Project Name: Medify

Prepared By:
Ahmed Sameh 202202151
Salma Rami 202201759
Shahd Tarek 202202263
Mohamed Tarek 202201058
Seif Amr 202201510

Functional Documentation for Front-End (HTML, CSS, JS)

Overview:

THIS FRONT-END PROJECT HAS DUAL LOG IN PAGE AND USER PAGE. USER PAGE HAS PROFILE MANAGE PATIENT, BOOK A DOCTOR AND APPOINTMENT REMINDER AND NOTIFICATION FUNCTIONS WHILE LOGIN PAGE CONSISTS OF SIGN UP FORM AND LOGIN FORM..

Login Page:

HTML:

- **Structure:**
    - **Form elements:**
        - Fields in signup form include username, email, password and role which can be selected from the drop down list – the role selection is optional.
        - The login form includes email and password fields.
    - View signup & login forms toggle button.
    - The forms are headermarked with appropriate labels and can be submitted or toggled when a button is pressed or targeted.

CSS:

- **General Styles:**
    - The forms are aligned to the center of the screen thorough flexing which is vertically centered as well.
    - Background includes gradient with blue color.
    - Styles for fonts and input types, buttons and fields labels.
- **Signup and Login Form Styles:**

- Input fields are covered with normal background colors with uniform enclosure margins with rounded corners for a neater and more pleasing design while buttons has hover actions that make them more appealing..
- The button has hover actions which help the user more
- Login and Signup views are controlled by toggle buttons that make use of CSS to alter the view of the forms.

JavaScript:

- **Checkbox State:**
  - This checkbox is used to select signup views or login views which can also be useful during sign ups.
  - The CSS would take over for transition along with active states of sections throughout the forms decorated by checkbox.

User Page:

HTML:

- **Navbar:**
  - Contains links for moving to other parts of the user page including Home, Profile, Doctors, Appointments and Notifications.
- **Sections:**
  - **Home Section:**
    - A very important section of the page, consisting of a header, a brief description text and a button that enables link to the doctors' section.
  - **Profile Section:**
    - Shows user's name, email and an "Edit profile" option which opens when clicked allowing users to change their name or email.
  - **Doctors Section:**
    - The available doctors are categorized according to their specialties, where each of them has an 'apply for appointment' button.
  - **Appointments: Offers the booked physician's vacant appointment schedules. The appointment schedule of the physician depends on the name of the doctor.**
  - **Notifications Section:**
    - It shows the messages intended for the user.

- **Footer:**
    - A simple footer with copyright information.

CSS:

- **General Styles:**
    - The navbar is made sticky on thetop and has a hover transition effect on links.
    - For the home section, the welcome section has a gradient background Photo, While the text is bold.
    - The profile section utilizes cards to store user data with an emphasis on transitions on input type of components.
    - Doctor cards feature vivid transitions on mouse over animation.
    - Bookable appointment slots have a table structure in this section with every time slot having a “book” option to facilitate scheduling appointments.
- **Button Hover Effects:**
    - Wherever buttons are available on the user page, such as on the doctor cards as well as the doctors’ profile, hovering the pointer over it activates an alternate effect where buttons’ backgrounds change accordingly or are slightly scaled for aesthetic purposes.

JavaScript:

- **Dynamic Content Handling:**
    - **`selectDoctor()` function**: The selectDoctor function is triggered by the user whenever they click the 'Book Appointment' button corresponding to the doctor. After this, it triggers available time slots that were previously created in an object (`doctorSlots`), fetches and displays them in a table for patients to see and allow them to create an appointment.
    - **Profile Editing Functions:**
        - **`editProfile()`**: Enables the user to enter their name and email in the fields that are going to be modifiable.
        - **`saveProfile()`**: After submitting the modifications, this function stores them in the system somehow.
        - **`cancelEdit()`**: This function halts the course of subject profile updates and articles return to standard.
    - **Notification Handling**:

    - Notifications are added in constant interval with respect to time and such actions are evident in the screen notification bar.

Components Interaction:

- **Login and Signup Toggle**: Log in page is integrated with a checkbox which enables the user to toggle between signup and login form and the connection of these forms is such that no page reload is required..

- **Profile Editing**: When a user gets to their user space, the Profile Section shall permit him to have control over his name and email aspects. Once clicked, the ‘Edit Profile’ activates the fields and with JavaScript functions, these fields are executing the commands to save the created changes.

- **Doctor Booking**: Rather than trying to look at a list somewhere, the Doctors Section creates a list for each available doctor and that list has cards ready for bookings where clicking it will call the selectDoctor() function that retrieves time available for appointments.
- **Appointment Slot Selection**: For this section, the Appointments Section gives the available appointments times of the selected doctor. Users are presented with the option to click the “Book” button to now confirm an appointment. The necessary API could extend listening if a user wish to keep it for real-info storage in the future solutions.

