information
 - Lots: an array of "LotWithModels" containers
 - Models: an array of "ModelWithLots" containers
 - LotWithModels maps lots to available models
 - ModelWithLots maps models to available lots
 - Both LotWithModel and ModelWithLots contains the lot and model metadata (e.g., model image url, lot and model names and statuses)

7.1.1 Authentication and Security

When the new web based sales configurator is launched from within the Sapphire Portal, an authentication token is created and delivered to the client application. The created token is then added to the ODATA web API calls in the REACT.js UI and verified on the back-end for each API call. The token is tied to a specific user in the system and the operations available one ODATA API are filtered based on the permissions of that user. User could for example have the right to view old sales worksheets, or play around with the options (if the web sales configurator would be made available for the home buyers) but not save any changes, i.e., submit sales worksheets.

7.2 React.JS User Interface

A root react component typically defines an element tree[17] and that was the paradigm used in this migration as well. The diagram 7.1 describes the high-level architecture of the web sales configurator.

The high-level components described in the diagram have been further divided to smaller pieces. Material UI[14] React components were used to get a consistent and modern look for the web configurator. Material-UI is a React library that contains UI elements that implement the Material Design look developed by Google[13]. Also, users are often already familiar with the Google styled user interface components which helps to make the configurator intuitive to use. Material-UI contains dozens of different ready-to-use React components of which we utilized switches, dropdowns, tabs, menus, time pickers, dividers and many others.

The React UI architecture was developed using a top-down approach. First, a graphical mockup of the desired configurator was created. That was then recreated using HTML after which the used HTML elements (mainly div-elements) were divided into logical entities and were converted to React components.

7.3 Proof of concept UI

The UI at this point of the sales configurator development was divided to four main level tabs. The final sales configurator will have extra functionality and controls (e.g., for mortgage, contingency and deposit management) whose layout hasn't been decided yet but they will be either added as extra tabs to the main level tabs or as sub-tabs to existing main-level tabs. Also, the

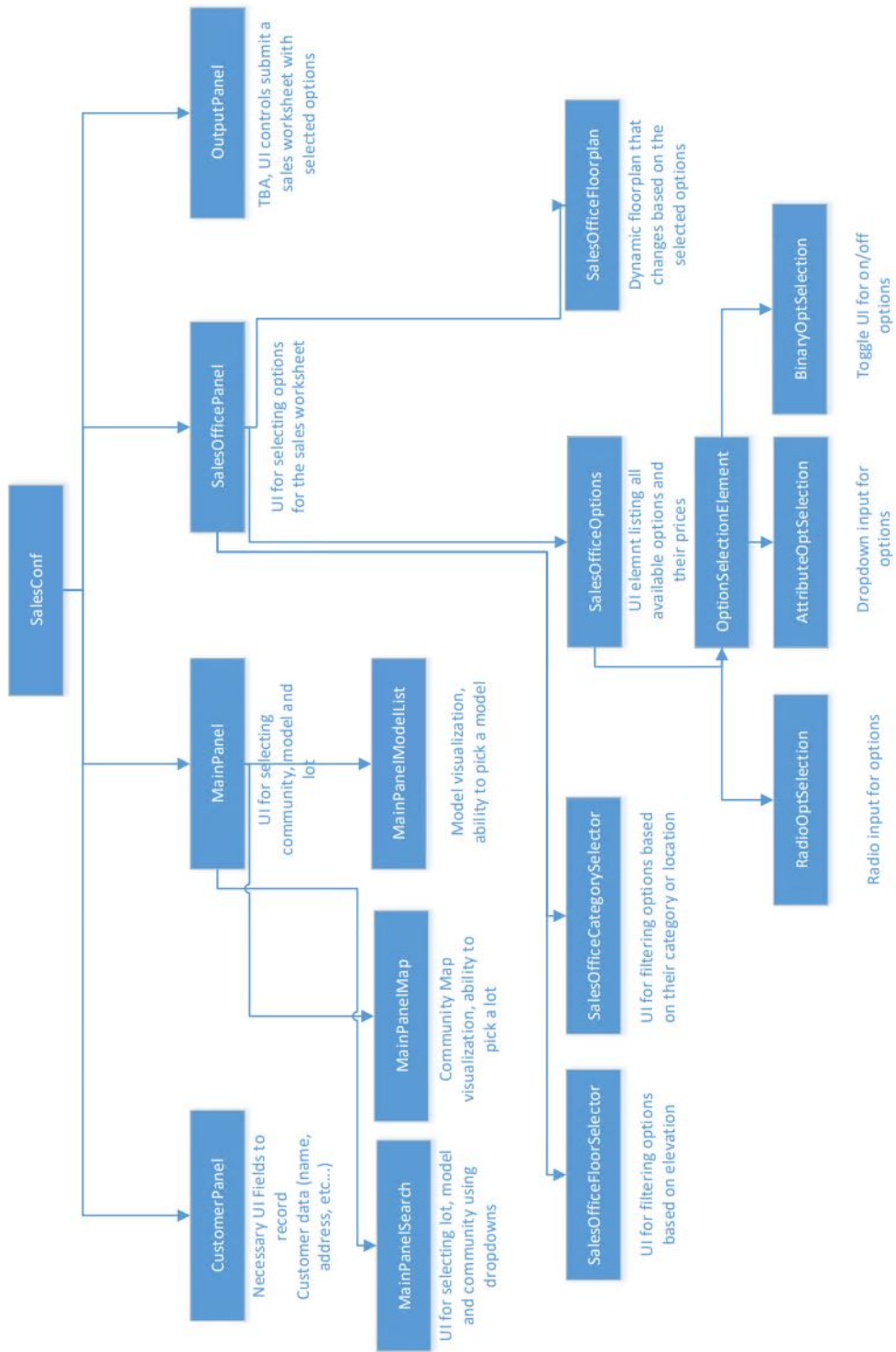
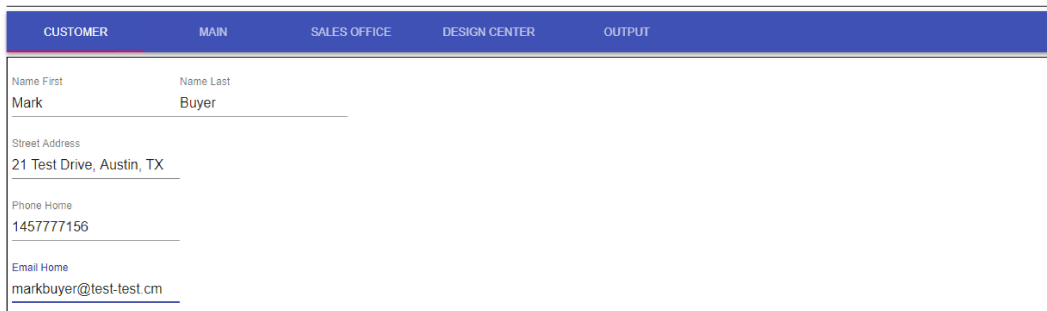


