A Comparison of Encryption Algorithms for Protecting Data Passed Through a URL

Authors: Adam Johansson & Mohamed Osman
Supervisor: Anders Haggren
Semester: VT 2017
Subject: Computer Science
Abstract

This project starts off with giving an overview of what sensitive data is, encryption algorithms and other required knowledge for this thesis project. This is because of the aim of this thesis project, that is to find the best way to encrypt data passed through a URL with a focus on protecting sensitive data in web applications. Data sent through the URL of a web application could be sensitive data, and exposure of sensitive data can be devastating for governments, companies and individuals. The tools and methods that are used for this thesis project are described. An overview is given of the requirements of the web application that was to be created, the development of it, implementation and comparison of encryption algorithms. After that the results of the encryption algorithms are compared and displayed together with a prototype of the web application and its encryption. The results are then analysed in two different sections: security of the encryptions and performance tests. With the results given we conclude which one of the encryption algorithms is the most suitable for our web application, and otherwise when encrypting data through the URL of a web application. The results shows that AES have a great advantage over 3DES both in security and performance when encrypting sensitive data passed through a URL. Those results are then used to build a secure web application to help and assist a broker during an open showing. The web application is then used to gather information from interested buyers so it can be easy for the broker to contact them after the showing.
Preface

The practical part of this thesis project was performed at the company Vitec Software in Kalmar, Sweden. Vitec wanted a web application that could be used for their broker customers on real estate showings. This project has given us a significant amount of new knowledge in programming and development, as well as some insight in how it is to work as a developer.

We would like to thank Ulf Johansson, Stephan Engström and Andreas Johansson from Vitec for helping us and making this thesis project possible. We would also like to thank Linnaeus University and the program of Computer Engineering for giving us this opportunity.
Contents

1 Introduction 6
  1.1 Background 7
  1.2 Purpose and Objectives 8
  1.3 Scope/Limitations 8

2 Theory 9
  2.1 Sensitive Data in Web Applications 9
    2.1.1 What is Sensitive Data 9
    2.1.2 Sensitive Data Exposure in Web Applications 9
  2.2 Web Application Security 10
    2.2.1 Encryption Overview 10
    2.2.2 DES: Data Encryption Standard 11
    2.2.3 Triple DES 11
    2.2.4 Double DES? 12
    2.2.5 AES: Advanced Encryption System 12
  2.3 Required Knowledge 12
    2.3.1 ASP.NET 12
    2.3.2 .NET Core 13
    2.3.3 MVC Pattern 13
    2.3.4 ASP.NET Core MVC 14
    2.3.5 JavaScript 14
    2.3.6 Typescript 15
    2.3.7 Knockout 15
    2.3.8 jQuery 16
    2.3.9 JSON 17
    2.3.10 REST API 17
    2.3.11 Hypertext Transfer Protocol 17
    2.3.12 URL and Query String 18
    2.3.13 AJAX 19

3 Method 21
  3.1 Literature Study 21
  3.2 Programming Languages 21
  3.3 Application Programming Interfaces 21
  3.4 Integrated Development Environment 22
  3.5 Web Browsers 22
  3.6 Other Software 22

4 Implementation 23
  4.1 Application Requirements 23
  4.2 Development 23
4.2.1 Design

4.3 Server Side Development

4.3.1 Query String Data

4.3.1.1 Query String Validation

4.3.1.2 Data Exchange

4.3.2 Form Data

4.3.3 Server to REST Communication

4.4 Protection of Data

4.4.1 Encryption Algorithm Class

4.4.2 Encryption Implementation

4.4.3 Decryption Implementation

4.4.4 Comparison of Encryption Algorithms

5 Result

5.1 Web Application

5.2 Comparison of Encryption Algorithms

5.2.1 Web Based Comparison

5.3 URL Encryption

6 Analysis

6.1 Security of Algorithms

6.2 Performance of Algorithms

7 Discussion and Conclusion

8 References
1 Introduction

People, companies, organizations and governments all have information that they consider to be sensitive. Governments have information that they consider private, for example militarily and diplomatic affairs. Companies have data that they want to keep private; product plans, key customers and new inventions to help them gain and keep competitive advantage over rival companies. Other organizations like schools and hospitals need to protect data of students and patients or donors. Protecting sensitive data from attacks and unwanted disclosure is essential for the privacy

Before the time of computers, data was usually stored in paper records, and sensitive data records were kept secure by locks and guards. This made it hard to steal or expose any form of data. Today, most of the data is stored on computers because of the storage advantage that computers offer. Since computers do not have locks, various of techniques has been developed to ensure the security of data. Nevertheless, different types of data-focused attacks still occur.

According to a report from Risk Based Security [1], the amount of disclosed sensitive information records has increased immensely in the last few years, from 822 million in 2013 to 4.2 billion in 2016. This is because of the internet becoming a more integral part to the economic and international life of governments, organizations, companies and individuals. They are all targets for ever-more frequent serious attacks. Web applications is one of the platforms that gets attacked frequently by various types of attacks, according to Web Application Security Statistics Report from whiteHat Security [2], 40% of internet breaches in 2015 were attacks on web applications. Many of today’s web application are not secure enough to protect their sensitive data, and this is both for the client-side and the server-side [3]. The sensitive information sent across the internet should be handled in a way that does not comprise the confidentiality and ensures that the data should only viewed by authorized objects.

The aim of this thesis project is to research paths of protecting sensitive data in web applications, with a focus on data that is sent through the URL of an application, as well as different ways of encrypting that data.
1.1 Background

Vitec Software have developed a web based broker system for various Norwegian brokerage agencies to help them with real estate selling. To be able to sell real estates, potential buyers must see the property before buying it. Brokers offer open viewings for potential owners to see the house and then latter decide if they want to buy or bid on it, or not. On these open viewings 20-30 people can appear and it might be hard for the real estate agent to keep track of everyone, therefore it would be easy for the broker if the information about the participants was saved online so that he/she could follow up on them via mail or telephone after the viewing and see if they are interested.

Vitec Software wants to provide a cross platform web application to their customers, the brokers, so that they can use it to gather information from the participants who are attending the viewings. The application should store name, phone number, current address and email address. This information can be gathered in two different ways:

- The broker sends an HTTP link of the web application to interested participants who want to attend the viewing so that they can pre-register themselves.
- People attending the viewing without pre-registration should be able to register themselves with the help of an iPad located in the building.

