DALARNA UNIVERSITY

Front-end and back-end separation:

Bachelor thesis

From an integrated MVC project to a web application with web APIs

Author: Zifan Xiao, Kennedy Walusimbi
Supervisor: Rueben Lareya
Examiner: Arend Hintze
Subject/main field of study: Informatics
Course code: GIK28T
Credits:15 HP
Date of examination:

Abstract

More and more front-end frameworks, such as React, Angular, and Vue, had come out and applied to web application development in recent years. The trend of using new web frameworks had increased since the concept of front-end and back-end separation had brought to software production. However, applying new frameworks in some of the old architectures may bring up challenges.
This study aimed at creating a solution to migrate an ASP.NET MVC web project into the new architecture of front-end and back-end separated web applications. What this study does contribute to is an in-depth understanding of the two different architectures in web application development with comparisons. Upon internet search for science sources concerning the separation of front-end and back-end coupling projects, limited literature was found. However, a number of posts found on different forums where developers argue about the demand for this architecture change. With knowledge archived from literature studies, we differentiate the ASP.NET MVC structure from Front-end and Back-end Separated architecture as a start of the study and understanding of these structures. With help from the book of Oates, different methods of performing research are found and used along with the research.
In this study, we identified the View Controller of an ASP.NET MVC project is the crucial point to migrate to the application of those popular web frameworks.

Keywords:

Front-end and back-end separation, MVC, Component-based architecture, front-end development, back-end development, angular

Table of contents

1. Introduction 1
   1.1 Background 1
   1.2 Problem and research questions 1
   1.3 Objectives 2
2. Theoretical background 3
   2.1 The MVC pattern and development of web applications 3
   2.2 A system proposal for rapid web application development 5
   2.3 Applications of front-end and back-end separation 6
   2.4 Popular web application frameworks - Angular as an example 7
3. Research methodology 9
   3.1 Research strategy – Case study 9
   3.2 Data Collection 10
      3.2.1 Literature review 10
      3.2.2 Interview 11
      3.2.3 Code analysis and document studies 12
4. Results and analysis 14
   4.1 The models of the two different architectures 14
      4.1.1 The model of the ASP.NET MVC project 14
      4.1.2 The model for Angular 8 web application with web APIs 15
   4.2 Architecture migration 16
   4.3 Code analysis 17
      4.3.1 The Models 17
      4.3.2 The Views 18
      4.3.3 The Controllers 19
   4.4 Task distribution in front-end and back-end separation project 23
5. Discussion 26
   5.1 Method reflection 26
   5.2 Future work 26
6. Conclusion 27

References 28

Appendix A 30

Interview questions **30**
Appendix B **31**
Interview result 1 ..... **31**
Appendix C **32**
Interview result 2 ..... **32**
Appendix D **33**
SLOC Counting Rules for C# **33**

List of Figures

Figure 1. The MVC pattern (Model-view-controller, 2020) 3

Figure 2. A sample application in thin-client architecture (Leff & T. Rayfield, 2001)..... 4

Figure 3 MVC structure proposal (Dragos-Paul & Adam, 2014)..... 5

Figure 4 Angular component-based architecture (Infragistics, 2017) 7

Figure 5 Model of the research process by Oates 9

Figure 6 Adapted Research Process 10

Figure 7 The structure of a fron-end and back-end coupled MVC project 3

Figure 8 The structure of an Angular 8 web application with web APIs..... 4

Figure 9 Migration between two architectures..... 5

Figure 10 Code resuse in migration..... 7

Figure 11 Tasks assignment in the development of the ASP.NET project..... 9

*Figure 12 Task assignment in a front-end and back-end separated project*10

Figure 13 A proposed way of task assignment.....14

List of Tables

Table 1 Keywords for literature search 15

Table 2 Code analysis of the ViewModels.....16

Table 3 Code analysis of the Controllers 22

Glossary

- Model View Controller (MVC)
  - A software architecture design pattern used to separate logic and interface functionality into three elements in an application.
- Front-end development
  - Front-end commonly referred to as the client-side development that focuses on everything that one sees, experiences, touches and interacts with on a website or in an application.
- Back-end development
  - Back end is commonly referred to as the server-side development where the focus is put on functionality on data or algorithms on how the application should act.
- Thin client approach
  - A computer that runs from sources stored on a central server as a substitute for a localized hard drive.
- Front-end and back-end separation
  - An industrial standard of web application development which decouples the services and functionalities in front-end and back-end.
- Angular
  - A component-based architecture for building web and mobile applications.
- Source lines of code (SLOC/LOC)
  - The number of lines in the source code of a program.
- Total Lines of code (TL)
  - The total number of lines of code in one version of a program
- Continuing lines of code (CL)
  - The number of lines of code that are present in a previous version(m) and are unchanged to the newer version(n).

1. Introduction

1.1 Background

As web technology evolves during the past decade, there are more and more choices for developing a web application. The MVC pattern, which was traditionally designed for graphical user interfaces in the 1980s (Krasner & Pope , 1988), later contributed a lot in designing web applications. This pattern and its Derivatives (MV*) continued its use in a lot of web technologies and frameworks, such as Java Web, ASP.NET MVC, and AngularJS.

In modern web development, developers have to consider more in the selection of web technologies when they are planning to migrate to a newer technology to meet business needs and take advantage of its new features.

According to developers in the local branch of CGI in Borlänge, in one ASP.NET web project, the code was not well documented and mixed with plain HTML files as extensions for some of their current needs. Maintenance was not conducted in a well-organized way. It created a demand for developing a new front-end to avoid costly maintenance and allow more efficient collaboration between developers.

This study analyzes the structure of an ASP.NET MVC web application and provides a possible method to enable it to connect to a rebuilt front-end in a component-based web framework. It was based on our previous work on this project, where we managed to implement the separation into a new Single Page Application (SPA) in Angular 8 with customized web APIs.

1.2 Problem and research questions

In the case of CGI, the project in ASP.NET MVC had its SQL server database aside with the web application in one project. The controllers in C# handles requests and decides which of the views to render. Its Views of front-end, cshtml files which also used AngularJS that enables more flexible and convenient data binding. It also had been extended with HTML modules and imported JavaScript during the maintenance throughout the years. The original front-end and back-end did not allow individual testing, maintenance, or deployment.

The programmers can naturally choose Angular 8, the newer and stable version of Angular, in the rebuilding of their web application. However, Angular 8 is an entirely different web application development framework in component-based architecture (CBA). It is used as a SPA thick-client approach, which is loaded on the client-side and kept updated by requesting data from the back-end (Flanagan, 2006). The original front-end that was built in ASP.NET MVC and AngularJS resides more on the server and requests for new Views to keep the client-side up to

date. Thus, the front-end and back-end of the original project needed to be decoupled (Shilpi, 2019), and build APIs providing the same data - used in generating the Views of an ASP.NET MVC project - in implementing an Angular 8 front-end.

It is unavoidable to make these adaptions in order to apply the advantage of the new technology of CBA. Therefore, it is believed that, in some practical projects in web programming, there is a demand for reforming the ASP.NET MVC web project to a front-end and back-end separated project. To analyze the need for front-end and back-end separation and find a way for this migration, we designed the following questions to conduct our research on the development:

- What is the difference between the two architecture of web applications?
- How could the original ASP.NET MVC code be adopted in the implementation of a new front-end and back-end separated system?
- How the task assignment among developers in a separated web application structure helps to make efficient development?

1.3 Objectives

This study aims to create a solution for ASP.NET MVC web projects to migrate to front-end and back-end separation web application web APIs. Thus, the research objectives are defined accordingly.

