CI6300 – INDIVIDUAL PROJECT

# UniHood Report


Daniel Carnovale

**K1336511**


**22/04/2017**

# Contents

# Introduction

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 "HOWTO" 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 assuming the reader is starting from a standard Windows workstation with no specific tools pre-installed. 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 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.

Readers should be able to use these instructions to build a completely different application. It is my hope that many individuals and organisations would find the HOWTO deliverables of this project valuable because of the significant time-saving when starting a project using similar tools.

Appendix A is the primary deliverable of this project work, and consists of a series of HOWTOs. 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.

The next most significant deliverable is the running Web Application system itself at http://www.uni-hood.co.uk. This is a dedicated Amazon EC2 Instance which I propose to leave running for the immediate future.

The research on infrastructure, technology choices, installation and documenting the HOWTOs, repeating steps to verify their reliability etc. was a very large task and did not allow me to apply the level of web/database application programming that I have learnt from my Programming and Database courses. I felt it was more appropriate to focus on infrastructure and HOWTOs as this would provide more unique value in this body of work.

GitHub Issues were used as a means of documenting my research and any tips relevant to the HOWTO document or the topic area in general. These can be found at

https://github.com/darkquake93/unihood/issues for inspection (each issue can be clicked on and their comments, status etc. will be displayed).

## Literature Review

The paper “Cloud Computing’s Effect On Enterprises [1]” details the current and developing stage of Cloud Computing, the associated costs and of course its impact on enterprises. This provides an insight on the rationale for moving to cloud-based resources and hence a justification for our goals in this project.

The paper “A Web-Based geographical information system prototype on Portuguese traditional food products [2] ” discussing PostgreSQL on page 14 provides (while brief) a concise and informative outline of the PostgreSQL Database system and strengthens a beginner’s understandings of its capabilities in general.

“User Guide - PostgreSQL and phpPgAdmin - Powered by Kayako Help Desk Software [3]” is a useful guide to the web-based phpPgAdmin tool, which played a vital role in Physical Database Design and Implementation.

The definitive reference for instructions on creation and management of the web server instance is “Amazon Elastic Compute Cloud (EC2) Documentation [5]”. This is exhaustive and covers many advanced topics beyond the scope of this report. One of our goals is to demonstrate that the initial build of a project such as UniHood does not require such volume and detail.

In order to provide an easier alternative to such detailed reference resources, there is a long tradition of “HOWTOs” in the open-source community. These started as simple text documents, but are now more typically found as well formatted web pages and blog entries. A good example of a HOWTO that is directly related to this report is “HOWTO: Get started with Amazon EC2 [6]”. This is indeed a useful guide to the creation of an instance under EC2, however it seems to assume the reader is already working in the context of a local Linux machine and therefore has SSH functionality. It is also somewhat limited about creation of database services and subsequent steps.

In technical forums and blogs, many similar guides can be found, 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 any typical example may be out of date by the time a reader finds it. A similar caution should be applied to this report; in 10 months’ time a different tool set or procedure may be preferable.

# Analysis

## 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 HOWTO document. To verify that this section could achieve the same results each time, these preliminary HOWTO 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 HOWTO 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.

## Documentation Strategy

I had to make a decision as to whether the HOWTOs should be maintained in the .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.md file satisfied this scenario.

## Server hosting In House VS Cloud

A strong candidate for selection of the server was to set up and manage a Linux installation locally, and work against it as distinct from working against a remote cloud-based service. I opted for cloud-

based because it's easily accessible from anywhere and is not affected by hardware or availability problems with local machinery.

In terms of choosing a cloud provider, 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.

## Registering for Amazon Web Services

The aim to minimise the costs associated with the project was a priority 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 new account, so I could conveniently utilise this instead of my current Amazon account. To distinguish this from my main account, 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 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.

Regards the associated email, any email address can be used for making your AMAZON account (the AMAZON WEB SERVICES ACCOUNT is different), Gmail was my preference. Make sure that this email is valid and not previously associated with Amazon or there may be conflicts. I suggest as a best practice to create a new one for this purpose.

