## Implementation

Now we get to the details.

* Describe your data structures and be sure to illustrate them with a diagram.
* If your user interface was a key feature describe how that was implemented.

**Figure 1:** An architecture diagram. Caption to go below figure.   
Note that LibreOffice handles this better than MS word



* Discuss the function of the most significant methods in each class. This may well require flowcharts, or sequence diagrams, in some cases.
* Any special relationship between the classes (e.g. friends) and why they exist.
* A description of any special programming techniques or libraries used.

## Program Validation and Verification

Tell us how you tested the system and why you believe it works. Describe the Quality Management Plan for your project, that is, software testing plan. The plan should indicate the types of testing that was performed and detail how they were done. This must include the reasons on why the chosen testing protocol was considered effective.

Create a table that summarizes the testing plan (see Table 1).

Table 1: Summary Testing Plan. A table caption goes above the table.

|  |  |
| --- | --- |
| Process | Technique |
| 1. Class Testing: test methods and state behaviour of classes | Random, Partition and White-Box Tests\*\* |
| 1. Integration Testing: test the interaction of sets of classes | Random and Behavioural Testing |
| 1. Validation Testing: test whether customer requirements are satisfied | Use-case based black box and Acceptance tests |
| 1. System Testing: test the behaviour of the system as part of a larger environment | Recovery, security, stress and performance tests |

Describe all the steps taken to validate the correctness of the program.

If you had user tests then say what you did and what the results were. Describe why these test data were chosen (what test conditions the data was testing). Table 2 provides an example of the sorts of results we are looking for. The full detail of the test runs should be appended to the report.

Table 2: Summary of tests carried out. A table caption goes above the table.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Set and reason for its choice | Test Cases | | |
| Normal Functioning | Extreme boundary cases | Invalid Data (program should not crash) |
| Preliminary test (see Appendix 3) | Passed | n/a | Fell over |
|  |  |  |  |
|  |  |  |  |

Follow your table of results with a discussions of them highlighting how useful and usable your system is for its intended purpose.

\*\* Also known as glass **box**, structural, clear **box** and open **box testing**. A software **testing** technique whereby explicit knowledge of the internal workings of the item being **tested** are used to select the **test** data. Unlike black **box testing**, **white box testing** uses specific knowledge of programming code to examine outputs. ?? so assert Statements ??

* 1. **Implementation**
* Discuss the function of the most significant methods in each class:

All classes in alumni.models inherit object-relational mapping in their constructor methods.

Classes in alumni.views generally return a rendered html template with object information passed via a context dictionary.

* Any special relationship between the classes (e.g. friends) and why they exist.
* A description of any special programming techniques or libraries used:

The Alumni Network was built with the Django 1.8 framework. Django is an open source web-based python framework, which mostly follows the well-established Model-View-Controller (MVC) pattern.

There are three critically important python files required to develop an application in the Django Framework (disregarding .py files for things like settings and test for now) - models.py, views.py and urls.py

Django comes with an object-relational mapper, allowing for the creation and maintenance of database tables which correspond to python objects in a **models.py** file. (This is the Model component of the architecture described above.) This orbject-relation mapping, as well as other features, is implemented by having custom models inherit from django models.Model class.

The **views.py** file (which serves as the ‘view’ in MVC) provides a way to process HTTP web requests within a web templating system.

A **urls.py** file provides a regular-expression based URL dispatcher, this serves as the controller in the MVC pattern.

This framework makes extensive use of type introspection ‘behind the scenes’ (see the Python function’s ‘dir’, ‘type’, ‘isinstance’ and ‘hastattr’ for examples of introspection. )

Further documentation on the framework is available here: https://www.djangoproject.com/start/overview/

**1.2 Program Validation and Verification**

**Class Testing:**

asdsad

**Integration Testing:**

asdasd

**Validation Testing:**

Customer requirements were tested via ‘client’ feedback. In addition to the required demo (i.e. the prototype / stage 3), our group did an additional demo prior to the project deadline. This feedback was directly related to the customer requirements and was instrumental in making feature edits to better satisfy client requirements. Although this method suffers from a lack of automation, it was chosen because it is a highly reliable way to draw attention to the most salient points with regards to customer requirements.

**System Testing:**

Asdasd

Table n: Summary Testing Plan – Alumni Project

|  |  |
| --- | --- |
| Process | Technique |
| 1. Class Testing: test methods and state behaviour of classes | Random, Partition and White-Box Tests |
| 1. Integration Testing: test the interaction of sets of classes | Random and Behavioural Testing |
| 1. Validation Testing: test whether customer requirements are satisfied | Client Feedback |
| 1. System Testing: test the behaviour of the system as part of a larger environment | Recovery, security, stress and performance tests |

Table 2: Summary of tests carried out. A table caption goes above the table.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Set and reason for its choice | Test Cases | | |
| Normal Functioning | Extreme boundary cases | Invalid Data (program should not crash) |
| Preliminary test (see Appendix 3) | Passed | n/a | Fell over |
|  |  |  |  |
|  |  |  |  |