To have a better understanding of the two web application structures, we will review relevant literature and create models based on it. Further, we analyzed the code of the three MVC elements used in the ASP.NET MVC web project and the code in the implementation of web APIs. In addition to this, we will conduct interviews aimed at the developers for a clearer insight into the change of task assignment in the front-end and back-end separated project.

2. Theoretical background

This section enriches with research found concerning MVC, web application development, component-based architecture, front-end and back-end separation. Which later results to how it contributes to our own case.

2.1 The MVC pattern and development of web applications

Model-View-Controller (MVC) is a software design pattern. By dividing the program logic into three connected elements, the internal representations of information are separated from the information representations related to the users. Initially designed in the 1970s for graphical user interfaces (GUIs), this pattern became more famous for web application development.

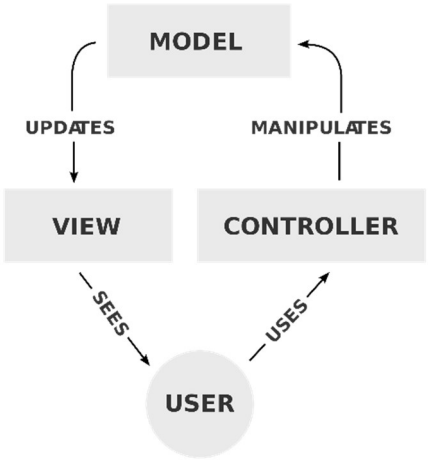

Figure 1. The MVC pattern (Model-view-controller, 2020)

Scott Guthrie of Microsoft created the framework of ASP.NET with MVC pattern later in 2007. Most of the IT companies like to use the MVC pattern to help with developing web projects. According to Aku Kolu (MVC Frameworks in Web Development, 2012), "The ASP.NET MVC simplifies the generation of views greatly." By using ASP.Net MVC with built-in template engine, called Razor, a web application with controllers manipulating the connected database can be easily created. The use of a thin client approach (Model-view-controller, 2020) in ASP.NET MVC web application development made it more responsive on the client-side when the server had enough performance. Lots of web applications remained MVC structured and are being served and maintained.

Since the ASP.NET MVC is using a thin-client approach that places almost the entire structure of MVC logic on the server, the client requests for updates of the web page by making hyperlink requests or form submissions. Thus, the controllers

handle end-user interaction and display the views with customized instances of models.

In the article about web development with the MVC pattern (2001), Leff and T. Rayfield explained the MVC pattern in detail. They also stressed that an appropriate partition for a web application is needed when applying the MVC pattern to its designing. When deployed as a thin client, only the Views of the web application is resident on the client; the Models and most of the Controllers resident on the server. When deployed as a thick client, all three MVC elements are on the client.

- Web application partitioning

As Leff and T. Rayfield stated, "the partitioning of a web application greatly affects the way that the application is implemented". A web application can be in a thin-client or fat-client approach depending on how the application is partitioned in its design. When most of the controllers run on the server, it is considered as thin-client; when most of the controllers run on the client, it is considered as fat-client.

- The structure of a web application in thin-client architecture

According to the sample MVC application, which was deployed to the thin-client architecture the structure of this web application should be like Figure 2.

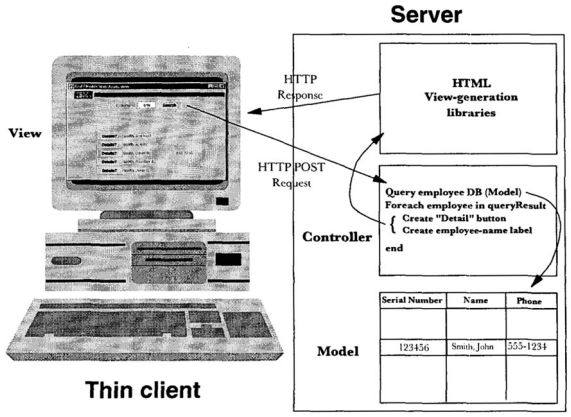

Figure 2. A sample application in thin-client architecture (Leff & T. Rayfield, 2001)

The client, its front-end, uses http requests to communicate with its server after the user made operations. By receiving the response of HTML views that are dynamically generated with controllers and models, the clients can be kept updated.

2.2 A system proposal for rapid web application development

In the research of Designing an MVC Model for Rapid Web Application Development (Dragos-Paul & Adam, 2014) intended to improve the existing design paradigms and patterns for web application development in order to reduce the time cost in production. They described how the efficiency of web application development can be improved by forming a certain structure. Dragos-Paul and Adam went deep into the MVC pattern with their understanding of current web technology and extended the original structure.

The development structure needed to be divided into several categories, as it is shown in Figure 3, including model, View, Controller, and data source, etc. They stated that, "for efficient development of web applications, there is an acute need to separate their presentation form logic and data storage."

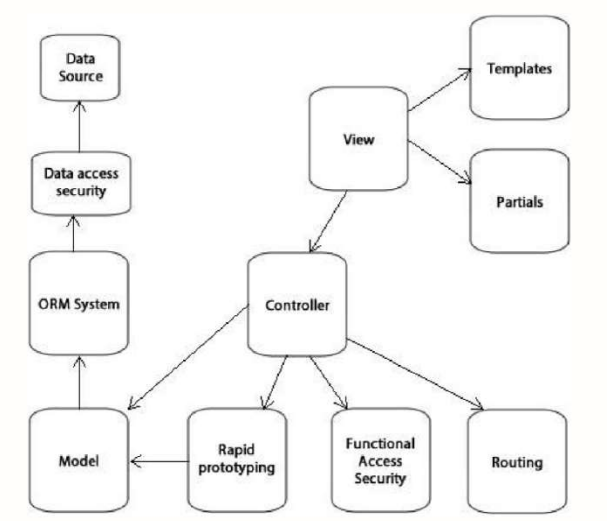

Figure 3 MVC structure proposal (Dragos-Paul & Adam, 2014)

They stressed out that there are three main roles for MVC, development, design, and integration, which allow for more efficient collaboration. The development role gathers programmers that are responsible for implementing the logic of the application. The design role is mainly about user interfaces and data display. The integration role is for the developers to connect the work of the other two roles. Correspondingly, these roles are the Model, View, Controller of an MVC web application.

2.3 Applications of front-end and back-end separation

The current studies that mentioned the software architecture of front-end and back-end separation are about the implementation of a software that has separated front-end and back-end.

As Shiqi Guan, et al. (2018), they implemented a remote-control laboratory system with front-end in React and a web API server as back-end. In modern IT-industry, it brings more security and stability when front-end and back-end are developed, tested, and deployed independently. To make an efficient collaboration between front-end developers and back-end developers, they communicate with data stored in JavaScript Object Notation (.JSON) format. Shiqi Guan, et al., stated in their study," The future trend of development is the separation of front-end and back-end."

In the research of Liu, K. et al. (2017), they divided the system architecture into three components – the client, the application server, and the database. Ajax was used in the data exchange between the front-end (the client) and back-end server. They concluded that their system development with front-end and back-end separation helped in designing the easy, reliable and advanced system and met the objectives and demands of long-term system development. The flexibility and adaptability of the system were enhanced. The architecture of front-end and back-end separation highlights the back-end servitization and the diversification of front-end frameworks.

Yunrui, Q. (2018), also indicated several advantages of conducting front-end and back-end separation in system development. In their implementation of a warehouse management system, the software architecture was designed to separate the front-end and back-end in individual deployments. They managed to reduce the pressure on the server and brought the benefit of less maintenance cost in the separation design.

2.4 Popular web application frameworks - Angular as an example

Different from ASP.NET MVC that uses a thin client, the relatively new font-end frameworks in Component-Based Architecture (CBA), such as Angular, React, Vue.js, and Ember, create thick-client applications to provide a rich, interactive user experience and better performance (Is React Fast Enough, 2018). This sort of component-oriented web development exploded into popularity due to the higher performance of client devices and browsers in the late 2010s. This improvement has released the pressure of the back-end server and allowed developers to take advantage of the clients for lightweight data processing work.