It took 3 days to resolve the above support issues, after which the Amazon Web Services were linked to my account and this free tier program was then available for use. It greatly helped minimise the costs when using 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.

I have elaborated on this in the "Amazon Free Tier Account Creation" HOWTO.

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

It was helpful to consider from the outset whether any newly created Amazon machine was vulnerable to any form of attack. I gathered advice on this and determined that we are protected

from this because Amazon Linux by default does not allow password-based SSH access. It is only possible to access the instance using a privately created key-pair.

## OS (Windows VS Linux)

I chose Linux as the development platform because it appears to be very much the leading platform for Cloud-Based technologies, especially as regards Open-Source components.

## Selection of Technologies

Project analysis involved the use of forums, help pages on the web and advice from colleagues 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, and I intended to make full use of their “free tier” procedure / offer.

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 HOWTOs.

## Docker consideration

The Docker Virtualisation environment was also considered, however it required 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).

## 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.

## Access from Local Workstation to Remote 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.

## Choice of Linux distribution (Amazon Linux)

I felt it was safe to assume that at this stage Centos 6 and Amazon Linux are similar 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.

Amazon’s “Free Tier” service allows this Operating System to be used for free or very low-cost during the first year.

## Technologies Utilised

- 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.

| Component | Choice |
|---|---|
| Cloud Service Provider | Amazon Web Services (AWS) |
| Database Backend | PostgreSQL |
| Linux Distribution | Amazon Linux |
| SSH Client | PuTTY |
| Programming Language | Perl |
| Javascript Library | JQuery |
| CSS Support | JQuery UI |
| Data Design Tool | StarUML |

Additional resources:

- GitHub for Source Control and as a Cloud-Based repository of project documentation. This is a free service for Open-Source project work.
- PostgreSQL for database back-end
- PuTTY as an SSH client
- Programming language Perl.
- HTML5 / CSS / Javascript for browser-side programming

Javascript UI Framework - JQuery UI

## Selection and purchase 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.

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

Every time an instance is restarted a different IP number is given, which is inconvenient yet free. Ultimately, a small fee is required for a single elastic (static) IP number which is then assigned permanently to your domain name using your DNS provider management settings (e.g. Zoneedit).

Obtaining an elastic IP address was a very simple procedure as well as its association with the instance. These are both covered in the HOWTO document under the "Creating and Associating Elastic IP to Instance" section.

This procedure is completely free as it is a single Elastic IP attached to an instance, and Amazon states "You can have one Elastic IP (EIP) address associated with a running instance at no charge." This reference also details all the pricing for standard instance uptimes, outside of the Free Tier. "EC2 Instance Pricing – Amazon Web Services (AWS)," [4]. My finding here was that ultimately, to reduce costs, the instance should be kept STOPPED but not TERMINATED.

