GAMCO HOSPITAL ADMINISTRATIVE SYSTEM

A project presented to the College of Computing Education,
University of Mindanao Matina, Davao City

In Partial fulfillment of the Requirements in the subject IT Elective 2

Submitted by:

Alarilla, Marie Cris

Oliva, Glenn Angelo

Submitted to:

Mr. Melvin Soriano

December 2022

Acknowledgement

We would like to convey our heartfelt gratitude to our IT5 instructor, Mr. Melvin Soriano for his expert direction as well as assistance throughout accomplishing our Final Project. Secondly, I'd like to commend my partner for her assistance in completing this project simply because she was able to design our project and make it good and fun in the allotted time. We highly appreciate all the help.

Introduction:

Our GAMCO hospital administration system is a database-driven computer or console-based system that allows for the effective management of a hospital's clinical, financial, and administrative tasks. It is intended to assist hospitals in improving patient care, increasing efficiency, as well as lowering expenses. Modules for maintaining patient records, scheduling visits, billing, and managing doctors are often included in the system. Additional elements, such as decision support tools for clinical decision-making as well as patient monitoring, may be included in some hospital management systems overall. This project includes user/patient registration, storing their information in our database, and computerized billing for each patient consultation. This program has the capability of assigning a unique id to each patient and automatically storing all patient and doctor data. It contains a search function to determine each user's current state. It contains Active or Inactive and will stay Archieve if the patient or Doctor does not exist or has retired on our system. In addition, users may search for friendly using as well as doctor's availability and data of a patient using their unique id. GAMCO Hospital Administrative System can be accessed using a username and password. It can be accessed by an administrator or a receptionist. Only they have access to the database. The information is easily accessible. The UI is easy to use. The data is highly safeguarded for personal use, and the processing is quite rapid.

System Goal:

A hospital management system's purpose is to increase the quality of care offered to patients while also making hospital operations more efficient and cost-effective. The goal of the project "GAMCO HOSPITAL ADMINISTRATIVE SYSTEM" is to computerize the hospital's front office management in order to produce software that is user pleasant, simple, rapid, and cost efficient. It is concerned with the gathering of patient information, consultation details, and so forth. It was traditionally done by hand. The basic role of the system is to register and store patient and doctor information, retrieve these facts as needed, and meaningfully change these details. System input comprises patient and diagnostic details, while system output is to display these details on the CRT screen. The GAMCO hospital administration system aims to improve patient care while also making hospital operations more efficient and cost-effective. This may be accomplished by establishing a centralized system for administering the clinical, financial, and administrative tasks of the hospital. Doctors and nurses may readily access patient information, arrange visits, and order drugs using a hospital management system, allowing them to deliver better care to their patients. Furthermore, the technology may assist the hospital in better managing its financial and operational resources, resulting in increased efficiency and cost savings. A hospital management system's ultimate purpose is to support the hospital's mission of providing high-quality, patient-centered care.

Discussion of Functionalities

Python is one of the computer languages that may be used to create GAMCO Hospital Administrative Systems. Python is a popular programming language for creating Hospital Administrative Systems. Because it is an interpreted high-level language that is simple to understand and write. Furthermore, because Python has a huge and active community, there are numerous modules and frameworks available for developing this system. To create a GAMCO Hospital Administrative System in Python, you must first create a database to hold the system's data. This is possible using a SQL database, which is a popular sort of database for storing structured data. After you've created the database, you may use Python to build the hospital management system's many features. You might use Python to develop Modules for our MAIN, Database Connection, and Database Function, which implies that all of our data, records, scheduling appointments, and billing would be saved on those modules. Overall, implementing hospital administration systems with Python and a SQL database may be a strong combo. Python's high-level, interpretative nature lends itself well to rapidly designing and testing the many capabilities of a hospital management system, whilst SQL provides a strong and scalable database for storing and managing the system's data. In addition, we included error handling so that if a user mistakenly enters incorrect information, our system will display a notification and the application will not terminate until the user decides to logout. We also utilize date and time to reserve our time in and time out sessions.

User Manual

Login with the admin account's master key. Before entering the system, the user must have basic Admiration user login details. If you do not already have a user account, please register by filling out all of the registration fields. To access the main menu, simply enter data into the login box.

