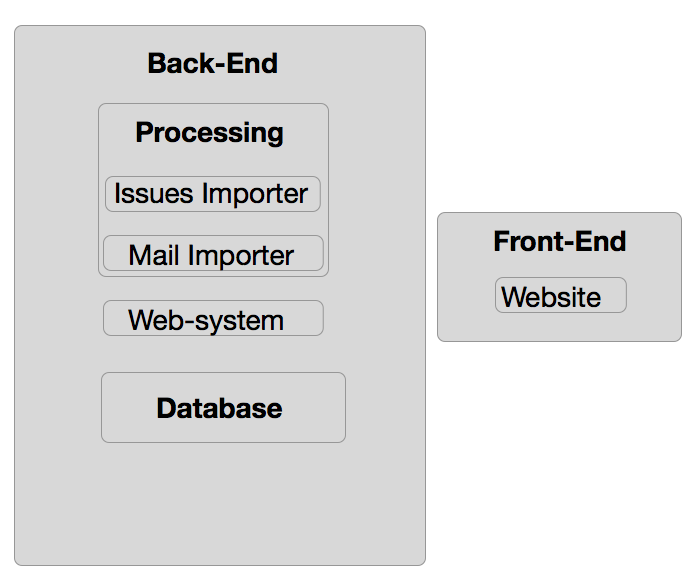
Architecture

# Introduction

The Optimus dashboard mainly consists of two systems, the front-end (website) and back-end (communications back and forth with databases and data).



The front end pulls JSON files from the back-end server and passes them using PHP and AJAX queries. Utilising bootstrap visualisation library and website template, the JSON data can be displayed graphically, allowing a fully customisable but seamless interface. This segmented design means that minor upgrades can occur over a long period of time, that will mean an eventual different product could potentially be created to serve the community at large

A singular script runs against the stored mail data, such that it parses and splits the information in to smaller manageable items which are then loaded into the database. The system starts by decompressing one month’s worth of mail data, splitting the file into individual parts and importing the data into an SQLite database table. This Ruby script traverses the files surrounding it looking for GZip files to decompress and import, but only if they are from the a year equal to, or less than the second command line argument offered.

A separate ruby method also runs against the entire Month’s mail archive in order to create a list of words and how many times they’ve been used. These word weightings are stored in a JSON file to be interpreted by the front-end webspace to create a word-cloud. The word-cloud will output a set amount of words, which are determined by the first command-line argument.

After the final file has been read it looks to another script that queries the database for information regarding number of emails per user, number of emails per subject, splitting the information up by year and month.

After decompression and parsing the GZip files are moved to a different directory to signal to the program that the file has already been entered into the database.

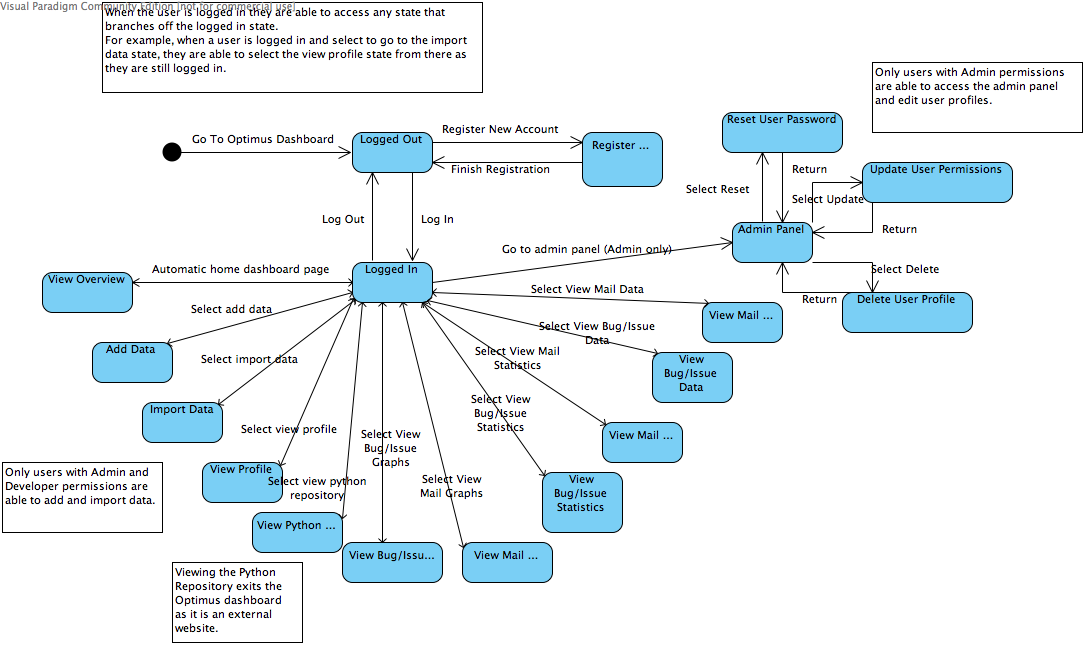
The modular nature of this design allows for easy additions, as well as new and improved queries being added to the files. Allowing modifications of the amount of information gathered by modifying the command line arguments allows future modifications and addition of data.

In the case of the Bugs import, Wombat web-crawler was able to pass over the site, and draw in the issues data, as well as the meta-data surrounding the users. Utilising ruby on rails, this data is passed to the front-end for interpretation and visualisation.