Figure 7.1: High-level architecture of the React element tree

UI at this point is not finished nor polished. Finishing the UI will be part of the last phase of development when the core functionality is implemented.

7.3.1 Customer Panel

Customer panel will host the controls for managing in the customer information. This panel is only partly implemented in the first phase of the migration since the same information can already be typed in using the Sapphire Build portal ASP.NET web interface. The web UI also provides the capability to modify more complex customer related information such as contingencies and down payments, mortgage and realtor information. Ultimately the goal is to allow the end users to be able to input this information from the sales configurator as well.



CUSTOMER	MAIN	SALES OFFICE	DESIGN CENTER	OUTPUT
Name First	Name Last			
Mark	Buyer			
Street Address				
21 Test Drive, Austin, TX				
Phone Home				
1457777156				
Email Home				
markbuyer@test-test.cm				

Figure 7.2: Customer panel

7.3.2 Main Panel

Main panel (figure 7.3) contains the controls to select a community, lot and a model. On the left there are dropdowns that contain the available business unit, community, model and lot selections. The business unit dropdown filters the available communities (business units and communities form a tree hierarchy where communities are leaves of that tree). Community dropdown filters available lots (each lot belongs to a single community) and lot dropdown filters available models (only specific models are available for a single community-lot combination). Also, selecting a model before selecting a model will filter the available lots.

Middle section shows the community map. Community maps can also form a hierarchy. The top level community map has the overall view of the community and it can be divided in to sections. Clicking a section will reveal



Figure 7.3: Main tab of the configurator

more detailed partial view of the community revealing the lots (figure 7.4). Selecting a lot can be done clicking the squares that represents the lots and have the lot number in them.

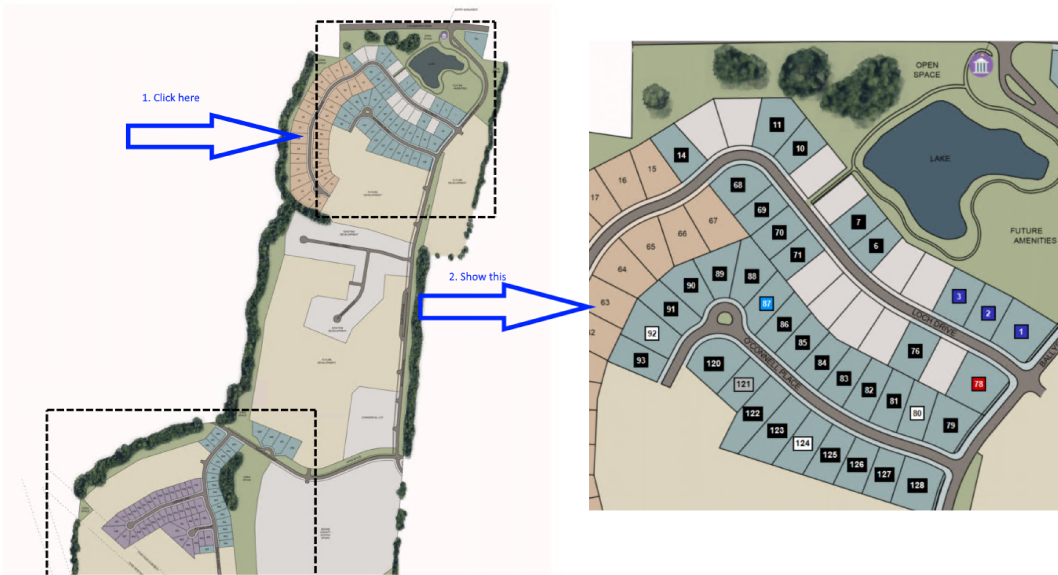


Figure 7.4: Community map hierarchy

The right section shows the available models images and their names.

