**INTI International College Penang School of Computing**

**3+0 Bachelor of Science (Hons) in Computer Science, in collaboration with Coventry University, UK**

**3+0 Bachelor of Science (Hons) in Computing, in collaboration with Coventry University, UK**

**Coursework cover sheet**

**Section A - To be completed by the student**

|  |  |
| --- | --- |
| Full Name: **LINKESH A/L JAYA PRAKASH RAO** | |
| CU Student ID Number: 15095824 | |
| Semester: APR2024 | |
| Lecturer: Mr. Shahriman Mohd Said | |
| Module Code and Title: 5001CEM Software Engineering | |
| Assignment No. / Title: Portfolio | % Of Module Mark: 10% |
| Hand out date: 25 April 2024 | Due date: 25 June 2024 |
| Penalties: No late work will be accepted. If you are unable to submit coursework on time due to extenuating circumstances, you may be eligible for an extension. Please consult the lecturer. | |
| Declaration: I/we the undersigned confirm that I/we have read and agree to abide by the University regulations on plagiarism and cheating and Faculty coursework policies and procedures. I/we confirm that this piece of work is my/our own. I/we consent to appropriate storage of our work for plagiarism checking. | |

**Section B - To be completed by the module leader**

|  |
| --- |
| Learning Outcomes  LO4 -Select, evaluate and use tools and techniques to successfully manage a large scale software  project, including configuration management and version control.  LO5 - Use a range of appropriate tools to contribute to the development of a solution to a real-world  problem.  LO6 - Select, evaluate and apply standards, tools and techniques for assuring software quality. |
| Lecturer’s Feedback |
| Internal Moderator’s Feedback |

Table of Contents

[**1.0** **Introduction** 3](#_Toc169967719)

[**1.1** **Methodology Used for this project.** 4](#_Toc169967720)

[**1.2** **Importance of the system in real world** 5](#_Toc169967721)

[**2.0** **Progress Cycle** 6](#_Toc169967722)

[**CYCLE 1** 6](#_Toc169967723)

[**CYCLE 2** 7](#_Toc169967724)

[**CYCLE 3** 8](#_Toc169967725)

[**Prototype Link:** 10](#_Toc169967726)

[**CYCLE 4** 11](#_Toc169967727)

[**CYCLE 5** 13](#_Toc169967728)

[**CYCLE 6** 14](#_Toc169967729)

[**ERD Diagram** 15](#_Toc169967730)

[**DATABASE** 16](#_Toc169967731)

[**CYCLE 7** 18](#_Toc169967732)

[**2.1** **Testing Process** 20](#_Toc169967733)

[**2.2** **Automated Testing** 26](#_Toc169967734)

[**3.0** **GitHub** 43](#_Toc169967735)

[**4.0** **References** 44](#_Toc169967736)

# **Introduction**

Sage Enterprise created and used the "Call a Doctor" (CaD) system, a revolutionary system designed to transform the way patients access medical care. By enabling patients to consult doctors from the comfort of their homes, this system bridges the gap between traditional healthcare and modern convenience. Through CaD, patients can effortlessly find and request home visits from registered clinics, enhancing the accessibility and efficiency of healthcare services.

# **Methodology Used for this project.**

The Agile Methodology is essential for this project. Agile focuses on cross-functional collaboration and continuous improvement in project management. It breaks down projects into manageable stages and guides teams through cycles of planning, execution, and evaluation.

The following are the main guiding principles for project management using the Agile methodology:

1. Evolution is welcomed at any stage of the process.
2. Products or services are delivered more frequently.
3. To achieve project success and optimal outcomes, all stakeholders and team members must remain motivated. Teams have all the necessary resources and support to complete the project.
4. In-person meetings are considered the most effective and efficient means of communication for project success.
5. The ultimate measure of success is a functioning final product.
6. Agile methods allow stakeholders and development teams to work at a steady and sustainable pace, fostering continuous progress.
7. A constant focus on technical excellence and solid planning will enhance agility.
8. Simplicity must be a priority at every stage of the project.
9. Self-organizing teams are more likely to generate the best ideas and projects while meeting the requirements.
10. Teams adjust their behaviour to boost productivity and efficiency.

# **Importance of the system in real world**

The Call a Doctor (CaD) system is extremely important. By providing patients with a centralized platform to request doctor visits at home, it streamlines and organizes the process of accessing medical care. The system automates tedious tasks such as managing appointment requests and verifying clinic registrations. This increases overall efficiency, reduces administrative burdens, and saves both patients and healthcare providers time and money.

The CaD system enhances collaboration and communication among various stakeholders in the healthcare process. Patients are promptly and clearly informed about the status of their doctor requests and any changes to appointment schedules. Through the system, clinic administrators, doctors, and patients can work together and communicate effectively. This fosters cooperation, increases transparency, and ensures that everyone involved is on the same page.

Additionally, the CaD system helps maintain accurate records of patient visits and medical prescriptions. The technology enables clinic administrators to track the status of doctors, manage patient requests efficiently, and ensure that resources are allocated appropriately. The system can also assist in modifying schedules or reallocating doctors to meet demand, ensuring optimal use of available medical resources. To ensure data integrity and accessibility for future reference, the system allows administrators to back up and securely store patient records.

Overall, the Call a Doctor system is essential for improving communication, better managing healthcare services, and maintaining accurate records. It enhances the efficiency, productivity, and cost-effectiveness of medical care delivery, ultimately benefiting both patients and healthcare providers.