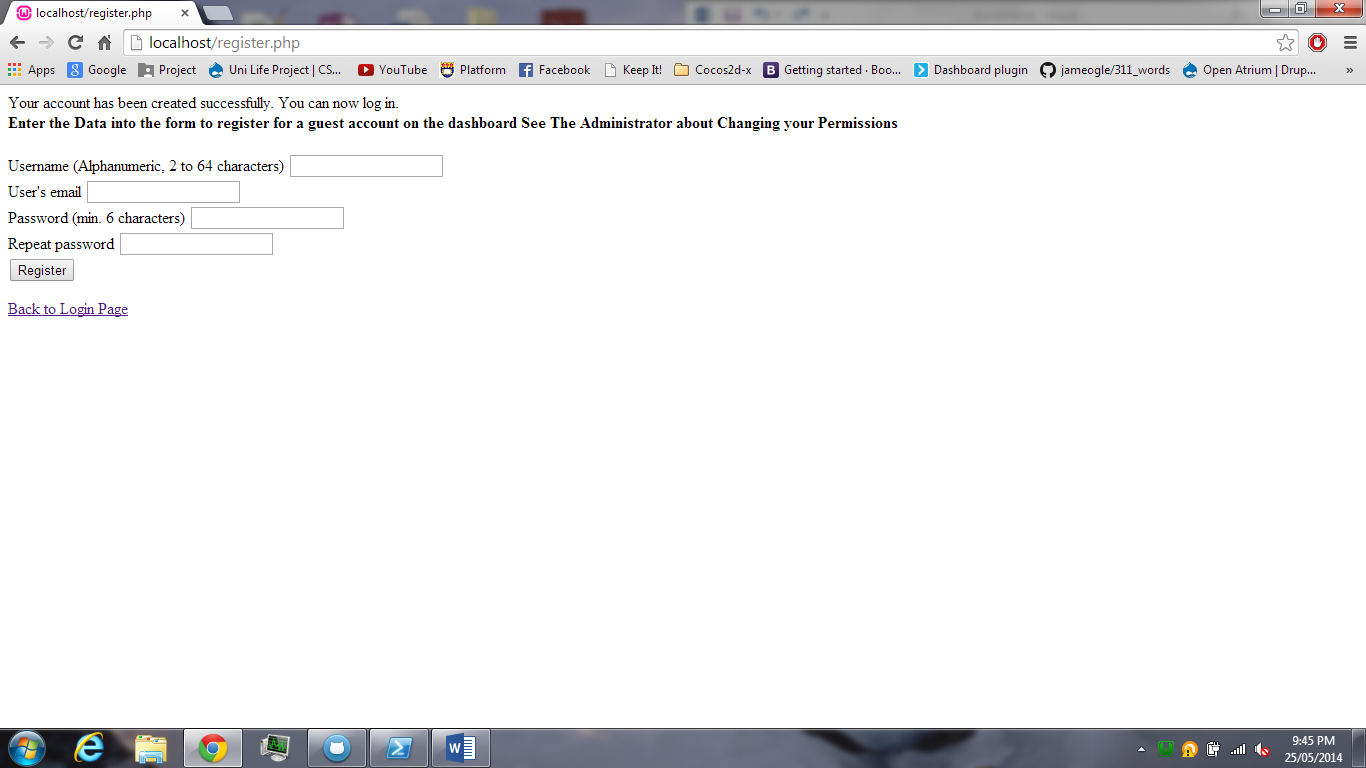
We have three classes of users with different permissions:

|  |  |  |
| --- | --- | --- |
| User | **Description** | **Permissions** |
| Guest | This is the regular user who just wishes to view data. These users are required to register in order to access the data. | Viewing data and graphs only. | |
| Developer | Developers are able to view data as well as adding data to their projects. They can also import a full list of stored data. | Viewing data and graphs.  Adding data.  Importing data. |
| Admin | An admin has the same controls as the developer as well as the ability to manage user profiles. | Viewing data and graphs.  Adding data.  Importing data.  Managing user accounts (reset password, update permissions, delete user) |