## Consideration of EBS service

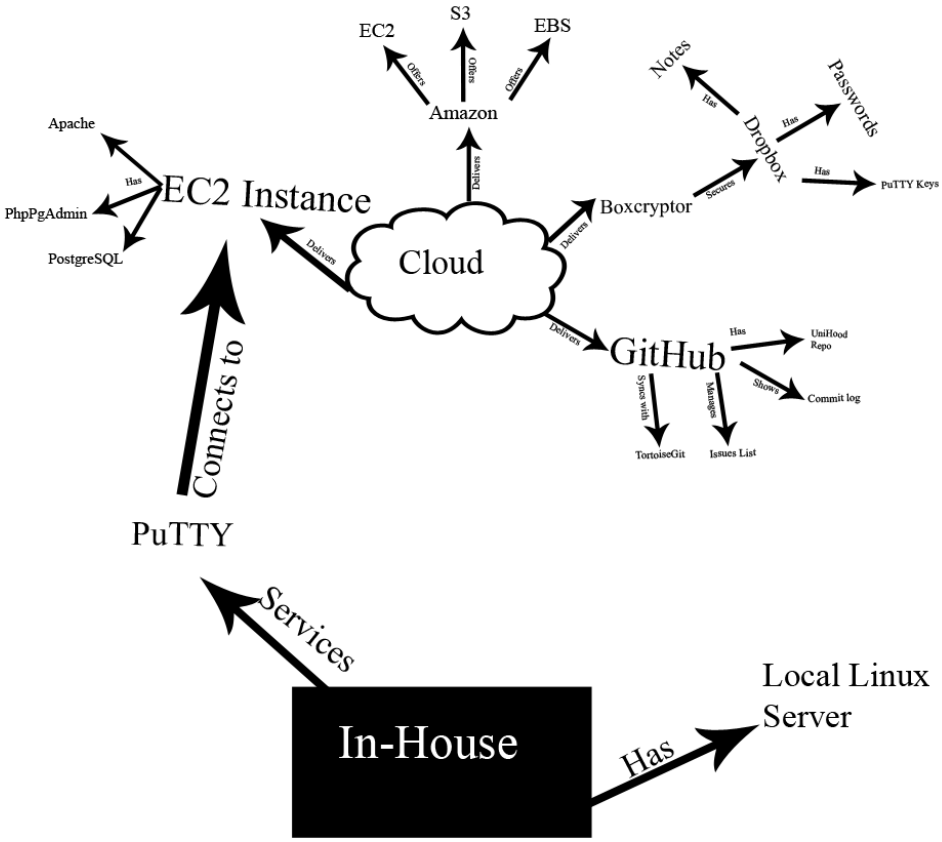
I wanted to investigate how to manage large amounts of data as a side-objective in case I needed to utilise this later, which led me to the experimentation of EBS Volume creation, discussed further under the HOWTO Deliverable in the "Creating / Removing EBS Volumes" and "Managing EBS Volumes" sections.

## Selection of Amazon Machine Image (AMI)

At this stage it was necessary to choose an "Amazon Machine Image (AMI)". The default choice is Amazon's own distribution of Linux titled "Amazon Linux". I observed under experimentation that this distribution behaves very similarly to the Centos 6.X version Operating Systems which I know to be a simple, clean, standard starting point.

