

**Faculty of Engineering**

**Department of Electrical and Electronic Engineering**

**Name: Swaraj Dube**

**Student ID: 011114**

**Name of Module: Web Based Computing**

**Code of Module: H63JAV**

**Module Convenor: Dr Lim Wee Gin**

**Effective Date: 19/4/16**

**Table of contents**

1. Introduction to the application………………………………………………………………………………………………………………….Page 1
2. Program design…………………………………………………………………………………………………………………………………………Page 1
3. Logging into MyManager…………………………………………………………………………………………………………………………..Page 1
4. MyManager sign up…………………………………………………………………………………………………………………………………..Page 3
5. Storing the user’s password safely………………………………………………………………………………………………Page 3 – Page 4
6. Database connection…………………………………………………………………………………………………………………..Page 4 – Page 5
7. To-do list manager…………………………………………………………………………………………………………………….Page 5 – Page 22
   1. Adding a new task…………………………………………………………………………………………………………..……Page 5 – Page 9
   2. Editing tasks……………………………………………………………………………………………………………………..Page 10 – Page 17
      1. Change name of task …………………………………………………………………………………………..Page 10 – Page 11
      2. Modify due date………………………………………………………………………………………………….Page 11 – Page 12
      3. Modify due time…………………………………………………………………………………………………..Page 12 – Page 13
      4. Change priority……………………………………………………………………………………………………Page 13 – Page 14
      5. Delete task………………………………………………………………………………………………………….Page 14 – Page 15
      6. Mark as done………………………………………………………………………………………………………Page 16 – Page 17
   3. Viewing tasks in different mode………………………………………………………………………………………Page 17 – Page 22
      1. View by priority…………………………………………………………………………………………………..Page 17 – Page 18
      2. View completed tasks………………………………………………………………………………………….Page 19 – Page 20
      3. View incomplete tasks………………………………………………………………………………………..Page 20 – Page 21
8. Contacts Directory…………………………………………………………………………………………………………………Page 22 – Page 32
   1. Adding a new contact……………………………………………………………………………………………………..Page 22 – Page 25
   2. Editing a contact……………………………………………………………………………………………………………..Page 25 – Page 29
      1. Editing name of contact……………………………………………………………………………………..Page 25
      2. Editing email of contact………………………………………………………………………………………Page 25 – Page 26
      3. Editing phone number of contact……………………………………………………………………….Page 26 – Page 27
      4. Delete contact……………………………………………………………………………………………………Page 28 – Page 29
   3. Sorting contacts by name ………………………………………………………………………………………………Page 29 – Page 30
   4. Searching contact by typing name in text field……………………………………………………………….Page 30 – Page 32
9. Software engineering practices used…………………………………………………………………………………….Page 32 – Page 33
   1. Inheritance……………………………………………………………………………………………………………………..Page 32
   2. Polymorphism………………………………………………………………………………………………………………..Page 32 – page 33
   3. Encapsulation…………………………………………………………………………………………………………………Page 33
10. External libraries included in the project………………………………………………………………………………Page 33
11. Introduction to the application

The application made has been named “MyManager” and it is designed to help users in their management. Often when people are very busy or in a rush, they tend to forget a lot of things. This is especially true when there are a lot of things on hand to do or when it is time to contact the right person at the right time. Hence what this application offers is a to-do list manager and a contacts directory. The users can quickly check the contacts directory to contact someone in case they do not remember the contact and also manipulate the contacts, the same goes for a to-do list manager where a person can add and edit tasks to help them keep track of their tasks.

1. Program design

The following block diagram shows how this application is designed:

Main applet form

Contacts directory JFrame

To-do list manager JFrame

The subclasses in this case perform their own functions by connecting to the database such as editing contacts or manipulating tasks.

The main database connection class is the one that creates the central database file and establishes a connection to that database which includes multiple tables for different purposes each of which can be accessed by the sub classes.

The main encryption class is used to encrypt the user’s password which returns it to the main applet form which then checks for a password match is the database if the user wants to log in or insert the credentials into the database when a new user signs up.

The JFrame forms provide the user interface to enjoy the contacts directory and to-do list manger features in the application which sends information to the sub classes for processing and receives processed information just for display.

1. Logging into MyManager

The very fact that the application contains contacts and tasks for people, there needs to be a security mechanism in order for other users to view other users’ data. This is why a login page has been created where the users are required to enter their username and password and only then the respective user can log into the application. The user credentials must be stored in a database so that at any given time a user tries to log in, the program can check for existing credentials and then allow the user to proceed.

Page 1

The log in screen is shown in the Figure 3.1 below:

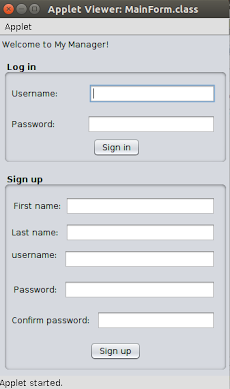


Figure 3.1

The database to store the user’s credentials is designed as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **first\_name** | **last\_name** | **email** | **pass** |
| aaa | abc | aaauser1 | ... |
| bbb | bcb | bbbuser2 | ... |

This table named as ‘user\_info’ is what that stores the user credentials and every time a new user signs up for this application, a new row is inserted that stores the new user’s credentials.  A unique property of this database design is that it does not allow duplicate emails just like any other huge internet company like Google. The following MySQL query is used to create this table:

CREATE TABLE user\_info(

first\_name varchar(100),

last\_name varchar(100),

email varchar(100)

pass varchar(100),

PRIMARY KEY (email)

);

As it can be seen that the email is the primary key here which means that duplicate records cannot exists in that particular column. The following MySQL query is used to insert data into this table:

“INSERT INTO `user\_info`(first\_name,last\_name,email,pass) VALUES(val1,val2,val3,val4) ”;

Where val1, val2, val3, val4 are the user’s credentials.

Page 2

1. MyManager sign up

This feature has been created for obvious reasons where if a user does not have an account and wants to get started using the application to help manage things then the user can comfortably sign in. As it can be seen in figure 3.1, the sign up section is where the user needs to sign up, the only thing that needs to be made sure is that the password should be at least 8 characters in length otherwise the application will not allow the user to proceed as shown in Figure 4.1 and alongside that the user must choose a different username if the application tells the user to choose another username if an existing username is found a shown in figure 4.2. Upon entering all the credentials correctly, these values are then updated in the table ‘user\_info’ using the following query:

“INSERT INTO `user\_info`(first\_name,last\_name,email,pass) VALUES(val1,val2,val3,val4) ”;

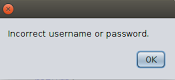
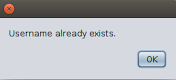
Where val1, val2, val3, val4 are the user’s credentials.