**State Diagram**

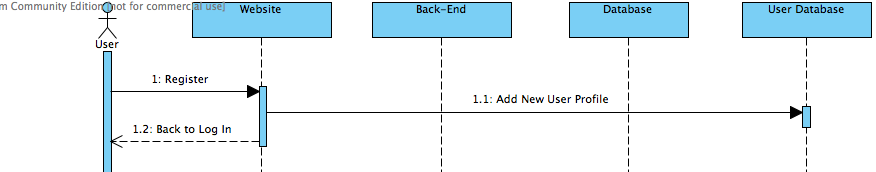
****

# Register New Account

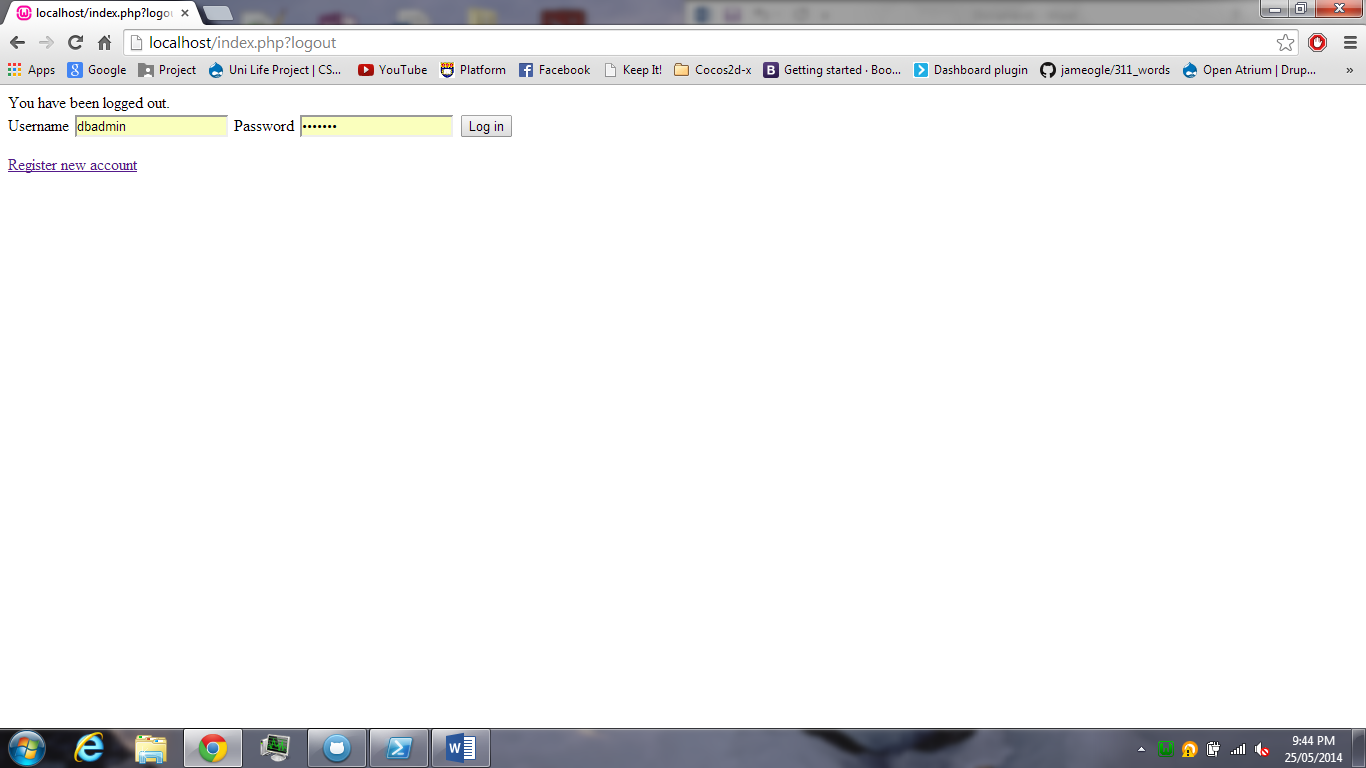


When you first arrive at the Optimus dashboard website you are prompted to register a new account. When selecting this option you are taken to this page that allows you to enter in a preferred username, your email and a password (along with password confirmation). This information is then sent to the user database which is where it will be saved. Once the user has registered they will then be automatically returned to the log in screen.

**Sequence Diagram**

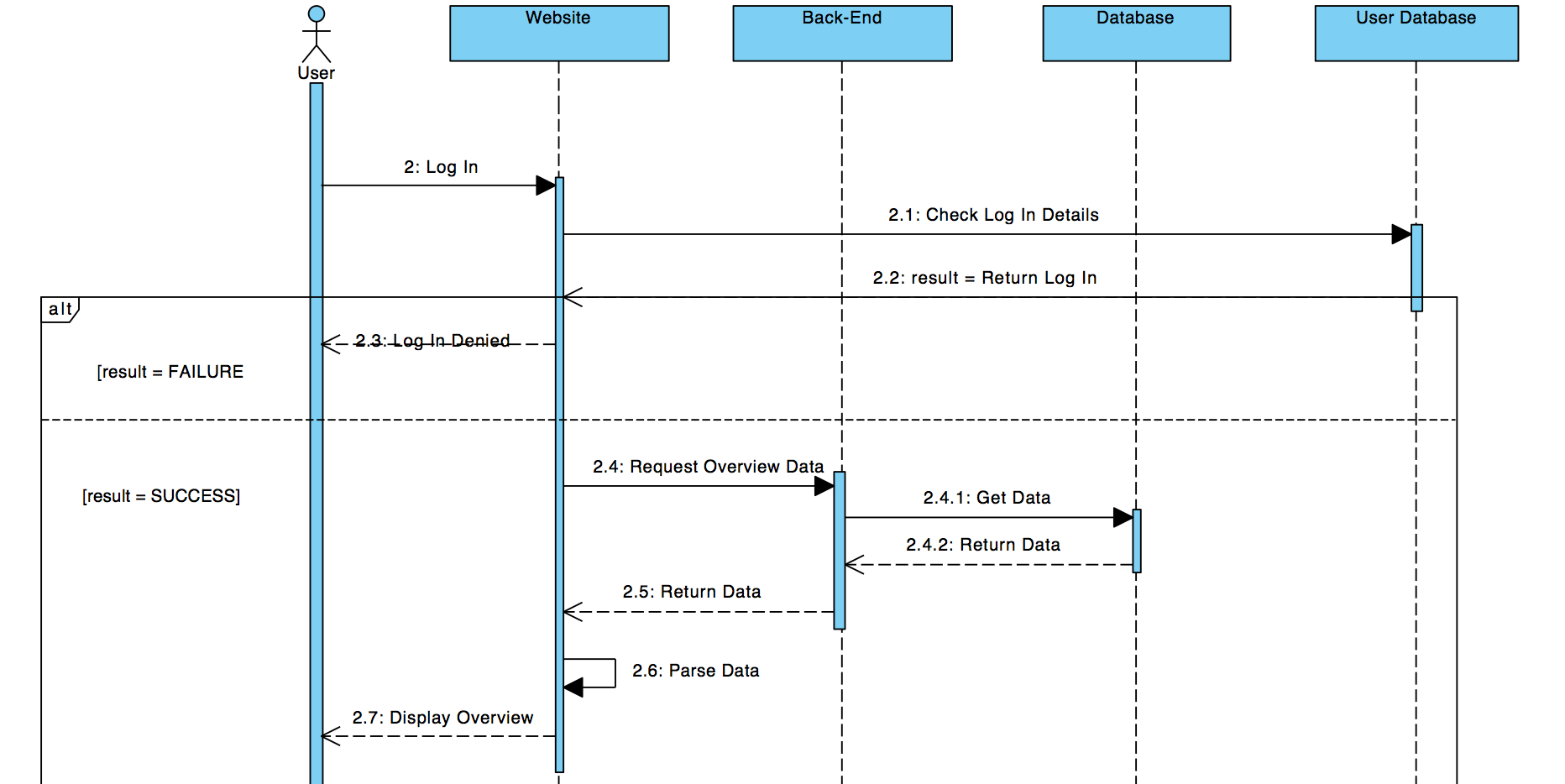


# Log In Page



Assuming the user has an account they are now able to log in to the Optimus dashboard by typing out their username and password (hidden). Then selecting the “Log in” button. If the username and password combination they entered does not match any profiles stored in the user database, then they will be automatically returned to the log in page. If the username and password combination is correct then the user will be granted access to the Optimus dashboard and they will be automatically taken to the home page, which provides an overview of all the data currently imported into the system. This data is taken