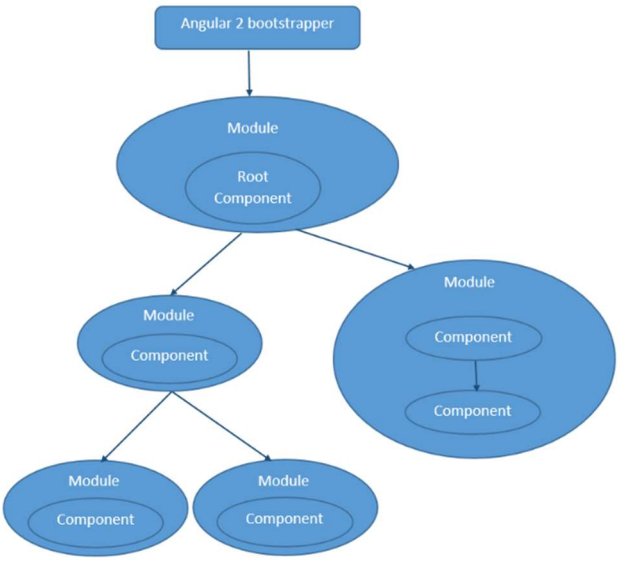

Figure 4 Angular component-based architecture (Infragistics, 2017)

Angular is a framework for building patron applications or webpages across the desktop, web and even native mobile apps. Web applications created in Angular are referred to *single web applications* which update pages by rewriting the current page with data from the web server. Angular's first version, AngularJS was originally developed back in the year 2010 by Google but rebuilt with typescript to their second version Angular In 2014. Angular has its base around components which are all modulized with both data and HTML. All pages can be generated with components in different size and levels (see Figure 4). Comparing to the traditional way of website development with HTML and Javascript, applying a framework, like Angular may bring convenience in data binding, code reuse and testing.

AngularJs and Angular (2 or later version), even though were both developed and maintained by Google, they are two completely different frameworks in the web development. The differences exist in many aspects:

- **Language Choices.**

In addition to TypeScript, one has an option of writing in languages like JavaScript, Dart, Elm, PureScript, etc. This gives the developer a variety of choice to choose the languages they are most comfortable with and leaving room into learning new languages.

- **Different structures**

Angular is completely component-based, which means lose components are created and merged into an angular application or webpage. Components are the building block of the applications, and with this, the application facilitates greater code reuse. AngularJS is Controller driven and with the a $scope the Controller is glowed to the views. Thus, becomes slightly slower of fact that different views are conducted whenever an action is carried out.

- **Mobile friendly**

Angular comes with native mobile support, making it possible correspondent to almost every mobile device, whereas Angularjs lacks mobile support, limiting it to only webpages through a desktop.

3. Research methodology

3.1 Research strategy – Case study

Case study involves inspecting an experience within its context. In the book of Oates, a case study is defined based on experimental current case analysis and focuses on the depth and not the breadth of the study, for a specific case. (Oates, 2006)

The purpose of our study is to create a pattern in order to help to separate the front-end and back-end from an MVC project into web APIs and a web application.

A case study is helpful when one wants to focus on one particular thing, such as an Information system or a development project, where a researcher wants to describe "what", "how", "who" or "why." There are different research strategies that can be applied in research. The most common research strategies are survey, design and creation, experiment, case study, action research, and ethnography. A case study was chosen in that we were looking to provide a very in-depth understanding of the case done at CGI. In a case study research, the focus lies upon a specific instance or the area to be researched. It is a research strategy that is well fitted when obtaining an in-depth knowledge of the research area. (Oates, 2006)

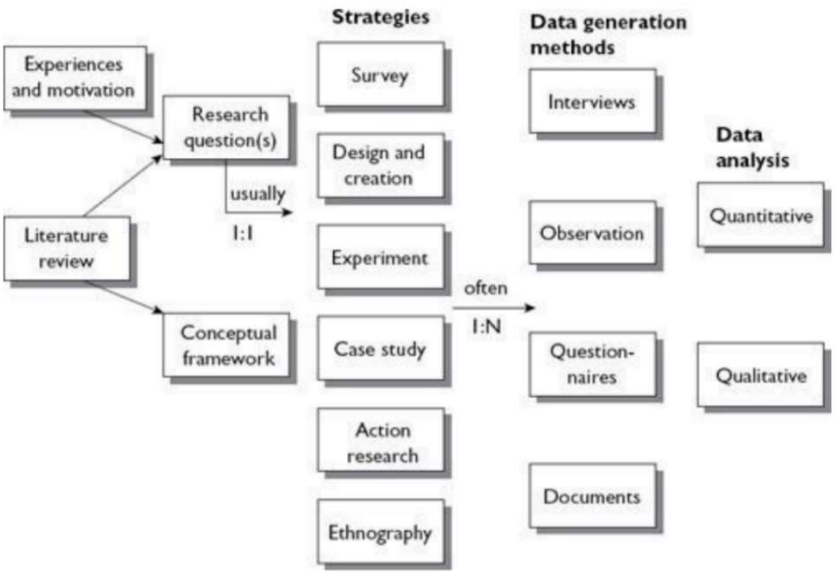

Figure 5 Model of the research process by Oates

We adopted the following processes from Oates research model to describe our research path.

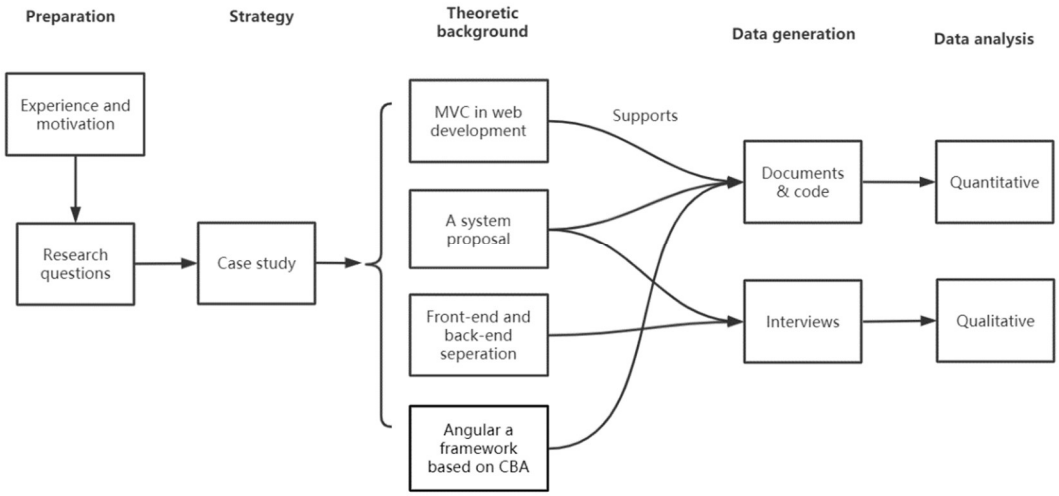

Figure 6 Adapted Research Process

The roadmap in figure 6 describes stages in our plan:

1. Defining motivation from experience from a certain case creating an awareness.
2. Following a suitable strategy to best describe our goal.
3. Motivating our case by finding relevant literature about:
    a. MVC in web development
    b. Web application development
    c. Front-end and back-end separation
    d. Angular and component-based architecture
4. Understanding:
    a. differences the two development architectures
    b. an insight from developers on the subject of separation
5. Determine
    a. the key element in migrating the software architecture
    b. how the task assignment can be improved after this change in architecture

3.2 Data Collection

3.2.1 Literature review

Literature is considered the published peer-reviewed sources on a narrow topic. As Oates stated (Researching information systems and computing, 2006), there are a variety of various sources to be used in literature studies. Literature review falls into two categories, in the first category, one reads literature to find a research case. In the second category, literature is selected to expand one's research.

