**CERTIFICATE**

**ACKNOWLEGMENT**

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **CONTENTS**  **INDEX** | **PAGE NO.** |
| 1. | INTRODUCTION |  |
| 2. | DESCRIPTION |  |
| 3. | MODULES |  |
| 4. | METHODOLOGY |  |
| 5. | CODING |  |
| 6. | SCREENSHOTS |  |
| 7. | SYSTEM REQUIREMENTS |  |
| 8. | ADVANTAGES AND DISADVANTAGES |  |
| 9. | CONCLUSION |  |
| 10. | REFERENCES |  |

1. **INTRODUCTION**

The **Organ Donation Management System (ODMS)** is developed mainly for general hospitals (GH), clinics and other health centres to manage the donor registration and user maintenance.

The public can retrieve information about organ donation through this desktop application. People who are interested in organ donation can register themselves through this system.

The application will be processed by the administrator and each donor will receive feedback about their application status. The donor can visit specified medical centres to perform a medical check-up or can ask for a medical assistance at home.

Only administrator has the authority and privileges to print organ list report and total donation report according to district from this system. An analysis study has been done based on the current manual system and all the problems statements and requirements have been identified.

Moreover, ODMS is three tier architecture systems which involve client tier, business tier and database management tier. The interfaces for ODMS have been designed according to the requirement and needs of the current market.

This Online Organ Donation Management System will help to improve the current situation and overcome the problems that arise nowadays.

**2. DESCRIPTION**

**Uses :**

**Application:**

The ODMS can be implemented in various sectors such as:

Hospital management system where the record section can keep organ transplant

data in secure state.

Government Agencies to keep vigil eyesight over organ transplantations thus ensuring arrest on illegal tissue trafficking.

Government schemes to promote organ donation.

**3. MODULES**

There are 3 major modules in ODMS as:

**1.Registration:**

In this module the registration of both donor and receiver will we carried out. Each user will get their unique Id type which they will use to log-in to system.

**2. Login:**

In this module the user can log into his/her profile, carry out periodic updates, and modify profile. The log-in Id will be one provide during the profile registration.

The login module is further divided into three sub modules.

The modules to pop up will be defined by the log-in Id type.

2A: Donor login:

This module will allow updates to the donor profile.

The user can report any recent updates to his medical conditions.

2B: Receiver login:

This module will allow updates to the receiver profile.

The hospitals, mediators will have partial control over such profiles.

