Here is a detailed explanation of each function and its purpose:

Function: `create_scrollable_frame(parent)`

Purpose:
Creates a scrollable frame inside a parent widget (e.g., a `tk.Tk` or `tk.Frame` object). This is useful when the content exceeds the visible area, and a scrollbar is needed to navigate.

What It Does:

1. Creates a `tk.Canvas` to act as the base for the scrollable content.
2. Adds a vertical scrollbar (`ttk.Scrollbar`) linked to the canvas for navigation.
3. Embeds a `tk.Frame` inside the canvas where widgets can be added.
4. Configures the canvas and scrollbar to work together.
5. Automatically adjusts the scroll region based on the size of the scrollable frame.

Returns:

- The `tk.Canvas` object and the `scrollable_frame` where widgets can be added.

Function: `admin_login()`

Purpose:
Provides a login mechanism for an administrator with a simple GUI window to input credentials.

What It Does:

1. Opens a new login window (`tk.Toplevel`).
2. Adds input fields (`tk.Entry`) for the admin username and password.
3. Includes a `Login` button that triggers the `check_admin()` function to validate credentials.
4. Checks if the entered username and password match predefined credentials (`admin` and `password`).
5. Displays a success message if the login is valid and closes the login window.
6. Shows an error message for invalid credentials and keeps the login window open.
7. Ensures the login window is modal (user cannot interact with the main application until the login window is closed).

Inner Function: `check_admin()`:

- Handles the validation of username and password.

Key Features:

- Protects the password field by hiding input using `show="*"`.
- Keeps the login window on top of the main application using `transient()` and `grab_set()`.

Here is a detailed explanation of the two functions you provided:

Function: `edit_member()`

Purpose:
Allows the admin to update a member's details, such as contact information, email, and membership expiration date.

What It Does:

1. Retrieves the member's name from the input field (`name_entry.get()`).
2. Searches for the member in the fitness center's member list by matching the name (case-insensitive).
3. Displays an error message if the member is not found.
4. Retrieves new data (contact info, email, and expiration date) from input fields.
5. Validates the expiration date format (`YYYY-MM-DD`). If invalid, shows an error message.
6. Updates the member's details in the database (the `fitness_center` dictionary).
7. Calls `save_data()` to persist the changes.
8. Displays a success message indicating the member's details were updated.
9. Clears the input fields by calling `clear_inputs()`.

Function: `delete_member()`

Purpose:
Allows the admin to remove a member from the fitness center's database.

What It Does:

1. Retrieves the member's name from the input field (`name_entry.get()`).
2. Loops through the list of members in `fitness_center["members"]`:
    - Adds all members except the one to delete into a new list (`updated_members`).
3. Replaces the original member list with the updated list, effectively removing the specified member.
4. Calls `save_data()` to persist the changes.
5. Displays a success message indicating the member was successfully deleted.
6. Clears the input fields by calling `clear_inputs()`.

Supporting Functions and Features:

1. **`save_data()`**:
    - This function is likely responsible for saving the updated `fitness_center` data to a file or database.
2. **`clear_inputs()`**:
    - Clears the input fields (`name_entry`, `contact_entry`, `email_entry`, and `expiry_entry`) to prepare for new input.

Suggestions for Improvement:

1. **Member Search Improvement**:
    - Allow partial matching for member names to handle typos or incomplete input.
2. **Confirmation Before Deletion**:
    - Add a confirmation dialog (`messagebox.askyesno`) before deleting a member to avoid accidental deletions.

Here's a detailed explanation of the `send_email_with_qr()` function and its purpose:

Function: `send_email_with_qr(member_email, member_name, qr_image_path)`

Purpose:
Sends an email with an attached QR code image to a member of the fitness center. The email includes a customized message with the member's name and an attached QR code.