# **Progress Cycle**

### **CYCLE 1**

**Meeting 1**

**Date:** 10/04/2024

**Method:** (Face to Face)

**Meeting Description:** Discussion of the intended purpose

**Attendance:** Linkesh, Harvind

A close-up of a paper

Description automatically generatedA spiral notebook with a pen and diagram

Description automatically generated

A screenshot of a computer

Description automatically generated

We discussed adding the feature to the system and use case diagram during this meeting depending on the assignment requirement. We have added kanban board and backlog for this meeting.

### **CYCLE 2**

**Meeting 2**

**Date:** 01/05/2024

**Method:** (Face to face)

**Meeting Description:** Discussion of the system prototype

**Attendance:** Linkesh, Harvind

A close-up of a login page

Description automatically generated

We discussed the color schemes to employ as well as the design of the prototype.

### **CYCLE 3**

**Meeting 3**

**Date:** 18/05/2024

**Method:** (Face to Face)

**Meeting Description:** Discussion and development of a system prototype

**Attendance:** Linkesh, Harvind

Pages were made during the meeting for each function, and the person in charge of designing the prototypes was given responsibility for each page.

I was assigned to develop prototype of doctor home page, doctor profile page, doctor edit profile page, clinic admin home page, add doctor page, delete doctor page, admin appointment summary page, admin home page and view clinic requests page.

The evidences of the prototypes I design is below:

A screenshot of a medical appointment

Description automatically generated

Doctor home page

A screenshot of a medical form

Description automatically generated

Add doctor page for clinic admin

A white sheet with black text

Description automatically generated

Delete doctor page for clinic admin

A screenshot of a medical form

Description automatically generated

Doctor profile page

**Prototype Link:** <https://www.figma.com/design/7xrhQRkcFjIAyDGaHRXet9/Sage-mode?node-id=0-1&t=SFWPJs8yHJBS5yu6-0>

### **CYCLE 4**

**Meeting 4**

**Date:** 20/05/2024

**Method:** (Face to Face)

**Meeting Description:** Discussion and development of a system prototype

**Attendance:** Linkesh, Harvind

There were still some pages left to complete our prototypes.

I was assigned to develop prototype of Patient home page, Patient notification page, Patient search/view page, Patient request for doctor page, Patient profile, Admin home page, Admin notification page, Admin patient management page and Admin doctor management page.

With this, we have completed our prototypes. The evidence of one of the prototypes I design is below.

A paper with writing on it

Description automatically generated A close-up of a paper

Description automatically generated

A screenshot of a computer

Description automatically generated

We have also created kanban board with backlog included before moving on to the database.

### **CYCLE 5**

**Meeting 5**

**Date:** 01/06/2024

**Method:** (Online)

**Meeting Description:** Database Creation

**Attendance:** Linkesh, Harvind

A screenshot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated

I was given the work to construct the database in mysql. The database was created to hold the data utilized in the system. But first, we discussed about the number of tables and the primary keys of each tables.