**Sequence Diagram**

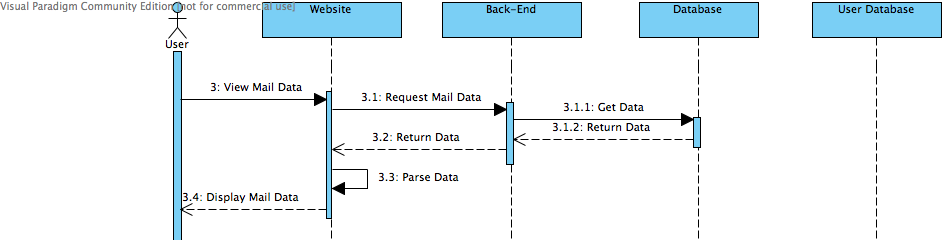
****

# Home (overview)

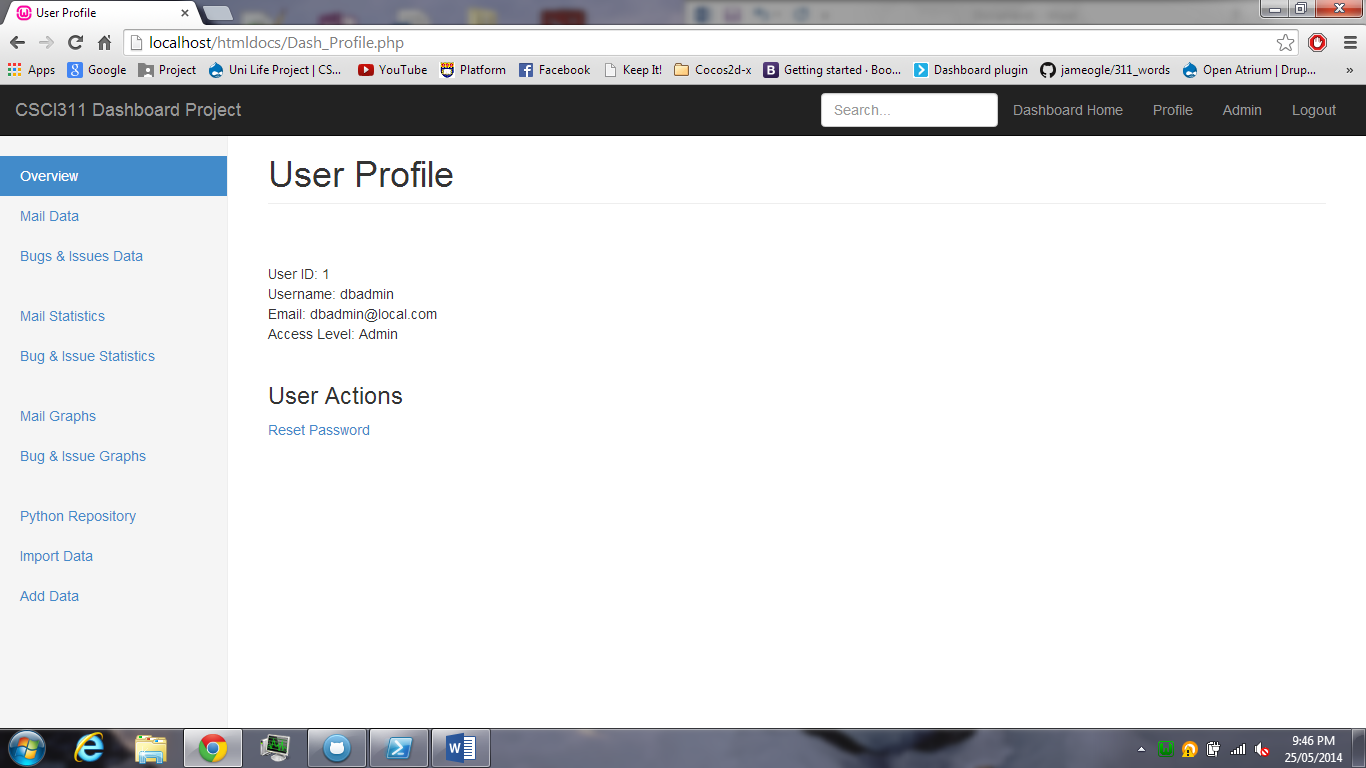
# 

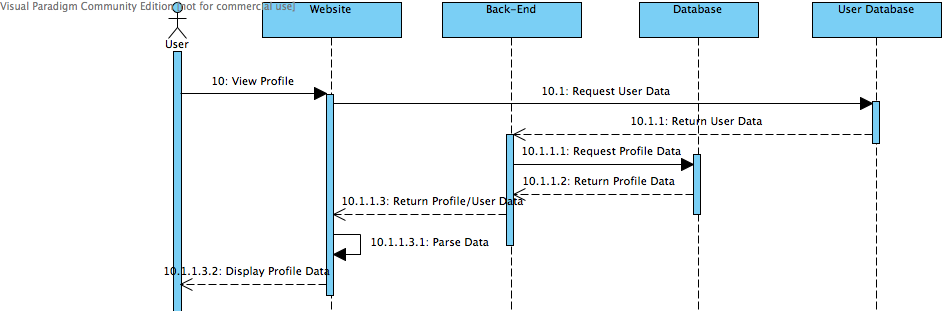