The link which is sent by the broker to the interested participants contains information that identifies the viewing. Some of the information is considered sensitive both by the brokerage agencies and Vitec Software, and this information should not be exposed to none other than authorized parties. Another security issue regarding registration onsite is that a participant should not be able to see what information another participant has entered prior to him/her by pressing the back button on the web application.
1.2 Purpose and Objectives

The purpose of this project is to investigate the possibility to transfer sensitive data in a URL. This is done by examining a few data encryption algorithms and other security measurements to later determine which of the investigated measurements are appropriate to protect the sensitive data that is sent through the URL. Objectives are to research which of the encryption methods works best for our purpose. This will be done by research in doing different tests comparing speed, security and response time when communicating with web servers.

1.3 Scope/Limitations

In the theoretical part of this thesis we set up to compare different data encryption methods and ways of protecting sensitive data. However, we do not cover every possible way of protecting sensitive data, as well as we do not cover every data encryption method that could potentially be used. We only cover a selection of the most popular data encryption methods and the ones that we thought were the most appropriate for the practical part of this thesis.
2 Theory

This chapter will give an overview of what sensitive data is, encryption algorithms and other required knowledge for this thesis project.

2.1 Sensitive Data in Web Applications

According to Open Web Application Security Project (OWSAP), many of today’s web applications have bad protection of their sensitive data. This section will give a brief overview of what sensitive data is, risk with sensitive data exposure and an overview of the life cycle of network data.

2.1.1 What is Sensitive Data

Sensitive data is information that must be protected from unauthorized accesses to ensure the privacy and integrity of an individual, company or organization. Sensitive data covers a wide range of information and can include information protected by civil rights: personal life, political opinion, healthcare details, ethnic or racial origin, religious or other beliefs and criminal or civil offences.

There is also sensitive information which is specific to you as a client, employee, student or patient. It could be information that identifies you as a person such as parent names, birth date, contact information, social security number etc. Businesses have information that they consider sensitive and it can include anything that poses a threat to the company in question if exposed to a rival company or the public. Business sensitive information can include trade secrets, business plans and strategies, financial data, key customers and suppliers amongst other things.

2.1.2 Sensitive Data Exposure in Web Applications

Number six on the 2017 OWASP top 10 [3] list of risks is sensitive data exposure. This is because of the bad protection of sensitive information in many web applications or APIs such as financial, credit card information, healthcare and other data that is considered sensitive by the subject. If the
data is not well protected attackers may steal or modify it, then use the exposed data to commit other crimes such as credit card fraud, identity theft or other type of crimes. Therefore, sensitive data requires extra protection like encryption or other special precautions when data is exchanged between client and server. Before protecting sensitive data, it is important to know three main things: what data is considered sensitive, where the data is located and who has the permission to access it [4].

**Which data is considered sensitive?** - Before protecting the sensitive data, it is important to know which of the data is considered to be sensitive. A company may have terabytes of data, but not all data may be equally sensitive. By identifying which data is sensitive it allows security professionals to secure it more properly.

**Where is the sensitive data located?** - It is typical that data is stored in a file server, cloud or back up disk. But what about data that is in transit? The data exchanged by server and client? Sensitive data in transit must also be protected by encryption to prevent man-in-middle-attacks or eavesdropping.

**Who has the permission to accesses it?** - Accessing sensitive information should be limited only to the authorized subjects who need to access it. This will make sure that attacks are prevented or data leaks from within the organization.

### 2.2 Web Application Security

Protecting sensitive data in web applications is especially important when data is in transit and it is vulnerable to attacks, such as man-in-middle-attack and other similar attacks. This section will discuss different encryption tools to ensure security when data is in motion between a client and a sever.

#### 2.2.1 Encryption Overview

One way to protect sensitive data from unwanted disclosure is to use encryption. Encryption is a mechanism to convert from a clear text format to a sophisticated and diffused text format that cannot be understood by anyone except authorized parties [5]. Over the past decade cryptologists have
invented different encryptions algorithms to ensure security in data or other text formats. Encryption methods are key based algorithms where the text is encrypted and decrypted by using a secret key. Algorithms that use same secret key for encryption and decryption are called symmetric-key encryptions and the methods that use different keys are called asymmetric-key encryptions. Two popular symmetric-key encryptions are Data Encryption Standard (DES) and Advanced Encryption System (AES). These encryption methods are described below.

2.2.2 DES: Data Encryption Standard

DES was developed for the U.S. government and intended to be used by the public [6]. The algorithm is complex, but it is repetitive which makes it very suitable for implementation on single-purpose chips. Many software and hardware systems have DES as an implemented encryption system. The algorithm was developed in the 1970’s by IBM. It is a combination of transposition and substitution algorithms, but it performs those algorithms repeatedly on top of each other. The algorithms are applied in a total of 16 cycles. It begins by encrypting blocks of the plaintext of 64 bits each with a key of 64 bits. The keys actual value is any 56-bit number and the other 8 bits are used for check digits that does not interfere with the encryption or the implementation of it.

2.2.3 Triple DES

Triple DES (3DES) uses three keys to improve the strength of the encryption. The keys have the same size as with regular DES (64-bit size with a strength of 56 bits). The application of 3DES is that you first encrypt with the first key, then with the second on top of that, and finally with the third. This gives the algorithm a strength of a 112-bit key [6]. 3DES is therefore significantly stronger than regular DES, but it requires more computation and more keys. There is a different version of 3DES that uses only two keys. This version uses the principle of encrypt, decrypt, encrypt. This means that it uses the first key to encrypt, then decrypts on top of that, and then the second key for encryption on top of that as well. This gives a strength of 80 bits, which is still stronger than DES, but weaker than the previous 3DES.
2.2.4 Double DES?

Double DES uses two keys with the same size and strength of DES. It encrypts the plaintext with the first key, and then the second one on top of that. This however only increases the theoretical strength of the key with 1, resulting in a 57-bit key value. Because of this, this encryption method is not as popular as DES or 3DES, since an increased strength of 1 is not worth the increased complexity that the algorithm offers.

2.2.5 AES: Advanced Encryption System

AES, also known as Randle algorithm, is a fast and strong encryption algorithm that can be implemented on simple processors. It has a robust mathematical foundation and it uses multiple elements such as substitution, transposition, the bitwise function “shift exclusive OR” and other additional mechanisms to achieve a strong encryption algorithm. AES uses repeat rounds of operation; 10, 12 or 14 rounds for the keys of sizes 128, 192 or 256 bits. Each round contains four steps that substitutes and scrambles bits. To add diffusion bits from the key they are frequently combined with bit results from the substation and transposition. Unlike DES rounds, which is limited to only 16 rounds, the rounds of AES can be increased to enhance the security of the encryption. If security professionals think that 10, 12 or 14 rounds are too few, the only change they have to do to is adjust the algorithm in the repeat loop and change the limit of rounds that it takes [6].

