|  |  |
| --- | --- |
|  |  |

## Contents Page

Contents

[Contents Page 2](#_Toc40237849)

[Introduction 3](#_Toc40237850)

[Planning & Design Process 4](#_Toc40237851)

[Normalisation Table 4](#_Toc40237852)

[User Case Diagram 5](#_Toc40237853)

[Class Diagram 6](#_Toc40237854)

[Activity Diagram 7](#_Toc40237855)

[Sequence Diagram 8](#_Toc40237856)

[State Diagram 9](#_Toc40237857)

[Entity Relationship Diagrams 10](#_Toc40237858)

[Database Creation 11](#_Toc40237859)

[Evaluation 14](#_Toc40237860)

[Conclusion 14](#_Toc40237861)

## Introduction

#### Scenario

This project was to adapt and create a data driven application using data from Facebook users, their friends and the messages shared between them. The idea behind it was to use MySQL as a backend database to store large quantities of data and then code a front-end from scratch to load, read and access those databases in a sample interface. The core focus being to learn an understanding of the planning process from an entrepreneurial perspective, enabling us to effectively plan and pitch a project to future clients.

#### The project idea

Creating a simple user interface required a user to log on to the website and the ability to view their credentials and information (only theirs) as well as view a list of their current friends, and a drop down box of the messages they have sent and received (if any). The credentials as stipulated in Table 1 are found in the UNF of the normalization table in appendix 1. This idea was taken from Facebook’s© current layout, to learn and understand how that information may be used. Henceforth, it was my first thought that the ID numbers found in the data would be purely used as a catchment and not something that should necessarily be read by the user.

#### Requirements

##### Functional

The functional requirements for the application should enable the user to:

* Login to their profile
* View a list of friends
* View messages sent and received between that user and others
* View profile information regarding their education, work, and personal details

##### Non-Functional

The non-functional requirements for the application should enable the user to:

* Be prompted on incorrect credentials for logging in
* Know that they are in their own account
* Confident that the information collected of others is limited to only what is necessary
* Have no errors when navigating to the profile page for the information it should contain

## Planning & Design Process

Normalisation TableA screenshot of a cell phone

Description automatically generated

#### Appendix 1 – ERD Normalisation Table

Normalisation makes information concise and minimizes redundancy. Through the processes of UNF to 3NF in Appendix 1 it shows how in its first raw form, all the information would be collected in one large database that would be difficult to read and edit in the future. There is also an allowance for repeating data. Initially, this normalisation was difficult due to the need to break apart occupation into education and workplace then creating a student and worker table for their relationship with the user database. This was vital to complete before moving on to creating the databases to remove the obscurity of a user having more than one job and one course of education. Highlighted in the table are the redundant repeating groups, and through the procedure were isolated, adapted or removed to provide a clear relationship between the necessary fields.

### User Case Diagram

A close up of text on a white background

Description automatically generated

#### Appendix 2 – User Case Diagram

The initial design was an adaptation of how Facebook© currently allows its users to navigate through their individual user accounts. This version was simplified to only contain processes adequate to the information provided in Table 1 of the scenario. All the functions shown here were initially projected to include a button to execute the query, however after trial and error, having too many buttons could diminish the user experience and so the final draft was adapted to exclude this for the time being. In an entrepreneurial perspective, there would be a back and forth communication with the client before a speculative adaptation. However, in this case this has been ignored.

### Class Diagram

A screenshot of a social media post

Description automatically generated

#### Appendix 3 – Class Diagram

Leading on from the final normalisation, the class diagram shows the initial variables required to create the database. This is the first instance of identifying the relationships between the tables and allows an easy understanding of the one-to-many rule. Each direction shows how many children can belong to one father, but only one father can have many children. The friendship table is the only one that is linked to the initial user table with no children of its own. This is because in the messages table, the friend ID is equivalent to the user ID and requires an identifier table to establish the relationship between the individual users.