Figure 4.1 Figure 4.2

1. Storing user’s password safely

Transposition cipher method is used for encryption of the password. When the user enters the password then that particular string is encrypted using this technique which is then stored on a database. This way one cannot tell directly about the user’s password as it is all encrypted. The way this encrypting mechanism works is as follows:

Consider the following string: “GO BUY SOME MILK AND EGGS”. In order to encrypt this string, a key is used and this key is in the form of an integer say the key is 12345, in this key ‘5’ is the largest number and this number stands for the number of columns in the encrypted matrix which in this case stands for 5 columns and the numbers of rows depend on the length of the string as follows:

[G, O,  , B, U]

[Y,  , S, O, M]

[E,  , M, I, L]

[K,  , A, N, D]

[ , E, G, G, S]

As it can be seen that there are 5 columns however the number of rows will not be fixed due to the length of the string as the string is written in row by row order in the matrix above including white spaces and thus is this is how the matrix is made. There however is an exception here because it is not compulsory that the last row will have the same amount of characters as the number of columns which means if the maximum number in the key is 5 then it is not compulsory for the last row to have 5 characters as well for example if the string is: “this15mypassword” then the matrix look like:

[t, h, i, s, 1]

[5, m, y, p, a]

[s, s, w, o, r]

[d,]

Page 3

As it can be seen that the last row does not have 5 characters and this could run into a problem hence what is done is that the remaining pending characters are filled with ‘#’ and thus the final matrix would look like:

[t, h, i, s, 1]

[5, m, y, p, a]

[s, s, w, o, r]

[d, #, #, #, #]

Most of the encoding process is complete by this point with the help of the matrix and thus the final step to encoding is to store the encoded password that reads the matrix column by column. In this case:

-Reading the first column would be read as: t5sd

- Reading the second column would read as: hms#

- Reading the third column would read as: iyw#

- Reading the fourth column would read as: spo#

- Reading the fifth column would read as: 1ar#

And thus the final string would be appending of all the columns string in order which would mean that the final encoded string is: t5sdhms#iyw#spo#1ar#.

Reason for choosing the encoding technique

* This is a very secure way of securing critical data as different keys can be used at different time for example a key of 326357 can also be used which means that 7 is the maximum number here and the matrix will have 7 columns, for another case the key can be 43215 which means the maximum number is 5 and the matrix will have 5 columns.

As it can be seen that the number of columns cannot be fixed hence it is impossible to come up with one universal decoding scheme that can decode any string encoded in this format as matrix generation is random.

1. Database connection

In order for the program to interact with a MySQL database, a MySQLite JDBC driver is required. This was downloaded and the driver’s .jar file was included in the project which would then be able to interact with the local SQLite database. The reason why a local database has been chosen is because if the internet connection is slow then the program could approach a state of freeze and occasionally the cloud server may be down as well. None of these problems occur if a local database is used. One other benefit of using this local database is that when this database is created it is automatically saved inside the respective project folder with the extension ‘.db’ hence if this project were to be sent across multiple stations then the original data still remains. The only concern that now remains is to add permissions to the .db file to limit the access which could be done using terminal commands. The class ‘Connection’ is used to establish a connection to the database, the class ‘Statement’ is used to execute MySQL queries and the class ‘ResultSet’ is used to fetch data from the database. The following code on the next page is used to create a database.

Page 4



The name of the database file is ‘MyManager.db’ in this case.

1. To-do list manager

This is one of the main features of this program. The user can save their tasks, set priorities , set due time and date and also edit these tasks by changing the due date, time, task name and also mark the tasks as done and delete the tasks as well. The figure 7 a) below shows the GUI designed for the To-do list manager:

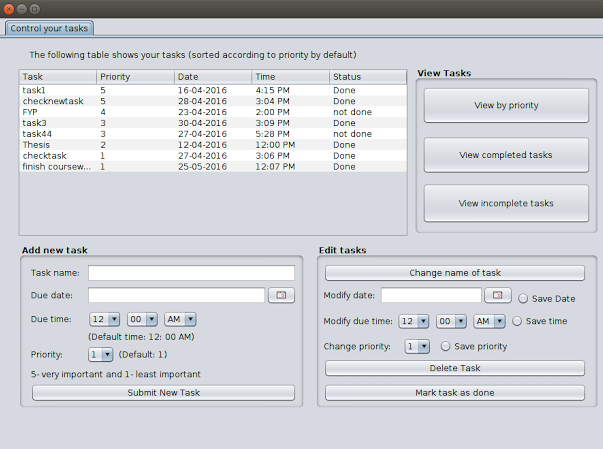


Figure 7 a)

7.1 Adding a new task

For a adding a new task successfully, the user must enter the task of the name and the due date else the a dialogue box appears the specifies the user must not leave these fields empty, the user could also select the due time and the priority however these are by default set to 12:00 AM and level 1 for the priority respectively.

The priority levels range from 1-5 where 5 is extremely important and 1 is least important. The task that is entered is sorted by priority and put into the appropriate row in the table.

Page 5

Furthermore whenever the user wishes to check the to-do-list, by default the table shows tasks that are sorted by priority from highest to lowest which is shown in the figure 7 a).

4 databases are used to help support ‘adding a new task’ feature which are listed as follows:

1. ‘user\_tasks’

This database stores the name of the task that the user just saved and the table is designed as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **user1** | **user2** | **user3** | **...** |
| 1 | Buy shoes | Report submission | Send email to boss | ... |
| 2 | Finish thesis | Print documents | Find the bug | ... |
| ... | ... | .... | ... | ... |

The following MySQL query has been used to design this table:

CREATE TABLE user\_tasks(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

The following query is used to add columns as users to the table:

“ALTER TABLE user\_tasks ADD COLUMN new\_user\_name\_here varchar(100)”

The ID in this case acts as the unique identifier for the task name and the ID can never be duplicated as a ‘PRIMARY KEY’ has been applied to that column as well as ‘AUTO INCREMENT’ which means every time a new row is inserted the ID counter increases as well which is what the dots represent in the table above i.e. the row will keep growing as a new task is added. The following query is used to add a task name into the database if user1 add a new task:

“INSERT INTO user\_tasks(user1) VALUES(‘name\_of\_task’)”

The columns represent the user. Every time a new user signs up a new column is allocated to that user to differentiate different tasks between different users and yet again the dots on the right most column indicate that the table will grows it columns as a representation for the new user every time a new user signs up

1. ‘user\_task\_priorities’

This table stores the priority of the task added. The design of the database is the same as ‘user\_tasks’ however the only difference here is that the cells now stores number (2-5) instead of characters, the table is designed as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **user1** | **user2** | **user3** | **...** |
| 1 | 2 | 3 | 1 | ... |
| 2 | 5 | 4 | 5 | ... |
| ... | ... | .... | ... | ... |

Page 6

With respect to ‘user\_tasks’ table , the way this table can be interpreted as the following: the task for user1 at ID=1 is “Buy Shoes” and the priority of the task is 2 because at ID=1 the priority of user1 is ‘2’ , in the same way the priority of the task “Report submission” of user2 is ‘3’. This table grows in a similar manner as ‘user\_tasks’ i.e. every time a new task is added, the priority values are also added and every time a new user signs up, a new column is also added.

