|  |
| --- |
| ci6300 – INDIVIDUAL PROJECT |
| PROJECT REPORT |
| Daniel Carnovale |
|  |
| **K1336511** |
| **17/04/2017** |

|  |
| --- |
|  |

Table of Contents

[Introduction and Literature Review 4](#_Toc480207605)

[Methodology 4](#_Toc480207606)

[Analysis 4](#_Toc480207607)

[Purchasing of the Domain Name 5](#_Toc480207608)

[Amazon 5](#_Toc480207609)

[Amazon EC2 6](#_Toc480207610)

[Console Access 6](#_Toc480207611)

[Physical Database Implementation 6](#_Toc480207612)

[Design 7](#_Toc480207613)

[Schematic of System Components 7](#_Toc480207614)

[Server hosting In House 7](#_Toc480207615)

[Choice of Amazon as a Cloud Provider 7](#_Toc480207616)

[Choice of Virtual Machine Hosting service 7](#_Toc480207617)

[Choice of Source Control (GitHub vs Dropbox) 7](#_Toc480207618)

[Local access to server 8](#_Toc480207619)

[Database (Technology, Management tool PHPPgAdmin) 8](#_Toc480207620)

[OS (Windows VS Linux) 8](#_Toc480207621)

[Choice of Linux distribution (Amazon Linux) 8](#_Toc480207622)

[Security (Approach, Confidentiality, Privacy, SSH Key Pair) 8](#_Toc480207623)

[StarUML Class Diagram 8](#_Toc480207624)

[Backup techniques (test data, use PhpPgAdmin frequently to export dump files, considered Ansible and Terraform etc) 8](#_Toc480207625)

[Apache / PostGres configuration 8](#_Toc480207626)

[Docker consideration 8](#_Toc480207627)

[How should I record information? Word .doc or GitHub .md files. Doc is such a standard, though for technical documentation I believe this should be in .md format. 8](#_Toc480207628)

[How to reliably reproduce runtime environment (research suggests that Docker is a strong solution to this, but as yet I have to learn Docker effectively) 9](#_Toc480207629)

[Choice of web-server (Apache, NginX ?) Apache because.. 9](#_Toc480207630)

[Hack attempt prevention, disabling clear-text password login access. 9](#_Toc480207631)

[Similarities between Amazon Linux and Centos 6 (why Centos 6.x instructions should be followed) 9](#_Toc480207632)

[Installation of PostgreSQL 9](#_Toc480207633)

[Regular yum update to enforce preventing of vulnerabilities regarding security issues 9](#_Toc480207634)

[Looked into SQL generation from UML but assumed it was very complex / couldn’t find anything / wouldn’t work for my version 10](#_Toc480207635)

[Benefit of public key usage with GitHub 10](#_Toc480207636)

[Considered enhancement of GitHub credentials 10](#_Toc480207637)

[Consideration of static (elastic) instead of dynamic ip 10](#_Toc480207638)

[Documentation of snapshots to accompany steps document 10](#_Toc480207639)

[Implementation 11](#_Toc480207640)

[Discussing How-To steps (Actual steps in appendix?): 11](#_Toc480207641)

[Testing & Evaluation 11](#_Toc480207642)

[Critical Review 11](#_Toc480207643)

[Appendix A 11](#_Toc480207644)

# Introduction

( Cloud computing services have become more and more desirable as a means of hosting and managing projects for business or personal requirements, as opposed to in-house services. These two options can be combined however to further strengthen a project and I will be exploring this as a review of what I have applied to my “Unihood” project. Ultimately for this project a database and web app are required, and hosting and managing these within a virtualisation environment provides several benefits including automation of the re-construction of the entire system. )

Many low-cost cloud-computing resources and new free open-source software components are now available to software developers. These have reduced the costs that were associated with the creation of web-based software systems. Although a large team of IT professionals and expensive hardware and communications resources are no longer necessary, and can be replaced by just a few individuals, it’s still essential for those individuals to be able to identify which tools and resources will meet their needs. They must also know how to effectively combine these tools to build their proposed system.

This project is an attempt to itemise and instruct (in a “HOW-TO” fashion) one such combination of tools and resources that will allow a complete web application system to be built with minimal code, costs and manpower. We address many of the considerations that are essential for real-world feasibility, such as Security, Source control, Issue management, Disaster Recovery, Scalability and Maintainability. To this end, I have built a system called “UniHood” whose ultimate goal is to allow a university community to review and comment on courses, their modules and lecturers in a more structured way than the typical forums and social networks.