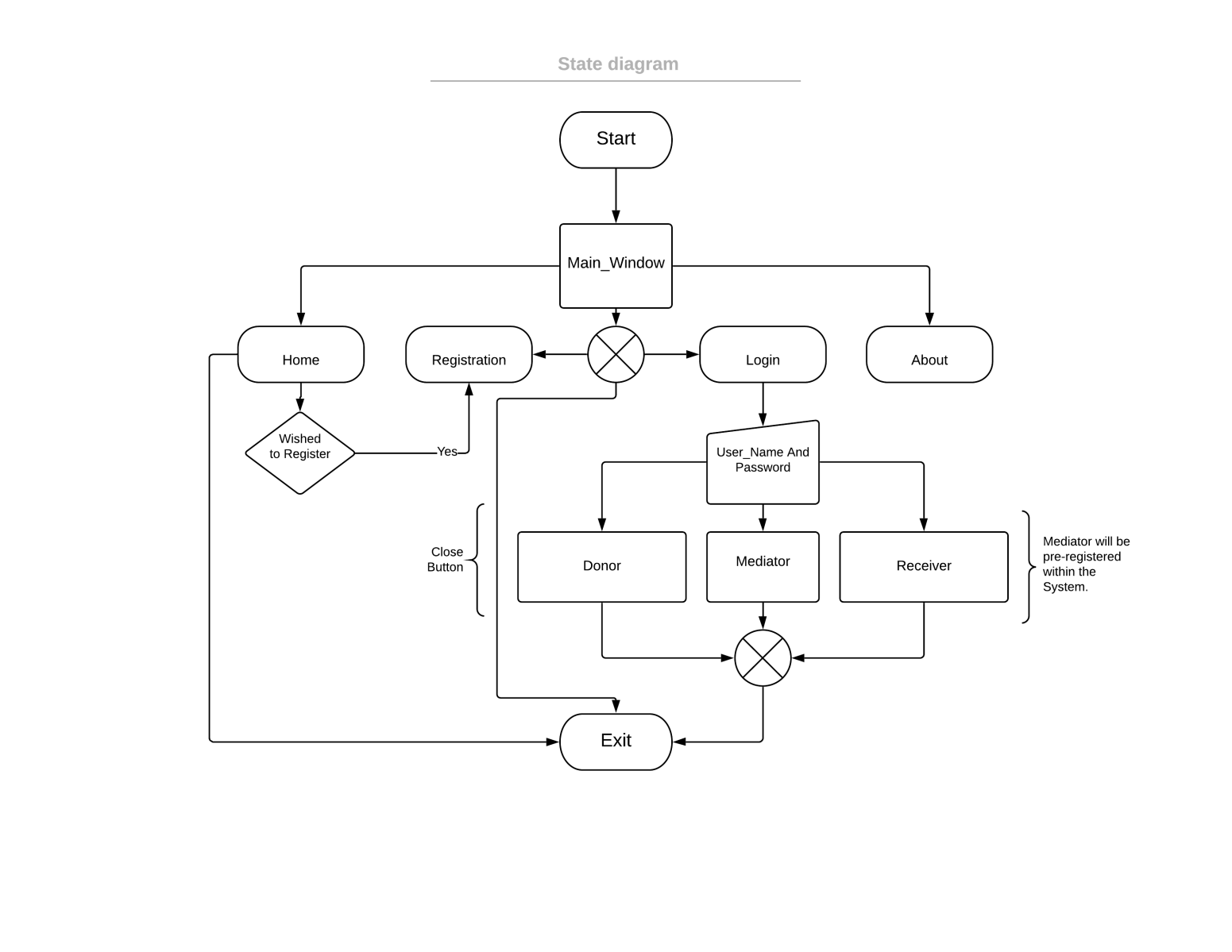
2C: Mediator login:

These profiles will be internally generated in the system.

Such profiles will not have any registration platform.

Mediator could be hospitals, GC, governing bodies.