# Mail Data

**Sequence Diagram**

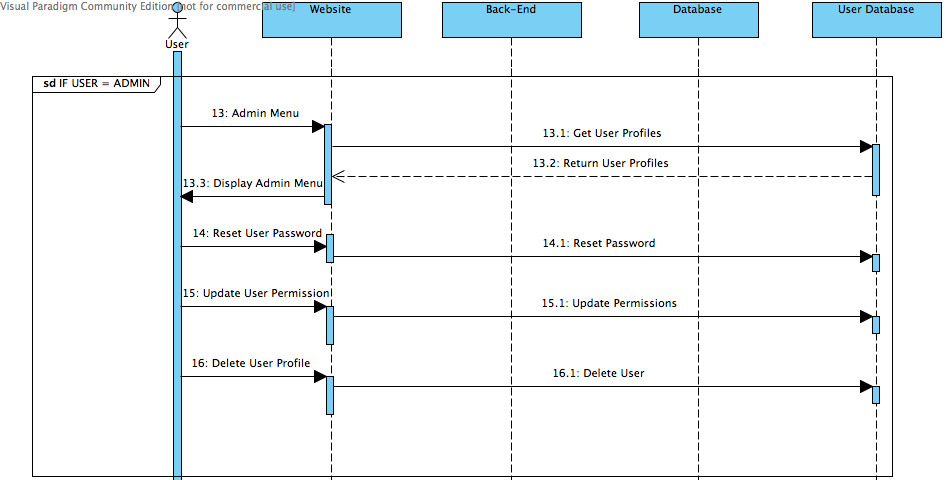
****

# User Profile

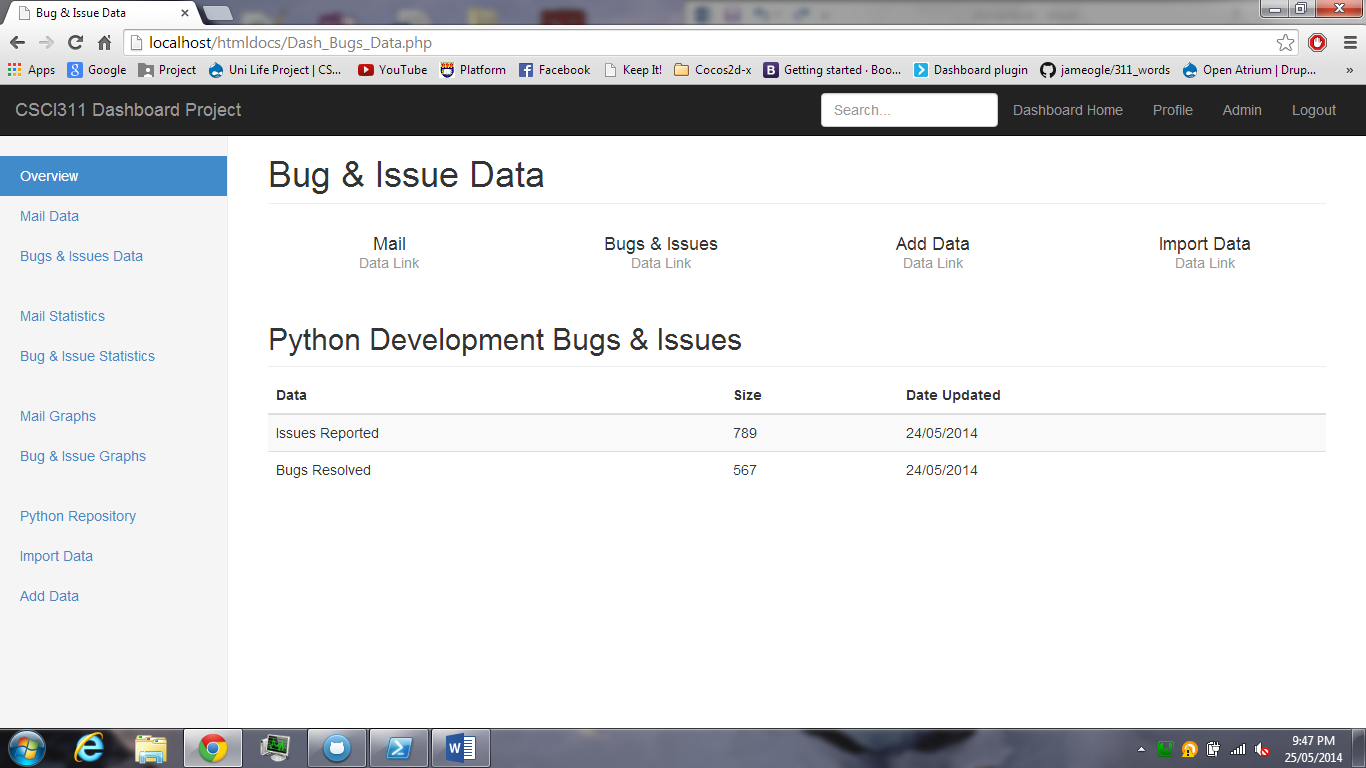