- **Notifications Section**: It could show in the notifications section any user-specific notifications. The notifications can be appended to the array through JavaScript technology.

Future Considerations:

- **Form Validation**: The presentation encourages the incorporation of form validation to the login wearing and signup pages to avoid the submission of erroneous information.
- **Back-End Integration**: Synchronise with a provided back-end Application Programming Interface to be able save data pertaining to user profile, user’s appointment overlay as well as notifications.

- **Session Management**: Session management was included and ideally the user stays authenticated in the application, within the same session while interchanging multiple pages..

This documentation is helpful for a simple understanding of the framework used in developing the front-end functions, styling, and interaction, as well as for the working features of the current version and the plans of this project development.

Functional Documentation for Back-End (Flask, SQLite, Python)

Overview:

This project is a user management and authentication system with admin and user capabilities. Additionally, it supports functions such as signup, login & session management and role based dashboards. Flask helps to control the views and processing tasks, SQLite the database, and session handling tasks to ensure some degree of personalization

---

Login and Signup Features:

Flask Routes:

- **`/` (Login Page)**:
  Serves a view page with login form offered.
    - **Forms**:
        - Fields: Email, Password.
        - Action: When this form is filled, information is submitted to the `/login` where it will be checked.
    - Users are taken to the appropriate dashboards depending on whether they are (`user` or `admin`).
- **`/signup` (Signup Handler)**:
  This action is performed after submitting the signup form.
    - **Logic**:
        - Validates email for `@admin` or `@user`.

    - Enrols the new account in the databases (`admins` or `users`) that belong there.
    - A welcome message is displayed and prompted to a login page if no unconsigned parts have registered it.

Database Design (SQLite):

- **Tables**:
    - **`users`**: which holds user details (email, password).
    - **`admins`**: which are admin details (email, password).
      Both of these use email and id, respectively, as unique identification fields within these two tables.
- **Initialization**:
    - Database; owing to the purpose of this server, needs to initialize in order to ensure that the required tables and their content will exist to intergrate this into the client application. `init_db()` will perform this function on server bootup.

Role-Based Dashboards:

Routes:

- **`/indexuser` (User Dashboard)**:
    - These work with the logged in users having the permissions identification of normal `user`.
    - If an attempt is made to access it without having an active session, the user is redirected to / (the login page).
- **`/indexadmin` (Admin Dashboard)**:
    - They are meant for logged in users with the identification of `admin`.
    - If this route is accessed without having an active session, it Redirects to `/` (login page)

Session Management:

- **Flask Sessions**:
    - **Storage**: `session['user_type']` determines role (`user` or `admin`).
    - **On Login**:
        - Valid credentials, the session with the corresponding type of the role is set.
    - **On Logout**:
        - `session.pop('user_type', None)` removes the session and the user is taken to the login page.

Security Features:

1. **Unique Email Validation**:
    - Emails can be said or stored in this case on the database, there is no single email that can be and as such duplication of accounts will be not allowed.
    - SQLite's `UNIQUE` constraint on the `email` field ensures integrity.
2. **Role Validation**:
    - Automatic signups are the role driven emails and these includes which are (`@admin` or `@user`).
    - This will prevent attempts such as the assignment of a role without authority
3. **Session Handling**:
    - Restricted access routes meaning that the user does not have permission to access them, lead to the action whereby such a user is taken back to the login page.

Scalability and Future Enhancements:

Current Features:

- Basic login/signup functionality.
- Role-based session management and dashboards.

Suggested Improvements:

1. **Password Encryption**:
    - passwords can be hashed using various libraries like bcrypt or Werkzeug before they are stored in the database
2. **Error Handling**:
    - : Provide error messages, for instance, if a user enters their credentials wrongly or when requested fields are empty
3. **Form Validation**:
   The signup and logon forms need to have a server-side validation during the registration process where invalid data should not be accepted for submission
4. **Role-Specific Features**:
    - Other features which are role based include:
        - **Admin**: Can manage the users and monitor usage through logs.
        - **User**: Can edit their profile and receive certain recommendations according to the log in profile
5. **Session Timeout**:
    - Proceed to include session timeouts, which enhance security features of the system
6. **Database Migration**:
    - Start using a distributed database (for example PostgreSQL or MySQL) to allow the system to be used in large scale systems

---

Summary:

The system described backs up the user and admin authentication coupled with the role management. It has been established that development and deployment is easy with the use of Flask and SQLite. The system has also been designed aiming at being developed further to meet required application standards which are advanced.