**4. Methodology**



**5. CODING**

|  |
| --- |
|  |
|  | import datetime  import re |
|  | import sqlite3 |
|  | import sys |
|  |  |
|  | from PyQt5.QtWidgets import QMessageBox |
|  | from PyQt5 import QtWidgets, uic |
|  | from PyQt5 import QtGui,QtCore |
|  |  |
|  |  |
|  | def delete\_record(): |
|  | noise() |
|  |  |
|  |  |
|  | class Test: |
|  | cursor: sqlite3.Cursor |
|  |  |
|  | def \_\_init\_\_(self): |
|  | self.ui = uic.loadUi('organ.ui') |
|  | self.ui.show() |
|  | self.create\_database() |
|  | self.ui.mainTab.setCurrentIndex(0) |
|  | self.ui.changeLoginTypeStack.setCurrentIndex(0) |
|  | self.ui.changeFormStack.setCurrentIndex(0) |
|  | self.ui.adminLoginStackTab.setCurrentIndex(0) |
|  | self.perform\_table\_sizeing() |
|  | self.about\_tab() |
|  | self.ui.loginButton.clicked.connect(self.login) |
|  | self.ui.homeTabRegisterButton.clicked.connect(self.tab\_change) |
|  | self.ui.formSelecter.currentIndexChanged.connect(self.change\_form) |
|  | self.ui.registerTabRegisterButton.clicked.connect(self.add\_record) |
|  | self.ui.renewApplicationButton.clicked.connect(self.renew\_application) |
|  | self.ui.goToLoginPageReceiver.clicked.connect(self.set\_login\_page) |
|  | self.ui.goToLoginPageDonor.clicked.connect(self.set\_login\_page) |
|  | self.ui.goToLoginPageAdmin.clicked.connect(self.set\_login\_page) |
|  | self.ui.adminLoginStackTab.currentChanged.connect(self.populate\_record) |
|  | self.ui.searchRecord.clicked.connect(self.admin\_populate) |
|  | self.ui.deleteRecord.clicked.connect(delete\_record) |
|  |  |
|  | def login(self): |
|  | user\_name: str |
|  | password: str |
|  | self.ui.statusNameLabel.setText('') |
|  | self.ui.statusPasswordLabel.setText('') |
|  | try: |
|  | user\_name = self.ui.userName.text() |
|  | password = self.ui.userPassword.text() |
|  | if user\_name == "" and password == "": |
|  | raise Exception('both') |
|  | if user\_name == "": |
|  | raise Exception('user\_name') |
|  | if password == "": |
|  | raise Exception('password') |
|  | except Exception as e: |
|  | if str(e) == 'both': |
|  | self.ui.statusNameLabel.setText('\* UserName Required') |
|  | self.ui.statusPasswordLabel.setText('\* Password Required') |
|  | if str(e) == 'user\_name': |
|  | self.ui.statusNameLabel.setText('\* UserName Required') |
|  | if str(e) == 'password': |
|  | self.ui.statusPasswordLabel.setText('\* Password Required') |
|  | else: |
|  | check = None |
|  | if 'Admin' in user\_name: |
|  | self.cursor.execute('select \* from loginId where username = ? and password = ?;', |
|  | (user\_name, password)) |
|  | check = self.cursor.fetchall() |
|  | if check: |
|  | self.populate\_record() |
|  | self.ui.changeLoginTypeStack.setCurrentIndex(1) |
|  | elif 'REC' in user\_name: |
|  | self.cursor.execute('select \* from loginId where userName = ? and password = ?;', |
|  | (user\_name, password)) |
|  | check = self.cursor.fetchall() |
|  | if check: |
|  | self.ui.changeLoginTypeStack.setCurrentIndex(3) |
|  | self.populate\_receiver(user\_name) |
|  | elif 'DON' in user\_name: |
|  | self.cursor.execute('select \* from loginId where userName = ? and password = ?;', |
|  | (user\_name, password)) |
|  | check = self.cursor.fetchall() |
|  | if check: |
|  | self.ui.changeLoginTypeStack.setCurrentIndex(2) |
|  | else: |
|  | self.ui.statusNameLabel.setText('Incorrect Username or Password') |
|  |  |
|  | def create\_database(self): |
|  | try: |
|  | self.connection = sqlite3.connect('organRecord.db') |
|  | self.cursor = self.connection.cursor() |
|  | self.create\_data() |
|  | self.ui.statusbar.showMessage("Connected to Database") |
|  | except Exception as e: |
|  | print(e) |
|  | self.ui.statusbar.showMessage(" Failed to Connect to Database: Exiting Application") |