****

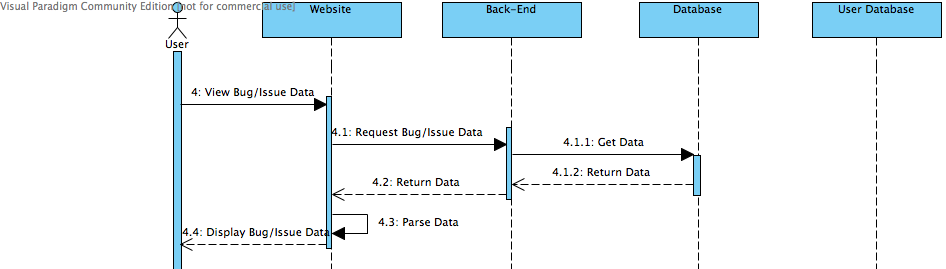
# Admin Menu



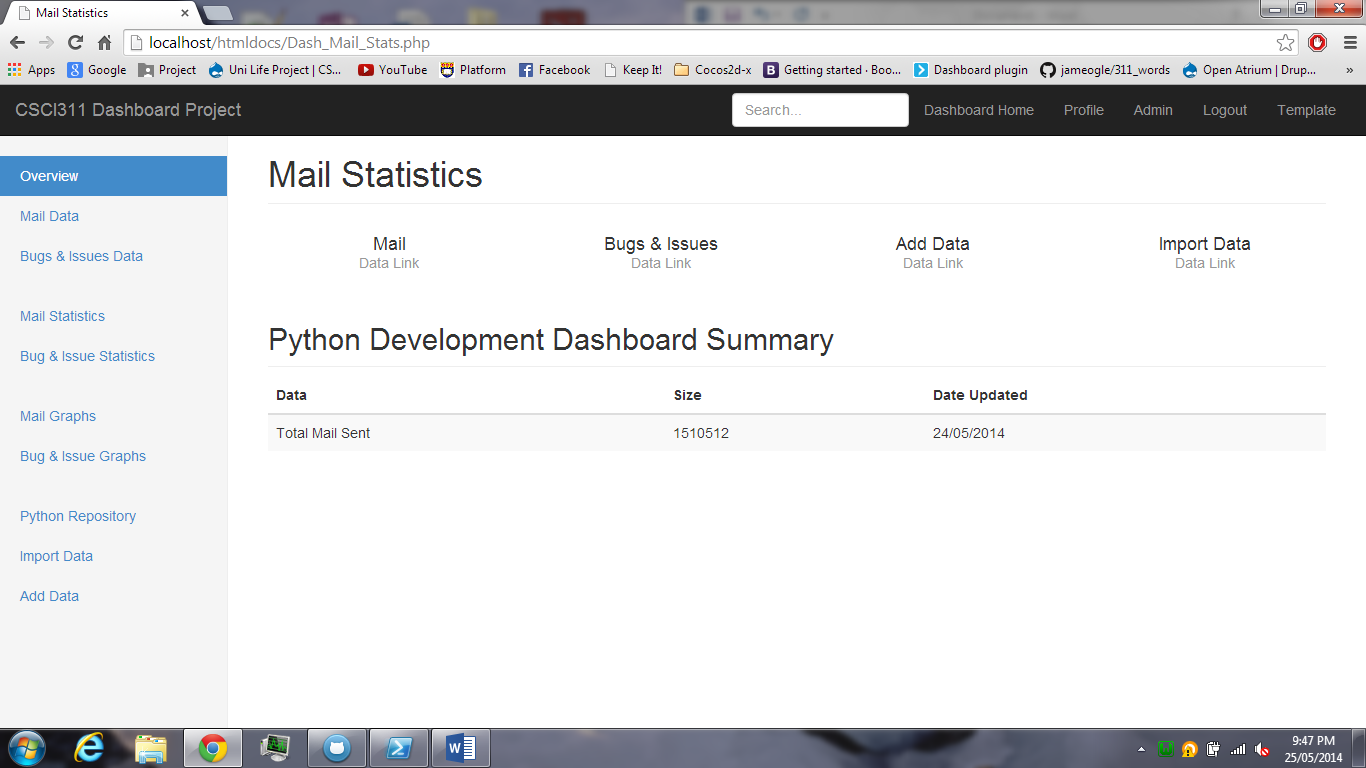
# Bug and Issue Data

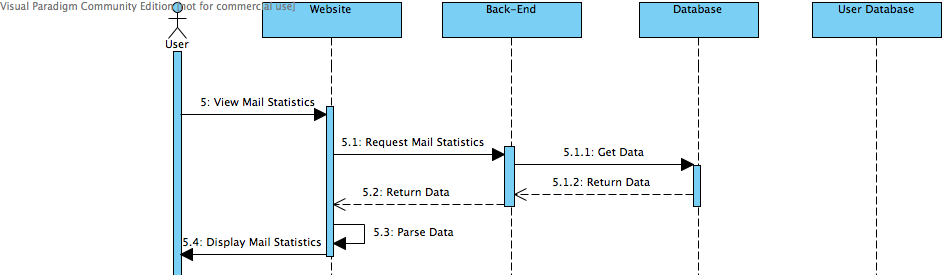