This work is particularly important given that the huge diversity of Open-Source components now available makes it extremely difficult for many students and individuals to get started with building a system such as UniHood. This project will help them begin to realise the cost-savings that are being achieved via cloud-based computing.

Stakeholders should be able to use these instructions to build a completely different application. Many individuals and organisations would find the HOW-TO deliverables of this project extremely valuable because of the significant time-saving if they want to start a project using similar tools.

# Literature Review

There is a long tradition of “HOW-TOs” in the open-source community, and some of these may have similar goals, however the range of available web-building tools and resources (such as Amazon AWS Free Tier and Google Single Sign-On) is evolving so rapidly that one might prepare the same report in 10 months’ time and recommend different tools in it.

# Primary Deliverables

The primary deliverables achieved were:

* Running cloud-based service, ie. the UniHood system operating as a live public-facing website
* Collection of specific HOW-TOs including screenshots
* Explanation of Components
* Design Document
* Discussion of functionality achieved
* Observations

# Intermediate Deliverables

There are many intermediate deliverables, each of which is self-contained and may be of value to many different types of projects. Some of these deliverables will be self-contained discussions of:

* Authentication
* Authorisation
* Security
* Configuration
* Source control
* Issue management
* Backup
* Disaster Recovery
* Scalability
* Productivity
* Automated testing
* Continuous Integration
* Continuous Delivery
* Continuity of Service
* Domain Name Management

**Technologies & Resources**

**Major resources required**

* Online documentation pages for Amazon AWS, MySql and other Software resources
* Forums and discussion sites such as StackOverflow
* Client-side PC workstation(s)

**Technologies Utilised**

For Cloud-Computing resources I expect to use various Amazon AWS Resources such as:

* EC2 – Elastic Compute Cloud, dedicated to managing Amazon Virtual Machines.
* S3 - Simple Storage Service, allows upload and download of very large files
* EBS – Elastic Block Storage, mainly used for adding extra drives to currently running Virtual Machine instances.
* Elastic IPs – A concept in which static IPs are able to be assigned to an Amazon Account and because of their “elasticity”, provide a means of mapping this same IP straight to another instance when the previously IP assigned instance fails.

Amazon offers a “Free Tier” service which allows these above resources to be used for free or very low-cost during the first year.

Additional resources to be chosen from:

* GitHub for Source Control and as a Cloud-Based repository of project documentation. This is a free service for Open-Source project work.
* MySQL for database back-end
* PuTTY as an SSH client
* Programming language eg. PHP, Python, Ruby or Perl.
* HTML5 / CSS / Javascript for browser-side programming
* Javascript UI Framework eg. Bootstrap, Angular, JQuery UI
* Database Technology eg. mySQL, PostGressQL
* Configuration tools eg. Ansible, Docker

# Data Sources

Sample test data will have to be improvised according to the evolution of the project. The reasoning for this is that it is always impossible to speculate how the final database structure will come together, so as it evolves with the project so will the test data. No alternatives for the test data will be discussed nor will they be necessary; although this is a practical project it is also an Open Source one as stated earlier, which means the data cannot be limited to just one organisation (in this case Kingston University).

# Use of Technologies

I will focus on an Amazon (Free Tier) virtual machine providing these functionalities and in particular the optimal / custom alterations of the machine's OS' specifications such as RAM, Hard Disk space (as well as the procedure of adding an extra virtual drive known as an Elastic Block Store (EBS) to the virtual machine and so on.

This project proposal aims to outline all objectives involved in the technical backend aspects of a practical website while aiming to minimise the associated costs. The objectives involved include setting up and managing a database environment (planning to use SQL Developer or similar), creation, configuration and connection to Virtual Machine (VM) instances (through Amazon Web Services), reservation of domain name through GoDaddy and its assignment to the UK registry.

Additional objectives regarding connection to the VM instance via an SSH client (PuTTY in this case) include a series of steps provided by Amazon on the "connection" window, and include implementation of private key files (both .pem and .ppk extensions) which provide a secure connection between PuTTY and the VM.