|  | app.exit() |
|  |  |
|  | def create\_data(self): |
|  | try: |
|  | query = """CREATE TABLE IF NOT EXISTS donor(registrationNumber VARCHAR(5), firstName VARCHAR(20), |
|  | middleName VARCHAR(20), lastName VARCHAR(20), dateOfBirth DATE, gender VARCHAR(10), |
|  | bloodGroup VARCHAR(3), mobileNumber VARCHAR(15), landLineNumber VARCHAR(15), email VARCHAR(100), |
|  | address VARCHAR(255), city VARCHAR(30), district VARCHAR(30), state varchar(30), pinCode VARCHAR(6), |
|  | pledgeDate DATE,donatingOrgans VARCHAR(100));""" |
|  | self.cursor.execute(query) |
|  | self.connection.commit() |
|  | query = """CREATE TABLE IF NOT EXISTS receiver(registrationNumber VARCHAR(5), firstName VARCHAR(20), |
|  | middleName VARCHAR(20), lastName VARCHAR(20), dateOfBirth DATE, gender VARCHAR(10), |
|  | bloodGroup VARCHAR(3), mobileNumber VARCHAR(15), landLineNumber VARCHAR(15), email VARCHAR(100), |
|  | address VARCHAR(255), city VARCHAR(30), district VARCHAR(30),state varchar(30), pinCode VARCHAR(6), |
|  | disease VARCHAR(50),doctorName VARCHAR(50), doctorRegistration VARCHAR(30), illnessStage VARCHAR(15) |
|  | ,priority VARCHAR(20),transplantingOrgan VARCHAR(10));""" |
|  | self.cursor.execute(query) |
|  | self.connection.commit() |
|  | query = """CREATE TABLE IF NOT EXISTS loginId(userName varchar(10),password varchar(10));""" |
|  | self.cursor.execute(query) |
|  | query = """SELECT \* FROM loginId;""" |
|  | self.cursor.execute(query) |
|  | if not self.cursor.fetchone(): |
|  | query = """INSERT INTO loginId(userName, password) values('Admin','Admin@0');""" |
|  | self.cursor.execute(query) |
|  | self.connection.commit() |
|  | except Exception as e: |
|  | print(e) |
|  |  |
|  | def tab\_change(self): |
|  | self.ui.mainTab.setCurrentIndex(1) |
|  |  |
|  | def set\_login\_page(self): |
|  | self.ui.changeLoginTypeStack.setCurrentIndex(0) |
|  | self.ui.userName.clear() |
|  | self.ui.userPassword.clear() |
|  |  |
|  | def add\_record(self): |
|  | if self.ui.formSelecter.currentIndex() == 0: |
|  | data = [] |
|  | registration\_number = 'DON1' |
|  | self.cursor.execute("select registrationNumber from donor;") |
|  | if self.cursor.fetchone(): |
|  | self.cursor.execute("select registrationNumber from donor ORDER BY registrationNumber DESC LIMIT 1;") |
|  | old\_registration\_number = self.cursor.fetchone() |
|  | registration\_number = int(re.search(r'\d', old\_registration\_number[0]).group()) |
|  | registration\_number = 'DON' + str(registration\_number + 1) |
|  | data.append(registration\_number) |
|  | data.append(self.ui.fristName.text()) |
|  | data.append(self.ui.middleName.text()) |
|  | data.append(self.ui.lastName.text()) |
|  | date\_of\_birth = self.ui.dateOfBirth.date().toPyDate() |
|  | data.append(date\_of\_birth.strftime("%d-%m-%Y")) |
|  | if self.ui.radioFemale.isChecked(): |
|  | data.append('Female') |
|  | if self.ui.radioMale.isChecked(): |
|  | data.append('Male') |
|  | else: |
|  | data.append('Other') |
|  | data.append(self.ui.bloodGroup.currentText()) |
|  | data.append(self.ui.mobileNumber.text()) |
|  | data.append(self.ui.landLineNumber.text()) |
|  | data.append(self.ui.email.text()) |
|  | data.append(self.ui.permanantAddress.toPlainText()) |
|  | data.append(self.ui.addCity.text()) |
|  | data.append(self.ui.addState.currentText()) |
|  | data.append(self.ui.addDistrict.text()) |
|  | data.append(self.ui.addPincode.text()) |
|  | pledge\_date = datetime.datetime.now() |
|  | data.append(pledge\_date.strftime("%d-%m-%Y-%I-%M-%p")) |
|  | donate = '' |
|  | if self.ui.donateBody.isChecked(): |
|  | donate = donate + 'Body' |
|  | if self.ui.donateEyes.isChecked(): |
|  | donate = donate + 'Eyes' |
|  | if self.ui.donateKidney.isChecked(): |
|  | donate = donate + 'Kidney' |