What It Does:

1. **Set up email details**:
    - `sender_email`: The email address of the fitness center (used as the sender's email).
    - `sender_password`: The app-specific password used to authenticate the sender's email. It is recommended to use an app-specific password for better security rather than using the account password directly.
2. **Email subject and body**:
    - `subject`: The subject of the email, dynamically including the member's name.
    - `body`: The body of the email, including a greeting with the member's name and a message about the QR code for signing in.
3. **Create the email message**:
    - The `EmailMessage()` object is used to build the email.
    - The subject, sender, and recipient (member's email) are set for the message.
    - The body of the email is attached using the `set_content()` method in plain text format.
4. **Attach QR code image**:
    - The function reads the QR code image from the given path (`qr_image_path`) in binary mode.
    - It attaches the image as a PNG file to the email with the filename being `{member_name}_qr.png`.
5. **Sending the email**:
    - The function connects to the Gmail SMTP server (`smtp.gmail.com` on port 587) using the `smtplib.SMTP` class.
    - It starts TLS encryption using `server.starttls()` for secure communication.

- It logs in to the Gmail account using `server.login()` with the sender's email and app-specific password.
- The email message is sent using `server.send_message(msg)`.

6. **Exception Handling**:
    - If any exception occurs during the email sending process (e.g., network errors, authentication issues), the exception is caught, but it's silently handled (`pass`). It would be a good idea to log the exception or inform the user in a real-world application.

Security Considerations:

- **App-Specific Password**: The `sender_password` should be an app-specific password, which is a safer approach than using your actual Gmail account password. This password is generated in your Google account settings specifically for applications like this.

- **Exception Handling**: The current exception handling (`pass`) is minimal. It's generally a good idea to log exceptions for debugging or notify the user if something goes wrong.

Here is a breakdown of the functions you've provided, along with explanations of their purpose and operation:

Function: `generate_qr_code(member_name, contact_info)`

Purpose:
Generates a QR code for a member containing their name and contact information. It then sends the generated QR code to the member via email.

What It Does:

1. **Combines Member Info**:
    - Concatenates the member's name and contact information into a single string (`qr_data`).

2. **QR Code Generation**:
    - Uses the `qrcode` library to create a QR code with specific settings:
        - **version=1**: Smallest QR code.
        - **error_correction=qrcode.constants.ERROR_CORRECT_L**: Low error correction level.
        - **box_size=10**: Size of each box in the QR code.
        - **border=4**: Thickness of the border in boxes.
3. **Creating and Saving QR Code**:
    - Generates a QR code image with black boxes on a white background.
    - Saves the generated QR code image as `{member_name}_qr.png` in a specified directory (`FACE_IMAGES_DIR`).
4. **Sending the QR Code**:
    - Calls the `send_email_with_qr()` function to email the QR code image to the member.
5. **Returns**:
    - Returns the path of the saved QR code image.

Function: `view_activity_log()`

Purpose:
Displays the activity log for the fitness center, showing the names and sign-in times of members.

What It Does:

1. **Clears Previous Widgets**:
    - Clears any previous widgets in the `log_frame` to refresh the activity log display.
2. **Check for Activity Log**:
    - Checks if there is an activity log in the `fitness_center` data and if it contains entries.
    - If no activity log is found, it displays a message indicating that no logs are available.
3. **Displays Activity Log**:

- Loops through the activity log entries and displays each entry in the `log_frame` with the member's name and sign-in time.
- Uses a `tk.Label` widget to display each log entry.

Function: `sign_in_with_qr()`

Purpose:
Allows members to sign in by scanning their QR code with the camera, logs the sign-in time, and adds it to the activity log.

What It Does:

1. **Opens Camera**:
   - Uses OpenCV to capture video from the webcam (`cv2.VideoCapture(0)`).
   - Displays a message to the user to position the QR code in front of the camera.
2. **QR Code Detection**:
   - Initializes a QR code detector using OpenCV (`cv2.QRCodeDetector()`).
   - Continuously captures frames from the camera until a QR code is detected.
   - If a QR code is detected, the function decodes the data and extracts the member's name and contact information.
3. **Search for Member**:
   - Searches the fitness center's database for the member whose name and contact information match the decoded QR code data.
4. **Sign-In Logic**:
   - If the member is found, the sign-in time is logged into the `activity_log` with the current timestamp.
   - The activity log is saved, and a success message is displayed with the member's name and sign-in time.
5. **Error Handling**:
   - If the member is not found, an error message is displayed.
   - If the camera fails to capture a frame, an error message is shown.
6. **Exit Camera Feed**:
   - Displays the live camera feed while waiting for a QR code.
   - Exits the loop and closes the camera feed if the 'q' key is pressed.