The following MySQL query has been used to create this table:

CREATE TABLE user\_task\_priorities(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

The following MySQL query has been used to add column to this table:

“ALTER TABLE user\_task\_priorities ADD COLUMN new\_user\_name\_here varchar(100)”

Where ‘new\_user\_name\_here’ is the username of the new user just signed in.

The following query is used to add rows and insert priority values:

“INSERT INTO user\_task\_priorities(username) VALUES(prioriry\_value)”

Where username is the name of the user currently signed in the application and priority\_value is the number ranging from 1-5.

1. ‘user\_task\_dates’

This table stores the due dates of the tasks saved. The design of this table is the same as ‘user\_tasks’ however the cells in this table just store the dates of the tasks, the table is designed as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **user1** | **user2** | **user3** | **...** |
| 1 | 24-4-16 | 31-5-16 | 1-5-16 | ... |
| 2 | 5-5-16 | 4-6-16 | 5-4-16 | ... |
| ... | ... | .... | ... | ... |

This table means that the task “Buy shoes” of user1 has a due date of “24-4-16” because the task “Buy shoes” has an ID of 1 and in this table user1 has a value of “24-4-16” at the same ID i.e ID=1 . This table grows in a similar manner as ‘user\_tasks’ i.e. every time a new task is added, the due dates are also added and every time a new user signs up, a new column is also added.

The following MySQL query has been used to create this table:

CREATE TABLE user\_task\_dates(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

Page 7

The following MySQL query has been used to add column to this table:

“ALTER TABLE user\_task\_dates ADD COLUMN new\_user\_name\_here varchar(100)”

Where ‘new\_user\_name\_here’ is the username of the new user just signed in.

The following query is used to add rows and insert due dates:

“INSERT INTO user\_task\_dates(username) VALUES(Due date)”

Where username is the name of the user currently signed in the application and Due date is the due date of the task set by the user.

1. ‘user\_task\_time’

This table stores the due time of the tasks saved. The design of this table is the same as ‘user\_tasks’ however the cells in this table just store the dates of the tasks, the table is designed as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | user1 | user2 | user3 | ... |
| 1 | 5:00 PM | 31-5-16 | 1-5-16 | ... |
| 2 | 3: 00 PM | 4-6-16 | 5-4-16 | ... |
| ... | ... | .... | ... | ... |

This table means that the task “Buy shoes” of user1 has a due time of “5:00 PM” because the task “Buy shoes” has an ID of 1 and in this table user1 has a value of “5:00 PM” at the same ID i.e ID=1 . This table grows in a similar manner as ‘user\_tasks’ i.e every time a new task is added, the due times are also added and every time a new user signs up , a new column is also added.

The following MySQL query has been used to create this table:

CREATE TABLE user\_task\_time(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

The following MySQL query has been used to add column to this table:

“ALTER TABLE user\_task\_time ADD COLUMN new\_user\_name\_here varchar(100)”

Where ‘new\_user\_name\_here’ is the username of the new user just signed in.

The following query is used to add rows and insert due dates:

“INSERT INTO user\_task\_time(username) VALUES(Due time)”

Where username is the name of the user currently signed in the application and Due time is the due time of the task set by the user.

Page 8

1. ‘user\_task\_status’

This table stores the status of the task which is either of ‘Done’ or ‘not done’. The design of this table is the same as ‘user\_tasks’ however the cells in this table just store the dates of the tasks, the table is designed as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | user1 | user2 | user3 | ... |
| 1 | Not done | Not done | Done | ... |
| 2 | Not done | Not done | Done | ... |
| ... | ... | .... | ... | ... |

This table means that the task “Buy shoes” of user1 has that task “not done” because the task “Buy shoes” has an ID of 1 and in this table user1 has a value of “not done” at the same ID i.e ID=1 . This table grows in a similar manner as ‘user\_tasks’ i.e every time a new task is added, the task statures are also added and every time a new user signs up, a new column is also added. However in this case every time a task is added the status of that task is set to “not done”.

The following MySQL query has been used to create this table:

CREATE TABLE ’user\_task\_status’(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

The following MySQL query has been used to add column to this table:

“ALTER TABLE user\_task\_status ADD COLUMN new\_user\_name\_here varchar(100)”

Where ‘new\_user\_name\_here’ is the username of the new user just signed in.

The following query is used to add rows and insert due dates:

“INSERT INTO user\_task\_status(username) VALUES(‘not done’)”

Where username is the name of the user currently signed in the application and by default all the task completion status are set to ‘not done’.

These are the 5 databases that are used to support “adding new task” feature and thus to sum this up , based on the 4 tables shown above, it can be said that “Buy shoes” task will have the following properties:

Task: Buy Shoes

Priority:  2

Due date: 24-4-16

Due time: 5:00 PM

Status: not done

This is because all these elements are under ‘user1’ column and all of them have the same ID.

Page 9

* 1. Editing tasks

In this feature, the user can edit any part of the task which includes task name, priority, due date, due time, deleting the task and also marking the task as done. The 5 databases mentioned above are again required to make this feature possible.

One important note here is that the table itself is not editable, the user has to select the row and only then edit the task otherwise a dialogue box will open up specifying to select a row first. The name of task acts a unique field in this table as the user could add as many tasks but 2 tasks of the same name cannot exists in this program, if the user tries to enter a new task whose name is pre-existing in the database then a dialogue box pops up telling the user to choose another task name.

Upon making any changes to the to-do-list, the main table in the GUI shown in figure x.x is updated real time so that the user can see the changes instantly.

* + 1. Change Name of task

As the name suggest, this feature helps the user to change the name of the task selected. First for this to happen, the user must first select the row of the table, upon selecting a row in the table, the task name in that row is selected as the task name is the unique identifier in this case as this data is what is going to be used to search the correct task in the database and edit the name.

Once the row is selected and the user chooses to change the name of the box, a confirmation dialogue appears that prompts the user to enter a new name of the task. Upon entering the new task name, the current task name is searched in the table “user\_tasks” and when the search is found, the ID of that row is selected and the new task name selected by the user is replaced in this cell for example if this is the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | user1 | user2 | user3 | ... |
| 1 | Buy shoes | Report submission | Send email to boss | ... |
| 2 | Finish thesis | Print documents | Find the bug | ... |
| ... | ... | .... | ... | ... |

Suppose user1 chooses to edit “Buy shoes” to “Buy slippers”, what the program does is that it looks for “Buy shoes” in the user1 column and once the match is found, it’s associative ID is also fetched which is ‘1’ in this case and now what’s done is the program replace the cell at column “user1” and ID=1 with “Buy slippers”, this way the name of the task is successfully changed.