The use of literature studies benefits the researcher in a form of gaining broader knowledge towards the research topic. Data collected regarding model view controller, front-end, back-end development, front-end and back-end separation along with component-based architecture is used to strengthen our research topic. This was done through internet search which according to (Oates, 2006) is a widely used source for gathering valuable information for one's research work.

We summarized a list of keywords which we mainly search for the literature with.

Table 1 Keywords for literature search

Keyword	**Purpose**
MVC + web development	Creating an understanding through web applications built based on the MVC pattern
Front-end and back-end separation	Studies regarding the implementation of Front-end and back-end separation in different projects
ASP.NET web	Findings about features of web applications built with APS.NET

With the keywords above, we searched in Summon and Google Scholar search engine in the finding of literature relevant to our research. In the determination of relevance, whether the results found through the search, we refined to look at the first page of the search results. We then looked at the headings of all results, considered the attractive titles and proceeded with the summary to determine if the literature is relevant.

3.2.2 Interview

An interview is seen as an act designed to obtain information to people through oral inquires. In the book of Oates, it is stated that in an interview, one person usually has a purpose for undertaking the interview. In our case, we conducted the interview to get piece of what developers have to say when it comes to the separation of front-end and back-end.

Interviews are categorized into three types, thus are structured, semi-structured and unstructured interviews. Each of these types carrying their own pros and cons with it. In our case, we designed our interviews according to the semi-structured type. A semi-structured interview is a popular method for qualitative research due to their way of being flexible. Semi-structured interviews let one change the wording or order of questions that one asks to make sure that you get the most detail from each participant.

A qualitative research methodology was conducted for this study, and this was to focus on constructivism along with quality experienced in the case. The interview took place via skype, and the interviewees were developers assigned to the ASP.NET project. In the selection of interviewees, the interviews conducted to the people with knowledge into the case. In addition, the questions were also sent by email to the interviewees in advance in order to give them enough time to answer our questions precisely and appropriately. By sending in questions, that gave the interviewees an insight on the topic and emerged into a convenient output.

To collect the information that we need with the interview, we designed our questions based on the following criteria:

1. The role of the developer in a project?
2. How are the tasks divided?
3. How do the developers in different roles communicate and cooperate?

The interview questions can be found in Appendix A.

In the selection of the interviewees, the interviews were conducted to the developers working within the case and in similar cases. Besides, the questions were sent by email to the interviewers beforehand in order to provide them with enough room to answer our questions precisely and appropriately.

3.2.3 Code analysis and document studies

Document studies as defined by Oates, are documents found or generated. Found documents are those that already exist whereas generated documents are put together on the cause of the research task. In our case, the existing code of the original was found, and the newly developed web application was one source of data. Document-based data is always much more comfortable to obtain than interviews or questionnaires (Oates, 2006).

With documents studies we proposed to create and find parts that can be divided in order to create a clean structure towards the separation of front- and back end separation.

In the process of analyzing code for the two different architectures, we count the source lines of code to determine the change of the software. We used the measurement method of logical Source Lines of Code (SLOC) which measures the number of executable "statements". A strict list of rules for C# logical counting (Vu , Sophia, Thomas, & Barry, 2007) were followed to identify the number of SLOC, see Appendix D. We also classified them into the following three categories based on certain definitions:

- **Adopted code:** the code that had been adopted with no changes or changes only in variable names.
- **Modified code:** the code that had been adopted with changes in function calls, return values and class of instances.
- **Removed code:** the code that no longer works or had to be removed to adapt to the new front-end.

According to Nikolaus Baer and Robert Zeidman (Baer & Zeidman, 2009), the CLOC's method brings precision and objectivity which helps to set a standard for quantitative source code measuring. In our case, we chose to calculate the SLOC without the common tools, like CodeDiff, so that they used so that we could generate more detailed data of code categorized by classes or functions.

Knowing about how many new LOC in an update can help developers to understand the change in size of a program in software development. We will use Changing LOC (CLOC) Growth (Zeidman, 2011), shown in Equation 1. TNL is the total new LOC which include lines that have changed, along with completely new lines.

Equation 1 $$CLOC\ Growth(n) = TNL(n)/TL(0)$$

It is also important to know the amount of code that have been kept or as the software evolves. To evaluate the efficiency of this migration, we will use the LOC Continuity value to describe how much of the original code was still presented during the change of software.

Equation 2 $$Line\ Continuity(n) = CL(n)) \ / \ TL(n)$$

We will also calculate LOC decay value, which represents the number of lines of code that does not come along while moving from the previous version (m) to a new version (n).

Equation 3 $$Line\ Decay\ (n) = 1 - \ (CL(n)) \ / \ TL(n))$$

4. Results and analysis

4.1 The models of the two different architectures

In this section, both angular and the MVC structure used in the ASP.Net project is introduced using the approach of graphs to illustrate how the elements of the architecture interact.

4.1.1 The model of the ASP.NET MVC project

To better understand the relationship between elements in the development of a web application, in this case, we created new models for both the original structure and the target structure in this migration by merging the models we had studied from previous literature.

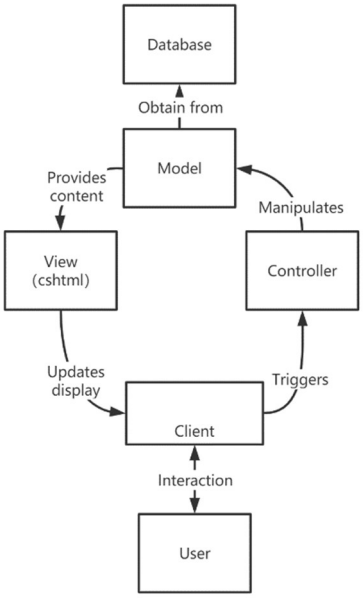

Figure 7 The structure of a fron-end and back-end coupled MVC project

In the figure above, we combined the model of the original MVC pattern (Figure 1), the model of the sample application (Figure 2), and the system proposal (Figure 3). The Views in cshtml are displayed on the client. They are continuously updated with the Views in the response the client receives. When the user operates the client by form submitting, data retrieving, and so on, the client usually makes requests to the Controller for an updated view. The Controller manipulates the model which handles the data in the database and generate a View as the response. In the original project, the main features were built in the architecture of integrated front-end and back-end.

4.1.2 The model for Angular 8 web application with web APIs

In the structure of a web application that uses Angular 8, a component-based web application framework, the communication between the client side and the server side is made by HTTP requests and responses with data rather than Views. When

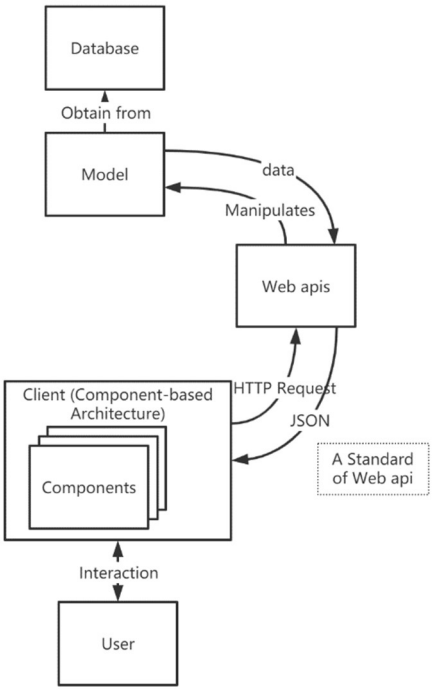

Figure 8 The structure of an Angular 8 web application with web APIs

a request for a set of data is received, the API will generate an HTTP response with corresponding data retrieved and formed by data Models. When a request for an update or delete operation in the database is received, the API can do the operation and return a response with a result information.