Here is an explanation for each function in your code:

1. Directory for Face Images:

```
FACE_IMAGES_DIR = "face_images"
if not os.path.exists(FACE_IMAGES_DIR):
    os.makedirs(FACE_IMAGES_DIR)
```

- **Purpose**: Creates a directory named `face_images` to store images (e.g., photos or QR codes) related to members.
- **Explanation**: It first checks if the directory already exists. If it doesn't, the directory is created.

---

2. Initialize Fitness Center Data:

```
fitness_center = {
    "members": [],
    "activities": {
        "Gym": [],
        "Yoga": [],
        "Swimming": []
    },
    "activity_log": []
}
```

- **Purpose**: Initializes the structure to store data about the fitness center, including members, activities, and an activity log.
- **Explanation**:
    - `members`: List to hold information about each member.
    - `activities`: A dictionary that tracks which members are participating in specific activities (e.g., Gym, Yoga, Swimming).
    - `activity_log`: A list that records each activity (sign-ins, etc.) performed by members.

---

3. `save_data()`:

```
def save_data():
    with open("fitness_center.json", "w") as file:
        json.dump(fitness_center, file, indent=4)
```

- **Purpose**: Saves the current state of the fitness center data (members, activities, logs) to a JSON file.
- **Explanation**:
    - The `fitness_center` dictionary is serialized into a JSON format and saved to `fitness_center.json` file.
    - The `indent=4` argument makes the output more readable by adding indentation.

---

4. `load_data()`:

```
def load_data():
    global fitness_center
    if os.path.exists("fitness_center.json"):
        with open("fitness_center.json", "r") as file:
            fitness_center = json.load(file)
    if "activity_log" not in fitness_center:
        fitness_center["activity_log"] = []
```

- **Purpose**: Loads the fitness center data from the `fitness_center.json` file.
- **Explanation**:
    - It checks if the JSON file exists. If it does, the data is loaded back into the `fitness_center` dictionary.
    - If the `activity_log` key does not exist in the loaded data, it initializes it as an empty list.

---

5. `generate_report()`:

```
def generate_report():
    for widget in report_frame.winfo_children():
        widget.destroy()

    if not fitness_center["members"]:
        tk.Label(report_frame, text="No members registered yet.", font=("Arial", 12)).pack()
        return
```

```
    for member in fitness_center["members"]:
        member_details = f"Name: {member['name']}\n" \
                         f"Contact Info: {member['contact_info']}\n" \
                         f"Membership Expiry: {member['expiration_date']}\n" \
                         f"Activities: {', '.join(member['activities']) if member['activities'] else 'None'}"
        tk.Label(report_frame, text=member_details, justify="left", font=("Arial",
10)).pack(anchor="w")

        photo_path = member.get("photo_path")
        if photo_path and os.path.exists(photo_path):
            img = Image.open(photo_path).resize((100, 100))
            photo = ImageTk.PhotoImage(img)
            photo_label = tk.Label(report_frame, image=photo)
            photo_label.pack(anchor="w")
            photo_label.image = photo

        qr_code_path = member.get("qr_code_path")
        if qr_code_path and os.path.exists(qr_code_path):
            qr_img = Image.open(qr_code_path).resize((100, 100))
            qr_photo = ImageTk.PhotoImage(qr_img)
            qr_label = tk.Label(report_frame, image=qr_photo)
            qr_label.pack(anchor="w")
            qr_label.image = qr_photo

        tk.Label(report_frame, text="-" * 50).pack()
```

- **Purpose**: Generates a detailed report of all the members in the fitness center.
- **Explanation**:
    - First, it removes any existing widgets (if there are any) from the report frame to refresh it.
    - If there are no members in the fitness center, it shows a message saying "No members registered yet."
    - For each member, it displays their name, contact info, membership expiry date, and activities they are enrolled in.
    - If the member has a photo, it resizes the photo to 100x100 pixels and displays it.
    - Similarly, if the member has a QR code, it resizes the QR code and displays it.
    - A separator line is added after each member’s details.

These functions together help in managing the fitness center’s data by loading and saving the data, generating detailed reports about the members, and ensuring that photos, QR codes, and

other member-related information are handled properly. Let me know if you need further clarification on any function!