# Context

The significance of this work to my study of the Computer Science course is of utilising my existing skills to create a unique system, unique in a way that it differs from my usual projects such as coded applications for external theoretical companies. At the same time, it will benefit the students of Kingston University through offering their lecturers insight as to which teaching methods / techniques are most effective, and is open for adaptation in the foreseeable future. The only relation to previous work is the implementation of the database hierarchy, the majority consists of working with new cloud-based technologies and implementing the services like Virtual Machine management they offer. My existing knowledge of Virtual Machines and the like however, will be fully applied to the construction of this database-backed website.

By the completion of the website, there will be situations where problems may still arise and will need to be investigated promptly; for the purposes of the final report these will be investigated ahead of time. If for example there is a case to recreate a VM instance, then resolution via the "Elastic IP" concept and ip reset for the domain name on Godaddy will need to be addressed. My hypothesis aims to test whether these problems can be resolved within a reasonable time that, theoretically, would not affect development of the main project.

# Ethical Considerations

The system will use Open-Source components as much as possible and will always acknowledge the source of any Open-Source software. Other resources, such as the Amazon Free-Tier, are encouraged for student use. For these reasons I see this system as having no ethical conflict, in terms of how it will be put together. Regarding the proposed application system "Uni-Hood", if it does actually get exposed to the community for use then this potentially raises questions of privacy and confidentiality of the users involved. If I succeed in getting the system to a stage where there is a genuine community of users, it will be necessary to get professional advice on whether there are any concerns here.

# Data Protection

A feature of my method and an illustration of the modern approach to this type of project is that all code and documentation will be committed to GitHub, giving in effect a high confidence level that not only all work is easily accessible from the cloud, but a full audit trail will always be available. In order to protect from the extremely unlikely scenario of a catastrophic failure of the GitHub ecosystem, the system itself being built will work from its own local Git repository, which is a clone of the GitHub repository. In order to demonstrate the database backup and recoverability concepts, I expect to use Amazon's S3 facility which is an extremely reliable data storage resource.

# Intellectual Property Rights / Access to Code

All of my own work and code will itself be Open-Source and visible to any GitHub user, and I will not be duplicating any similar work, hence I do not believe there are any issues regarding intellectual property rights. As is the convention with GitHub projects, I am allowing my work to be publicly available and freely fork-able by these users, with no limits in place such as freedom of distribution.

# Methodology

The deliverables for this project were a series of instructions covering a variety of technical components, where the treatment of each technical component involved applying a specific methodology covering the evaluation, selection and ultimately the documentation of the component.

In more detail, this methodology involved proposing a certain technology / technologies (e.g. the choice of PostgreSQL as a database), utilising that technology to achieve a certain aim in a phase for the project (e.g. a reliable database technology has been made available) and if successful, noting this down as a preliminary step under a separate section in the How-To document. To verify that this section could achieve the same results each time, these preliminary How-To steps were repeated and tested several times. If the same result was produced then the next step would be constructed in a similar way, in relation to the next part of the current aim. If however the results were different, then this step was reviewed and modified on a trial-and-error basis until it was “stable”.

As the project developed this verification procedure was found to be in fact not so straightforward, as it was found that on repeating steps in subsequent sessions, the “proper path through” for one day would not necessarily work for another. This is because assumptions about the exact technical context or false memories about the exact sequence of prerequisite steps played a significant factor in the invalidation of previous documentation.

The problem was addressed by ensuring that as many documented procedures as possible, even the earliest ones, were re-tried at later stages of the project, under constantly evolving technical conditions.

Ultimately when an entire section in the how-to document was completed, the steps undertaken to reach this phase in the project were reset to its initial state (e.g. in the case of PostgreSQL via termination of the entire EC2 instance, thus requiring re-installation of PostgreSQL when re-building the instance). From this point, the steps would once again be followed to observe whether the end-goal could still be achieved.

# Analysis

Project analysis involved the use of forums, help pages on the web and advice from friends and family members to start investigating available technologies and procedures. I considered technologies including GitHub, MySQL, PuTTY, PHP, Python, Ruby / Perl, HTML5, CSS, Javascript, Bootstrap, Angular, JQuery UI, mySQL, postgreSQL, Ansible and Docker, some of which were utilised in the project. The Amazon services EC2, S3, EBS and Elastic IPs were all implemented into the project as utilisation of the wide range of Amazon technologies was the main direction I wanted to head into for this area. I intended to make full use of their “free tier” procedure / offer, though through easy mistakes regarding instance up-time this was unfortunately not achieved.

