**COMPX-374  
MEDIC2MEDIC Project – Section 2  
Final Report  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Group L – Natch Sadindum, Tristan Brynildsen, Shashank Mylarapu, Taran Kern**

# Project Brief

The Medic to Medic Project is a web-based application designed for the Medic2Medic company and will provide a variety of services listed below.

**Prospective Student Management**– Collection and organisation of program applicant data. Functions include:

|  |  |
| --- | --- |
| Creating and filling application forms by student users | Managing/organising completed application forms |
| Viewing/modifying prospective student data | Manually adding prospective students |
| Retrieving/saving prospective student data |  |

**Graduate Data Management** – Collection and organization of annual survey data taken from graduates. Functions include:

|  |  |
| --- | --- |
| Adding and completing surveys | Restrict access to staff users only |
| Editing, viewing, retrieving and saving surveys | View and modify graduate students |
| Retrieving graduate students from database | Sending surveys |

**Donor Management Software** – Linking and allocation of students to donors and providing information to donor accounts

|  |  |
| --- | --- |
| Adding donor users | Send newsletters to donors |
| Edit donor user data | Allow donor users to log in |
| Create/edit newsletters | Allow donor users to donate money |
| Save and retrieve newsletters from database | Save/retrieve donor users from database |
| Allow donor users to modify their details |  |

Each of these functional requirements were set out in the original brief and have been successfully implemented in our program. More detailed descriptions and diagrams are in the Final Design/Architecture section of this report.

The program is intended to be used by designated staff users and untrained donor users and students alike. It should be designed with a simple, easy-to-use interface and should be designed to streamline completion of tasks for all user types.

The program and its associated database should be able to handle multiple staff, student, and donor users concurrently without major faults.

The database containing all data must be stored securely and have access restricted to only staff users and with limited access to data by donor users. Additionally the database should be able to be expanded indefinitely to allow for more student data to be stored as the client loads increase over time.

The program should have been designed and implemented in a way which minimises maintenance and all components should be documented thoroughly such that software maintainers can easily diagnose, bug-fix and improve the program.

# Final Design/Architecture

In this section, we will use a series of diagrams to show the functionality of our program as a whole, and thus how the program fulfils the functional requirements set out in the project brief above.. Each of the diagrams are updated to represent our program as it will be presented to the client.

|  |  |
| --- | --- |
| **Overall Context Diagram**  This diagram displays all services our program provides and shows each services’ connection to different user classes (staff, donors, graduates, prospective students).  These services all belong to one of the three main categories set out in the brief (prospective student management, graduate data management and donor management)  All services are linked into a single website, which will be described in greater detail in the next diagrams. |  |

|  |
| --- |
| **Prospective Student Diagram**  Prospective student data is stored in two parts: information on the student in D5, and information on each application in D4. This allows a single student to submit multiple applications if needed. Each prospective student fills out an application form and submits it through process 9. This data is stored in D4, which in turn creates a new data entry in D5 if necessary. Staff members can view these prospective student details and applications through processes 11 and 14 respectively and send responses back through process 10. Staff can also add prospective students manually through process 12 and make changes to submitted applications through process 13. |
|  |

|  |
| --- |
| **Graduate Student Diagram**  The graduates and survey systems are also quite interconnected, so are again shown on a single graph.  The datastore D7 contains the complete surveys and new surveys that need to be sent out.  Medic to Medic sends out annual surveys to their graduates. Surveys are stored in Datastore D7. Staff members can view all existing surveys and responses through process 17, as well as add new surveys through process 19 and edit them through process 18. Staff will start process 15 to send out new surveys, this gets the graduate information from datastore D6, via process 14, and sends the survey to the retrieved graduates. Graduates can then complete the form and return their data through process 16, which stores their responses back in D7 to be viewed by the staff. |
|  |

|  |
| --- |
| **Donor Management + Newsletter Diagram**  Donors are stored in datastore D1. Staff can view all donors and export their emails (say to use in a mail merge) through process 3. Staff are also able to edit donor details and add new donors through processes 4 and 1 respectively. Donors themselves can edit their own account details and their student preferences, also through process 4 (though locked to only their own account of course). Donors can view updates on their assigned students through process 7, which grabs information from the student database D3. All other interactions with this database are out of scope for our part of the project. Finally, donors are of course able to make donations through process 8.  Newsletters are stored in datastore D2. By going through process 5, staff can view all existing newsletters, and then either modify them if necessary or add new ones through processes 2 and 6 respectively, which each flow back into D2. Process 6, adding a newsletter, also sends a notification out to donors so they can stay up to date. |
|  |

# Test Results

The test results below will be divided into tests subsections consisting of functionality, usability, scalability, and performance tests.