### Activity Diagram

A screenshot of a cell phone

Description automatically generated

#### Appendix 4 – Activity Diagram

After the initiative to remove the need for multiple buttons, the activity of the interface shows an initial flow of the user inputting credentials to access their stored information on the database. There is a repeating element in the “Login” function to only allow access after the correct information is entered. This credential would be used to determine the user ID in the database to fetch the corresponding information for viewing. The flow moves on to allow the ability of the user to return home or exit the application. There is further improvement that can be implemented into this, with a more complex user interface, however, there is a limitation on time allowance and thus the user interface was simplified.

### Sequence Diagram

A screenshot of a social media post

Description automatically generated

#### Appendix 5 – Sequence Diagram

Moving on from activity, this diagram breaks down the interactions of the user with the front-end system and the database of the server. This allows an understanding of how the connection between the user and the software would transpire and a rough guideline as to what interactions would need to have an execution to the database server to retrieve information needed for the next stage in the user process. Justifiably, the initial design was to break apart the queries for the relevant information into separate buttons, which would then force an executable from the front-end to the SQL database. Fundamentally, the executables now all run on the navigation to the profile page, which means only one request is needed to the database from the system.

### State Diagram

A picture containing text, map

Description automatically generated

#### Appendix 6 – State Diagram

This diagram provides a clear structure in determining how the interface will be designed. It shows how the data in the database will be manipulated for each individual users’ purpose and as such, it helps with determining the flow of the different states within the application. Ultimately it was from this diagram that the initial interface was created due to how easy it was to see the state changes after events.

### Entity Relationship Diagrams

#### A close up of a white wall Description automatically generatedAppendix 7 – Initial ERD

In this simplest form it is inadequate to show how all the user information will be stored in databases.

#### A close up of a white wall Description automatically generatedAppendix 8 – Normalised ERD

In diagram form from the normalisation it is clearer to see how the many-to-one relationships now appear. There is a worker and student father to workplace and education respectively so to remove the potential of many-to-many connections due to the ability to have more than one job and to do more than one degree at university.

#### A screenshot of a cell phone Description automatically generated Appendix 9 – Final ERD

As above, the final ERD matches perfectly. There was a need to create foreign keys in both friendships and messages as both databases use a friend ID which is the same as the Primary Key (user ID) in the users table. Setting these as foreign keys enable us to identify them out separately in the SQL queries. As a result of this, the ERD shows two distinct many-to-one connections instead of the usual one.

A limit was made on the message text of 5000 characters which should be adequate for maintaining a distinct conversation with another user, without ruining the overall user experience. This number was taken from researching a similar social media site with private messaging functions (Twitter) who allow 10,000 characters and making an executive decision to conclude that 5,000 was more than adequate.

## Database Creation

#### User Table

|  |
| --- |
| CREATE TABLE ` isad157\_mriley-wallace ` . `users`(  `user\_id` INT NOT NULL AUTO\_INCREMENT,  `first\_name` VARCHAR(40) NOT NULL,  `last\_name` VARCHAR(40) NOT NULL,  `hometown` VARCHAR(40) NOT NULL,  `gender` VARCHAR(10) NOT NULL,  `relationshipStatus` VARCHAR(20) NOT NULL,  `currentTown` VARCHAR(40) NOT NULL,  PRIMARY KEY (`user\_id`)); |

Beginning with the source SQL table; this holds the unique identifier between users and their friends and the relationships with the other tables to ensure that any query to the database can find the correct credentials associated to each user. First off all variable declarations are set to NOT NULL to instantiate that there must be data in these fields for the rest of the database to function appropriately.

#### Messages Table

|  |
| --- |
| CREATE TABLE ` isad157\_mriley-wallace` . `messages`(  `friend\_id` INT NOT NULL AUTO\_INCREMENT,  `user\_id` INT NOT NULL,  `date\_time` VARCHAR(100),  `message\_text` VARCHAR(5000),  FOREIGN KEY(`user\_id`)references `users`(`user\_id`),  FOREIGN KEY(`friend\_id`)references `users`(`user\_id`)  On Delete CASCADE  On Update CASCADE ); |

There is no primary key in this table as both user\_id and friend\_id reference the same variable in the users database. A CASCADE declaration was added to any delete and update of the database to ensure that corresponding data is removed from the database once a user has been deleted. This helps to reduce any data redundancy.

