WePick – A dynamic web app that creates curated playlists based on multiple user's music preferences using Spotify

Edward Eldridge

Danielis Joniskis

Keith Higgins

B.Sc.(Hons) in Software Development

April 9, 2019

Final Year Project

Advised by: Martin Kenirons

Department of Computer Science and Applied Physics

Galway-Mayo Institute of Technology (GMIT)

INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO

GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Contents

About this project

0.0.0.0.1 Abstract A dynamic web app that creates curated playlists based on multiple user's music preferences using Spotify. Built with React, Flask, Python, Node.js, Mocha, AWS and Mongo.

0.0.0.0.2 Authors

Eddie Eldridge
Danielis Joniskis
Keith Higgins
Three students from the Galway-Mayo Institute of Technology

Chapter 1

Introduction

The introduction should be about three to five pages long. Make sure you use references [1]

Context for the Project.
Objectives of the Project.
Is the reader 100 percent of what the project is all about

Chapter 2

Context

- Provide a context for your project.
- Set out the objectives of the project
- Briefly list each chapter / section and provide a 1-2 line description of what each section contains.
- List the resource URL (GitHub address) for the project and provide a brief list of the main elements at the URL.

2.1 Filler

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam mi enim, interdum ut elit lobortis, bibendum tempus diam. Etiam turpis ex, viverra tristique finibus nec, feugiat at metus. Curabitur tempus gravida interdum. Donec ac felis a lorem scelerisque elementum. Vestibulum sit amet gravida tortor, a iaculis orci. Nam a molestie augue. Curabitur malesuada odio at mattis molestie. In hac habitasse platea dictumst. Donec eu lectus eget risus hendrerit euismod nec at orci. Praesent porttitor aliquam diam, eu vestibulum nisl sollicitudin vel. Nullam sed egestas mi.

Quisque vel erat a justo volutpat auctor a nec odio. Sed rhoncus augue sit amet nisl tincidunt, vitae cursus tellus efficitur. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque et auctor dui. Fusce ornare odio ipsum, et laoreet mi molestie sed. Cras at massa sit amet ipsum gravida aliquam. Nulla suscipit porta imperdiet. Fusce eros neque, bibendum sit amet consequat non, pulvinar quis ipsum.

2.1.1 More filler

Donec fermentum sapien ac rhoncus egestas. Nullam condimentum condimentum eros sit amet semper. Nam maximus condimentum ligula. Praesent faucibus in nisi vitae tempus. Sed pellentesque eleifend ante, ac malesuada nibh dapibus nec. Phasellus nisi erat, pulvinar vel sagittis sed, auctor et magna. Quisque finibus augue elit, consequat dignissim purus mollis nec. Duis ultricies euismod tortor, nec sodales libero pellentesque et. Interdum et malesuada fames ac ante ipsum primis in faucibus.

Donec id interdum felis, in semper lacus. Mauris volutpat justo at ex dignissim, sit amet viverra massa pellentesque. Suspendisse potenti. Praesent sit amet ipsum non nibh eleifend pretium. In pretium sapien quam, nec pretium leo consequat nec. Pellentesque non dui lacus. Aenean sed massa lacinia, vehicula ante et, sagittis leo. Sed nec nisl ac tellus scelerisque consequat. Ut arcu metus, eleifend rhoncus sapien sed, consequat tincidunt erat. Cras ut vulputate ipsum.

Curabitur et efficitur augue. Proin condimentum ultrices facilisis. Mauris nisi ante, ultrices sed libero eget, ultrices malesuada augue. Morbi libero magna, faucibus in nunc vitae, ultricies efficitur nisl. Donec eleifend elementum massa, sed eleifend velit aliquet gravida. In ac mattis est, quis sodales neque. Etiam finibus quis tortor eu consequat. Nullam condimentum est eget pulvinar ultricies. Suspendisse ut maximus quam, sed rhoncus urna.

2.2 Filler

Phasellus eu tellus tristique nulla porttitor convallis. Vestibulum ac est eget diam mollis consectetur. Donec egestas facilisis consectetur. Donec magna orci, dignissim vel sem quis, efficitur condimentum felis. Donec mollis leo a nulla imperdiet, in bibendum augue varius. Quisque molestie massa enim, vitae ornare lacus imperdiet non. Donec et ipsum id ante imperdiet mollis. Nullam est est, euismod sit amet cursus a, feugiat a lectus. Integer sed mauris dolor.