Explanation of `capture_face_image(member_name)`:

The purpose of this function is to capture a face image of a member using a webcam, allowing them to register or update their photo in the system. Here's a detailed breakdown:

---

1. Opening the Camera:

```
video_capture = cv2.VideoCapture(0)
```

- **Explanation**: This line opens the default camera (camera index `0`) for capturing video frames using OpenCV.

---

2. Showing an Informational Message:

```
messagebox.showinfo("Face Capture", "Please position your face in front of the camera.")
```

- **Explanation**: A pop-up message is shown to inform the user to position their face in front of the camera for the image capture.

---

3. Starting the Video Capture Loop:

```
while True:
    ret, frame = video_capture.read()
    if not ret:
        messagebox.showerror("Error", "Failed to access the camera.")
        return None
```

- **Explanation**:
  - This loop continuously captures frames from the webcam until the user either saves or cancels the capture.
  - If the camera fails to capture a frame, an error message is displayed, and the function returns `None` to indicate failure.

---

4. Displaying the Live Video Feed:
cv2.imshow("Face Capture (Press 's' to save or 'q' to cancel)", frame)

- **Explanation**: The live video feed from the webcam is displayed in a window, allowing the user to see themselves and adjust their position. The instructions for saving (`'s'`) or cancelling (`'q'`) are displayed on the window title.

---

5. Waiting for User Input:
key = cv2.waitKey(1) & 0xFF

- **Explanation**: This waits for a key press from the user. The `cv2.waitKey(1)` function waits for 1 millisecond, and `& 0xFF` ensures the key press is captured in a compatible format.

---

6. Saving the Image:

```
if key == ord('s'):  # If 's' is pressed, save the image
    image_path = os.path.join(FACE_IMAGES_DIR, f"{member_name}.jpg")
    cv2.imwrite(image_path, frame)
    video_capture.release()  # Release the camera
    cv2.destroyAllWindows()  # Close the OpenCV window
    return image_path  # Return the path to the saved image
```

- **Explanation**:
    - If the user presses the `'s'` key, the captured frame is saved as an image file in the `face_images` directory.
    - The image is saved using the member's name as the filename (e.g., `JohnDoe.jpg`).
    - After saving the image, the camera is released, and the OpenCV window is closed.
    - The path to the saved image is returned.

---

7. Cancelling the Capture:

```
elif key == ord('q'):  # If 'q' is pressed, cancel the capture
    video_capture.release()  # Release the camera
    cv2.destroyAllWindows()  # Close the OpenCV window
    return None  # Return None if the capture was cancelled
```

- **Explanation**:
    - If the user presses the `'q'` key, the image capture is cancelled.
    - The camera is released, and the OpenCV window is closed.
    - `None` is returned to indicate that the capture was cancelled and no image was saved.

Summary:

The function `capture_face_image()` captures a face image for a member by opening the camera and allowing the user to either save the image by pressing `'s'` or cancel by pressing `'q'`. The captured image is saved in the `face_images` directory with the member's name as the file name.

Explanation of Functions:

1. add_member():

This function is responsible for adding a new member to the fitness center, capturing their face image, generating a QR code, and sending the QR code to the member's email. Here's the step-by-step breakdown:

Step 1: Collect Member Information:

```
 name = name_entry.get()
contact_info = contact_entry.get()
email = email_entry.get()
expiry_date_str = expiry_entry.get()
```

-
    - The function collects input data from the user through text entry fields (name, contact info, email, and expiration date).

Step 2: Validate the Email Address:

```
if not email or "@" not in email or "." not in email:
    messagebox.showerror("Error", "Invalid email address!")
    return
```

-
    - The function checks if the entered email address contains the "@" and "." symbols, which are basic validation criteria for emails.

Step 3: Validate and Parse Expiration Date:

```
try:
    expiration_date = datetime.datetime.strptime(expiry_date_str, "%Y-%m-%d").date()
except ValueError:
    messagebox.showerror("Error", "Invalid date format! Please enter in YYYY-MM-DD format.")
    return
```

-
    - It attempts to parse the expiration date entered by the user into a `datetime.date` object using the `strptime()` function. If the format is incorrect, an error message is shown.