Model can be picked by clicking on the image.

7.3.3 Sales Office Panel

This panel represents the option selector part of the sales configurator. This view shows the available options for the selected lot and model and provides the controls to select the options. Behind the UI runs a Javascript based option configurator engine that evaluates the availability of the options based on the option rule setup and calculates the prices for the options based on the option price setup.

The left pane shows the available option locations and categories. Each option in Sapphire Build portal is part of one category and location. Categories group the options by similarity e.g., grouping them to plumbing, electrical and cabinet categories. Locations group the options by their location in the house, typically locations loosely correspond with available rooms (e.g., living room or master bedroom).

The middle pane shows the total price of the selected model/lot and the price of the options. As described before there are four different kinds of option selections, binary selections, quantity selections and multi-select options. Binary selections use Material UI Checkboxes, quantity selections use Material UI TextFields, multi-select options use Material UI RadioButtons or Selects (dropdowns).

The right pane of the Sales office panel renders a dynamic floorplan of the model with selected options. This allows the sales agent and customer to see immediately the effects of structure affecting options on a floorplan. This dynamic floorplan is also printed to sales order later on the process when the options selections and the sales order are printed and saved.

7.3.4 Output panel

This panel provides the user the ability to save the selected lot/model/option combination to the server. The page will also have the possibility to print a sales contract (with the selected options and prices) which the sales agent can print and have the customer sign. The Windows configurator currently supports electronic signing through DocuSign. That will be left for later phases in this project.

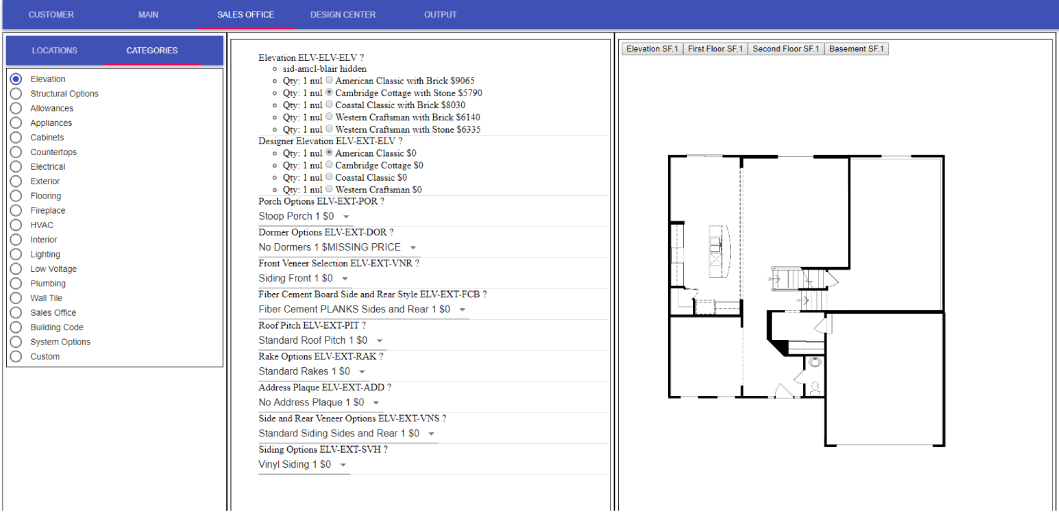


Figure 7.5: Option Selector part of the sales configurator.

Chapter 8

Evaluation

8.1 Overall success of the migration

8.1.1 General

In this phase one of the migration process, the desired core functionality was achieved and tested to be functional. However, lot of the non-critical functionality of the sales configurator was left for later phases. This makes some of the code quality and performance measurements hard to interpret. However, overall the new configurator seems to perform better than the old one. It seems snappier and less clunky than the old one and the performance measurements support that feeling. Also, there's less time when the configurator is unresponsive because the heavy background processes are better handled asynchronously or are significantly faster. On the old configurator some of the heavy operations (e.g., the option change) were much slower and run on the UI thread, which resulted the configurator being completely unresponsive at times. Also, the code base is better organized, functionality is better distributed into separate classes and files. In the old configurator, significant parts of the UI elements were handled in a single god-class and the class/file structure had nothing to do with the UI control hierarchy.

8.1.2 User Interface

The user interface is functional, but not in any way polished. Also, using the Material UI library was not the best choice for this fairly complex application that presents lot of data at single views and also is typically used with desktop browsers. It is clear that Material UI is mainly intended to be used with mobile browsers. The elements in Material UI the Material UI elements take way more space than, e.g., .NET WinForms elements. Luckily the

architecture allows changing the UI library fairly easily and that is something that will happen in later phases.

8.2 Performance measurement results

Even though the web based configurator is still lacking in functionality, the performance measurement results are good estimates of the actual performance enhancement of the migration. Since most of the time is spent, both in the original and the web based sales configurator, when running the core functionality of the configurator (initial fetching of option configuration information, selecting options and eventually saving the selected options to a sales order).