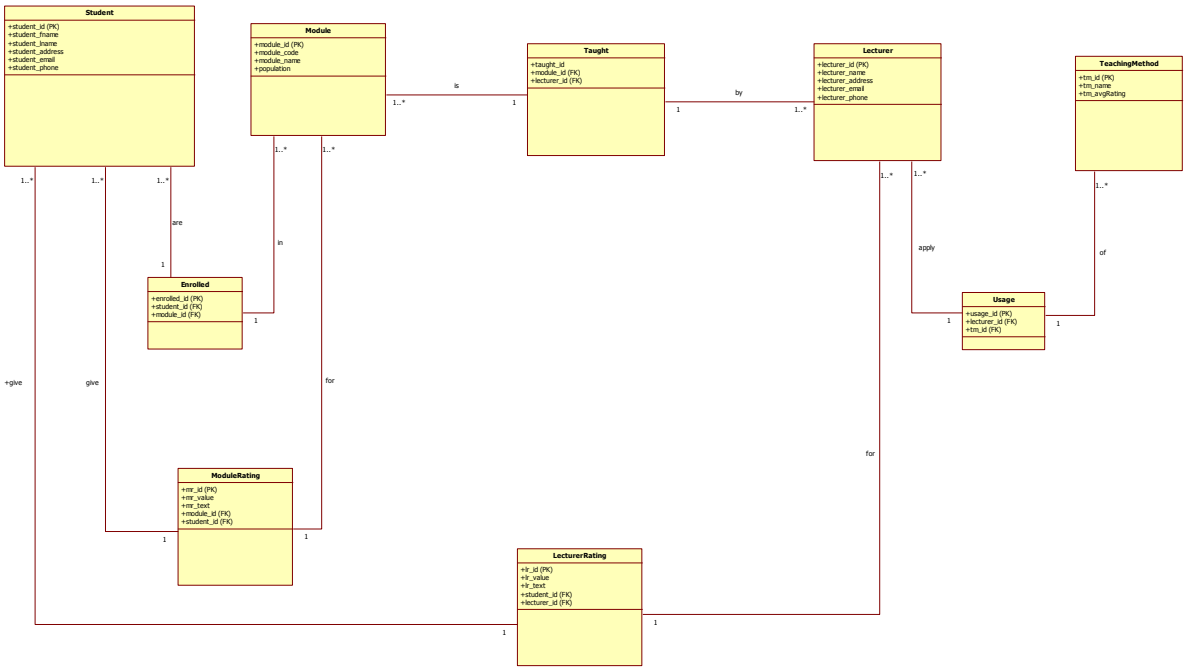
# Design

## Schematic of System Components



## 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.




## Database Design Strategy

To support the creation of this application, it was necessary to design an application database in the realm of the chosen functionality area (namely UniHood).

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.

# Database (Technology, Management tool PHPPgAdmin)

## Physical Database Implementation

The PHPPgAdmin environment was my choice for the database to run under. Initially there was confusion regarding the PostgreSQL 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 PostgreSQL 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. The open-source "Tables" facility 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 HOWTO 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.

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.

## Data Sources

Sample test data had 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 were to be discussed nor were they 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).

Once the database structure was set up, it was a simple process to insert the sample data through Structured Query Language (SQL) statements. SQL is very powerful when used correctly, and as mentioned above the data was improvised as there was no physical source of a student / lecturer list etc available, in addition to the high chance it would not be compatible with the current database structure. Hence when a query worked for a table, it was repeated with modified values. Finally, the open-source "tables" application was able to present this data in a more logical or meaningful way.

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.

# Implementation

## 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 HOWTO document under the “Installing and Configuring Postgres, setting up DB Owner” and “Installing Apache and PHP, and Managing the WebServer” sections. 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.

## Standard Connection Method : 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, however the easiest way is to simply identify and copy the instance’s IP. I needed to initiate a bash prompt connection onto the newly created instance which is discussed further under the HOWTO Deliverable in the Key Pair / Putty section.

## Optional Connection Method : Mobile Console Access

An extra convenient option for managing your AWS instance is to use the AWS Console Mobile Application. I found that sometimes the instance state is a bit sticky or completely doesn’t work using the desktop interface resulting in being charged for an instance I thought stopped. With the mobile application it is not only easy to review whether your instance is running but also (I found) have more reliability regards changing its state. Instructions on setting this up can be found under the “AWS Console for Mobile” section in the HOWTO document.

## Source Control from Windows : TortoiseGit

As a strong source-control mechanism, the GitHub facility can be used to record commits, issues and in general manage your work better regarding the instance. It is tied in with the TortoiseGit application which is available for Windows, and covered in more depth under the “TortoiseGit for Windows” section in the HOWTO document.

## Secure / Authenticated Command Line Access : Putty and PPK file

PuTTY is a Windows Application that can be used as a means of connecting to and managing your virtual Amazon EC2 instance, the procedure for which can be found under the “Connecting to the Instance” section of the HOWTO document. Once it is installed and your PPK file is available, it’s convenient to make the program Pageant startup and activate this file when you login to Windows. This tip and the global configuration of PuTTY and the PPK file are discussed in the HOWTO document under the “Setting up Putty and Encrypted PPK File” section. The preceding section “SSH KeyPair Generation” outlines the generation of the private key used in the setup steps. The passphrase you invent when following the steps will allow you to authenticate yourself one time only at the start of each Windows session (e.g. beginning of day etc.) and will subsequently allow seamless and secure connectivity to your EC2 instance without having to re-enter a conventional password at every login.