The following figure on the next page shows the name of the task ‘task 44’ which is shown in figure 7 a) being changed to the task name ‘print report’:

Page 10

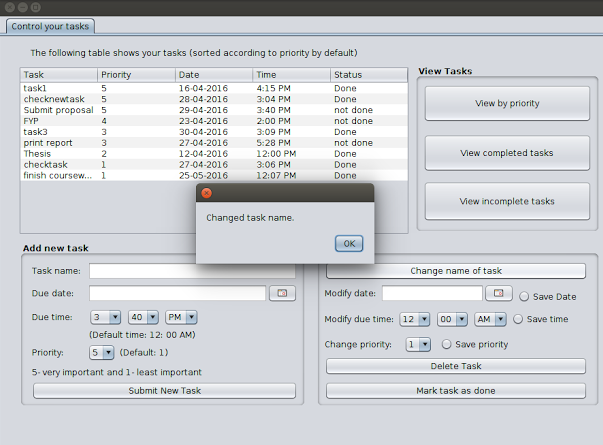


Figure 7.2.1 a) Changed task name ‘task44’ to ‘print report’

As it can be seen that the task ‘print report’ now appears in the table.

* + 1. Modify due date

This feature also works in the same way as changing the task name. First the user must select the row of which the due date has to be edited, upon clicking the row, the task name is stored in a temporary variable and then when the user chooses a new date using the ‘JDateChooser’ and presses the radio button ‘modify due date’, the due date is changed in real time mode using the following steps:

If user1 wished to change the due of task “Buy shoes” from 24-4-16 to 25-4-16 then:

* The task name “Buy shoes” is searched in the ’user\_task’ table.

“SELECT user1 FROM user\_task”

* Upon finding the match, its associative ID is stored which in this case is 1.

“SELECT ID FROM user\_task WHERE user1=’Buy shoes’”

* ‘user1’ column in table ‘user\_task\_dates’ is then replaced with “25-4-16” where ID is 1.

“UPDATE user\_task\_dates SET user1=’25-4-16’ WHERE ID=1”

This way a new date now is stored. In order for this to be successful , the user must first select a row in the main table in the GUI, specify a new date and then finally click the radio button, if any of these are not done then the program will not proceed to the database operations.

Page 11

As it can be seen from figure 7.2.1 a) that the due date of the task ‘finish coursework’ is 25-5-16 and based on the queries shown above the due date is changed and the following figure shows the due date being changed to 30-4-16.

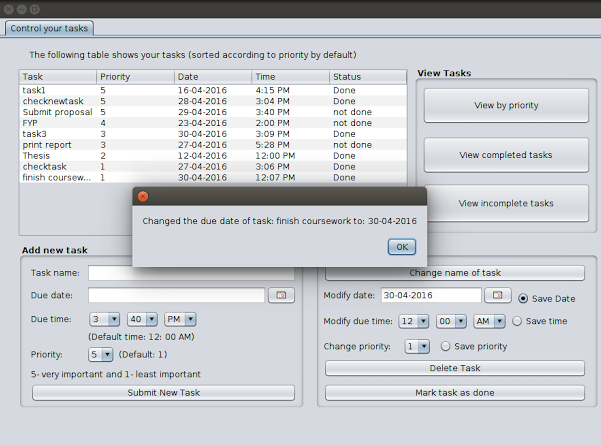


Figure 7.2.2 a) Changed due date of task ‘finish coursework’ to 30-04-2016

As it can be seen that the due date of the task has been changed.

* + 1. Modify due time

This feature also works in the same way as changing the task name. First the user must select the row of which the due date has to be edited, upon clicking the row, the task name is stored in a temporary variable and then when the user selects a new time from the combo boxes and then click the radio button, a new due date is saved. In this case the default due date is set to 12:00 AM hence it is not a must for the user to select a new time from the combo boxes however it is a must to select a row and click the radio button. When all the steps are completed, the program proceeds to the database operations which performs the following functions, so suppose user1 wants to change the dude time of the task “Buy shoes” from 5:00 PM to 6:00 PM then the following steps occur in database processing:

* The task name “Buy shoes” is searched in the ’user\_task’ table.

“SELECT user1 FROM user\_task”

* Upon finding the match, its associative ID is stored which in this case is 1.

“SELECT ID FROM user\_task WHERE user1=’Buy shoes’”

Page 12

* ‘user1’ column in table ‘user\_task\_time’ is then replaced with “5:00 PM” where ID is 1.

“UPDATE user\_task\_time SET user1=’5:00 PM’ WHERE ID=1”

As it can be seen from figure 7.2.2 a) that the due time of the task ‘finish coursework’ is 12:07 PM and applying the steps mentioned above, the program can change the task due time. The following figure shows the changed due time i.e. 11:55 PM:

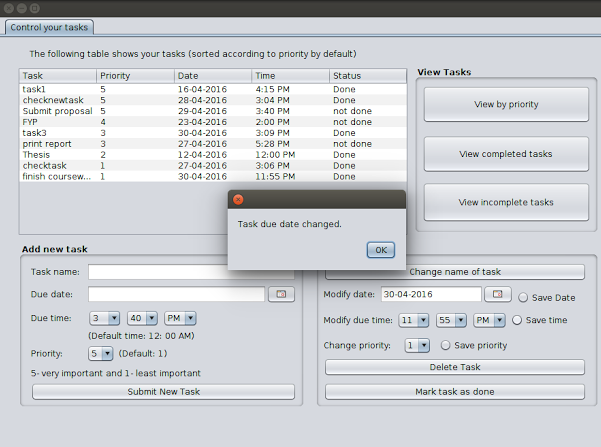


Figure 7.2.3 a) Changed due time of task ‘finish coursework’ to 11:55 PM

As it can be seen that the due time of task ‘finish coursework’ has been changed from 12:07 PM to 11:55 PM

* + 1. Change priority

This feature also works in the same way as changing the task name. First the user must select the row of which the due date has to be edited, upon clicking the row, the task name is stored in a temporary variable and then when the user selects a priority value from the combo box and finally click the radio button. Database operations proceed however it is not a must for the user to select the priority value from the combo box as the default value is already set to 1. Suppose user1 wants to change the priority of the task “Buy shoes” from 2 to 5, the following steps occur:

* The task name “Buy shoes” is searched in the ’user\_tasks’ table.

“SELECT user1 FROM user\_task”

* Upon finding the match, its associative ID is stored which in this case is 1.

“SELECT ID FROM user\_task WHERE user1=’Buy shoes’”;

Page 13

* ‘user1’ column in table ‘user\_task\_priorities’ is then replaced with 5 where ID=1.

“UPDATE user\_task\_priorities SET user1=5 WHERE ID=1”;

As it can be seen from figure 7.2.3 a) that the priority of the task ‘finish coursework’ is 1. By applying the steps discussed above, the priority of the task can be changed and the figure below shows the new priority of this task which is marked as 2.