Previously, users have reported long wait times during the launch of the configurator, when changing community/model and rendering the community map, fetching lot and model specific option information, changing selected options and saving the selected options.

The measurements were made using two different client environments (different databases and static image files stored on the file system of the server) since from experience it was known that the data setup could drastically affect the performance of the sales configurator. Event triggers were added to both the old and new configurator to get exact load times for each operation.

8.2.1 Start up performance

Start up performance was known to be much better for the new configurator since it is much lighter - only approximately 3 megabytes of static resources (of which the actual application JavaScript file is only 2.1 megabytes) need to be transferred to show the start screen of the web sales configurator. The Windows native sales configurator on the other hand is more than 90 megabytes. No measurable differences were found between client environments. The measurements were made on an environment where the background server and the clients were running on the same physical machine. If the measurements were made over the Internet, the differences would presumably have been greater. Figure 8.1 has the measurements.

8.2.2 Changing Community

Changing the community (and rendering the community map) is a relatively heavy operation since for each community the configurator needs to fetch

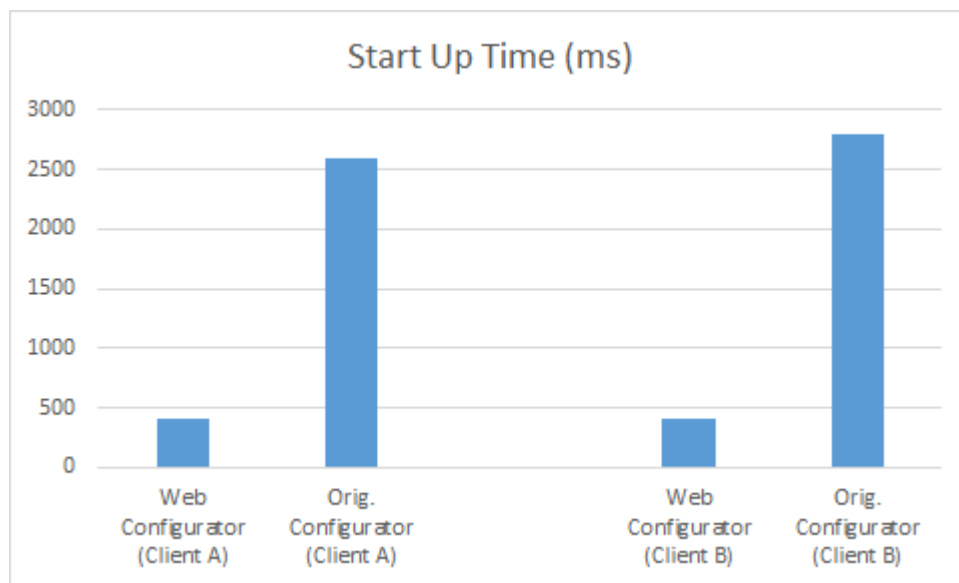


Figure 8.1: Start Up Load Time

the information required to render the community map, fetch all available lots and models (and model images) for the selected community. The load time is heavily dependent on the community that gets selected and therefore 10 random communities were picked for this test and the load times were measured on both configurators. The chart shows the average load times. Figure 8.2 has the measurements.

8.2.3 Load Option Data

This was expected to be the heaviest of all the operations. Here the configurator loads all of the available option data from the server. The data includes available options, option rules, prices, option categories, option locations and much more. This was the only category where the old configurator outperformed the web sales configurator. This isn't a cause for great concern since it is expected that the server calls that fetch this data have plenty of room for improvement in the form of performance optimizations and simplifications. Work is already on the way, which is expected to make the option data loading much faster than it currently is. Figure 8.3 has the measurements.