Mauris blandit neque tortor, consequat aliquam nisi aliquam vitae. Integer urna dolor, fermentum ut iaculis ut, semper eu lacus. Curabitur mollis at lectus at venenatis. Donec fringilla diam ac risus imperdiet suscipit. Aliquam convallis quam vitae turpis interdum, quis pharetra lacus tincidunt. Nam dictum maximus lectus, vitae faucibus ante. Morbi accumsan velit nec massa tincidunt porttitor. Nullam gravida at justo id viverra. Mauris ante nulla, eleifend vitae sem vitae, porttitor lobortis eros.

Cras tincidunt elit id nisi aliquam, id convallis ex bibendum. Sed vel odio fringilla, congue leo quis, aliquam metus. Nunc tempor vehicula lorem eu ultrices. Curabitur at libero luctus, gravida lectus sed, viverra mi. Cras ultrices aliquet elementum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed metus ante, suscipit sit amet finibus ut, gravida et orci. Nunc est odio, luctus quis diam in, porta molestie magna. Interdum et malesuada fames ac ante ipsum primis in faucibus. Mauris pulvinar lacus odio, luctus tincidunt magna auctor ut. Ut fermentum nisl rhoncus, tempus nulla eget, faucibus tortor. Suspendisse eu ex nec nunc mollis pulvinar. Nunc luctus tempus tellus eleifend porta. Nulla scelerisque porttitor turpis porttitor mollis.

Duis elementum efficitur auctor. Nam nisi nulla, fermentum sed arcu vel, posuere semper dui. Fusce ac imperdiet felis. Aenean quis vestibulum nisl. Integer sit amet tristique neque, at suscipit tortor. Morbi et placerat ante, vel molestie dui. Vivamus in nibh eget massa facilisis accumsan. Nunc et purus ac urna fermentum ultrices eget sit amet justo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Cras elementum dui nunc, ac tempor odio semper et. Ut est ipsum, sollicitudin eleifend nisl eu, scelerisque cursus nunc. Nam at lectus vulputate, volutpat tellus vel, pharetra mauris. Integer at aliquam massa, at iaculis sem. Morbi nec imperdiet odio. In hac habitasse platea dictumst.

Mauris a neque lobortis, venenatis erat ut, eleifend quam. Nullam tincidunt tellus quis ligula bibendum, a malesuada erat gravida. Phasellus eget tellus non risus tincidunt sagittis condimentum quis enim. Donec feugiat sapien sit amet tincidunt fringilla. Vivamus in urna accumsan, vehicula sem in, sodales mauris. Aenean odio eros, tristique non varius id, tincidunt et neque. Maecenas tempor, ipsum et sollicitudin rhoncus, nibh eros tempus dolor, vitae dictum justo massa in eros. Proin nec lorem urna. In ullamcorper vitae felis sit amet tincidunt. Maecenas consectetur iaculis est, eu finibus mi scelerisque et. Nulla id ex varius, ultrices eros nec, luctus est. Aenean ac ex eget dui pretium mattis. Ut vitae nunc lectus. Proin suscipit risus eget ligula sollicitudin vulputate et id lectus.

Chapter 3

Methodology

About one to two pages. Describe the way you went about your project:

- Agile / incremental and iterative approach to development. Planning, meetings.
- What about validation and testing? Junit or some other framework.
- If team based, did you use GitHub during the development process.
- Selection criteria for algorithms, languages, platforms and technologies.

Check out the nice graphs in Figure 3.2, and the nice diagram in Figure **??**.