4.2 Architecture migration

Upon the analysis of the two different architectures, we can easily find out how the migration between the two models was done to minimize the work.

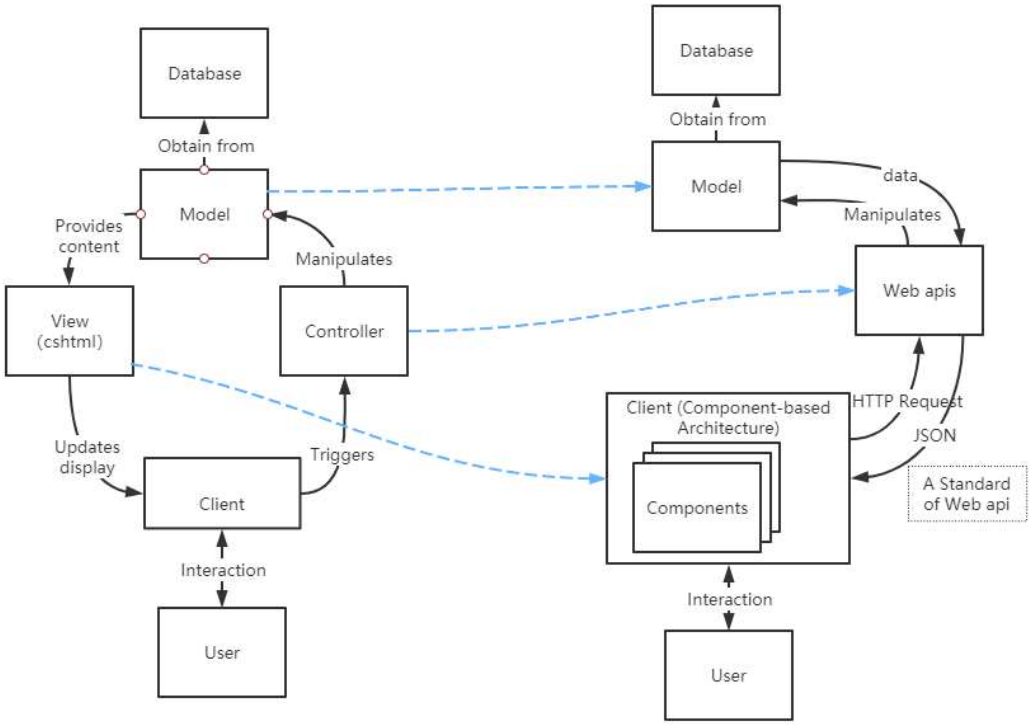

Figure 9 Migration between two architectures

Due to the need of retrieving data only by making requests with the front-end built in a new framework, the Controllers that provide Views with data to the client had to be converted into APIs providing the same information.

In the implementation of this migration, different types of elements of the original MVC structure were processed in different ways.

The Controllers were divided into View Controllers and Api Controllers. The original Api Controllers were remained to provide data when received corresponding requests. The View Controllers which handles redirection were removed, while those provides specific data were converted to Api Controllers with actions that return HTTP responses.

The Models were divided into View Models and Data Models. The data Models were remained to generate the type of data needed in the APIs. The View Models, which generate Views with certain data, were removed.

As the presentation layer of a web application, the Views were replaced with front-end with Angular 8. The new web framework can handle all the redirection within the client which reduce the pressure of the server side.

4.3 Code analysis

With the analysis of models in the migration, we had a clear map of how we could transform the original project into the development of a front-end and back-end separated project. During the implementation, we collected information from the code and did quantitative analysis on how to reuse the code in this migration. By evaluating how the previous code was resolved in the later development, we studied the amount of code that was adopted, modified, added and removed in the folder Controllers and Models.

4.3.1 The Models

The classes of Models in the ASP.NET MVC project can be divided into two categories. The first one is the View Model which forms a View with certain parameters. Since the Controllers that were converted into web API controllers will no longer return Views (discussed in 5.3.3 The Controllers), these Models can be removed. The other one is the data Model that are used to create instances of database entities. This sort of Model is still helpful in generating the response message in web APIs.

In terms of the models in the migration, we removed the View Models and kept all the data Models. In the current file structure, the folder Models contains all these models. The files named by *ViewModels.cs provide Models for the generation of Views and also some data Models, while the files named by *Models.cs provide data Models only.

Here is an example of the two kinds of Models which resides in one namespace:

```
public class RegisterViewModel
{
    [Required]
    [Display(Name = "UserRoles")]
    public List<SelectListItem> UserRoles { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.",
MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [System.ComponentModel.DataAnnotations.Compare("Password", ErrorMessage = "The
password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }
}


public class RegisterModel
{
```

```
    public string Email { get; set; }
    public string Password { get; set; }
    public string ConfirmPassword { get; set; }
    public string UserRoles { get; set; }
}
```

Based on this theory, we looked through all the files named by *ViewModels.cs and calculated the amount code in Model code reuse. The detail of each class of Models is shown in the table below.

Table 2 Code analysis of the ViewModels

Model name	***Name of the class***	***Adopted LOC***	***Removed LOC***
AccountViewModels	VerifyCodeViewModel		10
	ForgotViewModel		4
	LoginViewModel		13
	RegisterViewModel		17
	RegisterModel	5	
	ResetPasswordViewModel		15
	ForgotPasswordViewModel		5
ManageViewModels	IndexViewModel		6
	ManageLoginsViewModel		3
	SetPasswordViewModel		10
	ChangePasswordViewModel		14
	AddPhoneNumberViewModel		5
	VerifyPhoneNumberViewModel		8

4.3.2 The Views

The Views of an MVC web application are the presentations of the information which are displayed on the client-side.

The original Views in the MVC pattern of the ASP.NET MVC project can be either directly converted into other web frameworks or entirely redesigned depending on the business need.

In an upgrade or redesign of a web application, when needed to migrate to a completely different front-end framework, programmers usually need to rewrite most of it because of the different syntax and expressions. In our case, we need to migrate from the original cshtml with AngularJS to Angular 8 in Typescript, which are two completely different web programming languages. It is not cost-efficient to keep the original UI design when the code needed to be entirely rewritten. Since applying a new design is more reasonable, we think it's unnecessary to analyze the code reuse in the development of front-end, which is the View of the MVC structure.

4.3.3 The Controllers

Controllers in a web application based on MVC patterns are responsible for two things: handling requests and deciding which View should be returned and rendered in the front-end. In a component-based front-end development, there are web APIs to handle the requests. And there is no need to control the Views in the back-end because the web framework can always take care of this with its routing feature.

To make the best use of the original controllers, we can build the web APIs based on them to implement response generation to each kind of requests with the same business logic.

The MVC Controllers should be converted into web APIs which returns JSON data instead of Views. To achieve this, first of all, the class that the original Controller extends should be changed to System.Web.Http.ApiController so that this Controller can be recognized as a web API. And the Actions in original Controllers also needed to make adaptions to return the data that the front-end needs in JSON format.