### **CYCLE 6**

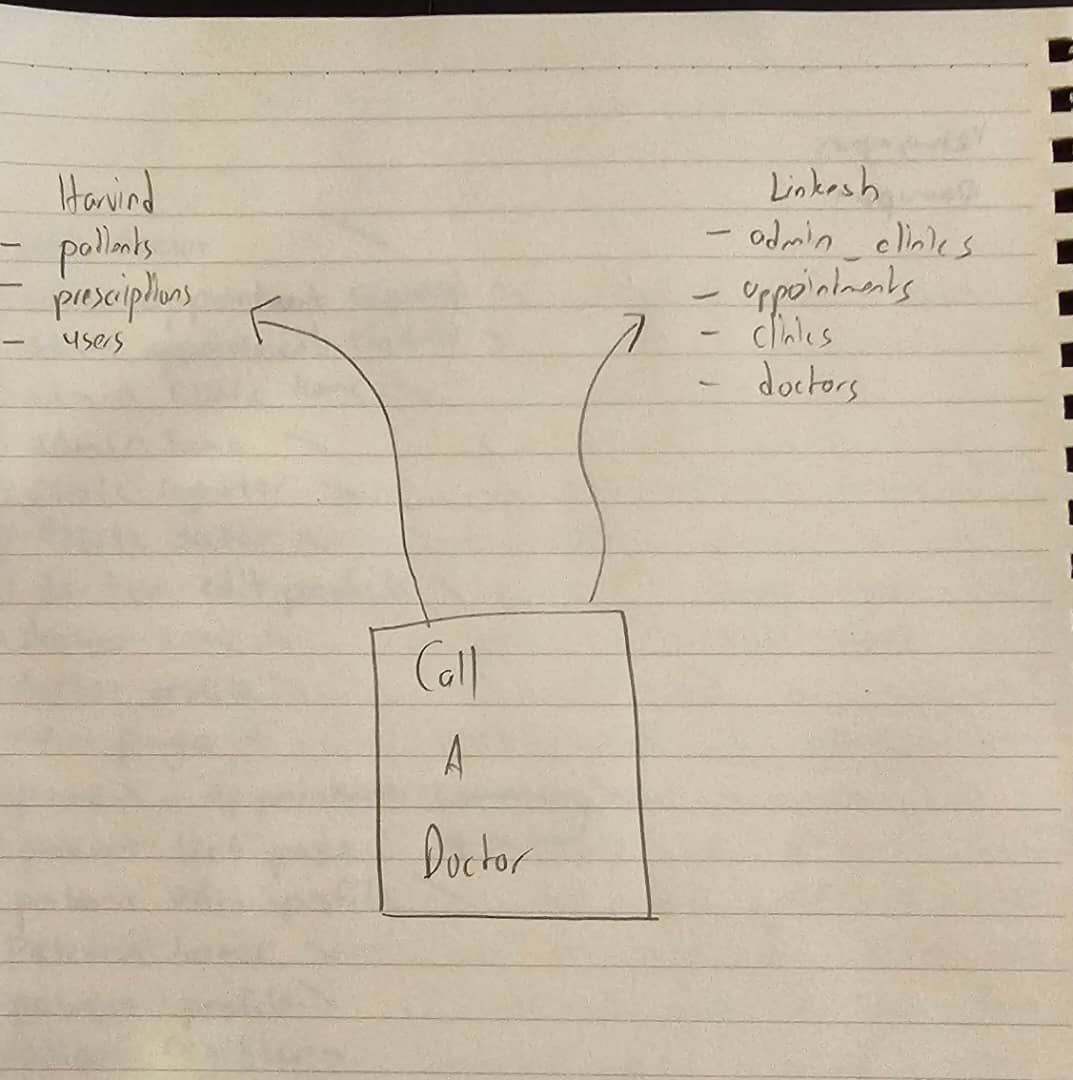
**Meeting 6**

**Date:** 12/06/2024

**Method:** (Face to face)

**Meeting Description:** ERD Making

**Attendance:** Linkesh, Harvind



I was the primary person who constructed the database in MYSQL and Harvind created the ERD while exchanging thoughts throughout the meeting. The database was created to hold the data utilized in the system. Based on the database, the ERD was created.

### **ERD Diagram**

A screenshot of a computer

Description automatically generated

### **DATABASE**

A screenshot of a computer

Description automatically generated

Diagram 1.0 admin\_clinics table

A screenshot of a computer

Description automatically generated

Diagram 1.1 appointments table

A screenshot of a computer

Description automatically generated

Diagram 1.2 clinics table

A screenshot of a computer

Description automatically generated

Diagram 1.3 doctors table

### **CYCLE 7**

**Meeting 7**

**Date:** 18/06/2024

**Method:** (Face to face)

**Meeting Description:** Progression Checking and Problem Solvement

**Attendance:** Linkesh, Harvind

The purpose of this meeting was to review systems and progress in order to make small improvements. I was now getting data from the database, having finished the most of the functions and rectified a few minor problems. I make sure that my group members are aware of my work by consistently updating GitHub with the rectified code.

A person sitting at a desk using a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

At this point, most of our functions were working, designs of the interface were done. The project was left with testing and documentation.

# **Testing Process**

**Doctor Home Page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Display the past appointments for the specific doctor that login from the database. | None | A table is displayed containing all the past appointments the specific doctor conducted. | PASS |
| Display the upcoming appointments for the specific doctor that logged in from the database. | None | A table is displayed containing all the upcoming appointments the specific doctor conducted. | PASS |
| When click “Generate prescription” in the upcoming appointment table, it should generate a prescription for the patient. | Patient selected | The system successfully generate a prescription for the patient. | PASS |
| When click “View medical record” in the upcoming appointment table, it should open view medical page and view the medical record. | Patient selected | The system successfully open view medical page and displayed the patients medical record. | PASS |
| When doctors click the “profile” button, the system should direct to doctor profile page. | None | The system closed the doctor home page and open the doctor profile page. | PASS |
| When doctors click the “logout” button, the system should direct to main page. | None | The system closed the doctor home page and open back the main page. | PASS |

**Doctor Profile**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Display the doctor’s information that logged in. | None | The system displayed the doctor’s profile that logged in. | PASS |
| When doctors click the “edit profile” button, the system should direct to doctor edit profile page. | None | The system closed the doctor profile page and open the doctor edit profile page. | PASS |
| When doctors click the “back” button, the system should direct back to doctor home page. | None | The system closed the doctor profile page and open back the doctor home page. | PASS |

**Doctor edit profile page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Display the doctor’s information that logged in. Doctors should be able to edit their address, phone number and email only. | Doctor’s email, phone number and address | The system let the doctors to edit their address, phone number and email only. | PASS |
| When doctors click the “confirm” button, the system should saved the current info and go back to doctor profile page. | None | The system saved the doctor’s info and open back the doctor profile page. | PASS |
| When doctors click the “back” button, the system should direct back to doctor profile page. | None | The system closed the doctor edit profile page and open back the doctor profile page. | PASS |
| When doctors click the “confirm” button without changing info, the system should still save the changes and go back to doctor profile page. | None | The system still saved the changes and close the doctor edit profile page. Then, open back the doctor profile page. | PASS |

**Clinic admin home page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Display the clinic admin’s information such as name, address and total number of doctors in the clinic the admin is working. | None | The system displayed the clinic admin’s information such as name, address and total number of doctors. | PASS |
| Display the appointment requests from patients that sent appointment requests to the clinic the admin is working. | None | The system displayed the appointment requests the patient sent to the clinic the admin is working. | PASS |
| When the clinic admin click the appointment request from the table, the clinic admin will then click “accept” button. It should then disappear from the table because the appointment has been approved. | Appointment request selected | The system approved the appointment request, then the appointment request will be disappear from the table. | PASS |
| When the clinic admin click the appointment request from the table, the clinic admin will then click “reject” button. It should then disappear from the table because the appointment has been rejected. | None | The system rejected the appointment request, then the appointment request will be disappear from the table. | PASS |
| A validation will be shown if the clinic admin click the “accept” or “reject” button without clicking any appointment request. | None | A validation message was shown “ Please choose an appointment request” | PASS |
| When hover over doctor management , it should show “add doctor” and delete doctor”. “Add doctor” will go to add doctor page and “delete doctor” will go to delete doctor page. | None | When hover , it successfully showed both “add doctor” that will go to add doctor page and “delete doctor” that will go to delete doctor page. | PASS |
| When click “appointment schedule”, it will go to admin appointment schedule page. | None | The system successfully went to admin appointment schedule page. | PASS |
| When click “logout” button, it should go back to the main page. | None | The system successfully closed the admin clinic home page and move back to the main page. | PASS |