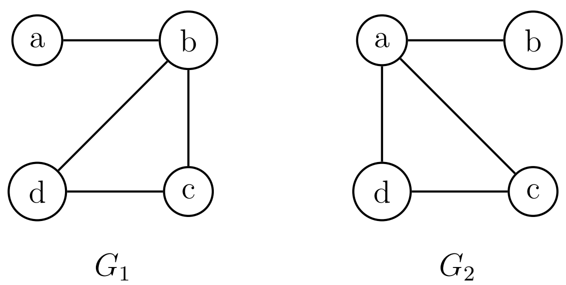


Figure 3.1: Nice pictures

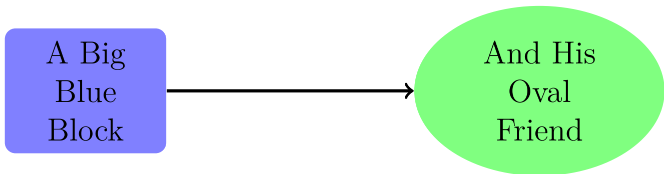


Figure 3.2: Nice pictures

Chapter 4

Technology Review

About seven to ten pages.

- Describe each of the technologies you used at a conceptual level. Standards, Database Model (e.g. MongoDB, CouchDB), XMl, WSDL, JSON, JAXP.
- Use references (IEEE format, e.g. [1]), Books, Papers, URLs (timestamp) – sources should be authoritative.

4.1 AWS

Amazon Web Services offers reliable, scalable, and inexpensive cloud computing services.

```xml
<this>
  <looks lookswhat="good">
    Good
  </looks>
</this>
```

4.2 MongoDB

MongoDB is an open-source, document database designed for ease of development and scaling. MongoDB stores data in JSON-like documents, which makes the database very flexible and scalable.[?]

4.3 Flask

Flask is a microframework for Python.

4.4 Python

Python is a programming language that lets you work more quickly and integrate your systems more effectively.

4.5 Mocha

Mocha is a JavaScript test framework running on Node.js. Mocha tests run serially, allowing for flexible and accurate reporting

4.6 React

React is a JavaScript library for building user interfaces. React is developed by Facebook. It is a tool for building UI components. React uses Babel to convert JSX (JavaScript XML) into JavaScript. Babel is a JavaScript compiler that can translate markup or programming languages into JavaScript. With Babel, you can use the newest features of JavaScript (ES6 - ECMAScript 2015). [?]

Chapter 5

System Design

As many pages as needed.

- Architecture, UML etc. An overview of the different components of the system. Diagrams etc... Screen shots etc.

Column 1	Column 2
Rows 2.1	Row 2.2

Table 5.1: A table.

Chapter 6

System Evaluation

In this chapter, we will discuss many aspects of the software. We will break the evaluation down into 4 headings.

- Robustness - How well the software can deal with problems, it's ability to handle change etc.
- Performance - Space and time complexity of the software, responsiveness etc.
- Security - How vulnerable is the application to attacks and security risks?
- Overall evaluation - Where the project succeeded/failed, limitations in our approach and technologies etc.

6.0.1 Robustness

Robustness in regards to software can be defined as the ability of the software to cope with errors during execution and cope with erroneous input. Measuring the robustness of a system is difficult as no system can be considered completely robust. However, there are certain methodologies and practices we can implement to increase the robustness of a system. *Behdis Eslamnour and Shoukat Ali* discuss these metrics in their paper entitled *Measuring robustness of computing systems*[2] Some of these proposed metrics include

- Error Handling/Error Catching
- Time between failures and time between recovery

6.0.2 Error Handling

Error handling refers to how a system handles errors should they occur. This can be as simple as logging the error in the console to rolling back the system to a previous stable release. It is a vital aspect for any system from both the perspective of the user and developer. As a developer, implementing proper error handling and catching is an important aspect of development as it enables the developer to work more effectively and efficiently as less time is spent fixing and locating the cause of bugs if proper error handling is done pre-emptively.

For the user, it's vital that error handling is done correctly as incorrect error handling can have a significant impact on the end-users experience of the system/software. For example, if a user attempts to login to a system and their login details are incorrect, a meaningful way of handling this error would be to notify the user that their login details are incorrect. If this is not done, the end user won't understand what's happening and why they can't use the system and as a result of this will become frustrated/unhappy with the service. As such it's important to make sure any critical problems such as this are caught and handled in an appropriate manner. However, it's also important to remember that complete robustness can not be achieved and as such one must try and figure out the most important and likely errors to handle. When making these decisions, one can consider many factors when deciding what errors to handle such as the risk of not handling the error (application crashing, unexpected behaviour, potential security risk, loss of data etc.) , how much time will be required to handle the error and the likelihood of the error occuring. However, as mentioned above complete robustness can not be achieved as to even consider the above factors, one must have the foresight to see that the error will occur in the firstplace.