## Functionality Test Results

The original test plan created earlier in the project has been updated to fit to our project scope more accurately.

All functionality unit tests have succeeded to within an 80% test success threshold within each test set, with the vast majority reaching 100% test success. Given that the unit tests in each set were intended to test the lowest level of functionality that the project relies on, this is not a surprise, for any faults here would cause the program as a whole to not function correctly.

On the integration level, the project successfully passed all test sets with a rate above the threshold of 80% as set in the original test plan. There were some failures early in the testing process in that data transfer between the program components did not function correctly, which meant that data would never reach other components and would essentially disappear.

On the system level, the project also succeeded on all tests. The tests were designed as such that failing them would mean the project would not be ready for release, as the system tests ensured that all components would work correctly under realistic conditions (black box).

## Usability Test Results

Multiple testers were used to test the usability of our program once it approached completion. Each of the developers were given the option to take part in usability testing, despite the possible adverse effect on test results.

Users unfamiliar with computers or with limited skill tended to perform the worst in the tests, with some test results falling outside of the acceptable range. Testers taking the role of untrained staff users performed worse than average. With basic training on how to use the program, results improved significantly, and all stayed within the acceptable range.

Most failed tests occurred in the staff user section. The staff section is technically the most complicated, with relatively large numbers of controls and options. This is concerning as our program was designed with simplicity in mind, making it the least complicated and most streamlined for all user types.

The most common issue noted was the user interface, where it was sometimes confusing, or the controls were inconsistent between pages.

Overall, the degree of failure in these tests were not very large, and due to our small tester base, it is possible that these results may simply be outliers, not requiring us to make significant user interface changes. We would need to standardise controls so the same types of controls perform the same tasks (such as sorting/organising data and editing data) and decluttering the user interface, such as with the drop-down menus used for navigation.

Testers with sufficient computer experience (based on personal description) all performed within the acceptable range for both the staff, donor user and student sections. We deemed this section an overall success, as it showed that staff users can easily adapt to and use our software without needing specialised training for our program.

## Scalability and Performance Test Results

All performance tests succeeded, with no problems found. This is not surprising, as our program was designed with a simple structure and the amount of data transferred is minimal. Due to this, the small size (and projected size) of the database among other factors, lead to acceptable performance for all tasks. Our requirement was set at 95% completion rate within 2 seconds, of which most tasks completed below 0.5 seconds. The longest operation was the donation function, which still passed the 95%/2s threshold. This is most likely due to the use of an external API/function that we did not create ourselves. We could improve its performance by finding a different and faster payment gateway, but the work required vs. the performance improvement is not compelling enough to do so. Should the volume of donations expected increase massively, then it may warrant a review and overhaul.

Scalability testing showed potential issues when very high loads were placed on our program and database, with the database being flooded with requests and degrading performance. However, the degree of load required for significant degradation of performance is far beyond what we estimate the program to deal with through its lifetime. We do not believe that any action is necessary at the current moment to address this.

Concurrency issues have been noted in our database, where two staff users attempt to modify the same indexes at the same time. At the moment, the program responds by implementing the most recent changes/actions. As staff numbers increase and concurrent unique users increase, this would eventually warrant a change in which staff users cannot successfully modify the same index at simultaneously.

Early in development, users were able to log in on the same user account, which would also cause concurrency and security issues. Discovered early on in testing, we implemented a different form of authentication and login security, which now prevents users from logging in on a unique user on two different instances simultaneously.

# Group Challenges

In this section, we will discuss the challenges our group experienced during the development process.