CSE471: System Analysis and Design

Project Report

Project Title: DigiCare

Table of Contents

Introduction

Digicare is a dynamic health management platform designed to empower users in tracking their health progress and receiving personalized assistance. Developed on Flask backend complemented by HTML, CSS and JavaScript frontend, this platform offers an array of features catering to various aspects of health management and assistance.

Functional Requirements

User Management:

- **Home Page:** The home page serves as a central hub, providing users with an overview of Digicare's functionalities.
- **Sign Up:** This feature enables users to register for a Digicare account, granting access to the platform's suite of features.
- **Log In and Log Out:** Users can securely access their accounts through the login feature and log out when the session is complete.
- **Password Recovery:** In the event of forgotten passwords, users can recover access via a secure process involving security question authentication.
- **Profile Setup:** Users can set up their profile data on the platform.
- **Editing profile:** Users can customize their profiles, facilitating personalized health tracking within the platform.

Health Data and Goal Tracking:

- **Set Health Data, Goals, and Medications:** This feature empowers users to input and manage personal health metrics, goals, and medication details.
- **View Health Records:** Users gain access to a comprehensive overview of their health data, goals, and medications, facilitating easy tracking.
- **Goal and Medication Completion:** Users can mark goals or medications as completed, with real-time updates reflecting their progress.
- **Progress Comparison:** Digicare provides comparative progress charts, allowing users to compare their individual achievements with aggregate user data.

Health Calculators and Blood Donation:

- **Health Calculators:** Interactive BMI, BMR, and Body Fat calculators provide users with insightful health metrics and feedback.
- **Blood Donor Network:** Users can connect with potential blood donors within Digicare, fostering a supportive community for blood donation.
- **Request Blood Donation:** This feature allows users to request blood donation, providing necessary contact information for potential donors.
- **Blood Request Notifications:** Real-time notifications keep users informed about incoming blood donation requests and acceptances.

Blood Donation and Admin:

- **Blood Request Management:** A notification panel empowers blood request receivers to efficiently manage incoming requests.
- **Accept/Reject Blood Requests:** Users can make informed decisions on blood donation requests, accepting or rejecting based on their availability.
- **Admin Panel:** An exclusive access point for administrators to oversee user management and platform activities.
- **User Management:** Administrators have the capability to view details of all users registered on Digicare.
- **Ban/Unban:** The admin can apply bans or lift them as necessary, ensuring a mechanism to address unusual or unethical user activities.

User Manual

Opening an Account / Sign Up / Registration:

In the system, after setting it up, users will be directed to the Home page. Here, they need to either open an account or register by clicking the Sign-up button, leading them to the registration page. On this page, users must fill in the form with accurate details. It's crucial to remember the email, password, and security question answers for future logins. While the Digicare system provides a password reset option, remembering the security question answers is necessary for successful access. So, to make it simplify for a successful registration, these are the steps to follow:

1. Upon system setup, navigate to the Home page.
2. Click on the Sign-up button to create an account.
3. Fill in the required details on the registration page.
4. Take note of the email, password, and security question answers for future logins.
5. With the registered information, you can access the Digicare system. If needed, there's an option to reset the password, but remembering security question answers is essential for

this process.