Below I will discuss some error handling that occured in our project and how it contributed to the overall robustness of the system.

Example 1 - Login/Register In the image below we can see how error handling and error catching can produce meaningful output and guide the user through login/register scenario of our application

—- user login diagram here —-

Example 2 - Authentication with Spotify In the image below we can see how error handling and error catching can produce meaningful output and guide the user through authenticating our applicatin with Spotify.

— authentication diagram here —-

The above error handling was done on both the server and client-side of the application. Both Python, MongoDB and Pythn provide useful tools for error handling/catching. Some examples include *Error boundaries* in React, *Try/Catch/Raise statements* and *Schema validation* in MongoDB.

- Error Boundaries - Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.

- Try/Catch/Raise statemetns - These commonly found in most programming languages and Python is no exception. We used these to catch certain errors such as HTTPErrors, null value errors, MongoDB errors and many others.

- Schema Validation - To ensure the data being passed into the Mongo database was in the correct format, we used a JSON Schema to validate our inputs against before passing it to Mongo. In the below example, we can see that for a user to be validated correctly, the minimum required properties are an email and password. This ensures that user's cant create accounts without these two required properties.

```
user_schema = {
  "type": "object",
  "properties": {
      "name": {
          "type": "string",
      },
      "email": {
          "type": "string",
          "format": "email"
      },
      "password": {
          "type": "string",
      },
      "spotifyUsername": {
          "type": "string",
      },

  },
  "required": ["email", "password"],
  "additionalProperties": False
}
```

6.0.3 Time between failures and time between recovery

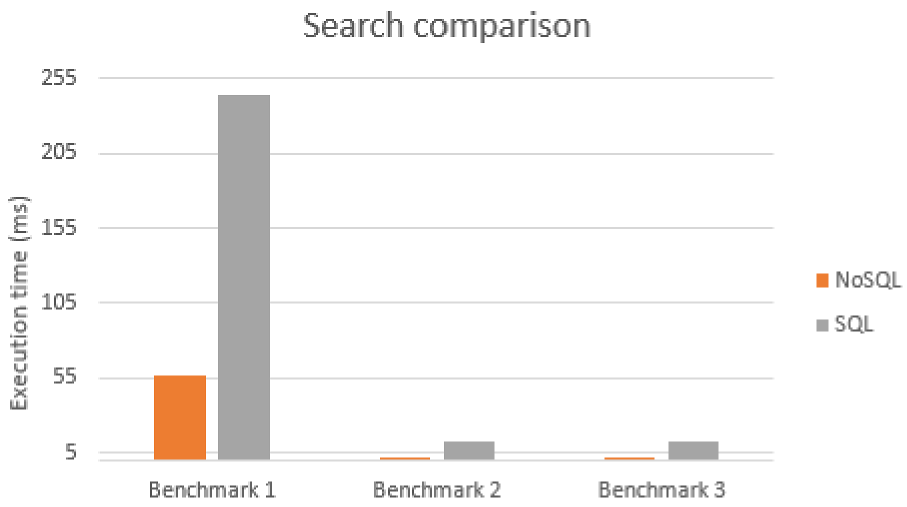
Time between failures and time between recovery are too very useful metrics in determining the robustness of a system. Systems that have a short mean time between failures and mean time between recovery either have excellent error handling or are designed in such a way that errors do not occur frequently. A good way of measuring this could be the downtime of your application over the course of a year. For example, on average, the top 50 e-commerce websites experienced 99.03% uptime. Over a year, 99.03% uptime would result in 3 days 15 hours and 39 minutes of downtime. 32 of these websites experienced 99.99% uptime. 9 out of 10 of users encountering a website that is not up will choose to use a competitors website [3]. From these statistics, we can see that time between failures anre time between recovery are extremely important aspects to any system/application. As such, they can be very helpful metrics when trying to measure the robustness of an application.