**Sequence Diagram**



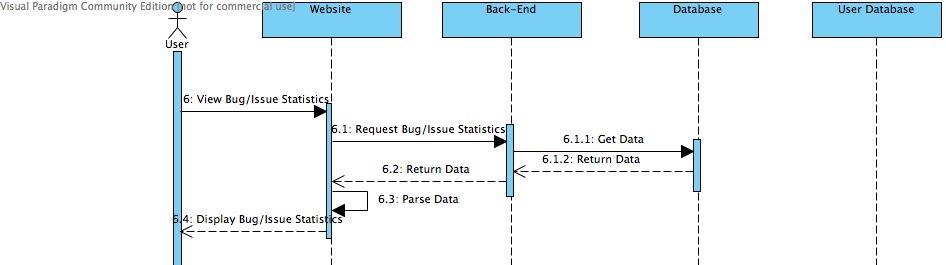
Mail Statistics



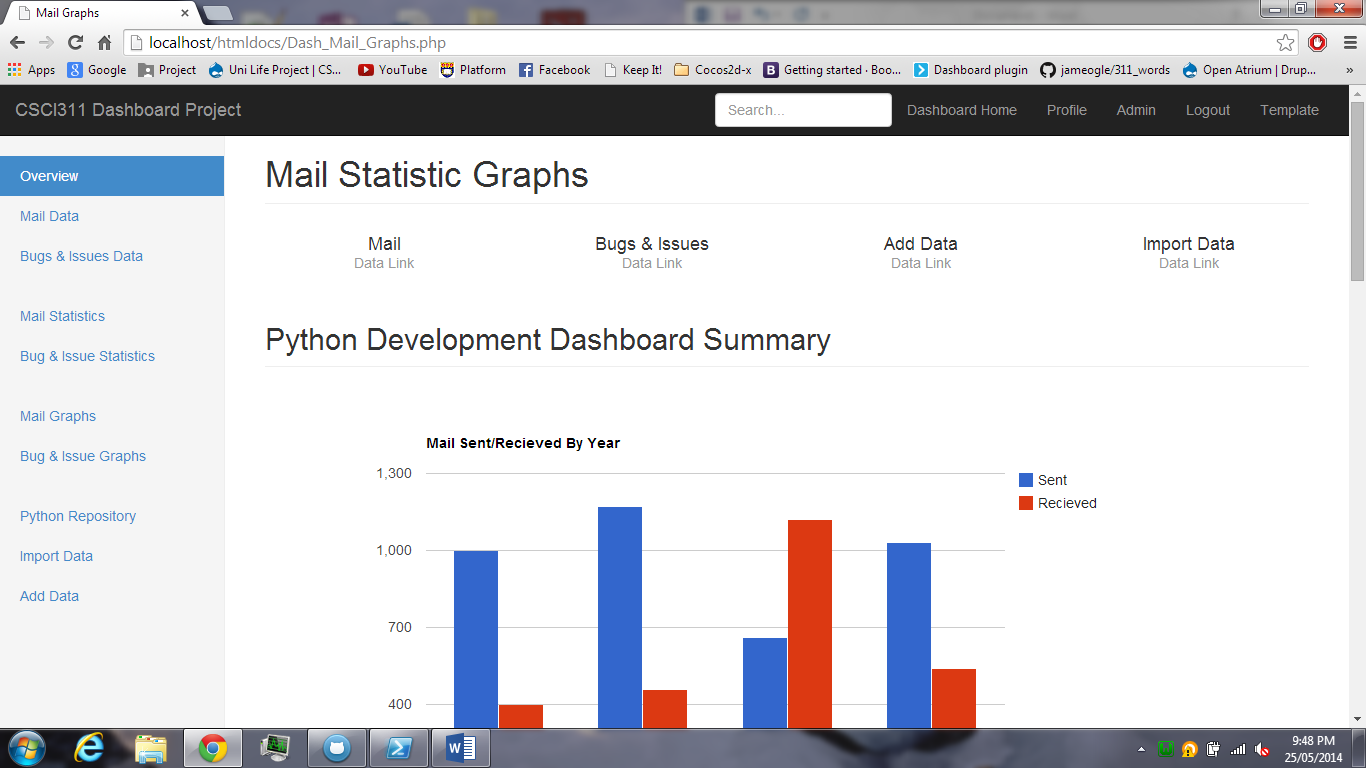
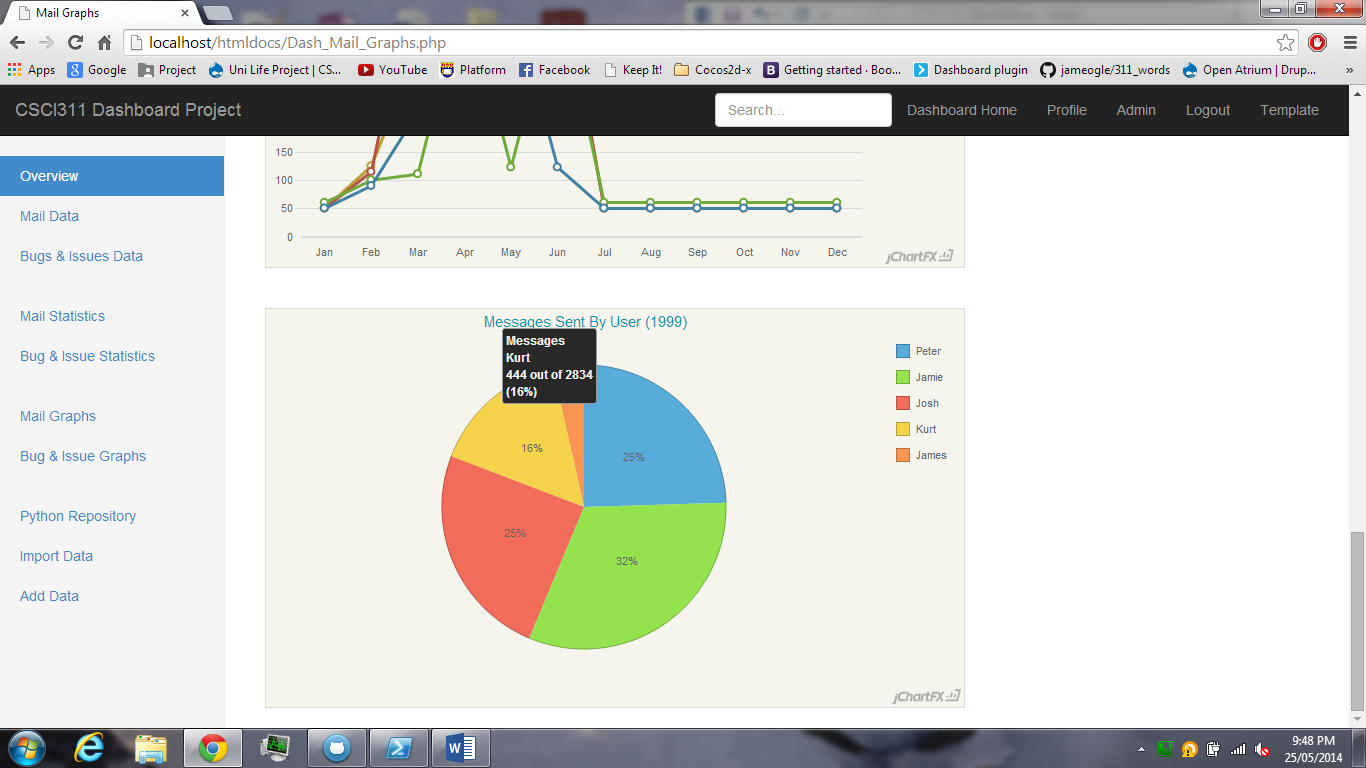
**Sequence Diagram**