|  | if self.ui.donatePancreas.isChecked(): |
|  | donate = donate + 'Pancreas' |
|  | if self.ui.donateHeart.isChecked(): |
|  | donate = donate + 'Heart' |
|  | if self.ui.donateLungs.isChecked(): |
|  | donate = donate + 'Lungs' |
|  | data.append(donate) |
|  | query = """INSERT INTO donor(registrationNumber , firstName ,middleName , lastName , |
|  | dateOfBirth , gender , bloodGroup , mobileNumber , landLineNumber , email , |
|  | address , city , district ,state, pinCode , pledgeDate ,donatingOrgans) |
|  | VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?);""" |
|  | self.cursor.execute(query, data) |
|  | self.connection.commit() |
|  | self.ui.statusbar.showMessage("Added Record") |
|  | word = data[1] |
|  | word2 = data[2] |
|  | word = word[2:5] + word2[2:5] |
|  | response = add\_sucess(registration\_number, word) |
|  | self.add\_login\_id(registration\_number, word) |
|  | if response: |
|  | self.renew\_application() |
|  | if self.ui.formSelecter.currentIndex() == 1: |
|  | data = [] |
|  | registration\_number = 'REC1' |
|  | self.cursor.execute("select registrationNumber from receiver;") |
|  | if self.cursor.fetchone(): |
|  | self.cursor.execute("select registrationNumber from receiver ORDER BY registrationNumber DESC LIMIT 1;") |
|  | old\_registration\_number = self.cursor.fetchone() |
|  | registration\_number = int(re.search(r'\d+', old\_registration\_number[0]).group()) |
|  | registration\_number = 'REC' + str(registration\_number + 1) |
|  | data.append(registration\_number) |
|  | data.append(self.ui.fristName.text()) |
|  | data.append(self.ui.middleName.text()) |
|  | data.append(self.ui.lastName.text()) |
|  | date\_of\_birth = self.ui.dateOfBirth.date().toPyDate() |
|  | data.append(date\_of\_birth.strftime("%d-%m-%Y")) |
|  | if self.ui.radioFemale.isChecked(): |
|  | data.append('Female') |
|  | if self.ui.radioMale.isChecked(): |
|  | data.append('Male') |
|  | else: |
|  | data.append('Other') |
|  | data.append(self.ui.bloodGroup.currentText()) |
|  | data.append(self.ui.mobileNumber.text()) |
|  | data.append(self.ui.landLineNumber.text()) |
|  | data.append(self.ui.email.text()) |
|  | data.append(self.ui.permanantAddress.toPlainText()) |
|  | data.append(self.ui.addCity.text()) |
|  | data.append(self.ui.addState.currentText()) |
|  | data.append(self.ui.addDistrict.text()) |
|  | data.append(self.ui.addPincode.text()) |
|  | data.append(self.ui.patientDisease.text()) |
|  | data.append(self.ui.doctorName.text()) |
|  | data.append(self.ui.doctorRegistration.text()) |
|  | data.append(self.ui.illnessStage.currentText()) |
|  | data.append(self.ui.patientPriority.currentText()) |
|  | data.append(self.ui.transplantOrgan.currentText()) |
|  | query = """INSERT INTO receiver(registrationNumber , firstName ,middleName , lastName , |
|  | dateOfBirth , gender , bloodGroup , mobileNumber , landLineNumber , email , |
|  | address , city , district , state, pinCode, disease, doctorName, doctorRegistration,illnessStage, |
|  | priority, transplantingOrgan) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?);""" |
|  | self.cursor.execute(query, data) |
|  | self.connection.commit() |
|  | word = data[1] |
|  | word2 = data[2] |
|  | word = word[2:5] + word2[2:5] |
|  | response = add\_sucess(registration\_number, word) |
|  | self.add\_login\_id(registration\_number, word) |
|  | if response: |
|  | self.renew\_application() |
|  |  |
|  | def change\_form(self): |
|  | self.ui.changeFormStack.setCurrentIndex(self.ui.formSelecter.currentIndex()) |
|  |  |
|  | def add\_login\_id(self, registration, word): |
|  | query = """insert into loginId (userName, password) values(?,?);""" |
|  | self.cursor.execute(query,(registration, word)) |
|  |  |
|  | def populate\_record(self): |