In regards to WePick's failure and recovery time, since the project was hosted on AWS' Elastic Beanstalk, there have been little to no outages as due to the continous integration solution integrated upon deployment, it was not possible for broken code to be deployed as any code proposed to be deployed had to first pass tests that were outlined by Travis CI and any problems that did arrive were dealt with quickly as Travis instantly notified us of any problems which we were able to fix quickly and promptly.

6.0.4 Performance

When planning and designing the project, one of our main aim's was that we wanted the project to perform operations and tasks quickly as well as being responsive to user input. Taking these considerations into account, we choose Python and MongoDB for our backend as they are generally considered lightweight and easier to handle performance than say something like Java. We chose React for the front-end of the application as React is well known for it's performance and responsiveness. This generally comes from the fact that React applications usually implement a single page design. React works by creating multiple components that can be re-used multiple times in the application. For example, in our application, instead of creating multiple pages with the same code (Navbar, Footer, Forms etc.), we can create seperate components for each of these items and only re-render what is required. So if the user needs a different form only that form will be re-rendered and the Navbar and Footer don't need to be re-rendered This choice of technology helped us achieve our overall goal in regards to perfromance and I think it was a good choice.

Our choice of MongoDB as a database was also important to the overall perfromance of our application as MongoDB is considered much faster than SQL and other relational databases in certain scenarios. This is because of it's non-relational, no-SQL way of storing data. Instead of storing data in rows, columns and tables, Mongo works by using a document style architecture. These documents are structured similarly to a JSON document and as such are easy to extract data from and manipulate without navigating through large sets of data like you would in a traditional SQL database. Generally, this performance increase can be seen most when you start to scale up the size of your database. For our application, it was important to be able to access user's favorite artists quickly so that they didn't have to wait a long time to generate a playlist.



From the above graph, we can see that when performing a search in the database, NoSQL databases like Mongo are considerably faster than a traditional SQL database /citeSQLvsNOSQL

6.0.5 Security

Securit was an important factor for us to consider when designing and creating the application as we had all experienced problems with group projects in the previous year. For example, many students dealt with problems where database tables were wiped and/or hijacked and held ransom to the owners. As such, we did not want to deal with such problems and thought it would be a good oppurtunity for us all to learn a bit more about how one goes about securing an application in a real world context. We also wanted user's information to be secure and not easily accessible by anyone.

To ensure our RESTful routes were not accessible to anyone, we implemented JWT Tokens into our application. A JWT Token or JSON Web Token is an open-standard that defines a way for securely transmitting and verifiying information between systems. JWT's are signed using a secret key or a public/private key using RSA encryption.

The way this ends up working in the application is as follows. When the user successfully logs in, a JSON Web Token is generated and each subsequent request made by the user will include that web token. Setting up the system in this way means that to perform a HTTP request on one of our many RESTFul routes, the user first has to verify that they are an authenicated user. This prevents from people performing actions on routes that they should not have access to.

A traditional JSON Web Token consists of a Header, Payload and Signature in the following format

`header.payload.signature`

The header details what type of token is being sent and the algorthim used to generate the token. The payload contains the information about the person sending the token and what they would like to do. The signature is essentially an your header and payload encrypted and is used to verify the payload wasn't altered during transit as well as verifying the sender is who they say they are.

An example token can be seen below

To implement this technology in our application, we used Flask-JWT, a simple Python library that provides JWT Support for Flask applications. A simple example of how this technology was implemented can be seen below.

```python
# Authentication Stuff
@application.route('/refresh', methods=['POST'])
@jwt_refresh_token_required
def refresh():
    current_user = get_jwt_identity()
    ret = {
            'token': create_access_token(identity=current
    }
    return jsonify({'ok': True, 'data': ret}), 200

@jwt.unauthorized_loader
def unauthorized_response(callback):
    return jsonify({
        'ok': False,
        'message': 'Missing Authorization Header'
    }), 401
```

As well as ensuring API calls were authenicated, we also made sure that our database and user information was secure. A secure admin account was created on the database, requring a password to perform any actions in the databse. This way, no operations could be performed on the database without knowing both the IP address of the server the database was hosted on and the password required to gain access. These two important pieces of information were known only to us and weren't available anywhere else.