Before

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
    var user = await _userManager.FindByNameAsync(model.Email);
    var result = await _signInManager.PasswordSignInAsync(
        model.Email, model.Password, model.RememberMe, shouldLockout: false);
    switch (result) {
        case SignInStatus.Success:
            return RedirectToLocal(returnUrl);
        case SignInStatus.LockedOut:
            return View("Lockout");
        case SignInStatus.RequiresVerification:
            return RedirectToAction(
                "SendCode", new {ReturnUrl= returnUrl, RememberMe= model.RememberMe });
        case SignInStatus.Failure:
        default:
            ModelState.AddModelError("", "Invalid login attempt.");
        return View(model);
   }
 }
```

After

```
[HttpPost]
[AllowAnonymous]
public async Task<HttpResponseMessage> Login([FromBody] RequestForm requestForm)
{
    var user = await _userManager.FindByNameAsync(requestForm._email);
    var result = await _signInManager.PasswordSignInAsync(
        requestForm._email, requestForm._pw, requestForm._rememberme
            || false, shouldLockout: false);
```

```
    switch (result) {
        case SignInStatus.Success:
            return Request.CreateResponse(
                HttpStatusCode.OK, new { Message = "Login succeed!" });
        case SignInStatus.LockedOut:
            return Request.CreateResponse(
                HttpStatusCode.BadRequest, new { Message = "Locked out." });
        case SignInStatus.RequiresVerification:
            return Request.CreateResponse(
                HttpStatusCode.Accepted, new { Message = "Need Verification." });
        case SignInStatus.Failure:
        default:
            return Request.CreateResponse(
                HttpStatusCode.NotFound, new { Message = "Invalid login attempt." });
       }
}
```

During the process of converting the MVC controllers to web API controllers, we found that the models called in the Actions of MVC controllers can be directly adopted for the web APIs. One main change that needs to be done here is that the return value should be an HTTP response rather than a View. Since the routing can be entirely handled in the client-side by Angular 8, the action method that only returns a View was removed.

By converting all the MVC Controllers, we got a result of the statistics of code reuse as it is shown below:

Table 3 Code analysis of the Controllers

Controller name	***Name of the function in the controller class***	***Adopted LOC***	***Modified LOC***	***Added LOC***	***Removed LOC***	***Description***
AccountController	Login (POST)	10	9		3	
	Login (GET)				4	Removed
	Register (POST)	4	7		1	
	Register (GET)				20	Removed
	RegisterMemberUser	7	2		1	
	VerifyCode (POST)	9	3		3	
	VerifyCode (GET)				5	Removed
	ConfirmEmail	4	2			
	ForgotPassword (POST)	9	3		2	
	ForgotPassword (GET)				3	Removed
	ForgotPasswordConfirmation				3	Removed
	ResetPassword (GET)				4	Removed
	ResetPassword (POST)	6	3		3	
	ResetPasswordConfirmation				3	Removed
	LogOff	3	1		1	

	Other sections of this class	12			9	
HomeController	Index				2	Removed
	Setting				2	
	Permissions				2	
	Other sections of this class				3	
ManageController	Index	15	1			
	AddPhoneNumber (GET)				2	Removed
	AddPhoneNumber (POST)	9	1		3	
	VerifyPhoneNumber (GET)	1	1			
	VerifyPhoneNumber (POST)	6	2		4	
	RemovePhoneNumber (POST)	9	2		1	
	ChangePassword (GET)				2	Removed
	ChangePassword (POST)	8	2		4	
	SetPassword (GET)				2	Removed
	SetPassword (POST)	8	3		2	
	Other sections of this class	13	1			
RoleController	Index	2	1		5	
	isAdminUser	10				
	Create (GET)				8	Removed
	Create (POST)	8	3			
	Other sections of this class	4	1			
	Total	**157**	**48**	**0**	**107**	

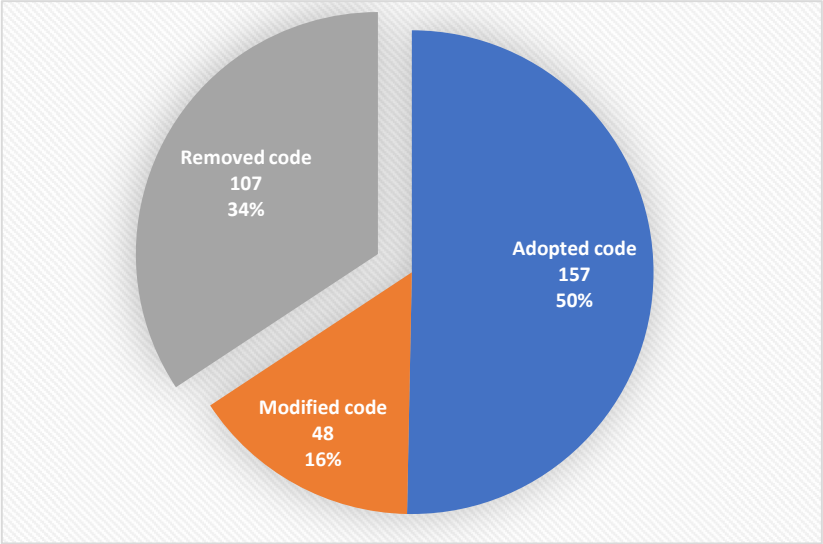

Figure 10 Code resuse in migration

As it is shown in the pie chart of code reuse, about 50% of the original code was directly applied, and 16% needed some change. The rest of the code which takes up 34% which handles redirection and routing were removed.

$$CLOC\ Growth(n) = 0.15$$

According to Zeidman's method, CLOC Growth calculates the amount of code kept while the software emerges. In the process of migration, 15% code is been retained.

$$Line\ Continuity(n) = 0.503$$

Line continuity calculates the LOC delay value that represents the number of LOC left while in the process of migration. An amount of 50,3% code was left out in the migration from the ASP.NET to the component-based architecture.

$$Line\ Decay\ (n) = 0.497$$

Line Dacey represents the number of lines deleted in the change. 49.7% of the code was not at any use so it was deleted in the migration.

4.4 Task distribution in front-end and back-end separation project

After our interview with the developer in CGI company, we collected information about how the developers collaborated in the previous project. The tasks are assigned according to the developers' skills. And in the previous ASP.NET project, our interviewee was responsible for the majority part of the project and usually reqires him to focus on multiple technologies in the implementation.

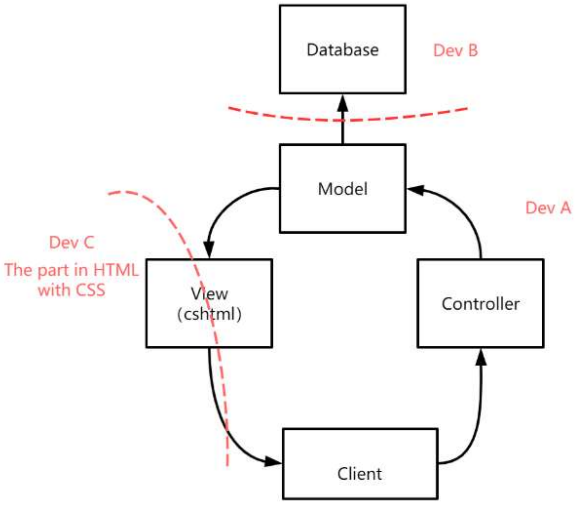

Figure 11 Tasks assignment in the development of the ASP.NET project

Based on the previous model we used to describe the structure of the MVC project, we interpret the task assignment as the interviewee had explained in Figure 10. There is a Developer B who is responsible for the database. Developer C helped with some of the implementation of Views in HTML. Developer A, who is the interviewee take the most part of the project. However, the task assignment can be different as the structure of the project had been changed.

Additionally, we had a second semi-structured interview with a developer from another IT company, Tencent Shenzen, with simmilar questions. As he described, most of their work is clearly divided for front-end developers and back-end developers in a program that owns millions of users. The back-end implement the services that store and provide data, and the front-end developers implement the presentation layer to display that data. In each updates of their product, the development and testing of front-end are done independently based on their API documentation.

From the view of an enterprise-level, an ideal efficient front-end and back-end separated development mode can be described as the conceptual model below:

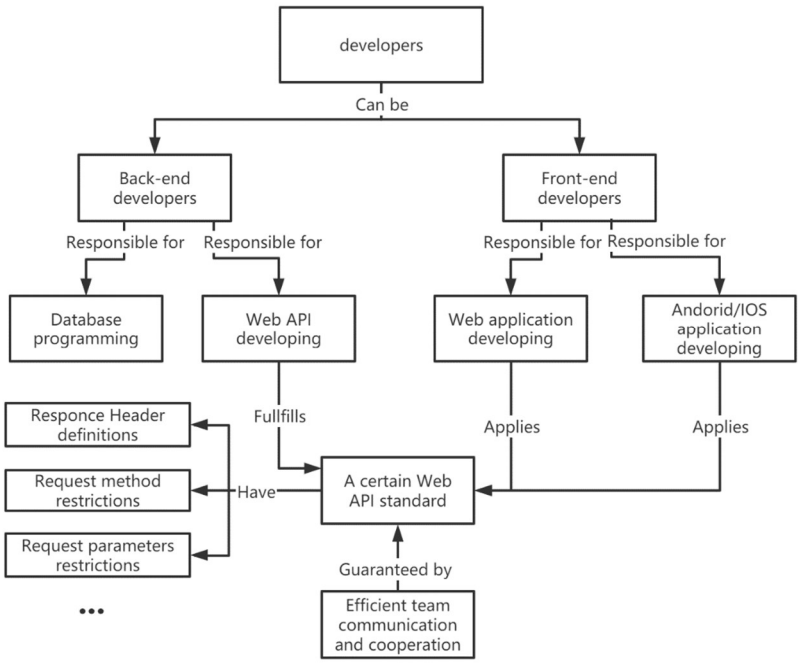

Figure 12 Task assignment in a front-end and back-end separated project

Developers can be divided into front-end and back-end developers who mainly work for in their own field. To reduce the cost of team communication and improve the efficiency of development, there is a need for the developers to have a web API standard that defines the request they need and the response they provide.

With a certain web API standard, the back-end and front-end developers may not need much direct communication. The back-end programmers develop APIs that provides the data needed in the front-end and generates the API documentation for the front-end developers to follow with. The front-end and back-end separated structure lead developers to more specific task assignment, where back-end developers implement data services and the front-end handles user interaction related work by using the data services.

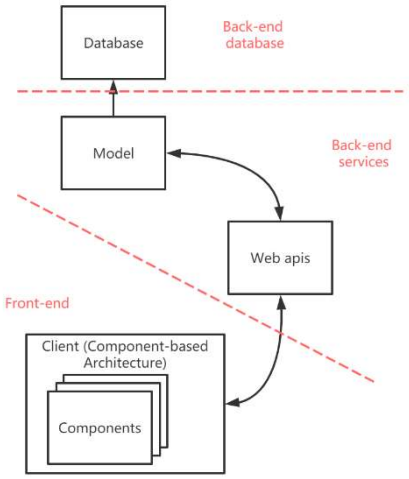

Figure 13 A proposed way of task assignment