## Optional GitHub Management Tool : OctoDroid

Similar to the AWS Console app for Mobile (above) the OctoDroid app allows easy viewing and modification of your GitHub repositories and general user preferences. Your repositories are listed initially, and after selection several insights become visible. More information is found under the "OctoDroid for Mobile" section in the HOWTO document.

## Instance Management : Stopping and Restarting the EC2 Instance

Sometimes you will wish to stop and / or restart your Amazon EC2 instance so as to save costs, or even to re-initialise services or similar requirements. This is explained under the "Stopping and Restarting the EC2 Instance" section of the HOWTO document.

## Application Development

A complex application programming phase is beyond the scope of this project. To help kick-start what will become a fully developed web application, I obtained advice and assistance in the selection and implementation of a suitable database / web application framework. This led to the installation of the Perl "Mojolicious" library plus some plugins which allowed almost immediate web-based browsing and management of the relational database. Readers of this report could consider this library or similar offerings in other web application languages such as PHP or Python. In my own case, I am not yet an experienced web application programmer, but the strategy I adopted here has given great visibility to all of the setup work covered in this report.

"Mojolicious - Perl real-time web framework [7]"

## SQL Injection Attack Prevention

The delivered web app is clearly a database-backed application, and is the type of app which is typically subject to SQL Injection Attacks. This vulnerability can be completely prevented by ensuring that the application logic uses bind variables whenever constructing SQL Queries. The Mojolicious framework and its plugins have allowed all database access to be managed internally using bind variables and it was never necessary to write any database management code that would have constructed literal SQL statements.

## Testing & Evaluation

- Testing for this project falls into two categories:
- A) Do the primary deliverables (HOWTOs) achieve their purpose?
- As discussed previously in the methodology, I have a very high level of confidence that HOWTOs are genuinely useful, however it must be accepted that this is a subjective evaluation and can only really be determined by others.
- B) Does the Web Application demonstrate that the system being build is a truly database-backed system?
- Testing consisted primarily of using the generated web app to create and browse structured data relevant to the UniHood application. It was convenient to make use of phpPgAdmin on many occasions to both load up and review consistency of information presented by the web app. For usability testing I exposed the application to several users for sample feedback. The most significant message coming back from this form of testing was that while the system succeeded in presenting and maintaining database information, it did not have the actual processing functionality required to fully deal with the original requirements. This was an expected result because the web framework is a CRUD style (Create, Read, Update, Delete) database navigational interface.

# Critical Review

I found the PhpPgAdmin Database tool very useful not just in terms of managing a database, but in terms of seeing the project's structure and data design model in action and as a backup strategy in terms of exporting the SQL files. I would strongly recommend it to anyone studying a similar project matter, it was very much an invaluable resource.

I hope my approach for this project was successful in that the HOWTOs in particular will serve as a very informative and valuable resource for others also investigating the same topic area. Overall I believe this report will be very helpful for those who want to start such a project. I strongly believe anyone would greatly benefit from my "HOWTOs" document, it even helps remind myself where I went wrong or extra steps not immediately obvious.

Through easy mistakes regarding instance up-time the ability to execute an entirely zero-cost exercise was not achieved. As shown in (bil-1 to bil-3) for Amazon Billing per month (only two are shown for the usual amount I was billed over the previous months, and one from the current month), I wasn't maximising the benefits on the Free Tier usage. You can see the difference however when the price almost halves for the month of April (though not over yet, but I have paid attention to my instance's uptime.

If I were to re-do this project under different design decisions, I would make strong use of the Docker environment. I suspect it would actually allow me to save a lot of infrastructure setup work however it needs more research like any other platform before I'd be comfortable using it.

## Ethical Considerations

The system uses Open-Source and/or free components as much as possible. Resources such as the Amazon Free-Tier are encouraged for student use, so for these reasons I see this system as having no ethical conflict. 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.

## Intellectual Property Rights / Access to Code

All of my own work and code is itself Open-Source and visible to any GitHub user, and I have not duplicated 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.