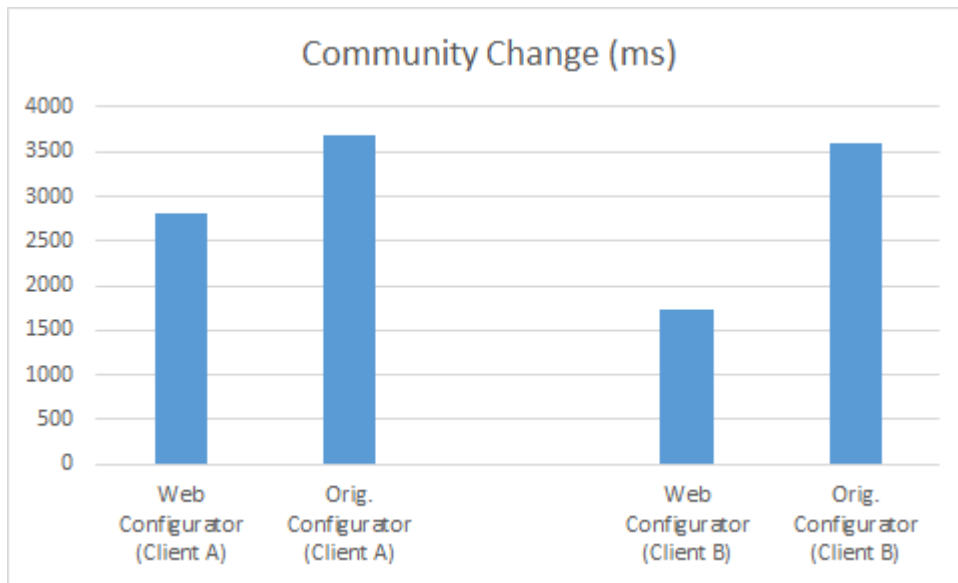


Figure 8.2: Community Change Load Time

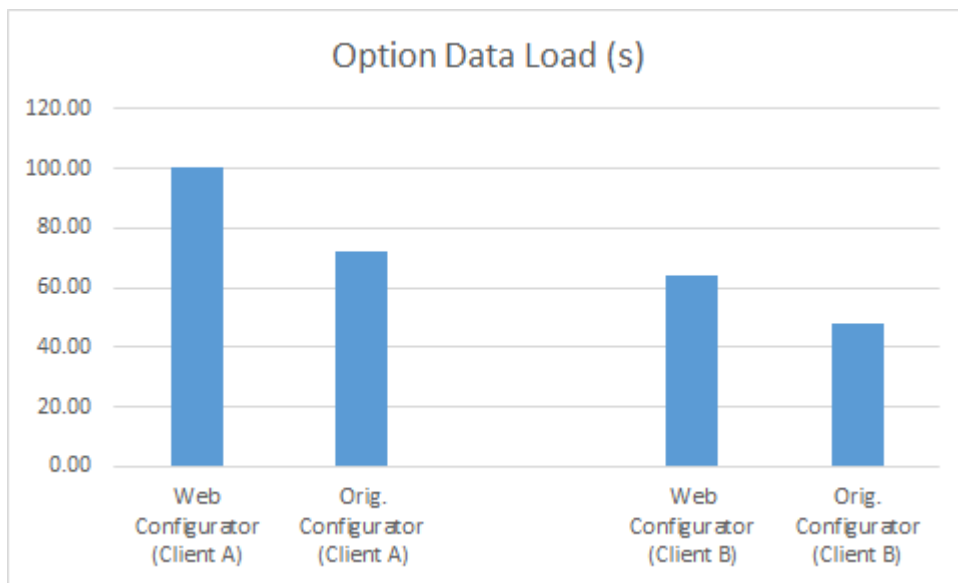


Figure 8.3: Option Data Load Time

8.2.4 Changing Options

Depending on the data setup this can be fairly heavy operation and unique compared to the others since in a typical scenario dozens of option changes can be done in a single session. Also, of all of the operations measured here, option change is the one, which users expect to be fast. This was known to be heavily dependent on the option rule data setup and the option rule engine running on the Windows native configurator was known to be slow and unnecessarily complicated. The average load times for the web sales configurator (66 and 120 milliseconds) fall very near or below the decisecond threshold which users generally feel are instantaneous. The data setup of the client A was deliberately selected as one the test environments since it was known that their extremely complicate option rule setup was causing performance problems on the option selector part of the configurator. Luckily, it seems that the new configurator engine is able to handle them much better than the old one. Figure 8.4 has the measurements.

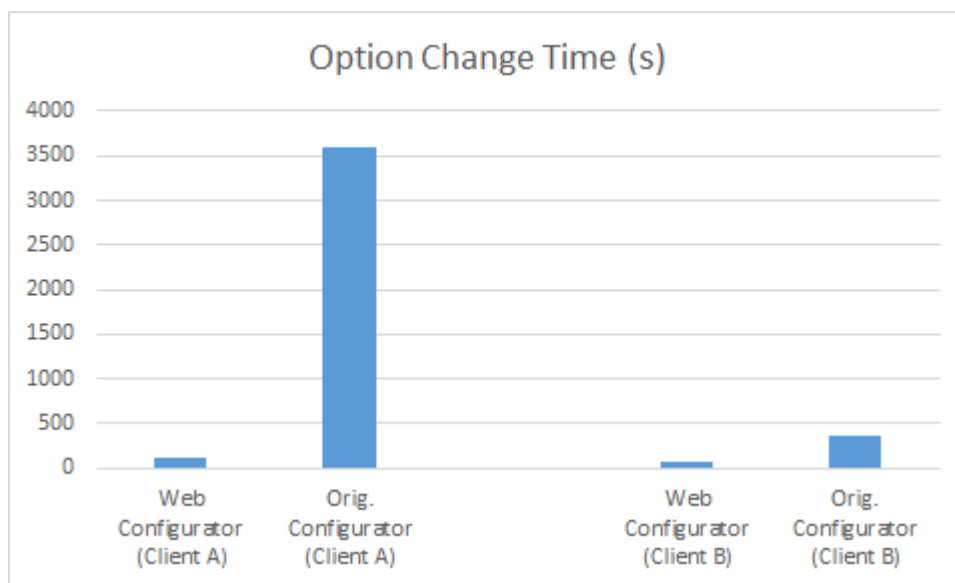


Figure 8.4: Option Data Load Time

8.3 Code quality measurement results

The code quality measurements were trickier to compare. First of all at least with the Windows native sales configurator it's not trivial to say where

the sales configurator ends and the back-end Sapphire build portal starts. They share lot of the same DLLs (even DLLs containing business logic) and it isn't in that sense a typical client-server software whereas the web sales configurator is. Also, the fact that at the moment the web sales configurator is missing a lot of the secondary functionality compared to the Windows native configurator made the comparisons difficult. It doesn't make sense, e.g., to compare the total code line count from both projects since the new configurator is lacking lot of the non-core functionality. A lot of manual work had to be done in order to identify the code base from the old configurator that is responsible for creating roughly the same functionality as the new configurator has.