|  | try: |
|  | donor\_query = """select \* from donor;""" |
|  | self.cursor.execute(donor\_query) |
|  | donor\_table = self.cursor.fetchall() |
|  | required\_rows = len(donor\_table) |
|  | self.ui.adminLoginStackDonorTable.setRowCount(required\_rows) |
|  | x = -1 |
|  | for i in donor\_table: |
|  | x = x + 1 |
|  | for j in range(0, 17): |
|  | self.ui.adminLoginStackDonorTable.setItem(x, j, QtWidgets.QTableWidgetItem(i[j])) |
|  |  |
|  | receiver\_query = """select \* from receiver;""" |
|  | self.cursor.execute(receiver\_query) |
|  | receiver\_table = self.cursor.fetchall() |
|  | required\_rows = len(receiver\_table) |
|  | self.ui.adminLoginStackReceiverTable.setRowCount(required\_rows) |
|  | x = -1 |
|  | for i in receiver\_table: |
|  | x = x + 1 |
|  | for j in range(0, 21): |
|  | self.ui.adminLoginStackReceiverTable.setItem(x, j, QtWidgets.QTableWidgetItem(i[j])) |
|  | except Exception as e: |
|  | print(e) |
|  |  |
|  | def populate\_receiver(self, user\_id): |
|  | try: |
|  | query = """select registrationNumber,transplantingOrgan,priority,bloodGroup,doctorName from receiver;""" |
|  | self.cursor.execute(query) |
|  | waiting\_table = self.cursor.fetchall() |
|  | required\_rows = len(waiting\_table) |
|  | self.ui.receiverLoginStackTable.setRowCount(required\_rows) |
|  | x = -1 |
|  | for i in waiting\_table: |
|  | x = x + 1 |
|  | for j in range(0, 5): |
|  | self.ui.receiverLoginStackTable.setItem(x, j, QtWidgets.QTableWidgetItem(i[j])) |
|  | query = """select transplantingOrgan from receiver where registrationNumber=?; """ |
|  | self.cursor.execute(query,(user\_id,)) |
|  | value = self.cursor.fetchall() |
|  | query = """select registrationNumber from receiver where transplantingOrgan=?;""" |
|  | self.cursor.execute(query,value[0]) |
|  | organ = self.cursor.fetchall() |
|  | count = 1 |
|  | for i in organ: |
|  | if i[0] is None: |
|  | Continue |
|  | elif i[0] == user\_id: |
|  | self.ui.place.setText(str(count)) |
|  | else: |
|  | count = count + 1 |
|  | except Exception as e: |
|  | print(e) |
|  |  |
|  | def renew\_application(self): |
|  | self.ui.fristName.clear() |
|  | self.ui.middleName.clear() |
|  | self.ui.lastName.clear() |
|  | self.ui.bloodGroup.setCurrentIndex(0) |
|  | self.ui.mobileNumber.clear() |
|  | self.ui.landLineNumber.clear() |
|  | self.ui.email.clear() |
|  | self.ui.permanantAddress.clear() |
|  | self.ui.addCity.clear() |
|  | self.ui.addState.setCurrentIndex(0) |
|  | self.ui.addDistrict.clear() |
|  | self.ui.addPincode.clear() |
|  | self.ui.patientDisease.clear() |
|  | self.ui.doctorName.clear() |
|  | self.ui.doctorRegistration.clear() |
|  | self.ui.donateBody.setChecked(False) |
|  | self.ui.donateEyes.setChecked(False) |
|  | self.ui.donateKidney.setChecked(False) |
|  | self.ui.donatePancreas.setChecked(False) |
|  | self.ui.donateHeart.setChecked(False) |
|  | self.ui.donateLungs.setChecked(False) |
|  | self.ui.illnessStage.setCurrentIndex(0) |
|  | self.ui.patientPriority.setCurrentIndex(0) |
|  | self.ui.transplantOrgan.setCurrentIndex(0) |
|  |  |
|  | def admin\_populate(self): |
|  | try: |
|  | if self.ui.adminRecordText.text() == '': |
|  | raise Exception("blankRecord") |
|  | else: |
|  | record = self.ui.adminRecordText.text() |
|  | if 'DON' in record: |
|  | query = """SELECT firstName, middleName, lastName, mobileNumber, dateOfBirth, gender, email |
|  | FROM donor WHERE registrationNumber = ?;""" |
|  | if query is None: |
|  | raise Exception("User Does not exists") |
|  | elif 'REC' in record: |
|  | query = """SELECT firstName, middleName, lastName, mobileNumber, dateOfBirth, gender, email, |
|  | priority FROM receiver WHERE registrationNumber = ?;""" |
|  | if query is None: |
|  | raise Exception("User Does not exists") |