2.3 Required Knowledge

This section will discuss the technologies that were used to develop the web application.

2.3.1 ASP.NET

ASP.NET is short for “Active Server Pages” and it is based on the Microsoft .NET Framework. It is a server side programming environment for creating and deploying dynamic web applications. This environment is created by Microsoft and was first released in 2002 with the release of the .NET framework. ASP.NET has been through some changes, and different versions
were released making it easy for the developer to develop dynamic and interactive web applications using C#, Html, CSS and JavaScript [7].

2.3.2 .NET Core

In December 2016 Microsoft released the latest version of ASP called ASP.NET Core, which is an open source and cross-platform framework. This means that developers can build and run applications on many platforms such as macOS, Linux and windows, as opposed to earlier versions which was only supported in Windows. This new framework helps developers to create modern, cloud based applications which are connected to the internet. Examples of applications are web applications and IoT applications. The applications built with ASP.NET Core can run both on the new cross-platform .NET Core or the full .NET Framework [8].

2.3.3 MVC Pattern

MVC is short for “Model-View-Controller” and it is a powerful, important design pattern in computer since which has been used for many years. The main idea of MVC is to apply the well-known design principle Separation of Concerns by separating concerns inside an application, such us separating the user interface logic from the data access logic. This principle fits perfectly in web applications because it facilitates the separation between the client side and the server side programming. Since its introduction, it has been used in many frameworks, and especially in .NET frameworks. Model, View and Controller are explained separately below:

- **Model** is a set of classes that represents the domain of the application. These classes are domain objects which often encapsulates data saved in the database, the code that validates the data, manipulates the data and enforces domain-specific business rules and logic to it.
- **View** defines how the application user interface should be displayed and the presentation of the web application within the web browser. The View often contains HTML, CSS and JavaScript files.
- **Controller** is a class that manages the association between the View and the Model. When the user enters a request URL in the web browser it is the Controller class who responds to the user by interacting with the Model and rendering the appropriate View.
2.3.4 ASP.NET Core MVC

ASP.NET Core MVC provides the same functionality as the previous ASP.NET MVC version, but it is now built on the .NET Core framework. The ASP.NET MVC is a collection of independent components that have well defined properties, and it is possible to replace those key components with your own components. This new framework works well with the HTTP protocol, giving the developer control over the requests that are passing between the client and the server.

The structure of the framework is that both the Model and the Controller are C# classes. The Controller class contains a method called “action” which is associated with a URL. When the user on the client side sends a URL request to the server it is the action method that is executed to perform an operation on the data model and then to present a View for the user. Figure 2.1 illustrates the client and the server interaction with each other using the Model-View-Controller pattern [9].

![Diagram](image)

*Figure 2.1: The Model, View and Controller’s interaction with each other.*

2.3.5 JavaScript

JavaScript is “the language of the web browser” [10]. It was created in 10 days by Brendan Eich in 1995 [11]. The deadline for the creation of this language was harsh because of the high pace in web innovation, with Microsoft working on making the internet the core feature of the new Windows 95 operating system. JavaScript was created for the company Netscape and it was to be used in their web browser.

JavaScript is not to be confused with the programming language Java. Java is
a heavyweight language that is able to support portable application, whilst JavaScript is mainly for lightweight web browser applications. JavaScript is the only language that is found in all browsers, and one of the reasons for that is due to that it is lightweight and expressive. The language might have many flaws, but being informed about its limitations it can become a powerful tool if used correctly. Some of the flaws in JavaScript lies in the user having to work with a model that is based on global variables.

2.3.6 Typescript

TypeScript is a programming language developed by Microsoft and is a superset of the programming language JavaScript. A superset of a programming language means that it is an extension of the original, keeping the original parts of the language whilst including new aspects. For TypeScript those aspects are extra functionalities like static typing and a bigger focus on object-orientation, such as classes, fields and different types in variables. TypeScript has taken much influence from other programming languages like C# and Java, which both are object-oriented also object-oriented.

Every TypeScript program is also a JavaScript program. The code that is written in TypeScript is translated to JavaScript in a separate file that is used for compilation. Because of this TypeScript is able to run with only JavaScript code, but that disregards its initial purpose. The main purpose is to help the developer in making it easier to write large and wholesome projects.

2.3.7 Knockout

Knockout is a standalone framework for JavaScript that uses the Model-View-ViewModel pattern. Knockout binds together the HTML (View) and the JavaScript with features like data bind. What data bind does, is that it connects content in the HTML file to be dynamically adjusted in the JavaScript file. Text and values from the HTML can be data bound so that some text element or some value of a HTML element is decided, fetched or changed by the JavaScript program. The Model keeps the data and other necessities to be able to be used in the View, which displays the data. The ViewModel is then there to observe the data changes in both the View and the Model.
**Observables** is one of the core features that Knockout is built around. You can declare a ViewModel as an observable, or parts of it as observables. This way the View updates automatically when/if the ViewModel changes. To do this there also needs to be a declaration (*data-bind*) in the View on that location where the value is supposed to be updated. Observables act like JavaScript “getters” and “setters”, where you can easily get the value of an Observable, or set it to a new value. But nevertheless they are still functions, and this is because JavaScript “getters” and “setters” are not supported in all web browsers. Since the Observables are essentially functions, you can call the function with no parameters to get the variable value, or do it with parameters to set the variable to a new value.

**Validation** is a Knockout plugin for Model and property validation. This plugin can be used to add restrictions to input-fields, such as max-length, required, email and so on. These properties are set to Observable variables in the ViewModel and act as validation for the client. A variable can have more than one validation property to it, for example if the field is *required* it can also have *min-length 2.*

### 2.3.8 jQuery

jQuery is a JavaScript library that helps a developer in having to write less code for certain JavaScript functionalities. Some examples of functions that can be done with less code are accessing elements in the Document Object Model (DOM) and adjusting aspects of the HTML document. You can modify the CSS and styling of the page, as well as using AJAX to make asynchronous calls via JavaScript and XML [12].

Despite being able to essentially write applications with lesser code for certain functions with jQuery, it might not always be worth importing an external library merely for those purposes.
2.3.9 JSON

JSON is a lightweight integrated part of JavaScript that provides a format for data exchanging. It is created to be easy for humans to read and write, as well as for computers to parse and generate [13]. A JSON object can take the following values: string, number, object, array, true, false and null.