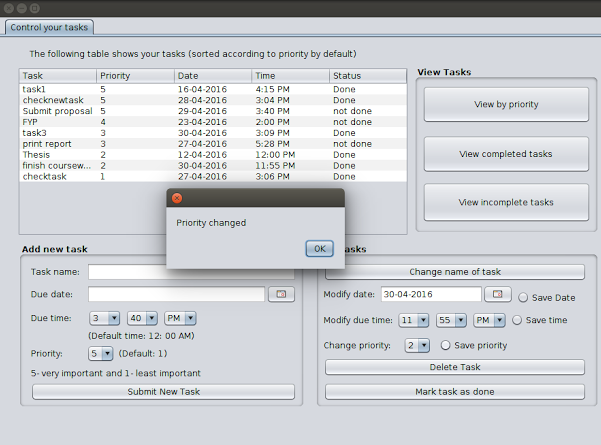


Figure 7.2.4 a) Changed priority of task ‘finish coursework’ to 2

As it can be seen that the priority of this task has been changed to 2 and more importantly, the task got automatically sorted by priority in reverse order as this program has been designed in a manner such that every time there is a new update, the table displays tasks that are sorted by priority in reverse order.

* + 1. Delete task

This feature works slightly different than all the 4 features explained above because when a task is deleted here then the task’s records from all the databases must be deleted which means the task name, priority, status, due date and due time must all be deleted. In this program if the user wishes to delete a task then the user must first select the row in the main table and then click the button to delete the task. Suppose user1 wants to delete the task with the following name: “Buy shoes” the following steps are occur in database processing:

* The task name “Buy shoes” is searched in the ’user\_tasks’ table.

“SELECT user1 FROM user\_task”

Page 14

* Upon finding the match, its associative ID is stored which in this case is 1.

“SELECT ID FROM user\_task WHERE user1=’Buy shoes’”

* The value in ‘user1’ column at ID=1 in table ‘user\_task\_priorities’ is deleted.

“DELETE FROM user\_task\_priorities WHERE ID =1”

* The value in ‘user1’ column at ID=1 in table ‘user\_task\_dates’ is deleted.

“DELETE FROM user\_task\_dates WHERE ID =1”

* The value in ‘user1’ column at ID=1 in table ‘user\_task\_time’ is deleted.

“DELETE FROM user\_task\_time WHERE ID =1”

* The value in ‘user1’ column at ID=1 in table ‘user\_task\_status’ is deleted.

“DELETE FROM user\_task\_status WHERE ID =1”

From figure 7.2.4 a) it can be seen that there is a task with the name ‘task3’ with a priority value of 3 being displayed in the table. Based on the steps discussed above any chosen task can be deleted and thus the following figure shows this task is found no more in the table once it is deleted:

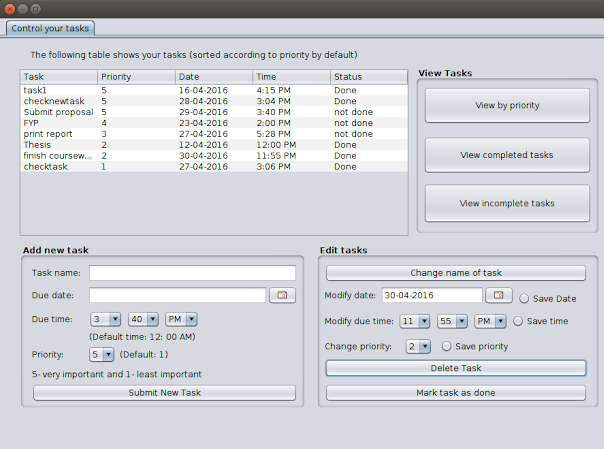


Figure 7.2.5 a) Deleted task ‘task 3’

Page 15

The following figure shows the algorithm in the form of flowchart used to delete a task:

* + 1. Mark as done

This feature also works in the same way as changing the task name. First the user must select the row of which the due date has to be edited, upon clicking the row, the task name is stored in a temporary variable and then when the user clicks the button to mark the task as done, database processing then takes place. Suppose user1 wants to change the task status with name “Buy shoes” the following steps take place:

* The task name “Buy shoes” is searched in the ’user\_tasks’ table.

“SELECT user1 FROM user\_task”

* Upon finding the match, its associative ID is stored which in this case is 1.

“SELECT ID FROM user\_task WHERE user1=’Buy shoes’”

* ‘user1’ column in table “user\_task\_status” is changed to “Done” where ID=1.

“UPDATE user\_task\_status SET user1=’Done’ WHERE ID=1”

From figure 7.2.5 a) it can be seen that the task ‘print report’ is not yet done and based on the steps discussed above, any task chosen can be marked as done.

Page 16

The following figure shows marking the task ‘print report’ as done:

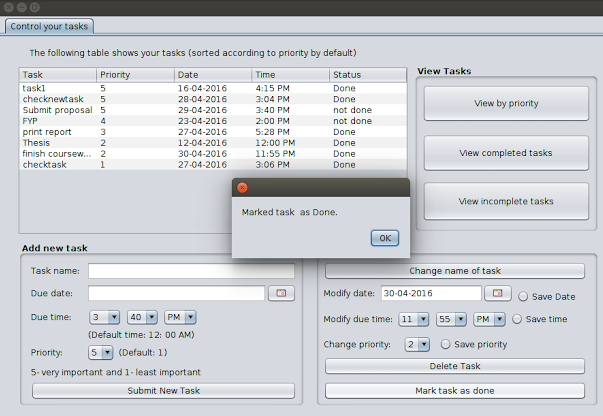


Figure 7.2.6 a) Marked task ‘print report’ as ‘Done’

* 1. Viewing tasks in different mode

In this program the user can view tasks in 3 modes which are as follows:

* View by priority
* View completed tasks
* View incomplete tasks
  + 1. View by priority

In the main GUI shown in figure 7 a) when the user clicks the button “View by priority”, the main table shows the tasks that are sorted by priority with the extremely important tasks at the top and the least important at the bottom of the table.

Suppose user1 wishes to view the tasks that are sorted by priority, the following steps occur in database processing:

* All the priority values are selected from the table ‘user\_task\_priorities’.
* All selected priority values are stored in an arraylist.
* Sort the arraylist.
* Iterate through the arraylist and get the task name, priority, due date, due time and status of that particular ID.
* Store this data and display in the table.

Page 17

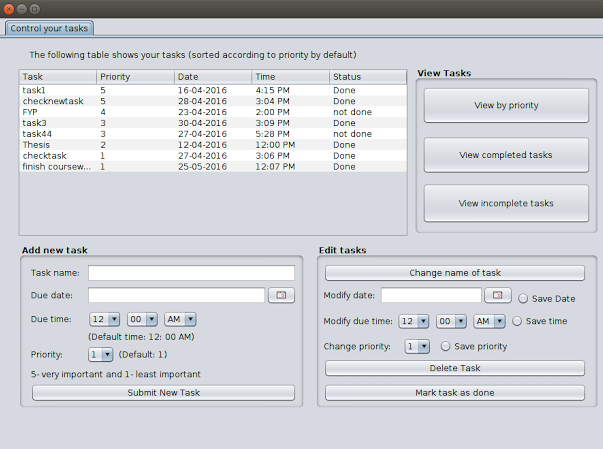


Figure 7.3.1 a) Tasks sorted by ID in reverse order

The following flowchart shows how the algorithm was implemented to achieve this task:



Page 18

* + 1. View completed tasks

In the GUI shown in figure 7 a) when the user clicks the button “View completed tasks”, the main table shows all the task details that have been marked as complete.