|  | else: |
|  | raise Exception("Invalid Record ID type") |
|  | self.cursor.execute(query, (record,)) |
|  | data = self.cursor.fetchall() |
|  | for i in data: |
|  | self.ui.adminTabName.setText(i[0] + ' ' + i[1] + ' ' + i[2]) |
|  | self.ui.adminTabNumber.setText(i[3]) |
|  | self.ui.adminTabDob.setText(i[4]) |
|  | self.ui.adminTabGender.setText(i[5]) |
|  | self.ui.adminTabEmail.setText(i[6]) |
|  | if len(i) == 8: |
|  | self.ui.adminTabPriority.setCurrentText(i[7]) |
|  | except Exception as e: |
|  | do = mesa() |
|  | do.call(str(e)) |
|  |  |
|  | def perform\_table\_sizeing(self): |
|  | self.ui.receiverLoginStackTable.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents) |
|  | self.ui.receiverLoginStackTable.resizeColumnsToContents() |
|  | self.ui.adminLoginStackDonorTable.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents) |
|  | self.ui.adminLoginStackDonorTable.resizeColumnsToContents() |
|  | self.ui.adminLoginStackReceiverTable.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents) |
|  | self.ui.adminLoginStackReceiverTable.resizeColumnsToContents() |
|  |  |
|  | def about\_tab(self): |
|  | image = QtGui.QPixmap('about\_image.jpg') |
|  | image = image.scaled(400, 400) |
|  | self.ui.aboutImage.setPixmap(image) |
|  | txt = "Copyright \u00A9 2019 CSE 3rd year corporation.\n All rights reserved." |
|  | self.ui.copyright.setText(txt) |
|  |  |
|  | def \_\_del\_\_(self): |
|  | self.connection.commit() |
|  | self.connection.close() |
|  |  |
|  |  |
|  | class add\_sucess(QtWidgets.QWidget): |
|  | def \_\_init\_\_(self, name, word): |
|  | super().\_\_init\_\_() |
|  | self.title = 'ODMS' |
|  | self.left = 300 |
|  | self.top = 300 |
|  | self.width = 320 |
|  | self.height = 200 |
|  | self.initUI(name, word) |
|  | self.show() |
|  |  |
|  | def initUI(self, user, password): |
|  | self.setWindowTitle(self.title) |
|  | self.setGeometry(self.left, self.top, self.width, self.height) |
|  | msg = QMessageBox() |
|  | msg.setIcon(QMessageBox.Information) |
|  | msg.setText("Record Added Sucessfully\n Do you want to add another") |
|  | msg.setInformativeText("USERNAME: " + user + "\nPASSWORD: " + password) |
|  | msg.setWindowTitle("Record Added") |
|  | msg.setStandardButtons(QMessageBox.Yes | QMessageBox.No) |
|  | buttonreply = msg.exec\_() |
|  | if buttonreply == QMessageBox.Yes: |
|  | return True |
|  | else: |
|  | return False |
|  |  |
|  |  |
|  | class noise(QtWidgets.QWidget): |
|  | def \_\_init\_\_(self, parent=None): |
|  | super(noise, self).\_\_init\_\_(parent) |
|  | layout = QtWidgets.QVBoxLayout() |
|  | self.setWindowTitle("Delete Record") |
|  | self.setGeometry(300, 300, 300, 200) |
|  | self.l1 = QtWidgets.QLabel("Confirm delete Record") |
|  | self.l2 = QtWidgets.QLabel("State reason to delete Record") |
|  | self.b1 = QtWidgets.QPushButton("Ok") |
|  | self.b1.clicked.connect(self.som) |
|  | layout.addWidget(self.l1) |
|  | layout.addWidget(self.l2) |
|  | layout.addWidget(self.b1) |
|  | self.setLayout(layout) |
|  | self.show() |
|  |  |
|  | def som(self): |
|  | Pass |
|  |  |
|  | def \_\_del\_\_(self): |
|  | Pass |
|  |  |
|  |  |
|  | class mesa(QtWidgets.QWidget): |
|  | def \_\_init\_\_(self, parent=None): |
|  | super(mesa, self).\_\_init\_\_(parent) |
|  | layout = QtWidgets.QVBoxLayout() |
|  | self.setLayout(layout) |
|  |  |
|  | def call(self, e): |
|  | choice = QtWidgets.QMessageBox.question(self, 'Warning', e, |
|  | QtWidgets.QMessageBox.Ok) |
|  | if choice == QtWidgets.QMessageBox.Yes: |
|  | print("Extracting Naaaaaaoooww!!!!") |
|  | sys.exit() |
|  | else: |
|  | Pass |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | app = QtWidgets.QApplication(sys.argv) |
|  | win = Test() |
|  | sys.exit(app.exec\_()) |