2.3.10 REST API

Representational state transfer, or RESTful web services is an Application Programming Interface (API). REST API is used in the client side of a web application to communicate with web services [14]. REST creates responses in XML, JSON, HTML or similar formats.

2.3.11 Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) protocol is frequently used in the internet, and it is described as the foundation of the web [15]. HTTP is the main transfer protocol that is used by a browser to interact with a web server. The browser requests data and the web server responds with the required data. HTTP lies in the application-level of the TCP/IP stack, using the TCP for transmitting data. HTTP allows the browser and server to use the request-response paradigm, where the client sends an HTTP request and the HTTP server sends a response to the request. The HTTP request-response paradigm uses pre-specified methods GET and POST. These two methods are frequently used and are described further in Table 2.1.

<table>
<thead>
<tr>
<th>Requests</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>The simplest of the request methods is the Get method, it requests a data item from the server and the webserver responds with status information and the item requested.</td>
</tr>
<tr>
<td>POST</td>
<td>The POST method is used to send a data item to the server and add the item to the server’s database.</td>
</tr>
</tbody>
</table>

*Table 2.1: Description of the GET and POST HTTP requests.*
When a web browser request data from the web server it sends a GET request over the transport layer and the server then responds by sending back an HTTP header, a blank line and the requested data. For example, a GET request is created as follows:

```
GET /item version
```

**GET** is the method used, **/item** is the requested data and **version** describes the version of the protocol used. HTTP/1.1 is the current version of HTTP. The web server responds with a header information and the requested data. The response header is received as:

```
HTTP/1.1 status code status text
Server: server-identification
Last-Modified: date document last changed
Content-Length: data-size
Content-Type: type of the data
```

The response contains an HTTP version and a **status code** that tells the web browser if the request was handled correctly in the server. For example, if there was no problem with the request and the server handled it correctly the server sends back a status code of “200 ok”, which means that no problems occurred. Otherwise, if a problem did occur, the sever sends back a status code depending on the error. The server might send back a “404” status code that specifies that the requested file cannot be found. The **Server-identification** file specifies the server domain name, **Content-Length** specifies the size (bytes) of the data transmitted and **Content-Type** is the type of the data, for example if the data transmitted is an image the type field is **image/jpeg** [16].

### 2.3.12 URL and Query String

Unified Resource Locator, known as URL, contains information that helps a browser locate a web page on the internet. The browser uses different characters to separate the URL into four different components. The components include a protocol (HTTP), the computer name, a document on
the server and its parameters. The protocol and computer name are used to form a connection to the server to retrieve the webpage that resides on the server, document name and parameters are used to request a specific page from the web server [16].

Additionally, to the four main URL components, a user sends pre-specified parameters called query strings back to the server to recover specific information from the server. The query strings follow a specific character paradigm, where a question mark (?) is a separating character and the data requested appears after this separating symbol. For example, consider the following URL:


HTTP is the protocol used, books.com is the endpoint name, books is a specific webpage followed by a question mark (?), which is then followed by parameters that specify a book with the name harry potter and author jk rowling. Those parameters are called key-value parameters, where bookname is a key and harry potter is its value. The parameters are separated by the “&” symbol which specifies that they are different keys/values. This is then sent to the server as a request of a book with the book name harry potter and author jk rowling. The server responds with a list of harry potter books with the author jk rowling [17].

2.3.13 AJAX

AJAX stands for Asynchronous JavaScript and XML. It is a set of techniques to make asynchronous calls from the client side of a web application to the server side. Web applications that otherwise does not use AJAX use synchronous programming where every action is run after the other. With AJAX, actions can happen in parallel, like when communicating between the client and server the user does not have to wait for a response from the server before being able to use the application again.
Figure 2.3: To the left is a classic web application model and to the right is an AJAX application model.

AJAX depends on the use of XMLHttpRequest. Connections, sending and receiving requests are handled using the XMLHttpRequest object. XMLHttpRequests are used to retrieve data from a URL without the need to do a full refresh of the web page [18].

Even though the X in AJAX stands for XML, JSON is used more than XML because of the many advantages JSON has over XML, like being lighter and an integrated part of JavaScript [19].
3 Method

In this chapter the tools and methods used in this thesis project will be described.

3.1 Literature Study

A part of our research have been in differentiating different data encryption algorithms and various security measurements that is used to protect sensitive data. Information about this subject have been gathered from books and reports that were relevant for our research.

3.2 Programming Languages

The web application was created using different programming languages, mainly C# and TypeScript. TypeScript was new to us and had to be researched upon on how to use it to its full extent. We had previous experience with JavaScript, but as mentioned earlier, TypeScript is in direct contact with JavaScript and any JavaScript code will work with TypeScript. But since TypeScript was to be used we needed to take advantages of the content given to us with TypeScript. TypeScript was used for the front-end of the application to create a dynamic environment connected to the HTML and CSS. C# and .NET was used in the back-end of development.

3.3 Application Programming Interfaces

Representational state transfer (REST) or RESTful web services were used as our Application Programming Interface (API). REST allowed us to send and receive data in a JSON format via HTTP/HTTPS. Commands like POST and GET were used to send and receive the information. This was for example useful when sending information that the back-end had access to, to the front-end for display.
3.4 Integrated Development Environment

Visual Studio Code was used to perform the programming of our application. Since MacBook’s were used for our work it seemed like the most appropriate Integrated Development Environment (IDE) since C# and .NET was to be worked with, which is not otherwise supported on MacBook’s operating system Mac OS X.

3.5 Web Browsers

Many different web browsers were used for testing purposes when creating the web application, for example Google Chrome, Mozilla Firefox and Safari. When performing data encryption tests on how fast the algorithms were through browsers, Mozilla Firefox was used.

3.6 Other Software

Git and GitHub were used for handling version-history. We have the most experience with this and found that it would be the best way to work, since there would be many different files and operations in both the front and the back-end of the application.
4 Implementation

This chapter will describe how the web application was created, it gives an overview of the requirements, development and implementation, as well as the comparison of encryption algorithms.

4.1 Application Requirements

The requirements of the application were that participants of a real estate viewing should be able to register themselves via the application by entering some of their personal information. The application should have knowledge about customer number, address of the viewing, time of the viewing, and who the responsible broker is. The application should be accessible either from email for pre-registration, or on an iPad if the viewing is public. Any personal information should be secured so that in the case of a public viewing, participants should not be able to somehow access other participants’ information. It should also not be possible for participants to change any information about the viewing. When participants register themselves, their information should be sent to Vitec’s own servers according to their already existing REST API.