Suppose user1 wishes to view the tasks that have been completed, the following steps occur in database processing:

* Select all ID’s from ‘user\_task\_status’ where all values in ’user1’ are marked as “Done”.
* Store the selected ID’s in an arraylist
* Iterate through the arraylist and get the task name, priority, due date, due time and status of that particular ID.
* Store this data and display in the table.

The following figure shows all the tasks that have been completed:

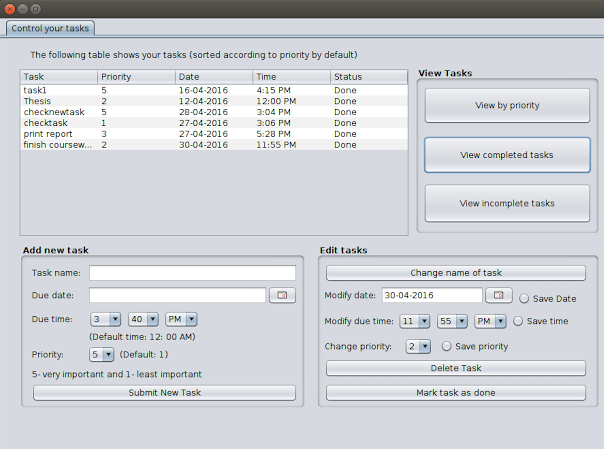


Figure 7.3.2 a) Tasks that are marked as ‘Done’

The flowchart on the following page shows how the algorithm was implemented to achieve this.

Page 19



* + 1. View incomplete tasks

In the GUI shown in figure 7 a) when the user clicks the button “View incomplete tasks”, the main table shows all the task details that have been marked as complete.

Suppose user1 wishes to view the tasks that are not done, the following steps occur in database processing:

* Select all ID’s from ‘user\_task\_status’ where all values in ’user1’ are marked as “not done”.
* Store the selected ID’s in an arraylist.
* Iterate through the arraylist and get the task name, priority, due date, due time and status of that particular ID.
* Store this data and display in the table.

The following figure on the next page shows the tasks that are marked as incomplete.

Page 20

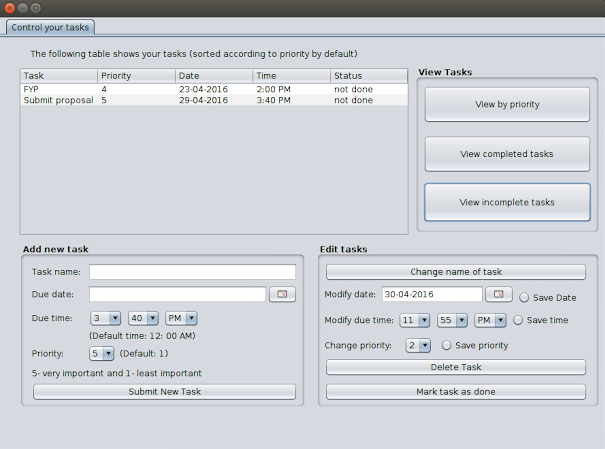


Figure 7.3.3 a) Tasks that are marked as ‘not done’

The following flowchart shows the algorithm used to implement this task:



Page 21

The reason for choosing the combo boxes for selecting the priority and time is so that there is no chance for the user to input exception data such as special characters like ‘\*’ or ‘@’ further more as for the time , in a 24 hour period there are many combinations of the current time which means a list needs to be provided to the user to select the appropriate time which is why the hour and the minute section has been separated else the list becomes even bigger which forces the user to scroll a lot more to find the appropriate time.

A date chooser has also been used which is part of an external library and its JAR folder needs to be included in the project path

1. Contacts directory

This is the 2nd main feature of this program. The user can save contacts in forms of names, phone number, Email contacts. The user can also edit the contacts such as changing the name of the contact, changing the phone number, changing the email contact, deleting the contact. The user can also view contacts that are sorted by names and the user can also search for contacts by typing the name of the contact in a text field.

The program here has been designed in such a manner that the phone number and the email contact cannot be duplicate because there is no point adding a contact which is already pre-existing. In case the user tries to add a pre-existing contact, a dialogue box appears that tells the user about pre-existing contact and the user must enter new credentials. The GUI of contacts directory looks like this:

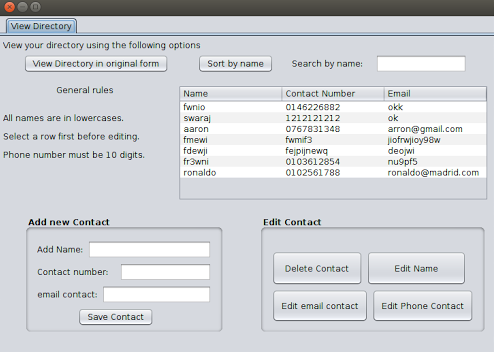


Figure 8 a) – Contacts directory GUI

8.1 Adding new contact

This feature allows the user to add a new contact which is inclusive of name, contact number and email and in order for the user to successfully save a new contact all the 3 fields shown in figure 8 a) under the “add new contact” section must be filled in, without filling all the 3 fields, the user cannot save the contact.

Page 22

Three main databases have been used to support this feature which are as follows:

* 1. ‘directory\_names’

This table stores the names of the people as part of contacts. The table below shows the design of the database:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | user1 | user2 | ... |
| 1 | joseph | zach | ... |
| 2 | Michael | Andrew | ... |
| ... | ... | ... | ... |

The following query is used to create the table:

CREATE TABLE directory\_names(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

The following MySQL query has been used to add column to this table:

“ALTER TABLE directory\_names ADD COLUMN new\_user\_name\_here varchar(100)”

Where ‘new\_user\_name\_here’ is the username of the new user just signed in.

The following query is used to add rows and insert names:

“INSERT INTO user\_task\_status(User name) VALUES(name)”

Where ‘User name’ is the name of the user currently signed in the application and ‘name’ as the name of the contact saved.

Each column represent a user and as new user signs up, a new column is made and also every time a new contact is added, a new row is inserted to this table with the ID automatically being incremented. The ID acts as a unique identifier of each contact.

* 1. ‘directory\_phone\_number’

The table designed here has the same design as ‘directory\_names’ table with the only difference being that the cells store phone numbers. The table looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **user1** | **user2** | **...** |
| 1 | 0103612855 | 0146226882 | ... |
| 2 | 0102561788 | 0102264722 | ... |
| ... | ... | ... | ... |

Page 23

The following query is used to create the table:

CREATE TABLE directory\_phone\_number(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

The following MySQL query has been used to add column to this table:

“ALTER TABLE directory\_phone\_number ADD COLUMN new\_user\_name\_here varchar(100)”

Where ‘new\_user\_name\_here’ is the username of the new user just signed in.

The following query is used to add rows and insert phone numbers:

“INSERT INTO directory\_phone\_number(User name) VALUES(Number)”

Where ‘User name’ is the name of the user currently signed in the application and ‘Number’ as the number of the contact saved.