**6. SCREENSHOTS**

**7. SYSTEM REQUIREMENTS**

**7.1 Software Requirements**

The minimum software requirement specifications for developing this project are as follows:

Programming Language: Python 3.7

Development Environment: Pycharm community Edition 2019

IDE of Jet Brains s.r.o

Backend Database: SQL Lite

UI Interfacing: PyQt

VCS (Version Control System): GITHUB of Microsoft Corporation.

**7.2 Hardware Requirements**

The minimum hardware requirement specifications for developing this project are as follows:

Processor: Intel core Duo 1.2 GHz or greater

Or AMD Ryzen 3 2.0 GHz

Hardisk: 20 MB of free space

Ram: 2 GB DDR3 or greater

Host OS: Windows 7 SP1 or greater Ubuntu Linux 16.07

**8. ADVANTAGES AND DISADVANTAGES**

**8.1 Advantages**

**8.2 Disadvantages**

**9. CONCLUSION**

Hence we have successfully developed Organ Donation Management System using the python 3.7 programming language. This software meets the objective and goals proposed earlier in the report. The user will find it useful compared to any other software in the society as it provides the better features with lesser complexities and better Graphical User Interface.

Also during this project we have learned process of developing the software.

**10. REFERENCES**

1. <https://en.wikipedia.org/wiki/Organ_donation>
2. <https://www.mohanfoundation.org/>
3. <https://www.python.org/>
4. <https://riverbankcomputing.com/software/pyqt/intro>