We also made sure that any important user information such as passwords were encrypted before being entered into the databse. For this, we used another Python library for Flask entitiled Flask-Bycrypt. Using this library, we can generate a hash of the user's password and store this in the database instead of storing their password in plaintext. Then, when checking if the user's login details are correct, we compare the generated hash of the passwords instead of the actual passwords themselves. This enables a secure login scenario and prevents passwords being revelead to anyone.

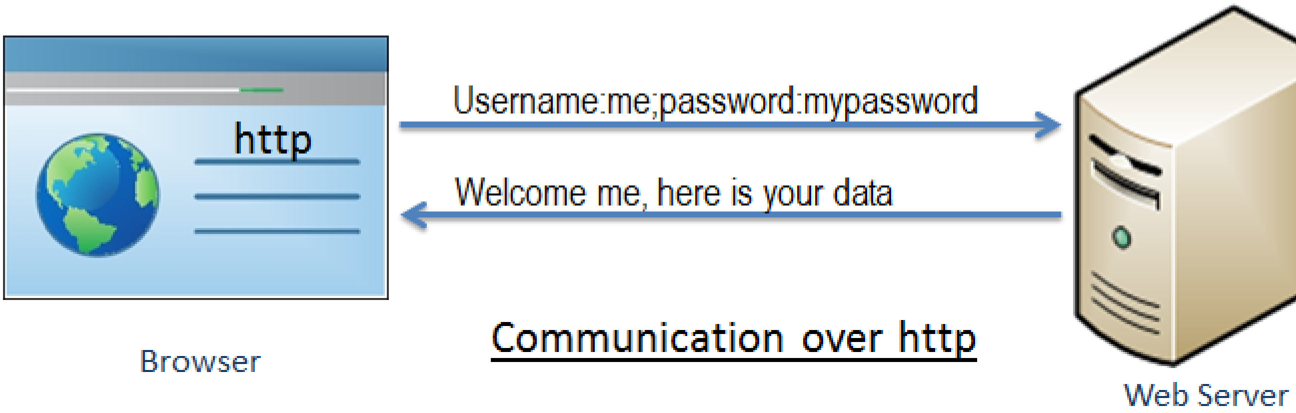
To improve overall security, we also chose to host our application on a different instance than our database. This meant that if one were to be compromised in any way, it would prevent the other in turn being compromised.

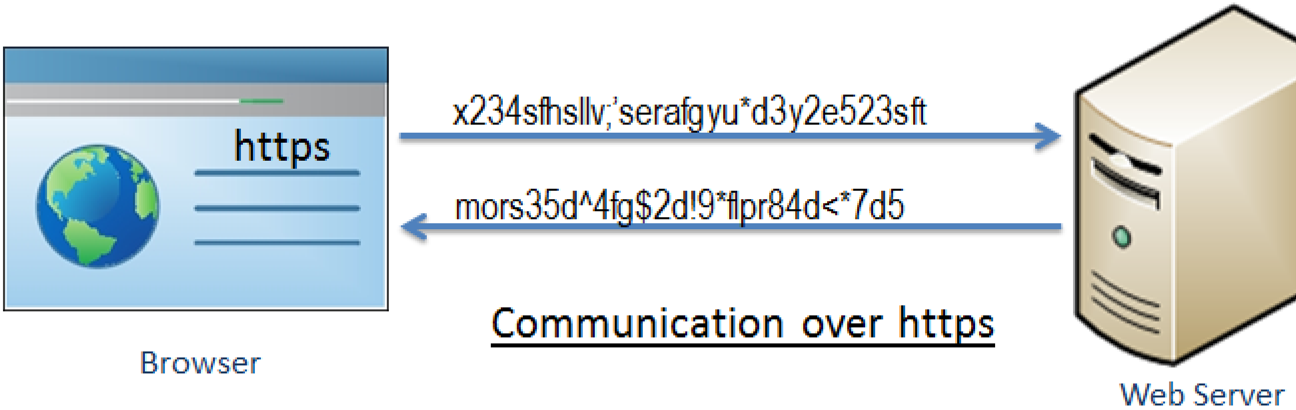
6.0.6 Improving Security

6.0.6.1 HTTP/HTTPS

One shortcoming of our project is the lack of a secure Hyper Text Transfer Protocol (HTTPS). HTTP can be described by Mozilla's standards /cite-HTTPviaMozilla as an application-layer protocol for transmitting data and documents. It was originally designed for communication between web browsers and web servers but it has since exceeded it's original purpose. It follows a client/server model. The client makes a request to establish a connection and waits for a response from the server. Despite this however, it is a /bfstateless protocol. This means that once communication between server and client, the server forgets all information regarding the communication. It can be used on any reliable transfer layer such as TCP/IP or UDP.