In our case, the web application has simple features and only used by their own employees. However, a similar way of task assignment can be applied. Based on the diagram we created (Figure 8), we divided the implementation work into 3 parts, as it is shown in Figure 13: back-end database, back-end services, and front-end. The developer who is responsible for the back-end database builds the database. The developer who is responsible for the back-end services connects to the database and implement data services, the APIs, for the front-end. And the front-end only need to build the application to present the data retrieved by making HTTP requests. We suggest that each of the developers in the previous project is responsible for one of the three parts, so that the variety of technologies the developers need to focus on can be narrowed.

5. Discussion

5.1 Method reflection

The study is conducted using techniques gained from our three years at the university. Researching information systems and computing by Oates has played a large role in the importance of choosing the most suitable methods throughout our research. Upon the case study, it was chosen based on its criteria "experimental current case analysis and focuses on the depth and not the breadth of the study, for a specific case".

Document study was chosen as our main method because we have had two versions of code-based document that were intended to be compared to each other. This was done by a scientific found calculation method that identified differences in code and could be listed, as well as analyzed.

Interviews were chosen as a second method in the collection of data to get an insight on the developer's knowledge about the two different architectures. And we planned a semi-structured type of interview, but time caught both all developers that were involved, which lead to us getting less than we expected. We believe that this causes a setback and not to rely on one person's perspective of a subject. Moreover, we lastly consider maybe to have tried getting with other developers conducting the same kind of work to generalize our subject a bit more.

5.2 Future work

In this research, we did not concern about security issues. Even though the original authorization feature remained in web APIs, the heavy-client method makes the web application possibly in an unsafe situation (local cache and files can be modified). However, the security problem can be solved by applying more advanced encryption method and mechanism with the authentication cookies, which can be considered as a topic for further studies.

In terms of the API standard we had mentioned, there is quite much vast knowledge that we can take advantage of, such as RESTful API and GraphQl. This research topic can be extended to how these technologies help in the collaboration between front-end developers and back-end developers.

6. Conclusion

Based on literature findings, we believe that there are significant differences between the two architectures. The traditional MVC web application uses a thin-client approach which loads the front-end in the server-side and sends rendered web pages to the client. And the front-end and back-end separated project uses a thick-client approach, which requires the client to render the front-end pages and communicates with its back-end with HTTP requests. The need for this front-end and back-end separation comes up when the project does not have all data provided in APIs while a new front-end framework is required.

From the two interviews, it is noticeable that both of the two architectures are popular solutions in web application development. However, the front-end and back-end separation can be more flexible in applying different front-end frameworks or different clients.

In the implementation of the front-end and back-end separation, one main step is to build APIs to provide all the data needed in its front-end. To achieve that, we converted the View Controllers into Api Controllers, which return HTTP responses instead of rendered Views. As the statistics are shown in Table 3 and Figure 10, nearly half of the code can be adopted in this conversion. And 16% of the code in the View Controllers needed some modification, which involves mostly changes of return values. 34% of the code in these Controllers can be removed. As for the Models, the classes that were used to generate Views were no longer used in the new architecture, while the classes used to generate certain formats of data were kept (see Table 2).

The front-end and back-end separation in a web application gave the project manager a better idea of dividing the work in its implementation. The development work was suggested to be separated into three parts: the back-end database, the back-end services, and the front-end.

In our case, the migration is from an ASP.NET MVC web project with AngularJS to a project with Angular. Our solution applies to the rebuilds of any kind of front-end framework and even different platforms. The web API enables the ability for the back-end to communicate with all kinds of clients and also allows the company to expand its business.

References

Baer, N., & Zeidman, B. (2009). Measuring Software Evolution with Changing Lines of Code. *ISCA 24th International Conference on Computers and Their Applications.* New Orleans, Louisiana, USA: CATA 2009.

Dragos-Paul, P., & Adam, A. (2014). Designing an MVC Model for Rapid Web Application Development. *Procedia Engineering*, 1172-1179.

Edu, C. (2020, 05 01). *Thin client*. Retrieved from Forcepoint: https://www.forcepoint.com/cyber-edu/thin-client

Flanagan, D. (2006). *JavaScript - The Definitive Guide, 5th ed.* Sebastopol, CA: O'Reilly.

Infragistics, T. (2017, 05 30). *Angular Component Architecture*. Retrieved from Infragistics: https://www.infragistics.com/community/blogs/b/infragistics/posts/angular-component-architecture-made-easy

Is React Fast Enough. (2018, August 1). Retrieved from Medium: https://medium.com/@mustwin/is-react-fast-enough-bca6bef89a6

Kolu, A. (2012). MVC Frameworks in Web Development.

Krasner, G. E., & Pope , S. T. (1988). A cookbook for using the model–view controller user interface paradigm in Smalltalk-80. *The Journal of Object Technology*, 26–49.

Kun, L., Jinmin, J., Xiaohan, D., & Hui, S. (2017, April). Design and Development of Management Information System for Research Project Process Based on Front-End and Back-End Separation. pp. 338-342.

Leff, A., & T. Rayfield, J. (2001). Web-application development using the Model/View/Controller design pattern. pp. 118-127.

Oates, B. (2006). *Researching information systems and computing.* London: Sage.

Qi, Y. (2018, September). Front-End and Back-End Separation for Warehouse Management System. *International Conference on Intelligent Computation Technology and Automation*, 204-208.

Shilpi. (2019, September 27). *Divide or Join: The Frontend Backend Dilemma*. Retrieved from opensenselabs: https://opensenselabs.com/blog/articles/frontend-backend

Shiqi, G., Wenshan, H., & Hong , Z. (2018, July). Front-end and Back-end Separation. *React Based Framework for Networked Remote Control Laboratory.*, pp. 6314-6319.

Vu , N., Sophia, D.-R., Thomas, T., & Barry, B. (2007). A SLOC Counting Standard. *Cocomo ii forum*, 1-16.

WikipediaContributors. (2020, April 30). *Application programming interface*. Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=954100408

WikipediaContributors. (2020, April 24). *Model-view-controller*. Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=952768599