Other requirements for the applications is that it should support the languages Swedish and Norwegian. The application should also adjust itself to if it is displayed on a private or public computer, in that the form should autocomplete or not, and if when a participant is registered there should be an option to register a new participant or not.

4.2 Development

Since the main purpose of the application is to assist a broker to collect participant information from the attenders of the house viewing, the development of this application is divided into two parts. The first part contains of extracting and validating the information that is passed through the url, this information is used to identify a specific viewing. The second part includes getting participant data from the website application form and forwarding it to the REST API where it gets sent to the company database. The data used in this section is test data that is used only to demonstrate how the application is implemented. This section will describe the design of the
application and how those two parts were developed using ASP.NET Core and other client side programing like JavaScript, Typescript and knockout.

4.2.1 Design

The design started with a goal of creating a form with appropriate inputs for the application. An HTML was created in combination with a CSS to get a base of development. The application was then equipped with an input field via the `<form>` tag in the HTML and inputs of phone number, first name and last name were added. At this point the application was only a form with inputs, checkboxes and a submit button. Some basic CSS styling was added to the application to make it look better.

Later on a design proposal from Vitec was received that shows how the form is supposed to be displayed, together with information on the page, styling and what is supposed to happen when the submit button is pressed. Figure 4.1 shows an overview of what the form HTML code looks like. There is more to the HTML code that is added for styling purposes and JavaScript bindings that are not displayed. Figure 4.2 shows how most of the styling is applied in the CSS. The design proposal can be seen in Figure 4.3 and Figure 4.4.

![Figure 4.1: HTML overview of the form and its different inputs.](image-url)
Figure 4.2: CSS styling.

Figure 4.3: Design proposal of the default form view. English version with a customer logo.
4.3 Server Side Development

The web application server is built using the latest .NET framework which is ASP.NET Core MVC. The structure of the application is divided into three main parts: Controllers which handles the requests from the browser, Models for creating data objects and Views for all the client side scripting. This section will describe how these parts were used to get information from the URL and how the registration information was handled.

4.3.1 Query String Data

In the application, a Controller class was created to handle all request specified with a Controller class name. The main Controller class was named RegisterController because the main purpose of the application is to let interested participants register for the viewing.

The RegisterController class contains methods that handle HTTP requests received from the browser. The return-type of the method is an ActionResult type, and this means that the method receives a request and response with an action. For example, returning a user interface page to the browser. Figure 4.5 shows how the ActionResult method looks like.
Figure 4.5: The method ActionResult.

If the user enters the URL in Figure 4.6, the previous method in Figure 4.5 gets called and responds with a method from the View class that renders the web page for the end user.

Figure 4.6: Example URL that fetches method from the View class.

4.3.1.1 Query String Validation

The query string validation method Index was modified so that it could receive a URL with query strings that had information about the real estate viewing. Figure 4.7 shows the URL with a query string entered in the browser and the modified method is shown below in Figure 4.8, together with additional methods that validates the received query string.

Figure 4.7: Query string example displayed in the URL.
public Actioners Index(string url)
{
    string query = Request.Query["url"];
    _queryStringProcessor.Processes(query);
    if (!_queryStringProcessor.IsQueryStringComplete() ||
        _queryStringProcessor.IsIntQueryValid() ||
        _queryStringProcessor.IsTimeValid())
    {
        return RedirectToAction("Error", "Register");
    }

    return View();
}

Figure 4.8: Modified ActionResult method.

This modified method receives a string parameter url which specifies to the method the values that exist in the query string. To extract the parameters from the query string received, a string variable with the name query was created and assigned to the received request. The query method takes the query string key and returns the value of it. The value of the key url is shown in Figure 4.9.

"id=12&objectNumber=F2323245-B21E-4474-99ED-EB1EB641D243dot&customerNumber=31&address=Timotejstigen+6&startTime=2017-03-31T11:00&EndTime=2017-04-10T16:00&isMail=true",

Figure 4.9: The value of the URL query string.

This set of parameters later gets processed and validated by a class named QueryStringProcessor. This class takes the parameters and checks if all the required key-pair values exist in the URL, and if they are correct. If the user entered the wrong URL parameters, or some other problem occurs, the end
user gets redirected to an Error method in the RegisterController class showed in Figure 4.10. The Error method returns a web page with an error message to the user, but if nothing is wrong the Index method returns the otherwise appropriate View to the user.

```csharp
public ActionResult Error()
{
    return View();
}
```

*Figure 4.10: Error method that returns a web page with an error message to the user.*

### 4.3.1.2 Data Exchange

The main purpose of the Index method described in the previous section was only to validate the request and respond either with a “success” View or an “error” View. For the actual data exchange between the client and the server a GetData method was created that receives the query string parameters from an AJAX request. This is because AJAX makes the data exchange between browser and server easier. To send query string parameters with AJAX, the URL is processed and parsed in the browser. The parsed query string was sent as a JSON object, where the key is `url` and the value is the data parameters. Figure 4.11 shows how the data is sent via AJAX and Figure 4.12 shows the sent JSON objet.
Figure 4.11: AJAX request header.

```
Host: "localhost:5000"
User-Agent: "Mozilla/5.0 (Macintosh; Intel ... Gecko/20100101 Firefox/53.0"
Accept: "application/json, text/javascript, */*; q=0.01"
Accept-Language: "sv-SE,sv;q=0.8,en-US;q=0.5,en;q=0.3"
Accept-Encoding: "gzip, deflate"
Referer: "http://localhost:5000/REgister...=2017-04-10T16:00:00&isMail=true"
Content-Type: "application/x-www-form-urlencoded; charset=UTF-8"
X-Requested-With: "XMLHttpRequest"
Content-Length: "225"
Connection: "keep-alive"
```

Figure 4.12: JSON Object with its query string parameters.

This request gets handled by the `GetData` method in the web server. The method receives the requested object and parses it to JSON format in order to use the data in the class. A string object with the variable name `urlData` was created to hold the parsed data. Figure 4.13 shows how the data was parsed with the `JObject` object, A .NET object that represents JSON.

```csharp
public ActionResult getData(string url)
{
    var urlFromAJAX = JObject.Parse(url)["URL"];
    string urlData = urlFromAJAX.ToString();
}
```

Figure 4.13: Parsing the JSON data received from browser.