8.3.1 Measuring JavaScript Code Quality

Typically React is written using JSX, a JavaScript extension that allows React developers to utilize HTML-like elements to describe UI within the JavaScript application logic. While great for the development, it proved to be a drawback when looking for tools that can calculate code quality metrics for JavaScript code. There seems to be only few tools that can calculate quality metrics out of JSX code directly.

First, a tool called complexity-report [6] was used to measure the JavaScript code quality metrics but like many other tools, this was unable to measure JSX code. To get the complexity report generated JavaScript compiler Babel [3] was first used to compile the .JSX files to .JS files and only then the complexity-report was run. This compilation naturally altered the files and even though they still were logically equivalent, e.g., the line count increased very significantly (up to 3 times the line count of the original .JSX file) due to this conversion. Using alternative code analysis tools and manual analysis it was noticed that the code quality metrics weren't applicable after the Babel compilation.

Another tool called es6-plato [7] is able to calculate code quality metrics directly from JSX files and was deemed to be better suitable for analyzing a React project. Plato calculates, file by file cyclomatic complexity values, total lines of code and an average maintainability index which happens to be by default comparable to the Windows-style maintainability index that is also capped at 100.

8.3.2 Measuring C# Code Quality

The original sales configurator has been developed using various Microsoft Visual Studio versions and Visual Studio also has decent build-in code qual-

ity measurement tools. Visual studio can out-of-box generate a code quality report which includes maintainability index, cyclomatic complexity, depth-of-inheritance, class coupling and lines of code metrics. The reports has these values calculated at the function, class and namespace level. The measurements for this thesis were generated by the Visual Studio 2017 Enterprise edition.

8.3.3 Comparing The Results

As stated before, the problem was how to confine the scope of the files that should be included in the report to get comparable results. A lot of manual code inspection was needed to get a source file set that as best as possible represents the current web sales configurator functionality. A C#-attribute "GeneratedCodeAttribute" can be used to identify computer generated code but also one of the side-effects the attribute has, is that the code flagged with that attribute is ignored when Visual Studio calculates code quality metrics. That attribute was used to flag methods and classes in the old sales configurator that contain functionality that was not yet migrated to the web sales configurator.

The measurements on both web and Windows native configurator focused on code that can be considered either UI representation logic or data model logic since that was the biggest and the most significant change between the two. Especially in the UI representation logic, it was clear that the WinForms code is significantly more complex and requires a lot more code lines.

8.3.4 Results

The result chart shows the average maintainability index of the included classes in the metrics calculation, the total sum of the cyclomatic complexity indexes and the total sum of the lines of code.

The biggest surprise was how much more complex (according to cyclomatic complexity) the C# code was compared to JSX code even when taking the increase in code line into account. The discrepancy was so big that there was doubt that the algorithms between Visual Studio and Plato differ somehow. Some C# code was manually converted to JavaScript to verify that the calculation logic is indeed the same.

Not surprisingly the maintainability index was also improved during the migration. The surprise was that according to maintainability index, even the old configurator was considered to be fairly well maintainable. According to Microsoft, maintainability index values above 20 get a "green" rating

[5]. That is from the developers' experience known not to be the case. Different metrics and manual reviews would be needed to find other problems in the configurator biggest of which are considered the messy architecture, complicated information flow and at times extremely poorly named variables and WinForm controls. A special tool was at one point developed where a developer can hover over the WinForms control to find out the name and id of that control - example name could be Label45.

Lines of code reduce was expected from the start. In JSX code you can in a single line define a user control type (e.g., a Dropdown), define its name, define its attributes and location in the UI. All of this in WinForms C# code requires, depending on the control, up to dozens of rows. Ideally this would be automatically generated code that would be generated by the Visual Studio WinForms designer functionality and could be excluded from the code calculation metrics but in the old configurator that is not always the case. The old sales configurator was in the early stages developed this way but the functionality available in the designer was deemed insufficient at the time and some of the designer generated code was then manually edited which gave flexibility but also was and is tedious.

The below chart has the results of the metrics calculation:

Version	Maintainability	Cyclomatic Complexity	Lines Of Code
Orig. Sales Conf.	55	4231	13245
Web Sales. Conf	70	420	3492

Chapter 9

Discussion

9.1 Migration Insights

The migration process all-in-all can be considered a success. The new web application compared to the old configurator is lighter, performs generally better and is platform independent - all features that one would expect to get out of this kind of migration project. The structured performance measurements gave additional confidence in that the migration went and is going in the right direction. The benefits of the code quality measurements are harder to evaluate. Based on this project I think that code quality metrics can be useful tool when trying to enhance any existing project but are difficult to use as a tool for comparing two different projects - especially if they are using different technologies.

React as the chosen UI framework seems to be the good choice for this kind of project. It is well documented, well supported and popular which gives confidence in its future. Generally, great care should be given when choosing the UI platform since there are dozens frameworks to choose from and new ones spawn and old ones fade away constantly.