## 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 is 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 is available. In order to protect from the extremely unlikely scenario of a catastrophic failure of the GitHub ecosystem, the system itself being built works from its own local Git repository, which is a clone of the GitHub repository. Regards database backup and recoverability concepts, note that even the test database dump file is managed and version controlled in GitHub.

# Future Directions for the Project

For UniHood to reach Production status, some or all of the following missing features are required. In the context of this report, this means that there would be opportunities to write HOWTOs for each of these areas because they are all common to most modern web apps.

## Regular System updates

Encorcing prevention of vulnerabilities regarding security issues has been strongly recommended for a while now, and a reminder is always present. It is necessary 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.

When this service enters the production phase, the plan would be to run a “yum update” via a Crontab job.

### Testing of HOWTOs

The HOWTOs have to be futher “field-tested” by publishing them and putting them in front of other people.

### Google Single Sign-On

It is essential for users of this application to be authenticated. Rather than implementing the traditional username / password scheme, it was my intention to adopt a Google Single Sign-On technique which allows reliable authentication of users without the application needing to see or manage their passwords.

“Integrating Google Sign-In into your web app [8]”

### HTTPS Certificate

It is necessary to purchase and apply an HTTPS certificate to ensure all data between user’s browser and the server is encrypted, as well as communication.

### Backup/Disaster Recovery

While our application software is already fully under Source-Control at GitHub, the database needs to be subject to a regular backup regime and a full disaster recovery procedure should be documented.

### Configuration Automation : e.g. Ansible

New tools such as ansible make it possible to automate some of the steps in these HOWTOs. It should be investigated to see if this has a positive or negative impact on setup complexity as documented so far.

### Scalability

One of the strengths of a Cloud-Based solution is that under massive load it should be possible to scale the number of instances accordingly. Procedures for this should be investigated and documented.

### Alternative Cloud Providers

It's possible that alternative Cloud Providers such as Azure, Mega and so on are easy to learn or are better value for money now or in the future. Ideally a similar project should be completed using these services.

# Appendix A: HOWTOs – As PDF

**Note: On spelling / punctuation I am following the classic UNIX / LINUX style of writing "HOWTOs", reference:**

**http://www.tldp.org/**

## Appendix B: Screenshots – As PDF

# Bibliography

[1] R.Saleem, 'Cloud Computing's Effect On Enterprises', Lund University, Jan-2011. https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=1764306&fileOId=1764311

[Accessed: 21-Apr-2017].

[2] Ye, Z., 2009. A Web-Based geographical information system prototype on Portuguese traditional food products.

https://run.unl.pt/handle/10362/2318

[Accessed: 19-Apr-2017].

[3] User Guide - PostgreSQL and phpPgAdmin - Powered by Kayako Help Desk Software [WWW Document] https://support.eapps.com/index.php?/Knowledgebase/Article/View/68/66/user-guide---postgresql-and-phppgadmin

[Accessed: 20-Apr-2017].

[4] EC2 Instance Pricing – Amazon Web Services (AWS) [WWW Document], n.d. . Amaz. Web Serv. Inc.

https://aws.amazon.com/ec2/pricing/on-demand/

(accessed 4.21.17)

[5] Amazon Elastic Compute Cloud (EC2) Documentation [WWW Document], n.d. . Amaz. Web Serv. Inc. URL //aws.amazon.com/documentation/ec2/ (accessed 4.17.17).

[6] Hull, S., 2012. How-to: Get started with Amazon EC2 [WWW Document]. InfoWorld. URL http://www.infoworld.com/article/2615510/cloud-computing/how-to--get-started-with-amazon-ec2.html (accessed 4.17.17).

[7] Mojolicious - Perl real-time web framework [WWW Document], n.d. URL http://mojolicious.org/ (accessed 4.21.17).

[8] Integrating Google Sign-In into your web app https://developers.google.com/identity/sign-in/web/sign-in