Each column represent a user and as new user signs up, a new column is made and also every time a new contact is added, a new row is inserted to this table with the ID automatically being incremented. The ID acts as a unique identifier of each contact.

* 1. ‘directory\_emails’

This table stores the email of the person as part of contacts. The database design for this table is the same as ‘directory\_names’. The table looks like:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | user1 | user2 | ... |
| 1 | [joseph@gmail.com](mailto:joseph@gmail.com) | zach@yahoo.com | ... |
| 2 | [michael@yahoo.com](mailto:michael@yahoo.com) | andrew@yahoo.com | ... |
| ... | ... | ... | ... |

CREATE TABLE directory\_emails(

ID INTEGER PRIMARY KEY   AUTOINCREMENT

);

The following MySQL query has been used to add column to this table:

“ALTER TABLE directory\_emails ADD COLUMN new\_user\_name\_here varchar(100)”

Where ‘new\_user\_name\_here’ is the username of the new user just signed in.

The following query is used to add rows and insert emails:

“INSERT INTO directory\_phone\_number(User name) VALUES(Email)”

Where ‘User name’ is the name of the user currently signed in the application and ‘Email’ as the email of the contact saved.

Page 24

All the 3 database design discussed above have the same design as the to-do list manager databases and literally the only difference is the content or the type of data each table stores.

* 1. Editing a contact

In this feature, there are 4 main things to edit which are deleting the contact, editing the name, email and phone number of contact.

* + 1. Editing name of contact

In order to edit the name of the contact, the user must select the table row first and only then click the button ‘edit name’. Upon selecting the row in the table, the email stored in that row is fetched and stored in a temporary variable as the email acts as a unique identifier here because duplicate emails cannot be saved in this table and then upon clicking the button, the database processing occurs so suppose user1 wants to change the name ‘joseph’ to ‘john’ the following steps occur:

* Search for the current email in the table ‘directory\_emails’ which is joseph@gmail.com in this case.
* Fetch the associative ID of the match found which is ID=1 in this case.
* Update user1 column value in table ‘directory\_names’ to ‘john’ where ID=1.
  + 1. Editing email of contact

In order to edit the name of the contact, the user must select the table row first and only then click the button ‘edit email contact’.  Upon selecting the row in the table, the email stored in that row is fetched and stored in a temporary variable as the email acts as a unique identifier here because duplicate emails cannot be saved in this table and then upon clicking the button, the database processing occurs so suppose user1 wants to change the email [joseph@gmail.com](mailto:joseph@gmail.com) to [john@gmail.com](mailto:john@gmail.com) the following steps occur:

* Search for the current email in the table ‘directory\_emails’ which is joseph@gmail.com in this case.

“SELECT user1 FROM directory\_emails”

* Fetch the associative ID of the match found which is ID=1 in this case.

“SELECT ID FROM directory\_emails WHERE user1=’joseph@gmail.com’”

* Update user1 column value in table ‘directory\_emails’ to ‘john@gmail.com’ where ID=1.

“UPDATE directory\_emails SET user1=’[john@gmail.com](mailto:john@gmail.com)’ WHERE ID=1”

Page 25

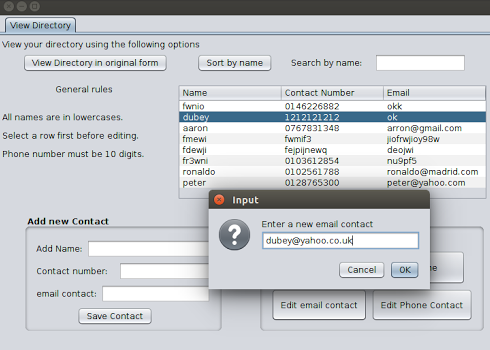


Figure 8.2.2 a) Changing email of name ‘dubey’ to ‘dubey@yahoo.co.uk’

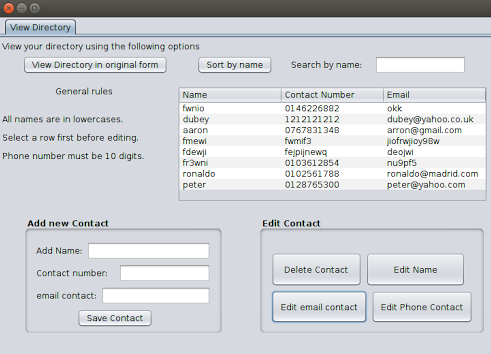


Figure 8.2.2 b) Changed email of name ‘dubey’ to ‘dubey@yahoo.co.uk’

* + 1. Editing phone number of contact

In order to edit the name of the contact, the user must select the table row first and only then click the button ‘edit email contact’.  Upon selecting the row in the table, the email stored in that row is fetched and stored in a temporary variable as the email acts as a unique identifier here because duplicate emails cannot be saved in this table and then upon clicking the button, the database processing occurs so suppose user1 wants to change the phone contact 0103612855 which is at ID=1 in user1 column to 0103612854 then the following steps occur:

Page 26

* Search for the current email in the table ‘directory\_emails’ which is joseph@gmail.com in this case.

“SELECT user1 FROM directory\_emails”

* Fetch the associative ID of the match found which is ID=1 in this case.

“SELECT ID FROM directory\_emails WHERE user1=’joseph@gmail.com’”

* Update user1 column value in table ‘directory\_phone\_numbers’ to ‘0103612854’ where ID=1.

“UPDATE directory\_phone\_numbers SET user1=’0103612854’ WHERE ID=1”

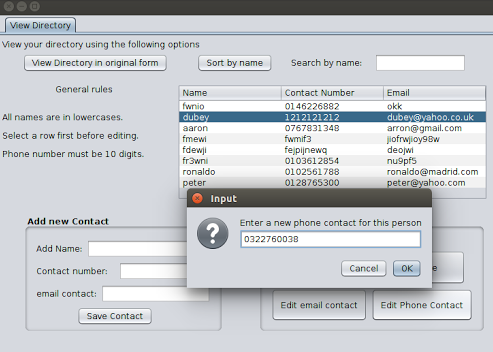


Figure 8.2.3 a) Changing number of name ‘dubey’ to 0322760038

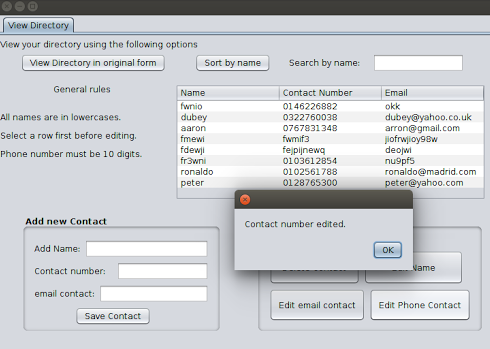


Figure 8.2.3 b) Changed number of name ‘dubey’ to 0322760038

Page 27

* + 1. Delete contact

In order to delete a contact, the entire record must be swiped off meaning, the name of contact, phone number, Email contact. All that needs to be done by the user here is to select the row in the table and click the button ‘Delete Contact’. Suppose user1 wishes to delete the contact whose email is ‘joseph@gmail.com’, the database processing occurs as follows:

* Search for ‘[joseph@gmail.com](mailto:joseph@gmail.com)’ in table ‘directory\_emails’.