The UI architecture and the top-down methodology how it was created proved to be a good way to come up with a React element architecture from scratch. The resulting hierarchy seems to be flexible for future enhancements and that will be put to a test when the rest of the UI functionality starts to get migrated.

Greater consideration should have been given for selecting the UI control library. The Material-UI was deemed not be the greatest choice for this kind

of application that mimics a full Windows native desktop application. While the intention is that the web sales configurator will be usable with a touch-screen, by far the more common use-case, at least for the moment, will be with a mouse and a traditional high-resolution large screen. The Material-UI can be configured to be more suitable for this scenario, but it probably is easier to find a library that is a better fit out-of-box.

9.2 Future Work

The web sales configurator is still in early phases of the development. Even though the core functionality was successfully migrated, great development effort is still needed to have all of the existing functionality migrated. As the sales configurator have been developed for more than a decade there's also a lot of functionality that is no longer needed and can be excluded from migration process since the customer requirements have changed. Figuring out, which functionality can be left out won't be a small task either.

The back-end functionality still needs revision and enhancement. Like the web UI, the back-end has still a lot of missing functionality and also was causing the slowdowns that made the web sales configurator lose to the old configurator in the option data load measurements. Improvements are on the way and it is believed that contrary to the old configurator there are still a lots of obvious performance bottlenecks that are easy resolve and thus enhance the performance.

The old configurator didn't have really any kind of unit test, but now the intention is to write unit tests at least for the core functionality. Since the configurator deals with prices and customer selection it is critical that the configurator produces correct results. Some automated test scripts were created for the old configurator but they covered only parts of the functionality and were difficult to setup. Automated test for the React UI should be easier to develop and maintain.

Chapter 10

Conclusions

This thesis aimed at designing and developing a web based sales configurator from a Windows native configurator. It touches on the reasons why such migration is a good idea and what drawbacks it entails. This thesis also includes a brief summary of different web application technologies, what kind of architectures are typically used and how web application quality and performance can be evaluated. Since the Windows native configurator was known to be suffering from poor code quality, part of this thesis focuses on ways how to evaluate and better the code quality.

The first step when designing a windows native application to a web application migration is to select suitable technologies for the client side as well as for the back-end server side. While the client-side framework selection was a success, more emphasis should have been aimed at selecting a suitable a UI library for the selected web development framework (in this case React). The selected web development framework React was found to be easy to understand, easy to develop and support resources easy to find.

The selected back-end technology (ODATA web API running on ASP.NET) was found to be a good match with the React - at least for this project. Even though at this point of the migration the ODATA capabilities were not used to their full extent, it is very likely that in the future the standard ODATA functionality will prove to be useful and make the further development of the configurator easier and quicker.

The code quality measurements gave some indication of the success of the migration but the results were hard to compare and somewhat indefinite. Better approach probably would have been to run the code quality metrics throughout web application development to catch possible issues early and give less focus for the final application comparison.

Bibliography

- [1] 2017 housing giants rankings. <https://www.probuilder.com/2017-housing-giants-rankings> Accessed 31.8.2017.
- [2] About eslint. <https://eslint.org/docs/about/> Accessed 21.03.2018.
- [3] Babel js. <https://babeljs.io/> Accessed 22.02.2019.
- [4] Clickonce security and deployment. <https://msdn.microsoft.com/en-us/library/t71a733d.aspx> Accessed 13.05.2018.
- [5] Code metrics ? maintainability index. <https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/> Accessed 22.02.2019.
- [6] complexity-report. <https://www.npmjs.com/package/complexity-report> Accessed 22.02.2019.
- [7] es6-plato. <https://www.npmjs.com/package/es6-plato> Accessed 22.02.2019.
- [8] escomplex. <https://www.npmjs.com/package/escomplex> Accessed 02.04.2018.
- [9] Hotframeworks. <http://hotframeworks.com/#rankings> Accessed 22.10.2017.
- [10] Html5 is a w3c recommendation. <https://www.w3.org/blog/news/archives/4167> Accessed 13.02.2018.
- [11] Introducing jsx. <https://reactjs.org/docs/introducing-jsx.html> Accessed 20.03.2018.
- [12] Java web start. <http://docs.oracle.com/javase/8/docs/technotes/guides/javaws/> Accessed 18.10.2017.

- [13] Material design - introduction. <https://material.io/design/introduction/#> Accessed 11.08.2018.
- [14] Material-ui - react components that implement google's material design. <https://material-ui.com/> Accessed 11.08.2018.
- [15] Netscape and sun announce javascript, the open, cross-platform object scripting language for enterprise networks and the internet. <http://tech-insider.org/java/research/1995/1204.html> Accessed 12.02.2018.
- [16] Open data protocol. <http://www.odata.org/> Accessed 17.06.2018.
- [17] React components, elements, and instances. <https://reactjs.org/blog/2015/12/18/react-components-elements-and-instances.html> Accessed 11.08.2018.
- [18] Rfc 1794: Dns support for load balancing. <https://tools.ietf.org/html/rfc1794> Accessed 13.05.2018.
- [19] Sapphire build enterprise management suite. <http://www.kovasolutions.com/>.
- [20] Virtual dom and internals. <https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom> Accessed 20.03.2018.
- [21] Code metrics values, 2017. <https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-value> Accessed 12.03.2018.
- [22] ADOBE. Flash and the future of interactive content, 2017. <https://blogs.adobe.com/conversations/2017/07/adobe-flash-update.html> Accessed 4.10.2017.
- [23] ARCELLI, F., BRAIONE, F. P., AND ZANONIA, M. Automatic detection of bad smells in code: An experimental assessment, 2011.
- [24] ARSENAULT, C. 14 important website performance metrics you should be analyzing, 2017. <https://www.keycdn.com/blog/website-performance-metrics/> Accessed 17.06.2018.
- [25] CARDELLINI, V., COLAJANNI, M., AND YU, P. S. Dynamic load balancing on web-server systems, 1999. May/June 1999, IEEE Internet Computing, pages 28-39.