After the user has registered and logged in, they will see our main menu. It will connect the Book Appointment, Pay Bills, and View Patient Details functions. After entering the admin function, the user will require the top-level admin username and password to proceed.

Log in to your administrator account. USERNAME = Admin PASSWORD = Authentication password

After login in as admin, it will choose our program, which manages patients, doctors, and appointments using basic crud, and it will save it to our database. If the user does not already have an account, he or she can create one by adding by. Users would normally need to log in with a unique username and password to use the hospital administration system. They will be able to access the system's different modules and functionalities after they have signed in. They would be able to access the patient records module to read, update, and add new patient records. They may also be able to access the appointments section in order to arrange and see upcoming appointments.

After logging in there as admin, it will have to choose our application, which uses simple crud to handle patients, physicians, as well as appointments, and save the information to our database. If somehow the user does not currently have an account, he or she might establish one by clicking on the plus sign. After entering his or her Username and Password, he or she will be able to login with a greeting on our system. Patient Information, this tool allows you to see all of the hospital's patients. It displays the patient's name, age, address, and phone number.

Following that, the patient will view their details. It is up to the user to decide whether to proceed with the pay bills that have been saved at 350 per hour and ask how many hours you stayed in the hospital, and it will automatically compute as well as the receipt and all the data contained in our database. Is it easy to use and knowledgeable especially our system as well as understand the flow beginner. User will love our System because we have onsite as well as reservation for their consult for hospital.

Curriculum Vitae

GLENN ANGELO OLIVA

COLLEGE STUDENT

09938374993

34 Matina San Antonio
Garnet St. Davao City

glennoliva122egmail.com

PROFILE

- 8 months experience of web design
- 4 months experience of customer service

EDUCATION

University of Mindanao
Batch 2020- 2024

LANGUAGES

ENGLISH

FILIPINO

SKILLS

- FUNNEL BUILDER
- SOCIAL MEDIA MANAGER
- DATA ENTRY

- COSTUMER SERVICE
- GRAPHIC DESIGN USING CANVA

- VIRTUAL ASSISTANCE
- SEO TITLE OPTIMIZER

MARIE CRIS ALARILLA

COLLEGE STUDENT

09480413851

Diamond Heights,
Communal, Davao City

mariecrisalarilla12092k@gmail.com

PROFILE

- 4 months of Customer Service

EDUCATION

University of Mindanao
Batch 2020- 2024

LANGUAGES

ENGLISH

FILIPINO

SKILLS

- **GRAPHIC DESIGN SKILLS**
- **COSTUMER SERVICE**
- **DATABASE MANAGEMENT SKILLS**
- **MARKETING**
- **DATA ENTRY**
- **SOCIAL MEDIA MANAGER**
- **VIRTUAL ASSISTANCE**

Source code:

Database_Connection

```
import mysql.connector

db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="gamco_hospital_management")


def DB_Open_Connection():
    try:
        if not db.is_connected():
            db.connect()
    except Exception as err:
        print("You got some error", err)


def DB_Close_Connection():
    try:
        if db.is_connected():
            db.close()
    except Exception as err:
        print("You got some error", err)
```