#### Friendship Table

|  |
| --- |
| CREATE TABLE ` isad157\_mriley-wallace` . `friendships`(  `user\_id` INT NOT NULL AUTO\_INCREMENT,  `friend\_id` INT NOT NULL,  FOREIGN KEY(`user\_id`)references `users`(`user\_id`),  FOREIGN KEY(`friend\_id`)references `users`(`user\_id`)  On Delete CASCADE  On Update CASCADE  ); |

For the friends list, there is a simple friendship table to establish the connection between a user and their friends. Messages do not rely on friendships as a user can message anyone who is not a friend, and a user doesn’t necessarily message every friend. The ID’s are used to reference the original user ID to allow the system to find the relevant information of each person such as their name.

#### Worker & Workplace Tables

|  |  |
| --- | --- |
| CREATE TABLE `isad157\_mriley-wallace` . `worker`(  `user\_id` INT NOT NULL AUTO\_INCREMENT,  `workplace\_id` INT NOT NULL,  `start\_date` VARCHAR(40) NOT NULL,  `end\_date` VARCHAR(40) NOT NULL,  FOREIGN KEY (`user\_id`)references `users`(`user\_id`),  FOREIGN KEY (`workplace\_id`) references `workplace`(`workplace\_id`)  On Delete CASCADE  On Update CASCADE  ); | CREATE TABLE `isad157\_mriley-wallace` . `workplace`(  `workplace\_id` INT NOT NULL AUTO\_INCREMENT,  `workplace` VARCHAR(40) NOT NULL,  PRIMARY KEY (`workplace\_id`)); |

The relationship between worker and workplace allows a user to have multiple workplaces, and the workplace table can be easily identified via its primary key.

#### Student & University Tables

|  |  |
| --- | --- |
| CREATE TABLE `isad157\_mriley-wallace` . `student`(  `user\_id` INT NOT NULL AUTO\_INCREMENT,  `education\_id` INT NOT NULL,  `start\_date` VARCHAR(40) NOT NULL,  `end\_date` VARCHAR(40) NOT NULL,  FOREIGN KEY (`user\_id`)references `users`(`user\_id`),  FOREIGN KEY (`education\_id`) references `education`(`education\_id`)  On Delete CASCADE  On Update CASCADE  ); | CREATE TABLE `isad157\_mriley-wallace` . `universities`(  `education\_id` INT NOT NULL AUTO\_INCREMENT,  `university` VARCHAR(100),  PRIMARY KEY (`education\_id`)); |

As with workers, students and university tables again link their relationship via an education ID to allow a student to belong to more than one University.

## Evaluation

The design aspect of the project took up the most time. This is all the “prep” work needed to ensure a smooth understanding of the overall project. The diagrams in this project fulfill the requirements for the software and functionally allow an individual user to access their personal information and only the relevant information of another user. This then ensures that there is some form of data protection within the system. The diagrams helped understand what was relevant within the software and helped with the efficiency of the databases by ensuring they went through the normalisation process first. From a business perspective, the diagrams will help a prospective client to see the state changes of the software and will be able to give more formative feedback on their criteria than would be possible in the initial ERD. The problem of time management was solved by simplifying the interface to reduce the number of buttons to execute the queries individually and instead executed it on the load function once the login credentials were successfully entered. This then allows the database to be read in a single instance and has everything available to the user.

## Conclusion

In conclusion, the design of the project has been completed to the initial criteria successfully and has led to the creation of a simplified demo interface. Ultimately the problems were overcome by looking at how a user would interact with the system and adjusting the application accordingly to provide a more streamlined version of database executions. There is room for improvement as the only issue was time management and with a better understanding on how to overcome this will allow for a much more in-depth demo than what was successfully created. The detriment of time has caused some design implementations to be discarded completely which resorted in the diagrams being adjusted, which turned out to be also time consuming. In hindsight, planning the project in a better time scale would have led to a much more efficient end application and this gives an understanding how delays on certain steps of the project can cause an overall delay on the entire finished product. The demo software is a functional and crude sample on the connection between front-end and backend development and the overall project gives knowledge into the complexity in fulfilling a client’s requirements for future projects.

Github Repository Link: <https://github.com/mriley-wallace/ISAD157-A-D-Project>

Final Word Count: 1826 words