**Add doctor page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Clinic admin should fill in all information then click “save”. It will then go back to clinic admin home page. | Doctor’s fullnaame, username, password, gender, IC, email, phone number, address and date of birth | The system successfully added the doctor and move back to admin clinic home page. | PASS |
| When clinic admin click “back” button, it should go back to the admin clinic home page. | None | The system successfully went back to the admin clinic home page and close the add doctor page. | PASS |
| A validation message will be shown if the clinic admin click “save” without filling all the informations. | None | A validation message “ please fill out all the fields” was shown. | PASS |

**Delete doctor page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Display all the doctors and their info that is in the same clinic as the clinic admin | None | The system successfully displayed all the doctors and their info in the clinic that the clinic admin in. | PASS |
| When click the “delete” button, it will delete the doctor from the clinic. | None | The system successfully deleted the doctor from the clinic. | PASS |
| When click “back” , the system should go back to the admin clinic home page. | None | The system successfully closed the delete doctor page and go back to the admin clinic home page. | PASS |

**Admin appointment schedule page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Once clinic admin selected the date and time, when click “search” button it should show available doctors in the clinic. | Select date and time | The system successfully showed available doctors to appoint them for appointment. | PASS |
| When click “back” , the system should go back to the admin clinic home page. | None | The system successfully closed the admin appointment schedule page and went back to the admin clinic home page. | PASS |

**Admin home page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Display all the registered clinics in the system | None | The system successfully displayed all the registered clinics. | PASS |
| When click “view clinic registration request” , it should go to view clinic requests page. | None | The system closed the admin home page successfully, and opened the view clinic requests page. | PASS |
| When click “logout” button, it should go back to main page. |  | The system successfully closed the admin home page and go back to main page. | PASS |

**View clinic requests page**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Input** | **Output** | **Status** |
| Display all the clinics that is waiting to be approve/reject | None | The system successfully displayed all the clinics that is waiting to be approve or reject. | PASS |
| When admin select a clinic, and then click “approve” or “reject”, the clinic should be approved or rejected and disappear from the table. | Clinic selected | The system successfully approve or reject the clinic, and the clinic will be disappear from table. | PASS |
| When admin select a clinic, and then click “view license”, they should be able to view license. | Clinic selected | The admin can successfully view the clinic license. | PASS |
| When admin click “back” button, the system should go back to admin home page. | None | The system successfully went back to the admin home page. | PASS |
| A validation message will be shown if admin click “approve” or “reject” or “view license”. | None | The system showed a message “Please shoose a clinic”. | PASS |

# **Automated Testing**

* + 1. Test for fetch\_doctors in adminappointmentschedule page

Using the unittest framework and ‘unittest.mock’, we can automate the testing of the ‘fetch\_doctors’ function, which is initiated when the search icon is selected, in order to simulate dependencies. This method guarantees that the function functions appropriately without the necessity of interacting with a live database or GUI elements. The following is a comprehensive guide to the configuration and execution of these tests.  
  
To begin, we establish the test environment and define the test class. In the ‘setUp()’ method, we generate phony objects for the function's dependencies, including the ‘treeview’ widget, hour and minute ‘comboboxes’, and date entry. To simulate the GUI environment, we also established a root window using Tkinter. This configuration enables us to simulate the behaviour of these dependencies during testing by generating prototype objects.  
  
Within the ‘TestFetchDoctors’ class, we define various methods representing distinct test cases that the function should handle. Each test case simulates a particular scenario by configuring the prototype objects to return specific values. For instance, we can modify the return value of the date entry to simulate a user-selected date and time, and set up the dummy database cursor to return various results based on the query conducted.  
  
For each test case, we prepare the necessary conditions by building up the prototype objects and their return values. For example, we can set the return value of the date entry's ‘get\_date’ method and the hour and minute ‘comboboxes’ to simulate user input. We also configure the simulated database cursor to return specified values for distinct queries.  
  
After setting up the conditions, we invoke the ‘fetch\_doctors’ function explicitly within the test code. To verify the function's behaviour, we use assertions provided by the ‘unittest.TestCase’ class. These assertions verify whether the function acted as anticipated. For example, we can use ‘assert\_called\_with’ to verify that the database cursor's execute method was called with the correct SQL query. We can also verify if the ‘treeview’ widget's append method was called with the anticipated values, indicating that the physicians were accurately fetched and displayed.  
  
Furthermore, we can mimic the messagebox.showinfo method to verify if the appropriate message is displayed when no physicians are available. This ensures that the function manages various circumstances, such as no available physicians or database errors, correctly and provides the expected feedback to the user.  
  
By executing these test cases, we can determine if the fetch\_doctors function performs as expected under various conditions. This automated testing approach helps identify and resolve issues early, ensuring the reliability and correctness of the function in different scenarios.  
  
In summary, the automated testing procedure for the fetch\_doctors function entails building up a prototype environment, defining test cases for various scenarios, invoking the function, and using assertions to verify its behavior. This approach ensures that the function manages various inputs and conditions accurately, providing confidence in its reliability and accuracy.