Step 4: Capture Face Image:

```
face_image_path = capture_face_image(name)
if not face_image_path:
    return
```

-
    - The function calls `capture_face_image(name)`, which opens the camera and captures the member's face. If the capture fails (i.e., the user cancels), the function returns early without proceeding further.

Step 5: Generate QR Code:

```
qr_code_path = generate_qr_code(name, contact_info)
```

-
    - A QR code is generated for the member based on their name and contact info using the `generate_qr_code()` function.

Step 6: Send Email with QR Code:

```
try:
    send_email_with_qr(email, name, qr_code_path)
except Exception as e:
    messagebox.showerror("Error", f"Failed to send email to {email}: {e}")
    return
```

-
    - The function attempts to send the QR code to the member's email using `send_email_with_qr()`. If sending fails, it shows an error message.

Step 7: Add Member to Fitness Center:

```
fitness_center["members"].append({
    "name": name,
    "contact_info": contact_info,
    "email": email,
    "expiration_date": str(expiration_date),
    "activities": [],
    "photo_path": face_image_path,
    "qr_code_path": qr_code_path
})
```

-
    - The new member's details, including their face image path and QR code path, are added to the `fitness_center["members"]` list.

Step 8: Save Data to JSON File:

```
save_data()
```

-
    - The updated fitness center data is saved to a file using the `save_data()` function.

Step 9: Show Success Message and Clear Input Fields:

```
messagebox.showinfo("Success", f"Member {name} added successfully, and QR code sent to {email}!")
clear_inputs()
```

-

- A success message is displayed, and the input fields are cleared using `clear_inputs()`.

---

2. register_for_activity():

This function allows a member to register for a specific activity (e.g., gym, yoga, swimming) in the fitness center. Here's the step-by-step breakdown:

Step 1: Collect Member Name and Activity Name:

```
 member_name = member_name_entry.get()
activity_name = activity_entry.get()
```

- - The function collects the member's name and the activity they wish to register for from input fields.

Step 2: Check If Activity Exists:

```
 if activity_name not in fitness_center["activities"]:
    messagebox.showerror("Error", f"Activity '{activity_name}' not found.")
    return
```

- - It checks if the entered activity name exists in the list of available activities (`fitness_center["activities"]`). If not, an error message is displayed.

Step 3: Find the Member:

```
 member = None
for m in fitness_center["members"]:
   if m["name"].lower() == member_name.lower():
       member = m
       break
```

- - The function searches for the member in the `fitness_center["members"]` list by comparing their name (case-insensitive). If the member is found, they are assigned to the `member` variable.

Step 4: Handle If Member Not Found:
```
if not member:
    messagebox.showerror("Error", "Member not found.")
    return
```

-
    - If no matching member is found, an error message is displayed.

Step 5: Register the Member for the Activity:
```
member["activities"].append(activity_name)
```

-
    - The member is added to the list of participants for the specified activity.

Step 6: Save Data to JSON File:
```
save_data()
```

-
    - The updated fitness center data is saved to the file.

Step 7: Show Success Message:
```
messagebox.showinfo("Success", f"{member_name} has been successfully registered for {activity_name}.")
```

-
    - A success message is displayed, indicating the member was successfully registered for the activity.

Summary:

- `add_member()` adds a new member to the fitness center by validating inputs, capturing a face image, generating a QR code, sending it to the member's email, and saving the updated data.
- `register_for_activity()` allows a member to register for a specific activity, checks if the member and activity exist, and updates the data.

Explanation of Functions:

1. clear_inputs():

This function clears all the input fields in the form, ensuring that the user interface is reset after a member is added or activity is registered. Here's the breakdown:

Step 1: Clear Name Entry:

```
name_entry.delete(0, tk.END)  # Clear name input
```

-
    - Clears the name field (entry widget for member name) by deleting all characters from the beginning (`0`) to the end (`tk.END`).

Step 2: Clear Contact Info Entry:

```
contact_entry.delete(0, tk.END)  # Clear contact info input
```

-
    - Clears the contact info field in a similar manner.

Step 3: Clear Expiry Date Entry:

```
expiry_entry.delete(0, tk.END)  # Clear expiration date input
```

-
    - Clears the expiration date field.