“SELECT user1 FROM directory\_emails”

* Find associative ID of the match found which is ID=1 in this case.

“SELECT ID FROM directory\_emails WHERE user1=’joseph@gmail.com’”

* The value in ‘user1’ column at ID=1 in table ‘directory\_names’ is deleted.

“DELETE FROM directory\_names WHERE ID=1”

* The value in ‘user1’ column at ID=1 in table ‘directory\_phone\_numbers’ is deleted.

“DELETE FROM directory\_phone\_numbers WHERE ID=1”

* The value in ‘user1’ column at ID=1 in table ‘directory\_emails’ is deleted.

“DELETE FROM directory\_emails WHERE ID=1”

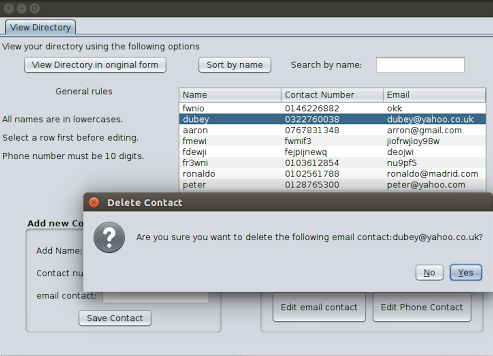


Figure 8.2.4 a) attempting to delete the contact with email: ‘dubey@yahoo.co.uk’

Page 28

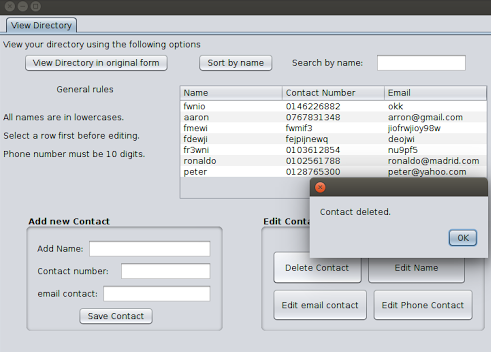


Figure 8.2.4 b) successfully deleted the contact with email: ‘dubey@yahoo.co.uk’

The following flowchart shows the algorithm used to implement this task:



* 1. Sorting contacts by name

Upon clicking the button ‘Sort by name’ all the contacts are displayed in the table that are sorted by names which means the names displayed are in alphabetical order.

Page 29

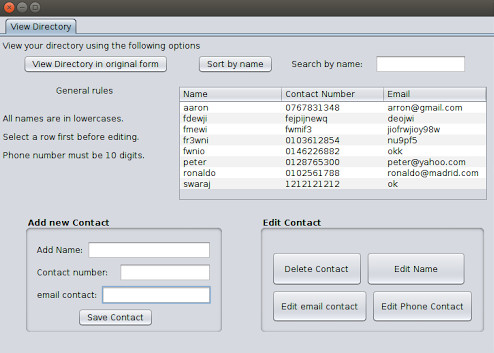


Figure 8.3 a) – Contact sorted by names in alphabetical order

The following flowchart shows the algorithm used to implement this task:



* 1. Searching contact by typing name in text field

In the GUI shown in figure 8 a), the user can search contacts by using the text field in the top right hand corner. As the user starts entering characters, the program finds the best match in the main database outputs the contacts:

Page 30

If the user presses ‘f’ then all names starting with ‘f’ are displayed as shown below:

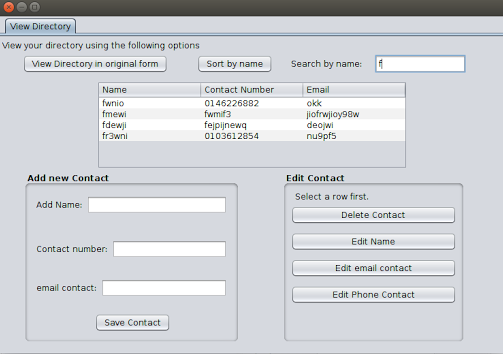


Figure 8.4 a) – All contacts with names starting with ‘f’

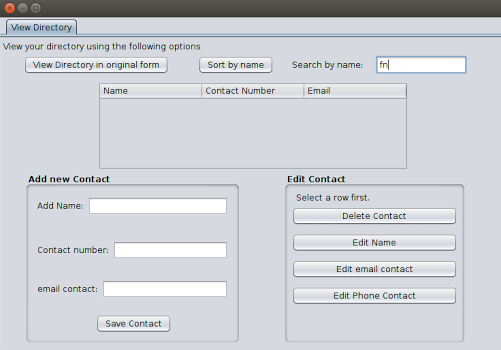


Figure 8.4 b) – All contacts with names starting with ‘fn’

No contacts are shown here because there are no names in the database that start with ‘fn’.

The following flowchart on the next shows the algorithm used to implement this task:

Page 31



1. Software engineering practices used

Due to the massive size of the program, separate classes have been used for separate functions for contacts directory and to-do list manager.

9.1 Inheritance

Inheritance is also used heavily in this program mainly because of the following reason: all the child classes of ‘SQLiteDatabaseConnection.java’ are also required to do database processing which means these child classes will need to inherit the following:

* ‘Connection’ class variable for connecting to the database.
* ‘Statement’ class variable for executing MySQL queries.
* ‘ResultSet’ class variable for fetching data from the database.

This is the main reason inheritance is used in this program i.e. mainly to inherit the database connection objects.

* 1. Polymorphism

Polymorphism is also very heavily used and is perhaps the most important aspect of this program else there is no way the program can run based on the design that is implemented. The reason is that when the central database is created, it is done so using just one object that is of type SQLiteDatabaseConnection and hence in order to write MySQL queries and execute them, it has to be done using this one object hence what is done is this object is passed to the main ToDoListManager.java and ContactsDIrectory.java file and the parent class i.e. ‘SQLiteDatabaseConnection.java’ is made to point to the child classes which are basically

Page 32

the subclasses of ToDoListmanger.java and ContactsDirectory.java having passed the main database connection object and making the parent class point to the child classes , the child classes can now use this one main database connection object to execute MySQL queries.

* 1. Encapsulation

Encapsulation has also been used widely throughout the program for data integrity purposes. All the functions that do not need to return values to other files use member variables that are private which basically means that all the void functions used private members. None of the variables are public as for this project not a single variable only protected and private variables have been used, protected members for inheritance and private members just for in class use.

1. External libraries included in the project

* MySQLite JDBC driver

This driver upon including in the project path allows the java program to create and establish a connection with a database, the driver also enables the programmer to use pure MySQL queries to manipulate the database.

* JCalendar package

Upon including this package’s JAR folder into the project path, the java program would now be able to use all widgets related to the calender, one of them being the JDateChooser that allows the user to pick dates in any format.

Page 33