Gamco_Main

```python
from db_function import db_functions
func = db_functions()
import datetime
import string
def admin_auth():
    while True:
        print("Welcome Admin menu")
        print("[1]. Manage Patient , [2]. Manage Doctor's , [3]. Manage Appointment , [4]. Logut")
        choose = input("Choose: ")

        match choose:
            case "1":
                print("-------------Manage Patient's------------------")
                print("[1]. Add Patient , [2] Delete Patient , [3]. Update Patient , [4]. Display Active Patients")
                choose = input("Choose: ")

                match choose:
                    case "1":
                        name = input("Enter your name: ")
                            age = int(input("Enter your age: "))
                            address = input("Enter your address: ")
                            if name == "" or age == "" or address == "":
                                print("Please Complete your details: ")
                            else:
                                func.insert(name, age, address)

                    case "2":
                        id = int(input("Enter the id that you want to remove: "))
                        func.delete(id)

                    case "3":
                        print("Enter new details!")
                        id = int(input("Enter the id that you want to update: "))
                        name = input("Enter new name: ")
                        age = int(input("Enter new age: "))
                        address = input("Enter new address: ")
                        func.update(name, age, address, id)
```

```python
                    case "4":
                        print("Displaying active patients")
                        func.display_active_patient()
                    case "5":
                        print("Displaying inactive patients")
                        func.display_Inactive_patient()
                    case "6":
                        admin_auth()
            #doctor_manage:
            case "2":
                print("-----Managing's Doctor-----")
                print("[1]. Add Doctor , [2]. Delete Doctor , [3]. Update Doctor , [4]. View Active Doctor , [5].
                choose = input("Choose: ")
                match choose:

                    case "1":
                        print("Inserting Doctor")
                        name = input("Enter doctor name: ")
                        age = int(input("Enter age of doctor: "))
                        address = input("Enter address of doctor: ")
```

```
            if name == "" or age == "" or address == "":
                print("Please complete your details: ")
            else:
                func.doc_add(name,age,address)

        case "2":
            id = int(input("Enter the id that you want to remove: "))
            func.doc_delete(id)

        case "3":
            id = int(input("Enter the id that you want to update: "))
            print("Enter new details!")
            name = input("Enter new name: ")
            age = int(input("Enter new age: "))
            address = input("Enter new address: ")
            func.doc_update(name, age, address, id)
```

```
        case "4":
            print("Displaying all Doctors")
            func.doc_display_Active()
        case "5":
            print("Archieved Doctors")
            func.doc_display_Inactive()
        case "6":
            admin_auth()
case "3":
    print("-----Managing Appointment-----")
    print("[1]. Delete Appointment , [2]. Update Appointment , [3] View Appointment , [4] View appointment Inacti
    choose = input("Choose:")

    match choose:
        case "1":
            id = int(input("Enter the id that you want to remove: "))
            func.Manage_Appointment_Delete(id)
```

```
        case "2":
            id = int(input("Enter the id that you want to update: "))
            print("Enter new details!")
            name = input("Enter new name: ")
            age = int(input("Enter new age: "))
            address = input("Enter new address: ")
            phone_num = input("Enter your new phone number: ")
            time_in = input("Enter your new Time_in: ")
            time_out = input("Enter your new Time_out: ")
            time_in_str = datetime.datetime.strptime(time_in, "%Y-%m-%d %I:%M %p")
            time_out_str = datetime.datetime.strptime(time_out, "%Y-%m-%d %I:%M %p")
            total = time_out_str - time_in_str
            print("Total time spent:" + str(total) + " Hour")
            func.Manage_Appointment_Update(name, age, address,phone_num,time_in,time_out,total,id)
        case "3":
            print("Viewing patient details! ")
            id = int(input("Enter id that you want to view: "))
            func.Manage_Appointment_View(id)
```

```
                    case "4":
                        print("Viewing all Inactive or cancelled appointment")
                        func.Manage_Appointment_InactiveView()

            case "4":
                main()
                break
def User():
    while True:
        print("[1]. Register , [2]. Login , [3]. logut")
        choose = input("Choose: ")
        match choose:
            case "1":
                print("----Register Page-----")
                username = input("Enter your username: ")
                password = input("Enter your password: ")
          if username == "" or password == "":
              print("Complete details please: ")
          else:
              func.reg_user(username,password)
      case "2":
          print("Login page")
          username = input("Enter your username: ")
          password = input("Enter your password: ")
          if not func.log_user(username,password):
              User()
              pass
          elif username == "" or password == "":
              print("Please Complete Details: ")
          else:
              print("Login Sucessfully Welcome: ", username)
              func.log_user(username,password)
              print("[1]. Book appointment , [2]. Paybills , [3]. View_details")
              choose = input("Choose: ")
match choose:
    case "1":
        print("Booking Appointment: ")
        name = input("Enter your name: ")
        age = int(input("Enter your age: "))
        address = input("Enter your address: ")
        phone_num = input("Enter your Phone number: ")
        time_in = input("Time_in: ")
        time_out = input("Time_out: ")
        time_in_str = datetime.datetime.strptime(time_in, "%Y-%m-%d %I:%M %p")
        time_out_str = datetime.datetime.strptime(time_out, "%Y-%m-%d %I:%M %p")
        total = time_out_str - time_in_str
        print("Total time spent:" + str(total) + " Hour")
        if name == "" or age == "" or address == "" or phone_num == "" or time_in == "" or time_out == "":
            print("Complete your details please: ")
        else:
            func.user_booking(name, age, address,  phone_num,  time_in, time_out,total)
```

```
    case "2":
        print("-----Paying bills------")
        name = input("Enter your name: ")
        total_spent = int(input("How many hours did you stay in hospital: "))
        pay = int(input("pay: "))
        total = (800 * total_spent)
        change = pay - total
        if pay >= total:
            if func.user_paybills(name, total_spent, pay, total, change) is False:
                return True
            else:
                print("------Receipt-----")
                print("Name: " + name + "\n" + "hours: " + str(total_spent) + "\n" + "pay: " + str(pay) + "\n"
                print("Thankyou For choosing GAMCO Staysafe always and Godbless!")
        else:
            print("insufficient Money")
            return False
```

```
                    case "3":
                        print("Viewing patient details! ")
                        name = input("Enter name that you want to view: ")
                        func.user_view_details(name)
            case "3":
                main()
def Admin():
    print("========Login Page==========")
    username = input("Enter your username: ")
    password = input("Enter your password: ")

    if username == "admin" and password == "123":
        print("Login Sucessfully")
        admin_auth()
    else:
        print("Invalid account")


def main():
    while True:
        print("===============Welcome to Gamco Hospital Management System==============")
        print("[1] Admin , [2] User")
        choose = input("Choose: ")
        match choose:
            case "1":
                Admin()
            case "2":
                User()


main()
```