# Function to fetch and display doctors based on selected date and time

def fetch\_doctors():

    selected\_date = date\_entry.get\_date()

    selected\_hour = hour\_combobox.get()

    selected\_minute = minute\_combobox.get()

    selected\_time = f"{selected\_hour}:{selected\_minute}:00"

    # Convert the selected date to YYYY-MM-DD format

    formatted\_date = selected\_date.strftime("%Y-%m-%d")

    print(f"Selected Date: {formatted\_date}, Selected Time: {selected\_time}, Clinic ID: {clinic\_id}")

    try:

        connection = mysql.connector.connect(

            host='localhost',

            user='root',

            password='calladoctor1234',

            database='calladoctor'

        )

        cursor = connection.cursor()

        # Clear the treeview before fetching data

        for item in tree.get\_children():

            tree.delete(item)

        # Query to get available doctors

        query = """

        SELECT d.fullname

        FROM doctors d

        LEFT JOIN appointments a ON d.doctor\_id = a.doctor\_id

            AND a.appointment\_date = %s

            AND a.appointment\_time = %s

            AND a.appointment\_request\_status = 'accepted'

        WHERE d.clinic\_id = %s

          AND d.is\_available = 1

          AND (a.doctor\_id IS NULL OR a.appointment\_request\_status != 'accepted')

        """

        print(f"Executing query with params: {formatted\_date}, {selected\_time}, {clinic\_id}")

        cursor.execute(query, (formatted\_date, selected\_time, clinic\_id))

        doctors = cursor.fetchall()

        # Insert fetched doctors into the treeview

        for index, doctor in enumerate(doctors, start=1):

            tree.insert("", "end", values=(index, doctor[0]))

        if not doctors:

            # When no doctors are available, list all available doctors regardless of appointment time

            query = """

            SELECT fullname

            FROM doctors

            WHERE clinic\_id = %s

              AND is\_available = 1

            """

            cursor.execute(query, (clinic\_id,))

            doctors = cursor.fetchall()

            for index, doctor in enumerate(doctors, start=1):

                tree.insert("", "end", values=(index, doctor[0]))

            if not doctors:

                messagebox.showinfo("No Doctors Available", "No available doctors found for the selected date and time.")

        connection.close()

    except mysql.connector.Error as error:

        print(f"Database Error: {error}")

        messagebox.showerror("Database Error", str(error))

* + 1. Test for accept\_appointment and reject\_appointment in adminclinichome page

In the context of clinic administration, the efficient management of appointment requests is essential for the optimization of operational workflows and the preservation of patient satisfaction. In this process, the functions ‘accept\_appointment’ and ‘reject\_appointment’ are essential, as they enable clinic administrators to promptly respond to incoming appointment requests from patients.  
  
In order to guarantee a smooth process, a clinic administrator must take several steps before accepting an appointment request. The system initially identifies the specific appointment by assigning a unique identifier to it, which is referred to as the ‘appointment\_id’, after the user selects an appointment request from the list displayed in the graphical user interface of the CAD system. This identifier is essential because it is directly associated with the appointment in the clinic's database.  
  
Once the appointment is selected, the ‘accept\_appointment’ function initiates an update in the appointment status to designate it as 'accepted'. This action not only affirms the appointment but also informs the patient and the clinic's personnel that the period has been reserved, thereby preventing any scheduling conflicts.  
  
The function is typically associated with a 'Accept' icon in terms of user interface incorporation, which improves the user experience by offering a distinct, actionable option. Administrators are informed of the successful update through visual feedback mechanisms, such as success message windows, which bolster their confidence in the system's responsiveness.  
  
In contrast, the ‘reject\_appointment’ function is designed to handle situations in which a clinic administrator must reject an appointment request. The function commences by selecting the appointment request from the list that is displayed, similar to the acceptance process. Upon selection, the system retrieves the ‘appointment\_id’, ensuring accuracy and specificity in the update process.

Once identified, the function updates the appointment status to 'rejected'. This action notifies the patient promptly, allowing them to reschedule or pursue alternative arrangements without ambiguity. It also facilitates administrative duties by maintaining an up-to-date record of appointment statuses, thereby facilitating efficient scheduling and resource allocation within the clinic.  
  
User interface elements such as the 'Reject' icon are integral to the function's accessibility and usability. Coupled with error management mechanisms, such as error message windows prompting administrators to select an appointment before proceeding, the function ensures procedural adherence and minimizes potential oversights.  
  
Both functions leverage database connectivity through frameworks like mysql.connector, facilitating seamless interaction with the clinic's database. This integration ensures that modifications to appointment statuses are reflected accurately and promptly, maintaining data integrity across the system.  
  
Overall, the’ accept\_appointment’ and ‘reject\_appointment’ functions exemplify the symbiosis between efficient clinic management and user-centric design in healthcare administration. By empowering administrators with intuitive tools to coordinate appointment requests, these functions contribute to enhanced patient care delivery and operational efficiency within healthcare facilities.