WikipediaContributors. (2020, May 1). *Single-page application*. Retrieved from Wikipedia : https://en.wikipedia.org/w/index.php?title=Single-page_application&oldid=954346925

Zeidman, B. (2011). *The Software IP Detective's Handbook: Measurement, Comparison, and Infringement Detection.* Cupertino, Califonia: Prectince Hall.

Appendix A

Interview questions

1. How long have you worked as a developer?
2. What computer language have you used for work?
3. How would you describe front-end and back-end development?
4. Which part of the project are you responsible for?

   Web design ☐ User experience ☐ Website security ☐

   Database ☐ Web APIs ☐ Testing ☐

5. Are you usually involved in multiple roles?
6. Do you usually need to focus on several technologies/computer languages to accomplish your tasks?
7. Cooperation and communication in a project:
   a) How do you divide the tasks when you start one new project? (ASP.NET MVC web project)
   b) What if there is a need to change a database model? How do you communicate to reach a consensus and accomplish the update in both back-end and front-end?
8. Do you face any specific difficulties in making changes or updates to the code according to a new business logic?
9. How do you think the separated structure of a project would help the cooperation between front-end and back-end developers?
10. How do you think the Component-Based Architecture would help in the task distribution in a web project comparing to an MVC thin-client web project?

Appendix B

Interview result 1

(A = Kennedy and Zifan, P1 = interviewee from CGI)

A: How long have you worked as a developer?

P1: Roughly a year

A: What computer language have you used for work?

P1: TypeScript, JavaScript, C#, PowerShell, HTML, CSS

A: Which part of the project are you responsible for?

P1: I am responsible for everything except business logic.

A: How do you make communications in collaboration, and how do you usually divide the tasks? Can you take the previous project as an example?

P1: Depends on experience and focus within the team. In the previous ASP.NET project, there was one developer responsible for the database, and one did some part of the HTML and CSS. I was responsible for the rest. It's not a small web project for the employees in this local branch, so it's quite enough to have only 2 or 3 developers working on this.

A: Do you usually need to focus on several technologies/computer languages to accomplish your tasks?

P1: There are always multiple technologies involved in my part of the work. Sometimes I need to navigate to the database for some modifications when I am implementing front-end.

A: What if there is a need to change a database model? How do you communicate to reach a consensus and accomplish the update in both back-end and front-end?

P1: if a database model needs to change, then it is changed. A change to the database model does not affect the data received by the front end if we do changes in the data models or APIs correspondingly.

A: Do you face any specific difficulties in making changes or updates to the code according to a new business logic?

P1: Depending on the type of change and its level of effort needed to be put in. Most of the changes and updates are relatively easy to make as long as the code involved is in my part of work.

Appendix C

Interview result 2

(A = Kennedy and Zifan, P1 = interviewee from Tencent)

A: How long have you worked as a developer?

P2: 1 year and 2 months

A: What computer language have you used for work?

P2: HTML JS CSS C++

A: Have you heard about front-end and back-end separation?

P2: The Front-end and back-end separation leads to a different way of rendering comparing to the MVC design pattern. The MVC applications use server-side rendering, SSR. And the front-end and back-end separated projects use client-side rendering, which moves the rendering task to the front-end and releases the pressure on the server. Based on the concept of the front-end and back-end separation, there came out more and more front-end frameworks.

A: Which part of the project are you responsible for?

P2: WebView programming, react web applications

A: Are you usually involved in multiple roles? Do you usually need to focus on several technologies/computer languages to accomplish your tasks?

P2: Usually not. As a front-end developer, the tasks I have is usually in the same area.

A: In terms of cooperation and communication in a project, how do you divide the tasks when you start one new project?

P2: There are many groups in a department which is responsible for one program. One group usually gets several tasks that are implemented in similar ways. The whole production is like a pipeline work.

A: What if there is a need to change a database model? How do you communicate to reach a consensus and accomplish the update in both back-end and front-end?

P2: All changes from new requirements will reflect in the API documentation. No change is needed in the front-end if the API provides the same data. Only big changes will require a change in API. And if there is a change in the API, we usually need to make sure the front-end is compatible with the old version.

A: How do you think the separated structure of a project would help the cooperation between front-end and back-end developers?

P2: Actually, in the company like Tencent, the amount of work for the front-end and back-end developers are nearly equal. The back-end only takes care of the data, the front-end deal with the GUI and render with the data requested by AJAX. Comparing to the integrated

Appendix D

SLOC Counting Rules for C#

Structure	Order of Precedence	Logical SLOC Rules
SELECTION STATEMENTS: **if, else if, else, "?" operator, try , catch, switch**	1	Count once per each occurrence. Nested statements are counted in the similar fashion.
ITERATION STATEMENTS: **For, while, do..while**	2	Count once per each occurrence. Initialization, condition and increment within the "for" construct are not counted. i.e. for (i= 0; i < 5; i++)… In addition, any optional expressions within the "for" construct are not counted either, e.g. for (i = 0, j = 5; i < 5, j > 0; i++, j--)… Braces {…} enclosed in iteration statements and semicolon that follows "while" in "do..while" structure are not counted.
JUMP STATEMENTS: **Return, break, goto, exit, continue, throw**	3	Count once per each occurrence. Labels used with "goto" statements are not counted.
EXPRESSION STATEMENTS: **Function call, assignment, empty statement**	4	Count once per each occurrence. Empty statements do not affect the logic of the program, and usually serve as placeholders or to consume CPU for timing purposes.
STATEMENTS IN GENERAL: **Statements ending by a semicolon**	5	Count once per each occurrence. Semicolons within "for" statement or as stated in the comment section for "do..while" statement are not counted.

BLOCK DELIMITERS, BRACES	6	Count once per pair of braces {..}, except where a closing brace is followed by a semicolon, i.e. };. Braces used with selection and iteration statements are not counted. Function definition is counted once since it is followed by a set of braces.
COMPILER DIRECTIVE	7	Count once per each occurrence.
DATA DECLARATION	8	Count once per each occurrence. Includes function prototypes, variable declarations, “typedef” statements. Keywords like "struct", "class" do not count.

Note. The Logical SLOC Counting Rules for C/C++, Java, and C# by Vu, et al. (A SLOC Counting Standard, 2007)