db_function

```python
import Database_Connection
import datetime

time_date = datetime.datetime.now()
db = Database_Connection
command_handler = db.db.cursor()

class db_functions:
    def __init__(self):
        pass

    #Patient's Functionality CRUD
    def insert(self, name_, age_, address):
        while True:
            try:
                db.DB_Open_Connection()
                command_handler.execute("SELECT * FROM patient_doc WHERE name = %s", [name])
                row = command_handler.fetchone()
                if row is None:
                    query = (name, age, address)
                    command_handler.execute("INSERT INTO patient_doc (name,age,address,privilages,status) VALUES
                    db.db.commit()
                    print("Sucessfully inserted", name)
                    return False
                else:
                    print("This information is Existed")
                    return True

            except TypeError as error:
                print("Sorry wrong places of details",error)
            finally:
                db.db.close()

    def delete(self, id):
        try:
            db.DB_Open_Connection()
            command_handler.execute("SELECT * FROM patient_doc WHERE id = %s ", [id])
            row = command_handler.fetchone()
            if row is None:
```

```
        else:
            #delete data
            query = [id]
            command_handler.execute("UPDATE patient_doc SET status = 'Inactive' WHERE id = %s", query)
            db.db.commit()
            print("Sucessfully deleted")
    except TypeError as er:
        print(er)

    finally:
        db.db.close()


def update(self,name,age,address,id):
    try:
        db.DB_Open_Connection()
        command_handler.execute("SELECT * FROM patient_doc WHERE id = %s", [id])
        row = command_handler.fetchone()
        if row is None:
            print("the id does not exist")
        else:
            query = (name,age,address,id)
            command_handler.execute("UPDATE patient_doc SET name = %s , age = %s , address = %s , status = 'Ac
            db.db.commit()
            print("Successfully updated details")
        except TypeError as er:
            print(er)
        finally:
            db.db.close()

    def display_active_patient(self):
        try:
            db.DB_Open_Connection()
            command_handler.execute("SELECT * FROM patient_doc WHERE status = 'Active' AND privilages = 'Patient'"
            records = command_handler.fetchall()
            for record in records:
                print(record)
        except TypeError as er:
            print(er)

        finally:
            db.db.close()

    def display_Inactive_patient(self):
        try:
            db.DB_Open_Connection()
            command_handler.execute("SELECT * FROM patient_doc WHERE status = 'Inactive' AND privilages = 'Patient
            records = command_handler.fetchall()
            for record in records:
                print(record)
        except TypeError as er:
            print(er)
        finally:
            db.db.close()
    # Patient's Functionality Ended here!
```

```python
#Doctor's Functionality!! CRUD!!
def doc_add(self, name_, age_, address):
    try:
        db.DB_Open_Connection()
        command_handler.execute("SELECT * FROM patient_doc WHERE name = %s", [name])
        row = command_handler.fetchone()
        if row is None:
            query = (name, age, address)
            command_handler.execute("INSERT INTO patient_doc (name,age,address,privilages,status) VALUES (%s,?
            db.db.commit()
            print("Sucessfully inserted", name)
        else:
            print("This information is Existed")

    except TypeError as error:
        print("Sorry wrong places of details", error)
    finally:
        db.db.close()
def doc_delete(self, id):
    try:
         db.DB_Open_Connection()
         command_handler.execute("SELECT * FROM patient_doc WHERE id = %s ", [id])
         row = command_handler.fetchone()
         if row is None:
             # check whether the data is have or none
             print("Patient doesn't exist!")
         else:
             # delete data
             query = [id]
             command_handler.execute("UPDATE patient_doc SET status = 'Inactive' WHERE id = %s", query)
             db.db.commit()
             print("Sucessfully deleted")
     except TypeError as er:
         print(er)
     finally:
         db.db.close()
 def doc_update(self,name, age, address, id):
     try:
         db.DB_Open_Connection()
         command_handler.execute("SELECT * FROM patient_doc WHERE id = %s", ([id]))
         row = command_handler.fetchone()
         if row is None:
             print("the id does not exist")
         else:
             query = (name, age, address, id)
             command_handler.execute("UPDATE patient_doc SET name = %s , age = %s , address = %s , status = 'Ac
             db.db.commit()
             print("Successfully updated details")
     except TypeError as er:
         print(er)
     finally:
         db.db.close()

 def doc_display_Active(self):
     try:
         db.DB_Open_Connection()
```

```
            command_handler.execute("SELECT * FROM patient_doc WHERE status = 'Active' AND privilages = 'Doctor'")
            records = command_handler.fetchall()
            for record in records:
                print(record)
        except TypeError as er:
            print(er)
        finally:
            db.db.close()

    def doc_display_Inactive(self):
        try:
            db.DB_Open_Connection()
            command_handler.execute("SELECT * FROM patient_doc WHERE status = 'Inactive' AND privilages = 'Doctor'")
            records = command_handler.fetchall()
            for record in records:
                print(record)
        except TypeError as er:
            print(er)
        finally:
            db.db.close()

    #Doctor's Functionality Ended here!

    #Register account for user!
    def reg_user(self, username,password):
        try:
            db.DB_Open_Connection()
            query = (username,password)
            command_handler.execute("SELECT username FROM register_log WHERE username = %s AND password = %s",query)
            row = command_handler.fetchone()
            if row is None:
                query = (username,password)
                command_handler.execute("INSERT INTO register_log (username,password) VALUES (%s,%s)",query)
                db.db.commit()
                print("Sucessfully Register")
            else:
                print("This account is existed please input another username!")
        except TypeError as er:
            print(er)
        finally:
            db.db.close()

    def log_user(self, username,password):
        try:
            db.DB_Open_Connection()
            query = (username,password)
            command_handler.execute("SELECT username FROM register_log WHERE username = %s AND password = %s",query)
            if command_handler.fetchone() is not None:
                return True
            else:
                print("Account does not exist")
                return False
        except TypeError as err:
            print(err)
```

```
    finally:
        db.db.close()
#Register Account Ended Here!

#USER MENU!!
def user_booking(self,name,age,phone_num,address,time_in,time_out,total):
    try:
        db.DB_Open_Connection()
        command_handler.execute("SELECT * FROM appointment WHERE name = %s", [name])
        row = command_handler.fetchone()
        if row is None:
            query = (name, age, phone_num, address, time_in, time_out,total)
            command_handler.execute("INSERT INTO appointment (name,age,address,phone_num,time_in,time_out,tota
            db.db.commit()
        else:
            print("This information is Existed")

    except TypeError as er:
        print(er)
    finally:
        db.db.close()

def user_paybills(self, name,total_spent,pay,total,change):
    try:
        db.DB_Open_Connection()
        command_handler.execute("SELECT * FROM pay_bills WHERE name = %s", [name])
        row = command_handler.fetchone()
        if row is None:
            query = (name, total_spent, pay, total, change)
            command_handler.execute("INSERT INTO pay_bills (name,hours,pay,total,transition) VALUES (%s,%s,%s,
            db.db.commit()
        else:
            print("This information is Existed")
            return False

    except TypeError as er:
        print(er)
    finallv:
        db.db.close()


def user_view_details(self, name):
    appointment_col = ["Name: ", "Age: ", "Address: ", "Phone_number: ", "Time_in: ", "Time_out: ", "Status:
    try:
        db.DB_Open_Connection()
        query = [name]
        command_handler.execute("SELECT * FROM appointment WHERE name = %s ",query)
        row = command_handler.fetchone()
        if row is None:
            print("Name does not exist in database")
        for rows,col_list in zip(appointment_col,row):
            print(rows,col_list)

    except TypeError as er:
        print(er)
    finally:
```

```
            db.db.close()

        #User menu ended's Here

    #Manage Appointment Function Start's Here!
    def Manage_Appointment_Delete(self,id):
        try:
            db.DB_Open_Connection()
            command_handler.execute("SELECT * FROM appointment WHERE id = %s", ([id]))
            row = command_handler.fetchone()
            if row is None:
                print("the id does not exist")
            else:
                # delete data
                query = [id]
                command_handler.execute("UPDATE appointment SET status = 'Inactive' WHERE id = %s", query)
                db.db.commit()
                print("Sucessfully deleted")
        except TypeError as er:
            print(er)
        finally:
            db.db.close()
    def Manage_Appointment_Update(self,name, age, address,phone_num,time_in,time_out,total,id):
        try:
            db.DB_Open_Connection()
            command_handler.execute("SELECT * FROM appointment WHERE id = %s", ([id]))
            row = command_handler.fetchone()
            if row is None:
                print("the id does not exist")
            else:
                query = (name, age, address,phone_num,time_in,time_out,total, id)
                command_handler.execute("UPDATE appointment SET name = %s , age = %s , address = %s ,phone_num =
                db.db.commit()
                print("Successfully updated details")
        except TypeError as er:
            print(er)
        finally:
            db.db.close()

    def Manage_Appointment_View(self,id):
        appointment_col = ["Id", "Name: ", "Age: ", "Address: ", "Phone_number: ", "Time_in: ", "Time_out: ", "Tot
        try:
            db.DB_Open_Connection()
            query = [id]
            command_handler.execute("SELECT name,age,address,phone_num,time_in,time_out,total_spent,status FROM ap
            row = command_handler.fetchone()
            if row is None:
                print("Id does not exist in database")
            else:
                for rows, col_list in zip(appointment_col, row):
                    print(rows, col_list)

        except TypeError as er:
            print(er)
```

```
    finally:
        db.db.close()
def Manage_Appointment_InactiveView(self):
    try:
        db.DB_Open_Connection()
        command_handler.execute("SELECT * FROM appointment WHERE status = 'Inactive'")
        row = command_handler.fetchall()
        for rows in row:
            print(rows)
    except TypeError as er:
        print(er)

    finally:
        db.db.close()

#Manage_Appointment Ends Here!!!
```