# Accept and Reject buttons

def accept\_appointment():

    selected\_item = appointment\_table.selection()

    if selected\_item:

        row\_id = selected\_item[0]

        appointment\_id = appointment\_ids[row\_id]

        update\_appointment\_status(appointment\_id, 'accepted')

    else:

        messagebox.showerror("Error", "Please select an appointment to accept.")

def reject\_appointment():

    selected\_item = appointment\_table.selection()

    if selected\_item:

        row\_id = selected\_item[0]

        appointment\_id = appointment\_ids[row\_id]

        update\_appointment\_status(appointment\_id, 'rejected')

    else:

        messagebox.showerror("Error", "Please select an appointment to reject.")

accept\_btn = ctk.CTkButton(button\_frame, text="Accept", command=accept\_appointment, fg\_color="green", text\_color="white", font=("Arial", 12, "bold"))

accept\_btn.pack(side=tk.LEFT, padx=10)

reject\_btn = ctk.CTkButton(button\_frame, text="Reject", command=reject\_appointment, fg\_color="red", text\_color="white", font=("Arial", 12, "bold"))

reject\_btn.pack(side=tk.LEFT, padx=10)

* + 1. Test for create\_prescription\_form in doctorhome page

The ‘create\_prescription\_form’ function plays a crucial role in the medical appointment management system by providing an efficient and user-friendly interface for physicians to create and save patient prescriptions. This function ensures that prescriptions are accurately linked to the correct patient and appointment, and securely stored in the database.  
  
When a doctor initiates the ‘create\_prescription\_form’ function, the primary window of the application is momentarily disabled to concentrate the doctor's attention on the new prescription form. This is accomplished by invoking the ‘root.withdraw()’ method. Subsequently, a new top-level window is created using ‘ctk.CTkToplevel(root)’, which functions as a dedicated space for inputting the prescription details. The window is titled "Prescription" and its dimensions are set to 400x250 pixels, providing a condensed yet sufficient area for the task at hand.  
  
Within this new window, several graphical user interface (GUI) elements are incorporated to facilitate the prescription input procedure. A label prompts the doctor to input the prescription for a specified patient, ensuring clarity and context. Additionally, a text box is provided for composing the prescription, offering ample space for detailed medical instructions. The text box is sized at 90x180 pixels, mediating between sufficient input space and maintaining an orderly window layout.  
  
A discrete frame is created within the prescription window to contain action keys, ensuring an organized and user-friendly layout. Two controls are added: a save button and a return button. The save icon is responsible for initiating the process of storing the prescription, while the back button enables the doctor to terminate the operation and return to the main window.  
  
The function incorporates an internal ‘save\_prescription’ function, which manages the actual storing of the prescription. This function retrieves the inputted text from the text field and assures it is not empty. If the text field is vacant, a warning message is displayed, informing the doctor that the prescription cannot be empty. This phase prevents incomplete or inadvertent entries from being preserved.  
  
Before recording the prescription, the function prompts the doctor for affirmation using a message box. This prompt informs the doctor that saving the prescription will designate the appointment as complete and asks for their confirmation to proceed. This step ensures that the doctor is fully aware of the consequences of storing the prescription and provides an opportunity to retract if required.  
  
Upon affirmation, the function attempts to connect to the database and implement the necessary SQL queries to save the prescription and update the appointment status. The prescription is inserted into the prescriptions table, connecting it to the correct appointment and doctor. Simultaneously, the appointment's treatment status is updated to "done" in the appointments table. The database connection and cursor are closed after these operations to ensure no resources are left open, which could contribute to potential issues.  
  
The function also incorporates robust error handling mechanisms. If any errors occur during the database operations, they are detected and appropriately addressed. An error message is displayed to the doctor, informing them of the issue, and the error details are printed for diagnostic purposes. This ensures that the doctor is aware of any concerns and can take appropriate action if necessary.  
  
Upon effectively storing the prescription, a success message is displayed to the doctor, confirming that the prescription has been saved. The prescription window is then closed, and the primary window is re-enabled, allowing the doctor to continue using the application. Additionally, the ‘refresh\_appointments’ function is called to update the appointments list, reflecting the alterations made by the newly saved prescription.

# Function to create prescription form and save the prescription

def create\_prescription\_form(appointment\_id, doctor\_id, patient\_name):

    """Create a form for entering a prescription and save it to the database."""

    def save\_prescription():

        """Save the prescription to the database."""

        prescription = text.get("1.0", ctk.END).strip()  # Getting the entered prescription text from the textbox.

        if prescription:  # Checking if the prescription is not empty.

            response = messagebox.askyesno("Confirmation", "This will result in completing the patient's appointment. Do you want to proceed?")  # Asking for confirmation to proceed.

            if response:  # If the user confirms.

                try:

                    connection = mysql.connector.connect(

                        host="localhost",  # Database host.

                        user="root",  # Database user.

                        password="calladoctor1234",  # Database password.

                        database="calladoctor"  # Database name.

                    )

                    cursor = connection.cursor()  # Creating a cursor object to execute SQL queries.

                    cursor.execute("""

                        INSERT INTO prescriptions (appointment\_id, doctor\_id, patient\_id, medical\_report)

                        VALUES (%s, %s, (SELECT patient\_id FROM appointments WHERE appointment\_id = %s), %s)

                    """, (appointment\_id, doctor\_id, appointment\_id, prescription))  # SQL query to insert the prescription into the database.

                    cursor.execute("""

                        UPDATE appointments

                        SET treatment\_status = 'done'

                        WHERE appointment\_id = %s

                    """, (appointment\_id,))  # SQL query to update the appointment's treatment status to 'done'.

                    connection.commit()  # Committing the transaction.

                    cursor.close()  # Closing the cursor.

                    connection.close()  # Closing the database connection.

                    refresh\_appointments()  # Refreshing the appointments list.

                    messagebox.showinfo("Success", "Prescription saved successfully.")  # Showing a success message.

                    prescription\_window.destroy()  # Closing the prescription window.

