Design Document

CMPUT 291: Mini Project 1

GROUP c291g19: Kelly Chin (1391757), Calvin Ho (1436255), Jessica Huynh (1394832)

Table of Contents

[General Overview 2](#_Toc465376671)

[User Guide 2](#_Toc465376672)

[Initial welcome screen 2](#_Toc465376673)

[Adding a new user 2](#_Toc465376674)

[Login as an existing user 2](#_Toc465376675)

[Doctor 2](#_Toc465376676)

[Nurse 2](#_Toc465376677)

[Administrator 2](#_Toc465376678)

[User Guide Diagram 2](#_Toc465376679)

[Detailed Design 2](#_Toc465376680)

[File: mini\_proj\_1.py 2](#_Toc465376681)

[Function: main() 2](#_Toc465376682)

[Functions: addUsers() & addUserSQL() 3](#_Toc465376683)

[File: sqlConnection.py 3](#_Toc465376684)

[Function: openConnection() 3](#_Toc465376685)

[Function: closeConnection() 3](#_Toc465376686)

[File: userInfo.py 3](#_Toc465376687)

[File: UserController.py 3](#_Toc465376688)

[Functions: userController(…), and other controllers 3](#_Toc465376689)

[File: dnaActions.py 3](#_Toc465376690)

[Files: testing .sql files 3](#_Toc465376691)

[Testing Strategy 3](#_Toc465376692)

[General Strategy 3](#_Toc465376693)

[Test Case Coverage 3](#_Toc465376694)

[Bugs & Assumptions 4](#_Toc465376695)

[Group Work Break-Down Strategy 4](#_Toc465376696)

[Kelly Chin 4](#_Toc465376697)

[Calvin Ho 4](#_Toc465376698)

[Jessica Huynh 4](#_Toc465376699)

# General Overview

This program implements a hospital database that allows users (doctors, nurses, and administration) to login and execute a number of tasks respective to their role in the hospital. The program also allows a user to be added to the database.

## User Guide

### Initial welcome screen

Prompts the user to select whether they want to login as an existing user, add a user to the database or quit the program.

### Adding a new user

Select role of new user [d]octor, [n]urse, [a]dmin that you wish to add. Then add user information (i.e. name, username, and password). Note: if you open the database in a visual manager, the password for each staff will be encrypted.

### Login as an existing user

Login with valid username and password

### Doctor

Specific tasks: [0] List all charts for a given patient, [1] Add a symptom, [2] Add a diagnosis, and [3] Add a medication. These will prompt you for required information.

### Nurse

Specific tasks: [0] List all charts for a given patient, [1] Add a symptom, [2] Create a new chart, and [3] Close an open chart. These will prompt you for required information.

### Administrator

Specific tasks: [0] Create report for all doctor prescriptions, [1] List total amount prescribed for specific drug, [2] List all possible medications for a specific diagnosis, and [3] List all diagnoses made prior to prescribing specific drug. These will prompt you for required information.

# Detailed Design

## File: mini\_proj\_1.py

### Function: main()

First function to be called when the program is started. It calls other functions that prompt user for what action (login, add, quit) to execute. Calls for ‘promptForLoginInfo’ to ask the user for username and password when user enters login command. When user enters their username and password, the function ‘verifyLoginInfo’ is called to check if the inputted information is accurate and exists. If the information is valid, it calls the ‘userController’ function to display available actions for that user and takes in a command for the action that the user prompted for. If the username and password information is invalid, the user is notified, and prompted to re-enter their information.

### Functions: addUsers() & addUserSQL()

If the user inputs the add command, the ‘addUsers’ function is called to ask for the new user role, name, username, and password, and then that user is added into the database. Once the new user has been added successfully, this ‘main’ function is called again.

## File: sqlConnection.py

### Function: openConnection()

Returned the connection and cursor to be used throughout the code. It also creates a new hospital database if one is not found.

### Function: closeConnection()

This commits and closes the connection to the database.

## File: userInfo.py

These are the prompts for initial user inputs

## File: UserController.py

### Functions: userController(…), and other controllers

Depending on the user’s role a controller is invoked (doctor, nurse, admin). Displays possible actions, and reads the action command the user enters to perform that task. Once the task is complete, it prompts for action command again.

## File: dnaActions.py

These functions are the back-end implementations of the actions each user role can do. This is where the sql queries are used to retrieve, input, and update values in the database. Note: file Admin.py is just separated to keep the files shortened and disassociated.

## Files: testing .sql files

There are two main .sql files that test the functionality of the sql queries for the doctor and the administrator. The nurse was tested mainly with the interface.

# Testing Strategy

## General Strategy

Our approach to the test the functionally of our code was done in two ways. For the SQL queries that were used to accomplish the tasks for each user (doctor, nurse, and administrator), they had been tested as if they were .sql files. In other words, these queries were tested without integration to a host programming language (i.e. Python). Once the queries passed the tests, they were merged into the python host source files. It was then further tested with the interface to ensure that the correct data was displayed.

## Test Case Coverage

Most of the test cases did not include garbage data being inputted through the user interface. For the sql portions which were tested individually, the database was populated in such a way to quickly verify if our queries were correct in very small data sets, often expecting 1 or 2 results. After porting the sql to the python user interface, the queries were tested again to ensure it was working seamlessly. Encryption was visually verified using the db manager .

## Bugs & Assumptions

Important validations were completed such as valid hcno for patients, valid chart id’s, etc. It was under the assumption that the user would not input garbage data and so no checks for these errors were written. In order to create a medication entry, a corresponding entry for that drug must exist in the drug table beforehand.

# Group Work Break-Down Strategy

This mini project was coordinated and tracked using Github. Git issues were created, assigned, and closed as a to-do list as each member tracked their tasks. The versioning of different code was addressed using Git branches.

## Kelly Chin

Main responsibilities were doctor tasks.

* Initial login interface
  + Base implementation asking to log in or add user, Time spent: 1 hour
* Doctor SQL Queries
  + Task 1: List all charts for a given patient, Time spent: 2 hours
  + Task 2: Add a symptom, Time spent: 20 minutes
  + Task 3: Add a diagnosis, Time spent: 20 minutes
  + Task 4: Add a medication, Time spent: 1.5 hours

## Calvin Ho

Main responsibilities were administration tasks.

* Administrator SQL Queries
  + Task 1: Create report for all doctor prescriptions, Time spent: 1 hour
  + Task 2: List all prescriptions for specific drug, Time spent: 45 minutes
  + Task 3: List all possible medications for a specific drug, Time spent: 1 hour
  + Task 4: List all diagnoses made prior to prescribing drug, Time spent: 45 minutes
* Design Documentation
  + Detailed Design, Time spent 1 hour

## Jessica Huynh

Main responsibilities were python interface and nurse tasks.

* Error Checking
  + Throughout (i.e. check if exists, check valid), Time spent: 2 hours
* Adding a user
  + SQL and interface code, Time spent: 10 minutes
* Login as an existing user
  + Password Encryption, Time spent: 3 minutes
  + Prompting Actions that respective user (D, N, A) can do, Time spent: 1 hour
* Nurse SQL Queries
  + Task 2: Create a new chart, Time spent: 30 minutes
  + Task 3: Close an open chart, Time spent: 30 minutes
* Design Documentation, Time spent: 2.5 hours