The difference between HTTPS and HTTP is that HTTPS is considered 'secure' where HTTP is not. Essentially, the difference between the two can be illustrated cleary in the following diagram.





In standard HTTP, the information is transmitted in hypertext format whereas with HTTPS, the information is first encrypted before sending the information. The issue with standard HTTP is that the data is transmitted in plaintext. This means that anyone who can 'see' the data or is monitoring the network can potentially intercept this data and/or modify it before being passed on to the server. This can lead to serious security issues. This could be potentially devestating to an application/system that relies on sensitive data such as passwords or bank/card details.

HTTPS guarantees that even if the message is intercepted, it would be of no value as it is encrypted. HTTPS provides these guarantees using a security standard known as SSL or Secure-Socket Layer. SSL is a standard for establishing an encryped link between a client and server. To first use SSL, one must aquire an SSL certificate. These certificates are small files that bind a domain name/IP address/hostname to an organizational identity and location.

6.0.6.2 But who exactly has the authority to issue these certificates and how can we trust them?

To issue SSL certificates and guarantee their authenticity, one must become a Certificate Authority. This title is usually limited to private companies and governments that have proven that the certificates they issue are secure. The more secure certificates they authorize, the more certificates they are able to distribute. This means that when aquiring an SSL certificate, you can guarantee it's authenticity. A certified certificate should contain your domain name, company name, address, city, state, country. It also contains an expiration date upon which the certificate must be renewed as well as the Certificate Authority that issued the certificate in the first place. All these factors provide some kind of accountability in regards to the certificate if something were to go wrong. Expiring certificates is a common occurence, even for large companies. According to GlobalSign /citeGlobalSign, between just October 2017 and February 2018, large organizations such as LinkedIn, PokemonGO the British Conservative Party and astonglishly The White House have all let their SSL certificates expire. Any users arriving on these websites would have been instantly greeted with a warning that these websites are not secure and sensitive information could have possibly been leaked in the process. For something like a goverment website, this kind of behaviour is a perfect example of why we have SSL and HTTPS in the first place and why it's vital that these systems are maintained and updated.

6.0.6.3 How does SSL work?

Let's use a traditional client/server architecture as our example. The browser/client connects to the server secured with SSL. The server will then prompt the client to identify itself. The server then sends a copy of it's SSL certificate. These certificate is then verified by the client. If verification is successful and the client is comfortable sending messages, it sends a response to the server which consists of a digitally signed acknowledgement which initiates an SSL encryped session. Any data exchanged between these two parties is now sent over the Secure-Socket Layer established by the client/server.

6.0.6.4 Benefits of SSL/HTTPS

We can describe the overall benefits of SSL/HTTPS as the following

- Utilize HTTPs, which elicits a stronger Google ranking.
- Create safer experiences for your customers.

- Build customer trust and improve conversions.
- Protect both customer and internal data.
- Encrypt browser-to-server and server-to-server communication.
- Increase security of your mobile and cloud apps.

6.0.6.5 Why didn't we implement SSL/HTTPS?

Due to the nature of aquiring an SSL certificate, one usually has to purchase a certificate and pay a yearly fee for using it. We made a decision based on the information stored by our application and by factoring in the cost, decided that we didn't think it would be critical if HTTPS wasn't implemented. We understand it makes our application insecure, but we also understand why it makes it insecure and how this could be dealt with appropriately in future projects.

Chapter 7

Conclusion

About three pages.

- Briefly summarise your context and ob-jectives (a few lines).
- Highlight your findings from the evalua-tion section / chapter and any opportuni-ties identified.

Bibliography

[1] A. Einstein, "Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]," *Annalen der Physik*, vol. 322, no. 10, pp. 891–921, 1905.

[2] B. Eslamnour, "Measuring robustness of computing systems,"

[3] Pingdom, "Downtime for the world's top-50 e-commerce websites."