Figure 1 The Homepage of DigiCare

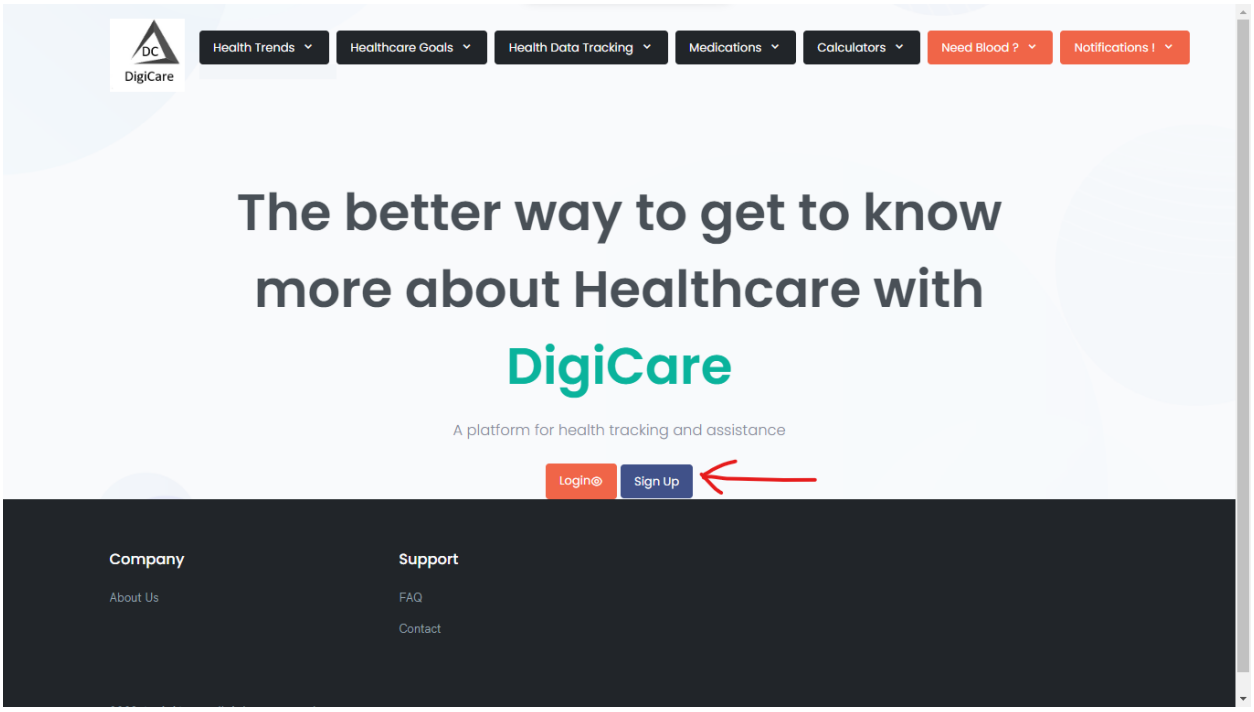


Figure 2: Log in and Sign-up button in Homepage

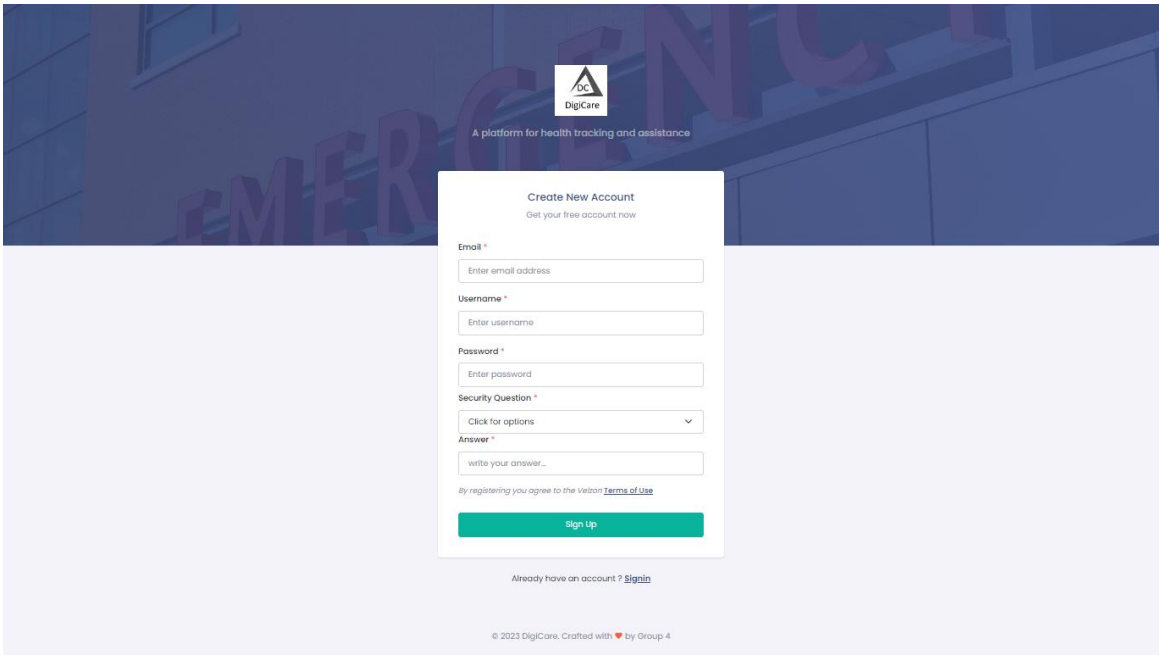


Figure 3: Registration / Sign-up page of DigiCare

Log in

To access the DigiCare system, locate and click the "Log in" button. This action will direct the user to the login page, where the user needs to enter the email and password associated with their account. Upon correctly inputting the credentials, you will successfully log in to your account. So, to make it simplify for a successful log in, these are the steps to follow:

1. Find and click the "Log in" button in the DigiCare system.
2. Upon clicking, the user will be redirected to the login page.
3. Input the email and password linked to your account.
4. Ensure the accuracy of the entered email and password.
5. If the credentials are correct, the user will be successfully logged into your DigiCare account.