Checklist:

IT5 FINAL PROJECT GUIDELINES AND CHECKLIST, you may do it yourself or do it with a pair. To conclude this course and apply all the things that we have discussed in python, we must apply it to create a simple program. Kindly refer to this document as your guidelines and checklist for the final project.

You or with your partner are tasked to create a simple and well-functioning program written in Python. Kindly refer to the table below for the complete checklist.

Item	Function	Complied	Not Complied
Data Types and Arithmetic Operators	The system will be able to utilize correct and proper data types as well as the use of arithmetic operators.	✓	
Control Structures: Selection and Repetition	The system applies the concept of control structures to solve certain problems.	✓	
Python Data Structures	The system utilizes at least one (1) data structure available in python.	✓	
Python Date and Time	The system utilizes date and time class for log purposes.	✓	
Methods	The system contains user defined methods that performs various function to achieve the system's goal.	✓	
Object Oriented Programming	The system must have written in object-oriented approach.	✓	
Database	The system utilizes a database for data storage and can perform SQL CRUD functionalities.	✓	
Error Handling	The system can properly handle possible errors whether syntax or logic error.	✓	
Transaction	The system can perform at least one (1) transactional functionality.	✓	
Coding	The system is well-written, utilizes comments for documenting code, organized, and clean.	✓	