This was essentially an “agile” approach in that I chose to investigate tools and make quick assessments on which ones to deploy, without performing a very exhaustive research on the merits of each. I continued use of a tool or resource if it could be quickly and easily deployed to achieve an end, however when this did occur I made notes on how each such tool was deployed. This resulted in the recording of sufficient material to turn these notes into fully elaborated How-To’s.

## Purchasing of the Domain Name

Several titles for this project were considered far before any implementation was done, though once UniHood was decided and searched for to verify it was a unique name the next step was to “morph” it into a domain name format, and purchase it. GoDaddy was the main go-to here, so a GoDaddy account was created and the domain name of uni-hood.co.uk was bought for a very cheap price of one pound.

## Amazon

The aim to minimise the costs associated with the project was at the top of my mind when setting up a new Amazon account which would be linked to the Amazon Web Services. After some research, I discovered there was a “Free Tier” period that is included in the creation of a fresh account, so I could conveniently utilise this instead of my current Amazon account. To distinguish this from my main account, I felt it was necessary to link it to a fresh Gmail account also.

After linkage between the Gmail account and Amazon account was completed (through simply providing it as the email address during sign-up), I started the process of linkage between the Amazon account and the Amazon Web Services. It is a requirement of the AWS sign-up process to link this account as the services will be tied to it. The sign-up process does not provide you with full access to the AWS services right away, as it is seen as a kind of request form for which you will need to wait on a response from and/or keep in contact with Amazon themselves regards if and when they allow you to use these services.

This took longer than expected however, as there was some small issue between my bank (Barclays) accepting or managing the small transaction used to verify this linkage. I contacted Amazon support and started this as a small case, for which after a few message exchanges was resolved very quickly. I was very impressed with the speed and quality of support from Amazon, and it helped me proceed with little disruptions to the actual creation of the Unihood virtual machine. I have elaborated on this aspect in the How-To document under Amazon “Free Tier” Account Creation”.

It may take some weeks, though when the AWS are linked to your account this free tier program will then be available for use. It will greatly help minimise the costs when using (in this case) virtual machines. Options which are eligible for this are indicated clearly on each step of the instance creation process, so there is strong assurance that the right choice is being made cost-wise.

## Amazon EC2

At this stage it was necessary to choose an “AMI (Amazon Machine Image)” namely “Amazon Linux”. I realised under later experimentation this behaved very similar to the Centos 6.X version Operating Systems. I wanted to investigate how to manage large amounts of data as a side-objective in case I needed to utilise this later, which led me to the experimentation of EBS Volume creation, discussed further under the How-To Deliverable in the EBS Volumes section.

## Console Access

There is normally no “console” access as expected in stand-alone workstations or in VPlayer-type environments; alternate instructions were provided upon clicking the “connect” link for the instance. I needed to initiate a bash prompt connection onto the newly created instance instead as directed, discussed further under the How-To Deliverable in the Key Pair / Putty section.

The “StarUML” application was particularly useful here as it was much faster and easier than sketching the tables on paper in some respects, however each approach has their advantages. I was able to export this work as an image I could always refer to, while maintaining a top-down approach in that classes with no foreign key linkages were at the top of the diagram, and the rest below them. This made it very easy to start off with creation of the top classes and work downward, instead of having a “foreign key trap” of sorts in that a column depended on another column however that column depended on the one referencing it, or the existence of another table that could not yet be created due to a similar situation. Database design and physical implementation go hand-in-hand, in that it took me several iterations to perfect the structure of the physical database whilst changing the design diagram the whole time in sync. Once at a good stage however, it was ready to be utilised in the main web application later down the track.

## Physical Database Implementation

The PHPPgAdmin environment was my choice for the database to run under. Initially there was confusion regarding the postgres version to use (9,91,92,93, etc). In the end, I decided to go with the latest one (95) but later found serious compatibility problems with phppgadmin! A workaround was a quick edit to the “Connection” php service file, the link for which is in the Appendix.