Step 4: Clear Member Name Entry for Activity Registration:

```
member_name_entry.delete(0, tk.END)  # Clear member name input for activity registration
```

-
    - Clears the member name field used for activity registration.

Step 5: Clear Activity Entry:

```
activity_entry.delete(0, tk.END)  # Clear activity input
```

-
    - Clears the activity field used to register a member for a specific activity.

Step 6: Clear Email Entry:

```
email_entry.delete(0, tk.END)  # Clear email input
```

-
    - Clears the email field.

This ensures all input fields are empty, which is useful after a member is added or an activity is registered, readying the form for the next action.

---

2. search_and_filter_members():

This function allows users to search for members by name or filter them based on their registered activities. The GUI creates a separate window for these operations.

Step 1: Create `search_members()` Function:

```
def search_members():
    search_query = search_entry.get().lower()  # Get the search query and convert it to lowercase
    result_text.delete(1.0, tk.END)  # Clear previous search results

    for member in fitness_center["members"]:
        if search_query in member["name"].lower():  # If search query matches part of the member's name (case-insensitive)
            member_details = f"Name: {member['name']}\n" \
                             f"Contact: {member['contact_info']}\n" \
                             f"Email: {member['email']}\n" \
                             f"Expiry Date: {member['expiration_date']}\n" \
                             f"Activities: {', '.join(member['activities']) if member['activities'] else 'None'}\n"
            result_text.insert(tk.END, member_details + "-" * 50 + "\n")
```

-
    - **Search Query**: The function takes the search query entered by the user, converts it to lowercase for case-insensitive comparison, and clears the previous results.
    - **Loop Through Members**: It loops through the list of members in the `fitness_center["members"]` dictionary and checks if the search query exists in the member's name.
    - **Display Member Details**: If a match is found, it formats the member's details and inserts them into the result text area.

Step 2: Create `filter_members()` Function:

```
 def filter_members():
    filter_activity = filter_entry.get().lower()  # Get the filter query for activity and convert to lowercase
    result_text.delete(1.0, tk.END)  # Clear previous filter results

    for member in fitness_center["members"]:
        if any(filter_activity in activity.lower() for activity in member["activities"]):  # If activity is found in member's activities
            member_details = f"Name: {member['name']}\n" \
                             f"Contact: {member['contact_info']}\n" \
                             f"Email: {member['email']}\n" \
                             f"Expiry Date: {member['expiration_date']}\n" \
                             f"Activities: {', '.join(member['activities']) if member['activities'] else 'None'}\n"
            result_text.insert(tk.END, member_details + "-" * 50 + "\n")
```

-
    - **Activity Filter**: Similar to the search function, this function takes a filter query for an activity, converts it to lowercase, and clears previous results.
    - **Loop Through Members**: It loops through the list of members and checks if any of their activities match the filter query. If a match is found, the member's details are displayed.

Step 3: Create Search and Filter Window:

```
 search_window = tk.Toplevel(root)  # Create a new top-level window for search and filter
search_window.title("Search and Filter Members")  # Set the title of the window
search_window.geometry("500x400")  # Set the size of the window
```

-
    - A new window (`Toplevel`) is created to handle the search and filter functionality separately from the main window.

- **Step 4: Create GUI Components for Search and Filter**:
    - **Search Section**: A label and entry widget are added for the search functionality. The user can type the member's name to search for them. A button triggers the `search_members()` function.
    - **Filter Section**: A similar label and entry widget are added for filtering based on activities. A button triggers the `filter_members()` function.

- **Results Display**: A `Text` widget is used to display the search or filter results, with the member details shown line by line, and a separator is added after each member's details.

Summary:

- `clear_inputs()` clears all input fields in the form to reset the UI for the next action.
- `search_and_filter_members()` provides search and filter functionality in a new window, allowing users to find members by name or filter them by activity. The results are displayed in a `Text` widget with the members' details formatted and separated for easy readability.

This section of code is for setting up the graphical user interface (GUI) for your Fitness Center Management System. Below is a breakdown of the components and their functionality:

1. Search and Filter Members Button:

search_filter_button = tk.Button(main_frame, text="Search and Filter Members", command=search_and_filter_members)
search_filter_button.pack()

- **Button**: This button, when clicked, opens the search and filter window for searching members by name or filtering them by activity.