After the data gets parsed, it gets validated with the same mechanisms as in the previous section. To make the data that was received to be in a more object oriented format, a C# Model class was created called Viewing. The Viewing class contains proprieties to hold the query data. Those properties are then used in the `GetData` method to hold the data received, as shown in Figure 4.14.

```csharp
Viewing viewing = new Viewing();
viewing.Id = _queryStringProcessor
```
Before the different data parameters gets stored in the `viewing` object, it gets parsed by the `QueryStringProcessor` class to make sure the data is parsed into the right datatype. Now that the data is stored as a properly formatted object, the server has its own representation of the data. Since this whole process was a server request, the browser awaits a response from the sever. The data received from the browser contains two types of specific information; Viewing and broker/company information. The Viewing information includes address of the real estate. It contains start and end time of the viewing as well as a Boolean type called `isMail`. `isMail` is true if the URL link was received by email, meaning it was sent from the broker to the interested participant, or false if the application is running on an iPad device at the real estate showing.

The company/broker information includes broker-id, object-number and customer-number. Since this data is considered sensitive it is not part of the data that gets sent back to the browser. The only data sent back to the browser is the address, time of the viewing and the `isMail` variable. To send the data a JSON object model was created with address and time of the viewing. This is shown in Figure 4.15.

```javascript
var model = new {
    address = viewing.Address,
    startTime = viewing.startTime,
    endTime = viewing.EndTime,
};
```
isMail = viewing.isMail

};

return Json(model);

Figure 4.15: The JSON object that is sent to the browser, containing information about the real estate showing.

The browser receives the data and appends it to the web page. How the browser client receives the data is shown in Figure 4.16, and the web page with the response data is shown in Figure 4.17.

Figure 4.16: Display of how the browser receives the real estate showing data.

Figure 4.17: The web application with the received showing data applied to it.
4.3.2 Form Data

When the information from the URL is parsed and validated, the end user of the application will be able to register his/her information as shown above. The information needed from the end user to fill in first name, last name, phone number, email, address and if he/she wants to receive additional information from the broker agency by email. Before this information is sent to the web server, the input fields must be validated in the browser. This was partly done in the HTML files, but mostly it was handled in TypeScript files. In the HTML there were attributes on the input fields for `maxlength`. This was so that it was not possible to have an input text longer than the `maxlength` attribute stated. This is displayed in Figure 4.18.

```
<input type="text" name="lastName" maxlength="30">
<input type="text" name="emailAddress" maxlength="30">
<input type="text" name="postNumber" maxlength="7">
```

*Figure 4.18: HTML code displaying the usage of the `maxlength` attribute.*

In TypeScript, advantage was taken of the framework Knockout and its validation plugin. This was of great use and what was desired for the application. This made it possible to dynamically validate inputs so that when the submit button is pressed, each of the inputs with the validation plugin applied to it will be checked for errors. If there are errors, appropriate messages are displayed, and if there are no errors the application shows the receipt view. To make this possible the `data-bind` attribute was used in the HTML on the input fields. The bindings are connected to the ViewModel `inputModel` variable that is located in a TypeScript file. The HTML code displaying this can be seen in Figure 4.19 and the `inputModel` in the TypeScript file can be seen in Figure 4.20.
The `inputModel` shows attributes of `lastName`, `email` and `postalCode`. These attributes are all Knockout Observables. `lastName` and `email` are strings whilst `postalCode` is a number. `lastName` and `email` also extends further validations in that they are both “required”, meaning that they cannot be left empty. `lastName` also is required to be of “minLength” (minimum length) 2, and `email` has the extension of “email” which means that it needs to be written as an email address (x@y.com). Figure 4.21 displays the resulting view of the application if incorrect data is applied to the form.
When all the user inputs are validated the information is sent to the server using AJAX. A separate class called jsonGetter was created with an AJAX function that takes in a participant that contains all the input data of the end-user. This participant is parsed into JSON format and then sent to the web server that also validates the data and eventually sends it to the REST API.

The function that takes the participant is displayed in Figure 4.22 and is called sendFormInput. It starts off by parsing the participant from a string format to a JSON format, then with the help of AJAX the participant is sent with the post HTTP-request to the getForm method in the RegisterController class at the web server.

![Figure 4.22: Displaying the function sendFormInput and how it sends a participant via AJAX.](image)

### 4.3.3 Server to REST Communication

The participant information posted from the browser is received by the GetForm method in the Register class shown in Figure 4.23. In ASP.NET Core there is a command called model-binding which maps JSON objects to C# object. This approach was used to map the participant object sent from the client and the participant object in the web server. To send the participant information to the REST API that the company uses, a SaveParticipant class was created to handle the communication between the web server and the REST API. A SetRestUrl method was created and called in the GetForm method to set the REST URL to be able to save the participant in the company data servers. Before sending the information to the REST API, the process must be authorized, as displayed in Figure 4.23. The SaveParticipant
class has an authorization method which is called in the *GetForm* method. The method takes in a username and a password. Due to confidentiality the username and password is obscured. After the authorization, the participant data is sent to the REST API by using a *PostAsync* method, which sends the data to the API and awaits a response. If the API receives the information without problems, a successful result is sent to the browser and a new participant can start registering, otherwise the browser receives an error message and the participant need to correct the errors in the form and resubmit.

![Figure 4.23: The GetForm method that communicates with the REST API to save participants.](image)

### 4.4 Protection of Data

As described in previous sections, the data that specifies a specific real estate viewing is passed through the URL as a query string. Vitec considers some of this data to be sensitive of the lower degree. This data should not be viewed by the end user and should be protected during communication. Therefore, this issue was solved by implementing an encryption algorithm to the query string, so that the data is not exposed to none other than Vitec and the broker agencies. Vitec will also implement HTTPs and SSL when they integrate the application to their system to protect the data in transit. This section will describe how the encryption is implemented and how the data should be protected.
4.4.1 Encryption Algorithm Class

An encryption class was created to encrypt the data. The encryption class has a constructor, as shown in Figure 4.24. The constructor takes the parameter cipher, which is an enum class (shown in Figure 4.25), with the variable name provider that specifies the AES or 3DES Key length. It also takes a string key parameter which is the key for encrypting and decrypting. If no key is provided from the user of the method, the constructor returns an error. Otherwise the Prepare method in Figure 4.26 is called, which takes in the provider variable (key length size chosen by the developer) and the key is encoded to bytes.

![Figure 4.24](image)

**Figure 4.24:** Crypt method that takes provider and key as parameters that aims at encoding the provided key to bytes.

![Figure 4.25](image)

**Figure 4.25:** The enum class Cipher.
The `Prepare` method was created to prepare the key and the key length to later encrypt the intended text. The method uses switch-case mechanism to use different key length settings depending on the choice of the user. For example, if the user chooses the TripleDES cipher as the `provider` variable, the TripleDES settings are used which sets the key size using the "`sa`" variable. The "`sa`" variable is declared earlier in the class and it represents the SymmetricAlgorithm class which is used in the .NET framework to create symmetric algorithms.