                    root.deiconify()  # Re-enabling the main window.

                except mysql.connector.Error as err:  # Catching any MySQL database errors.

                    print(f"Database Error: {err}")  # Printing the error message.

                    messagebox.showerror("Database Error", f"Error: {err}")  # Showing an error message box.

        else:  # If the prescription is empty.

            messagebox.showwarning("Warning", "Prescription cannot be empty.")  # Showing a warning message.

    def go\_back():

        """Close the prescription window and go back to the main window."""

        prescription\_window.destroy()  # Closing the prescription window.

        root.deiconify()  # Re-enabling the main window.

    root.withdraw()  # Disabling the main window.

    prescription\_window = ctk.CTkToplevel(root)  # Creating a new top-level window for the prescription form.

    prescription\_window.title("Prescription")  # Setting the title of the prescription window.

    prescription\_window.geometry("400x250")  # Setting the size of the prescription window.

    label = ctk.CTkLabel(prescription\_window, text=f"Enter Prescription for {patient\_name}:")  # Creating a label for the prescription textbox.

    label.pack(pady=10)  # Packing the label with some padding.

    text = ctk.CTkTextbox(prescription\_window, height=90, width=180)  # Creating a textbox for entering the prescription.

    text.pack(pady=10)  # Packing the textbox with some padding.

    button\_frame = ctk.CTkFrame(prescription\_window)  # Creating a frame to hold the buttons.

    button\_frame.pack(pady=10)  # Packing the button frame with some padding.

    save\_button = ctk.CTkButton(button\_frame, text="Save", command=save\_prescription)  # Creating a button to save the prescription.

    save\_button.pack(side=ctk.LEFT, padx=5)  # Packing the save button with some padding.

    back\_button = ctk.CTkButton(button\_frame, text="Back", command=go\_back)  # Creating a button to go back to the main window.

    back\_button.pack(side=ctk.LEFT, padx=5)  # Packing the back button with some padding.

* + 1. Test for save\_doctor in adddoctor page

The ‘save\_doctor’ function in the medical clinic administration system is crucial for adding new physicians to the database accurately and securely. This function begins by collecting data from various input fields in the user interface, such as the doctor's full name, username, password, gender, identification number (IC), email, phone number, address, and date of birth.  
  
The function ensures all fields are completed, displaying an error message if any are left vacant. It validates the IC to affirm it contains precisely 12 digits and only numeric characters, and examines the phone number for numeric characters only. The date of birth fields are also validated to ensure they are correctly filled.  
  
Password confirmation is essential; the function verifies that the password and confirm password fields match to maintain account security. Once validations are complete, the function connects to the MySQL database to execute SQL queries for inserting the new doctor's data.  
  
If the ‘clinic\_id’ is not provided, the function retrieves it based on the admin's full name, ensuring the new doctor is linked to the correct clinic. It then inserts the generic user data into the users table, setting the role to 'doctor' to distinguish this user from others in the system. The function also inserts additional doctor-specific information into the physicians table, including the user ID, entire name, clinic ID, availability status, gender, and IC.  
  
The function manages potential database errors by displaying appropriate error messages, ensuring the user is informed of any issues. Upon effective data insertion, a success message is shown, and the user is redirected to the clinic's admin main page.  
  
In summation, the ‘save\_doctor’ function efficiently validates and saves new doctor information, ensuring data accuracy and system integrity while providing a user-friendly experience.

# Function to validate and save the new doctor's information to the database

def save\_doctor():

    fullname = fullname\_entry.get()

    username = username\_entry.get()

    password = password\_entry.get()

    confirm\_password = confirm\_password\_entry.get()

    gender = gender\_combobox.get()

    ic = ic\_entry.get()

    email = email\_entry.get()

    phone = phone\_entry.get()

    address = address\_entry.get()

    year = year\_combobox.get()

    month = month\_combobox.get()

    day = day\_combobox.get()

    # Validate that all fields are filled

    if not (fullname and username and password and confirm\_password and gender and ic and email and phone and address and year and month and day):

        messagebox.showerror("Error", "Please fill all the fields.")

        return

    # Validate IC length and content

    if not ic.isdigit() or len(ic) != 12:

        messagebox.showerror("Error", "IC must contain exactly 12 digits.")

        return

    # Validate phone number

    if not phone.isdigit():

        messagebox.showerror("Error", "Phone number must contain only digits.")

        return

    # Validate date of birth

    if year == "Year":

        messagebox.showerror("Error", "Please fill in the year column.")

        return

    if month == "Month":

        messagebox.showerror("Error", "Please fill in the month column.")

        return

    if day == "Day":

        messagebox.showerror("Error", "Please fill in the day column.")

        return

    dob = f"{year}-{month}-{day}"

    # Validate passwords match

    if password != confirm\_password:

        messagebox.showerror("Error", "Passwords do not match.")

        return

    try:

        # Connect to the database

        connection = mysql.connector.connect(

            host='localhost',

            user='root',

            passwd='calladoctor1234',

            database='calladoctor'

        )

        cursor = connection.cursor()

        # Ensure clinic\_id is assigned correctly

        global clinic\_id

        if clinic\_id is None:

            cursor.execute('''

                SELECT admin\_clinics.clinic\_id

                FROM admin\_clinics

                JOIN users ON admin\_clinics.admin\_id = users.user\_id

                WHERE users.fullname = %s

            ''', (admin\_fullname,))

            clinic\_id\_result = cursor.fetchone()

            print(f"Debug: SQL query executed with admin\_fullname = {admin\_fullname}")  # Debug print statement

            if clinic\_id\_result:

                clinic\_id = clinic\_id\_result[0]

                print(f"Debug: Retrieved clinic\_id = {clinic\_id}")  # Debug print statement

            else:

                messagebox.showerror("Error", "Clinic ID not found.")