Firstly however it was necessary to install the Postgres and Httpd packages on the virtual machine instance as well as some additional packages. The whole database management environment is included in these packages, so after starting them and appending /phpPgAdmin to the instance ip you will be presented with an interface to which implementation of the database diagram (from StarUML) can be applied. Creation of the database and tables through the interface is fairly straightforward, however logically you must pay attention to the usage of foreign keys and make sure they match the flow of the diagram. Once satisfied with the table creation and their related columns within the tables, test data can be inserted as rows and reviewed within a table’s overview page. You may encounter a scenario where only id’s are displayed instead of the actual piece of data they are referencing, so a more visual interface allowing foreign data to be recognised was desirable. My father’s open-source “Tables” facility for his current work database was adapted to account for my data instead, and foreign key values were correctly resolved to the values they linked to and visualised. All database data from the original phpPgAdmin environment, including the test data, can be exported to a “dump file” and this process is outlined in the How-To document under the “Backup/Restore Test Data” section. This file was then able to be imported into the new facility, as both environments were capable of recognising these SQL dump files.

# Design

## Schematic of System Components

(Make in Photoshop)

## Server hosting In House

I chose the Amazon cloud-based environment due to it being a very well known company; offering great customer service and reliability, a free tier program and having a wide range of virtualisation tools and file storage available to its users. The EC2 and S3 services are the most relevant for this project, and when combined can immensely strengthen a virtual Amazon instance. As an additional benefit, through hosting the virtual server in the cloud, it is both accessible anywhere.

## Choice of Amazon as a Cloud Provider

Amazon already has a very high reputation of offering great Customer Service as well as being a massive marketing platform. After having used their S3 service in particular their storage seems very stable and offers a large amount of data to be stored so it seemed a strong candidate to make use of this as a cloud system for my project. Linking the storage facility with the virtualisation environment resulted in a very strong cloud-based solution.

## Choice of Virtual Machine Hosting service

I chose Amazon EC2 because I was already impressed with solely their Amazon Simple Storage Service (S3) at first, after having experimented with it for a while for general purposes, so had no doubt the EC2 service would serve just as well. It presented me with a very simple interface, quick response times, great support and so on and only had the slight issue with linkage of these services to my account however these were resolved very quickly.

Choice of Source Control (GitHub vs Dropbox)

I chose GitHub as a repository for all screenshots, source code, version control aspects and documentation (issues etc) needs. I manage it locally using the Windows application “TortoiseGit” which allows me to easily set up a new Git repo, and have access to many Git features though mainly I simply work off the local files, then commit and push them to the master repository link on GitHub. I also on the web interface manage “issues” which are small paragraphs of current bugs, reminders or just general notes regarding the project. I also have the opportunity to revert back to a past commission if something goes wrong in the current stage.

Dropbox on the other hand was a great repository for note taking also, and all important notes and even Putty files and similar were encrypted using the “Boxcryptor” app, which means that if my Dropbox account was compromised the encrypted files could only be read through an additional layer of security through this app. Locally, it’s very easy for me to login to the Boxcryptor app and view these files on a separate virtual drive safe from the eyes of the public.

## Local access to server

I decided to use the Windows Putty application because I am already very familiar with using it to connect to remote servers through the SSH (Secure Shell) protocol and by default it provides an excellent terminal interface to work off. Linux, being a terminal-based operating system itself, works very well with Putty and tips for connection through it are linked to from the connection window on the Amazon EC2 Management Console.

## Database (Technology, Management tool PHPPgAdmin)

## OS (Windows VS Linux)

## Choice of Linux distribution (Amazon Linux)

I needed to broaden my knowledge of the Linux world, and having the general view of Linux as a powerful, user-friendly and pro open-source platform persuaded me to select this as an AMI for the Amazon Instance.

## Security (Approach, Confidentiality, Privacy, SSH Key Pair)

## StarUML Class Diagram

After a significant level of experimentation with the “StarUML” application through my previous University years, I was able to quickly apply my knowledge to this project, and through the click-and-drag nature of the graphical environment I was able to quickly construct a basis for the actual database structure. I then was able to export this as an image file and commit & pushed to GitHub, so I could always refer to it whilst implementing the database.

## Backup techniques (test data, use PhpPgAdmin frequently to export dump files, considered Ansible and Terraform etc)

Test data was unchanged across database variations, in other words it was used consistently and unmodified so as to work within a true testing environment. The test data as well as the whole database tables and their columns were frequently exported as dump files as mentioned earlier. Platforms including Ansible, Terraform etc were considered, though the simple exporting process seemed adequate enough for the project’s needs.