For encryption an `Encrypt` method was created, as shown in Figure 4.27, that uses the pre-defined settings displayed in Figure 4.26. The encryption method takes in a string of plain text, which is the text the user desires to encrypt. The text entered by the user then gets converted to bytes using the `GetBytes` method from the .NET system library. The converted bytes are then encrypted using the `EncryptBytes` method. Since the purpose of the method is to encrypt the URL, a method called `ToURLToken` had to be used in the encoding class called Encoders, which was previously created by the company. The encoding method encodes the encrypted URL according to URL standards.
A decryption method was also created to decrypt the encrypted URL received from the browser. The decryption method is shown in Figure 4.28. The method uses the same mechanisms that are used in the encryption method. The main differences are that the `FromUrlToken` method was used, and that the decrypted bytes were changed back to string format.

4.4.2 Encryption Implementation

A separate program that implements the encryption algorithm was created to encrypt the query string. This program represents the system that Vitec will use to generate the encrypted query string. Figure 4.29 shows the code block of the program. The key and encryption settings were specified in the Encryption class constructor. A string `text` variable was created to hold the encrypted query string that was entered to the `Encrypt` method. The query string contains the following parameters:

- Broker id
- Object number
- Costumer id
- Address
- Start time
- End time
These parameters get encrypted and then added to the URL. The full link is printed out to the console and is ready to be used in the browser.

![Figure 4.29: Using the Encryption class with key and query string to get the encrypted URL.](image)

### 4.4.3 Decryption Implementation

We have described in the previous section how the AJAX request that is sent from the browser is handled. Now that an encrypted link is implemented, the decryption algorithm had to be added to the `GetData` method as well. The same encryption class that was used in the program that generated the link is also used in the web server, with the same settings and key. Figure 4.30 shows how the decryption was implemented in the `GetData` method.

![Figure 4.30: Showing how the decrypt algorithm was added to the `getData` method.](image)

### 4.4.4 Comparison of Encryption Algorithms

A performance comparison between two of the most popular data encryption algorithms AES and 3DES was implemented to decide which one should be used in the web application. The encryption class described in the previous section was used to compare the two different algorithms. The comparison is separated into two different parts: A general performance test and a web based performance test.

- The general performance test was about comparing the throughput of
the algorithms. This was implemented by encrypting and decrypting files and measuring the time required for each operation. The file size was increased for each test performed to see how the time varied for each attempt.

- The web-based test was performed to compare how long the request and response times were by using the different algorithms in the web application. We used the Mozilla Firefox debugger to analyse the algorithms.
5 Result

This chapter shows the results of the comparison between the encryption algorithms and a prototype of the web application and its encryption.

5.1 Web Application

The result of our implementation is a web application that receives an encrypted URL with query string parameters that identifies a specific real estate viewing. This parameter contains information about the real estate viewing such as address and time for viewing, it also contains company specific information such as client number, broker id and installation id for the participant.

The encrypted link gets parsed by JavaScript code to extract only the encrypted part and send it as an AJAX request to the server. The server receives the encrypted query string, decrypts it and checks if the parameters are valid. If the validation went through successfully, the server keeps the company specific parameters and responds with address and time for the viewing. If there were any problems with the query string and the validation did not go through successfully, the server responds with an error message and the user gets redirected to an error page.

The end user of the application fills the form with first name, last name, phone number and address information and then submits the form. The data collected from the user gets sent to the server with an AJAX request as a JSON object. The server receives the data and validates it. If the data is correct, the data gets combined with the company specific information and gets posted to the REST API, where it then gets saved to the database and the server responds with a confirmation to the end user.
Figure 5.1: Display of the start of the web application with the Swedish language selected.

Figure 5.2: The web application when the input fields are entered incorrectly. Norwegian language selected whilst mouse is hovering the flags.
Figure 5.3: Receipt screen of registration. The Figure shows when a user is on a private device with no option to register a new participant.

Figure 5.4: Receipt screen of registration. The Figure shows when a user is on a public device, and is given option for a new registration.

5.2 Comparison of Encryption Algorithms

Table 5.1 shows a comparison of AES and 3DES. This table together with our research gives us the following results of the algorithms security:
• 3DES keys are of shorter length and are weaker than AES keys [20].
• AES uses larger block sizes and lengthier keys, which provides more security [21].
• AES is resistant against differential, linear, interpolation and square attacks, whilst 3DES is weak against differential and linear cryptanalysis, as well as substitution tables [20].

<table>
<thead>
<tr>
<th>Factors</th>
<th>AES</th>
<th>3DES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key length</td>
<td>128, 256 bits</td>
<td>key1 = 168</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key 2, key 3 = 112 bits</td>
</tr>
<tr>
<td>Block size</td>
<td>128, 192 or 256 bits</td>
<td>64 bits</td>
</tr>
<tr>
<td>Created</td>
<td>2000</td>
<td>1978</td>
</tr>
<tr>
<td>Cipher type</td>
<td>Symmetric block cipher</td>
<td>Symmetric block Cipher</td>
</tr>
<tr>
<td>Security</td>
<td>Excellent security</td>
<td>Adequate Security</td>
</tr>
<tr>
<td>Rounds</td>
<td>10, 12 or 14</td>
<td>48</td>
</tr>
</tbody>
</table>

*Table 5.1: A comparison of the data encryption methods AES and 3DES*

Following tables and figures show the result of our tests comparing the data encryption algorithms AES and 3DES. First three tests are displayed of when encryption is used, then the same three tests are done but with decryption instead. The tests show time it takes for each algorithms to encrypt/decrypt files of varying sizes, average time and throughput.
Table 5.2: Statistics of encryption time in milliseconds on files of varying sizes, with average and throughput, AES vs 3DES.