                print("Debug: Clinic ID not found.")  # Debug print statement

                return

        # Insert new user into users table

        cursor.execute('''

            INSERT INTO users (username, password, email, phone\_number, fullname, role, date\_of\_birth, address)

            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)

        ''', (username, password, email, phone, fullname, 'doctor', dob, address))

        user\_id = cursor.lastrowid

        # Insert new doctor into doctors table

        cursor.execute('''

            INSERT INTO doctors (user\_id, fullname, clinic\_id, is\_available, gender, identification\_number)

            VALUES (%s, %s, %s, %s, %s, %s)

        ''', (user\_id, fullname, clinic\_id, 1, gender, ic))

        connection.commit()

        messagebox.showinfo("Success", "Doctor added successfully!")

        root.destroy()

        os.system(f'python adminclinichome.py {clinic\_id} "{admin\_fullname}"')

    except mysql.connector.Error as err:

        messagebox.showerror("Database Error", f"Error: {err}")

    finally:

        if connection.is\_connected():

            cursor.close()

            connection.close()

* + 1. Test for delete\_doctor in deletedoctor page

The ‘delete\_doctor function is a vital element of the medical clinic administration system, designed to eliminate a doctor’s record from both the doctors and users tables in the database. This procedure begins by establishing a connection to the MySQL database using ‘mysql.connector.connect’, ensuring access to the correct database with the provided credentials.  
  
Once connected, the function creates a cursor object to execute SQL queries. It first deletes the doctor’s record from the physicians table using a parameterized DELETE statement to assure security and prevent SQL injection. Following this, an identical DELETE statement is executed to remove the corresponding user record from the users table. These measures ensure that all associated data for the clinician is completely removed.  
  
The function then commits the transaction to conclude the deletions, making the alterations permanent in the database. Upon successful completion, a success message is displayed using ‘ctk.CTkMessageBox.show\_info’, informing the user that the doctor has been deleted.  
  
In case of any database errors, the function captures the exceptions and displays an error message with ‘ctk.CTkMessageBox.show\_error’, ensuring the user is aware of any issues. Finally, the function ensures the database connection is terminated appropriately to prevent resource leakage.  
  
Overall, the ‘delete\_doctor’ function efficiently manages the removal of doctor records, assuring data integrity and providing explicit feedback to the user throughout the process.

# Function to delete a doctor

def delete\_doctor(doctor\_id, user\_id):

    try:

        # Connect to the database

        connection = mysql.connector.connect(

            host='localhost',  # Database host

            user='root',  # Database username

            passwd='calladoctor1234',  # Database password

            database='calladoctor'  # Database name

        )

        cursor = connection.cursor()  # Create a cursor object to execute SQL queries

        # Delete doctor from doctors table

        cursor.execute('DELETE FROM doctors WHERE doctor\_id = %s', (doctor\_id,))

        # Delete user from users table

        cursor.execute('DELETE FROM users WHERE user\_id = %s', (user\_id,))

        connection.commit()  # Commit the transaction to the database

        ctk.CTkMessageBox.show\_info("Success", "Doctor deleted successfully!")  # Show success message

        load\_doctors()  # Refresh the table after deletion

    except mysql.connector.Error as err:

        ctk.CTkMessageBox.show\_error("Database Error", f"Error: {err}")  # Show error message if any database error occurs

    finally:

        if connection.is\_connected():

            cursor.close()  # Close the cursor

            connection.close()  # Close the database connection

# **GitHub**

Contribution:

A graph on a screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Link for the GitHub: <https://github.com/linkesh31/SoftwareEng>

# **References**

*What is Agile Methodology in project management?*. Versatile & Robust Project Management Software. (n.d.). <https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/>

Staff, A. T. (2021, November 16). *Agile methodology: Advantages and disadvantages of an innovative method*. Twproject. <https://twproject.com/blog/agile-methodology-advantages-disadvantages-innovative-method/>

*What is software project management?*. Versatile & Robust Project Management Software. (n.d.-b). <https://www.wrike.com/project-management-guide/faq/what-is-software-project-management/>

*Agile model (software engineering) - javatpoint*. www.javatpoint.com. (n.d.). <https://www.javatpoint.com/software-engineering-agile-model>

*A guide to the Entity Relationship Diagram(ERD) - databasestar*. www.databasestar.com.. <https://www.databasestar.com/entity-relationship-diagram/>

*What is an appointment scheduling system? - Qmatic*. www.qmatic.com.. [https://www.qmatic.com/blog/appointment-scheduling-solution-what-how-and-why#:~:text=An%20appointment%20scheduling%20system%2C%20also,Mobile%20app%20booking](https://www.qmatic.com/blog/appointment-scheduling-solution-what-how-and-why%23:~:text=An%20appointment%20scheduling%20system%2C%20also,Mobile%20app%20booking)

*What is the purpose of an appointment system? - Engageware*. www.engageware.com.. [https://engageware.com/blog/purpose-of-appointment-system/#:~:text=At%20their%20core%2C%20appointment%20scheduling,a%20robust%20customer%20engagement%20strategy.](https://engageware.com/blog/purpose-of-appointment-system/%23:~:text=At%20their%20core%2C%20appointment%20scheduling,a%20robust%20customer%20engagement%20strategy.)