## Apache / PostGres configuration

## Docker consideration

The Docker Virtualisation environment was also considered, however from word-of-mouth it was apparently very slow and required quite a further level of configuration compared to that of the Amazon EC2 instances. The idea was abandoned however noted down as a suitable fallback in case there was some issue with Amazon (however very unlikely).

## How should I record information? Word .doc or GitHub .md files. Doc is such a standard, though for technical documentation I believe this should be in .md format.

Information contained in document files in the current folder was questioned as to whether their saved format should be in the standard .doc format or the special .md format (incorporating GitHub markup). I ultimately decided to stay with the .doc standard for general files, both because it is the widely accepted and expected format and it would take a while to write the markup for everything, and may not be worth the prevention of layout issues (a small chance for which .doc files may be slightly altered unexpectedly). For technical documentation I decided to stick with .md format though only the Readme satisfied this scenario.

## How to reliably reproduce runtime environment (research suggests that Docker is a strong solution to this, but as yet I have to learn Docker effectively)

## Choice of web-server (Apache, NginX ?) Apache because..

## Hack attempt prevention, disabling clear-text password login access.

Bind variables were used as a measure against hackers, in addition to the disabling of clear-text password login access. Although this is a private project, there is still a connection log file available for analysis if this ever goes truly public. This means a trail can be seen of all attacker attempts on brute-forcing (or other methods) for usernames and passwords, but with bind variables the protection against the possibility of SQL injection is also enforced.

## Similarities between Amazon Linux and Centos 6 (why Centos 6.x instructions should be followed)

Amazon Linux is in face based off RHEL (Red Hat Enterprise Linux) version 5, and some parts of version 6. They state now that “one of the goals of the most current Amazon Linux AMI (2013.09) is to be as compatible as possible with RHEL 6” as discussed in the link:

<http://serverfault.com/questions/798427/what-linux-distribution-is-the-amazon-linux-ami-based-on>

I felt it was safe to assume that at this stage Centos 6 and Amazon Linux are almost identical in terms of feel and functionality. Any instructions that were followed were followed in reference to steps outlined specifically for Centos 6, though in the rare case they would fail then the steps for Centos 5 would be followed as a fallback.

## Installation of PostgreSQL

PostgreSQL is a Database Management package and its installation is as straightforward as that of the Apache or phpPgAdmin packages (using apt-get or similar, outlined in the HOW-TO document under the “Installing Apache and PHP, and Managing the WebServer”. It supplies all the standard Database aspects including columns and their tables, a range of different datatypes, import/export functionality, multiple character sets and so on. When this is utilised within the Apache webserver context it allows a powerful way to put the Database diagram from StarUML into effect.

## Regular yum update to enforce preventing of vulnerabilities regarding security issues

It has been strongly recommended for a while now, and a reminder is always present, to run a “yum-update” command on a machine running a Linux OS. This helps to enforce the prevention of vulnerabilities regarding security issues that may have come about, due to flaws or loopholes in older versions of the installed packages.

## Looked into SQL generation from UML but assumed it was very complex / couldn’t find anything / wouldn’t work for my version

I tried to convert the UML Diagram from StarUML straight into SQL queries, however though I attempted to install certain add-ons, or find certain functionalities (different across StarUML versions) or whatever seemed necessary to enable this feature I could not manage to proceed. In the end I had to create these statements manually when required, as mainly the graphical interface already allowed creation of these properties.

## Benefit of public key usage with GitHub

In addition to my

## Considered enhancement of GitHub credentials

## Consideration of static (elastic) instead of dynamic ip

## Documentation of snapshots to accompany steps document

# Implementation

## Discussing How-To steps (Actual steps in appendix?):

Appendix A is the primary deliverable of this project work, and consists of a series of HOW-TO’s. These are designed to be simple, practical and readable instructions that allow a particular technical objective to be achieved, and yet do not assume a high existing level of expertise to understand and execute.

# Testing & Evaluation

* Succeeded in getting a web app up and running? (self assessment.. I reckon I did)
* (put the how-tos in front of somebody else! See if they can follow them !)

# Critical Review

Was my approach successful? Will anyone really benefit from my How-Tos?

Which design decisions would I do differently?

# Appendix A

<https://tech.enekochan.com/en/2014/04/11/fix-error-column-spclocation-does-not-exist-in-phppgadmin/>

(Append all How-To’s here, so they’re shown in contents section)