<table>
<thead>
<tr>
<th>DATA (kB)</th>
<th>AES</th>
<th>3DES</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>56</td>
<td>51</td>
</tr>
<tr>
<td>78</td>
<td>62</td>
<td>55</td>
</tr>
<tr>
<td>124</td>
<td>74</td>
<td>69</td>
</tr>
<tr>
<td>247</td>
<td>110</td>
<td>108</td>
</tr>
<tr>
<td>356</td>
<td>174</td>
<td>158</td>
</tr>
<tr>
<td>559</td>
<td>186</td>
<td>192</td>
</tr>
<tr>
<td>789</td>
<td>216</td>
<td>228</td>
</tr>
<tr>
<td>931</td>
<td>248</td>
<td>273</td>
</tr>
<tr>
<td>2239,13</td>
<td>557</td>
<td>661</td>
</tr>
<tr>
<td>8412,123</td>
<td>1291</td>
<td>1432</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>297,4</td>
<td>322,7</td>
</tr>
<tr>
<td>Throughput</td>
<td>2,22</td>
<td>1,41</td>
</tr>
</tbody>
</table>

Figure 5.5: Milliseconds it takes to encrypt text files of different sizes, AES vs 3DES.
Figure 5.6: Throughput (Mbyte/Sec) when encrypting, AES vs 3DES.

Table 5.3: Statistics of decryption time in milliseconds on files of varying sizes, with average and throughput, AES vs 3DES.
Figure 5.7: Milliseconds it takes to decrypt text files of different sizes, AES vs 3DES.

Figure 5.8: Throughput (Mbyte/Sec) when decrypting, AES vs 3DES.
5.2.1 Web Based Comparison

This section shows the results of the response times when the URL is decrypted and sent back to the browser.

![Figure 5.9: 3DES response time using the web application.](image1)

![Figure 5.10: AES response time using the web application.](image2)

5.3 URL Encryption

The result of the URL encryption is an encryption created with AES. The plain text contains information about the viewing that the participant/participants are going to partake in, as well as information about the participant/participants themselves. As you can see in Figure 5.5 there are some keys of id, objectNumber and customerNumber. There are also attributes following those, for example address, time, date and a variable called isMail. All variables are self-explanatory except for isMail, which is a variable that is either true or false depending on if the URL was received via email, or not.

The resulting URL can be seen in Figure 5.6. This URL contains no visible variable names, keys or other values. If a user tries to adjust the URL it will lead to an error screen, saying that the URL was not entered correctly.
Figure 5.11: Displaying the plain text URL with keys and its values.

Figure 5.12: Displaying the encrypted URL with no visible keys or values of them.
6 Analysis

In this chapter we will analyse the results from the data encryption algorithm comparison. The data will be divided into two parts: security of encryptions and the performance tests.

6.1 Security of Algorithms

When it comes to security, AES have great advantages over 3DES because AES is considered unbreakable in practice. To break AES encryption with a key-length of 128 bits takes approximately $5 \times 10^{21}$ years. However, to break 3DES with a key-length of 112 takes approximately 800 days. Another disadvantage with 3DES is that the rounds are fixed to 16, whilst in AES the default rounds are 10, 12 and 14, but it can be increased if a higher level of security is desired.

6.2 Performance of Algorithms

The speed and performance of the algorithms differ very little in small files, where 3DES is faster by a small margin. When increasing the file sizes AES becomes gradually faster, but the difference is still small. This goes for both encryption and decryption of the files. When encrypting the files AES passes 3DES in speed at 559 kB, which means prior to that 3DES is faster. When decrypting, AES passes 3DES in speed at 789 kB instead. This means that the amount of files when 3DES is faster than AES is larger when decrypting, compared to encrypting. However, AES is faster than 3DES on average, even though the average speed is reduced for both algorithms when decrypting, especially for 3DES. Other than that, AES also has a higher throughput both when encrypting and decrypting the files.

On the response times of the algorithms, 3DES beat AES by having a response time less than half of AES’ time (105 MS compared to 256 MS).
Discussion and Conclusion

The goal of the theoretical part of this thesis project was to create a safe and secure way for Vitec to pass sensitive information over the URL, even though security professionals advice to not pass sensitive information via the URL and instead pass it through HTTP headers. Therefore, we were given the task to find which security measurements were appropriate for protecting data passed through a URL and later implement it to the application and analyse the result. It was crucial for Vitec that the data was well protected and the request/response time was optimal.

In the beginning we had to do some research on which security measurements that we should consider and later implement. Search results showed that encryption was the optimal solution for protecting the information from outside exposure. Since there are many different encryption algorithms we decided to compare different algorithms to see which one we should implement. At first we wanted to compare 3 different algorithms; Blowfish, 3DES and AES, since those are the most popular data encryption methods. But we realized that the .NET framework did not have support for the Blowfish algorithm and decided to not include it in our research.

The goal for comparing the algorithms was to find which of them was the safest and fastest when encrypting and decrypting. In our comparison we concluded that 3DES was faster than AES when it comes to small amount of data, but the difference is barely noticeable. When it comes to security AES was favoured over 3DES because of it being so hard to break, as well as some other aspects. Therefore, we chose AES as our main encryption algorithm because of its security.

We think that security should be favoured over performance when it comes to handling sensitive data. This is because if the performance differences are small, but the strength of the algorithms vary widely, then performance should not be a determining factor of what algorithm to choose.

Before starting the practical part of the assignment at Vitec we had to do research on topics that we did not have experience in, or only had some experience in. These topics were for example TypeScript and .NET. We also had to refresh our knowledge in subjects that we had worked on earlier, like HTML and GitHub.
When starting the development of the application requested, we took our knowledge we had in HTML and CSS and made a simple application that we could have as a base of operation. After we then received a template for the design and functionality of the application from Vitec, we could trail that and then implement it to our temporary application. Alongside this we had to do further research of how the development was going to proceed. Since we only had knowledge in TypeScript and .NET from the research we did prior to this project, we had to continue that study to be able to get the sought result.

Whilst we studied the knowledge that was required, we still continued the development of the application with the information that we were gathering. Attention was put in the application that it was supposed to be reusable, so we implemented previous experience we had with MVC patterns and some other development strategies. This is because the goal was that this application was to be implemented for Vitec to use, and integrate it with their working systems.

With a working application that supports many of the base requirements, we are now satisfied with what we have accomplished and how we have dealt with how sensitive information is sent through the URL from client to web server by protecting it with appropriate encryption methods.

One thing that could have been improved is the time spent on researching the differences between the encryption algorithms, so that we could have more data to work with which would improve our results. We could have improved on the planning of the project, defining stricter goals and strategies. The result of this was that many things took longer time than they needed to, for example defining our research question.

We are happy with the cooperation between us and Vitec, we had good communication that led to a successful work ethic. The teamwork between us two as a group has been great since each of us knew what duties needed to be done and performed them in a good way.

There are still steps left regarding the development of web application that could be implemented to the application, for example offline support and automatically rendering a company logo for the broker agency that uses the application. However, we did not have enough time to implement these steps,
but we did our best to make our code as reusable and documented as possible, so Vitec easily can continue working on the application and eventually integrating it to their systems.
8 References