---

2. Inputs for Adding a New Member:

These widgets collect the member’s details:

Name: Text input for the member's name.
name_label = tk.Label(main_frame, text="Name:")
name_label.pack()
name_entry = tk.Entry(main_frame)
name_entry.pack()

-

Contact Info: Text input for the member's contact information (e.g., phone number).
contact_label = tk.Label(main_frame, text="Contact Info:")
contact_label.pack()

```
contact_entry = tk.Entry(main_frame)
contact_entry.pack()
```

-

Email: Text input for the member's email address.

```
email_label = tk.Label(main_frame, text="Email:")
email_label.pack()
email_entry = tk.Entry(main_frame)
email_entry.pack()
```

-

Expiry Date: Text input for the membership expiry date, expecting the format `YYYY-MM-DD`.

```
expiry_label = tk.Label(main_frame, text="Membership Expiry (YYYY-MM-DD):")
expiry_label.pack()
expiry_entry = tk.Entry(main_frame)
expiry_entry.pack()
```

-

---

3. Buttons for Adding, Editing, and Deleting Members:

Add Member: Adds a new member using the `add_member()` function.

```
add_member_button = tk.Button(main_frame, text="Add Member", command=add_member)
add_member_button.pack()
```

-

Edit Member: This button triggers the `edit_member()` function, but this functionality has not been implemented yet.

```
edit_member_button = tk.Button(main_frame, text="Edit Member", command=edit_member)
edit_member_button.pack()
```

-

Delete Member: This button triggers the `delete_member()` function, but this functionality has not been implemented yet.

```
delete_member_button = tk.Button(main_frame, text="Delete Member",
command=delete_member)
delete_member_button.pack()
```

-

---

4. Inputs for Registering a Member for an Activity:

These inputs collect information for registering a member for an activity:

Member Name: Text input for the member's name.

```
 member_name_label = tk.Label(main_frame, text="Member Name:")
member_name_label.pack()
member_name_entry = tk.Entry(main_frame)
member_name_entry.pack()
```

-

Activity Name: Text input for the activity name.

```
 activity_label = tk.Label(main_frame, text="Activity Name:")
activity_label.pack()
activity_entry = tk.Entry(main_frame)
activity_entry.pack()
```

-

Register for Activity: Registers the member for an activity using the `register_for_activity()` function.

```
 register_activity_button = tk.Button(main_frame, text="Register for Activity",
command=register_for_activity)
register_activity_button.pack()
```

-

---

5. Sign-In with QR Code:

This button allows members to sign in by scanning a QR code. The actual functionality will be handled by the `sign_in_with_qr()` function.

```
sign_in_button = tk.Button(main_frame, text="Sign In with QR", command=sign_in_with_qr)
```

sign_in_button.pack()

---

6. Buttons for Viewing Activity Log and Generating Reports:

These buttons trigger functions to view the activity log and generate reports:

View Activity Log: Triggers the `view_activity_log()` function.
view_log_button = tk.Button(main_frame, text="View Activity Log", command=view_activity_log)
view_log_button.pack()

-

Generate Report: Triggers the `generate_report()` function.
generate_report_button = tk.Button(main_frame, text="Generate Report",
command=generate_report)
generate_report_button.pack()

-

---

7. Frames for Displaying Logs and Reports:

These frames are created to display logs and reports, though the actual content will be added later:

Activity Log Frame: A frame to contain the activity log.
log_frame = tk.Frame(main_frame)
log_frame.pack(pady=10)

-

Report Frame: A frame to contain generated reports.
report_frame = tk.Frame(main_frame)
report_frame.pack(pady=10)

-

---

8. Loading Data on Startup:

This function call loads the data from a file when the program starts:

load_data()

---

9. Starting the GUI Mainloop:

This starts the Tkinter event loop, making the GUI interactive:

root.mainloop()

Summary of the GUI Components:

- This GUI provides a comprehensive set of features for managing members in a fitness center. It includes forms for adding members, registering them for activities, and signing them in with QR codes.
- There are buttons for searching/filtering members, generating reports, viewing activity logs, and a section for managing member details.