- [26] CRAVEN, J. What is a production home builder? <https://www.thoughtco.com/what-is-a-production-home-builder-175921> Accessed 24.5.2016.
- [27] CROCKFORD, D. Javascript: The good parts, 2008.
- [28] DIANA M. SELFA, MAYA CARRILLO, M. D. R. B. A database and web application based on mvc architecture, 2006.
- [29] DYKES, S. G., ROBBINS, K. A., AND JEFFREY, C. J. An empirical evaluation of client-side server selection algorithms, 2000.
- [30] FLANAGAN, D. Javascript: The definitive guide, 2006.
- [31] FOWLER, M. Refactoring: Improving the design of existing code, 2002.
- [32] GORTON, I. Software quality attributes, 2011. Essential Software Architecture, Chapter 3.
- [33] GROUP, O. M. How to deliver resilient, secure, efficient, and easily changed it systems in line with cisq recommendations, 2006.
- [34] HOETZLEIN, R. C. Graphics performance in rich internet applications, 2012.
- [35] HOMER, A. Components and web application architecture, 1999. <https://msdn.microsoft.com/en-us/library/bb727121.aspx> Chapter 13 from Professional Active Server Pages 3.0 Accessed 8.10.2017.
- [36] HONKANEN, J. Reactjs, 2017.
- [37] KEMP, J., APPELQUIST, D., MALHOTRA, A., AND RAMAN, T. Web applications architecture, 2010. <https://www.w3.org/2001/tag/2010/05/WebApps.html> Accessed 8.10.2017.
- [38] KRASNER, G. E., AND POPE, S. T. A description of the model-view-controller user interface paradigm in the smalltalk-80 system, 1988. Journal of object oriented programming.
- [39] LINCKE, R., LUNDBERG, J., AND LÖWE, W. Comparing software metrics tools, 2008.
- [40] LOURIDAS, P. Static code analysis, 2006. Jul/Aug 2006, IEEE Software, pages 58-61s.

- [41] MASOUD, F. A., HALABI, D. H., AND HALABI, D. H. Asp.net and jsp frameworks in model view controller implementation, 2006.
- [42] MEIER, J., FARRE, C., BANSODE, P., BARBER, S., AND REA, D. Performance testing guidance for web applications, 2007.
- [43] MICROSOFT. Microsoft silverlight release history. <https://www.microsoft.com/getsilverlight/locale/en-us/html/Microsoft%20Silverlight%20Release%20History.htm> Accessed 8.10.2017.
- [44] MOLYNEAUX, I. The art of application performance testing, 2009.
- [45] MUKHERJEE, S. Source code analytics with roslyn and javascript data visualization, 2016.
- [46] NIELSEN, J. Response times: The 3 important limits. <https://www.nngroup.com/articles/response-times-3-important-limits/> Accessed 23.04.2018.
- [47] OFFUT, J. Quality attributes of web software applications, 2002. March/April 2002, IEEE Software, pages 25-32.
- [48] PIERO FRATERNALI, GUSTAVO ROSSI, F. S.-F. Rich internet applications, 2010.
- [49] PIZKA, M. How to effectively define and measure maintainability, 2012.
- [50] REENSKAUG, T. The original mvc reports, 1979.
- [51] RENEE PUUSEPP, T. L., AND KIVI, K. Enabling customer choice in housing, 2016.
- [52] SCHMITT, C., DOMINEY, T., LI, C., MARCOTTE, E., ORCHARD, D., AND TRAMMELL, M. Professional css: Cascading style sheets for web design, 2008.
- [53] SHKLAR, L., AND ROSEN, R. Web application architecture - principles, protocols and practices, 2009.
- [54] SMITH, J. Moving to html5 premium media. <https://blogs.windows.com/msedgedev/2015/07/02/moving-to-html5-premium-media/> Accessed 8.10.2017.
- [55] SONEWAR, P. A., AND THOS, S. D. Detection of sql injection and xss attacks in three tier web applications, 2016.

- [56] TOPIC, D. Moving to a plugin-free web. <https://blogs.oracle.com/java-platform-group/moving-to-a-plugin-free-web> Accessed 8.10.2017.
- [57] VORA, P. Web application design patterns, 2009.
- [58] YAMASHITA, A., AND MOONEN, L. Do code smells reflect important maintainability aspects?, 2012. 2012 28th IEEE International Conference on Software Maintenance.