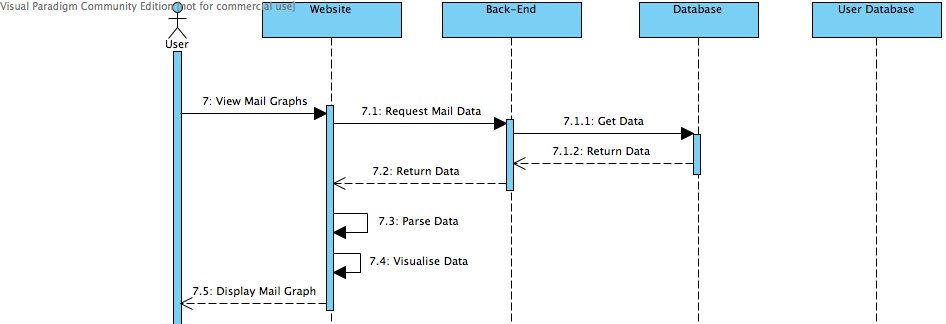
# Bug and Issue Statistics

**Sequence Diagram**

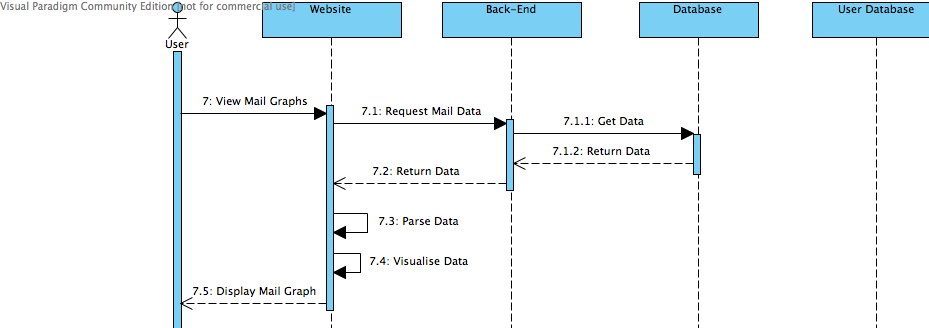


# Mail Statistics Graphs

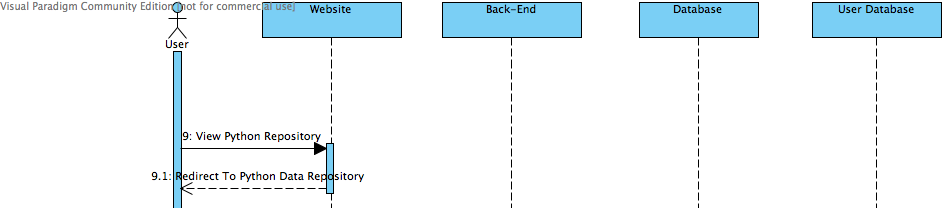




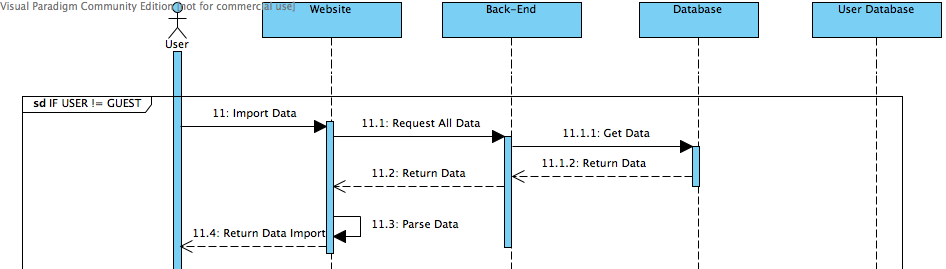
# Bug and Issue Graphs



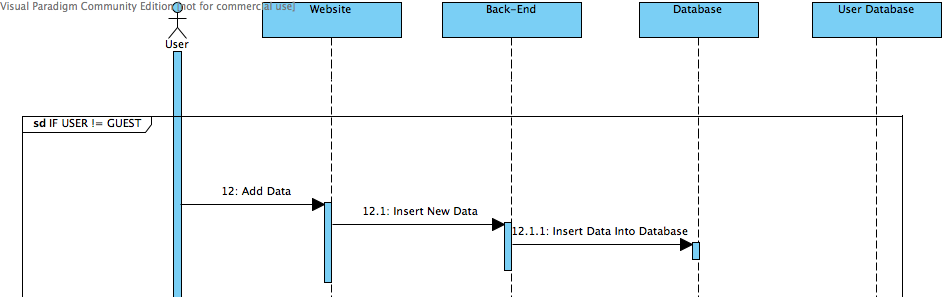
# Python Repository



# Import Data



# Add Data



Testing

Mail Parsing Test Suite

WordCloud Testing utilised a known test string and passed this to the function responsible for hashing the text and determining the ‘word weightings’ for use on a front-end graphical word cloud.

If the output expected resulted in the words found in the common words list and the main words correctly outputting the number of words in that string, then the test was considered passed and completed.

Mail Insert testing tested a number of elements associated with the storing of the mail metadata.

The ‘creatortest’ function tested whether or not the database could be correctly created to the schema required. It did this by utilising the create function (that the parser also uses) and testing whether it could insert test values. If an exception was thrown, the test was considered failed since it would mean the basic insert statement could not recognise the ‘mails’ table.

The Author parsing aspect of the test suite tested a variety of email formats, and utilised the function from the parser that would split the username away from the entire address. This address parsing tool would split it from addresses with and without ‘@‘ symbols as well as having the users full name in the address. a number of test values all with the same username were tested against, and if the resulting function did not provide the same result then the test was considered failed.

The final test was the insert test. This test was created to see if the data was correctly inserted into the table. utilising a fake email, the mail function would parse the text and insert the specific data into the table. This test function would then create the table, insert the values and then query the table. If the response did not produce the expected output, then it was considered failed and the program would then exit.