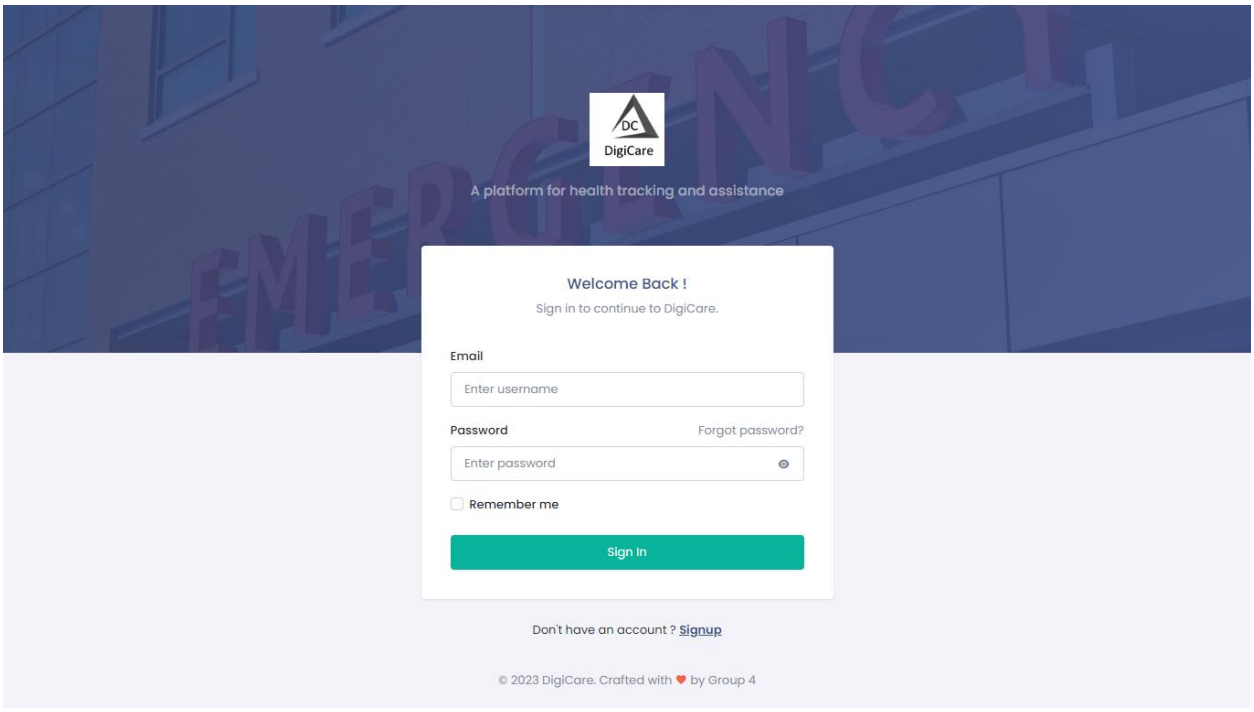


Figure 4: Log in Page

Forgetting Password / Reset password

In instances where users encounter password-related issues such as forgetting or mistyping, the DigiCare system provides a "Forget Password?" feature for resolution. To initiate the password reset process, users should locate the "Forget Password?" button on the login page. Upon clicking, they will be directed to the password reset page. Here, users need to enter their email, confirm the accurate security question used during sign-up, and provide the correct answer. If all

the information aligns accurately, the reset page will appear, allowing users to set a new password. So, to make it simplify these are the steps to follow:

1. On the login page, locate and click the "Forget Password?" button.
2. The click will redirect you to the forgotten password page.
3. Input your email and verify the security question and its answer used during sign-up.
4. Ensure that all information entered is correct and matches the records.
5. If the details are accurate, the reset page will appear, allowing you to set a new password.

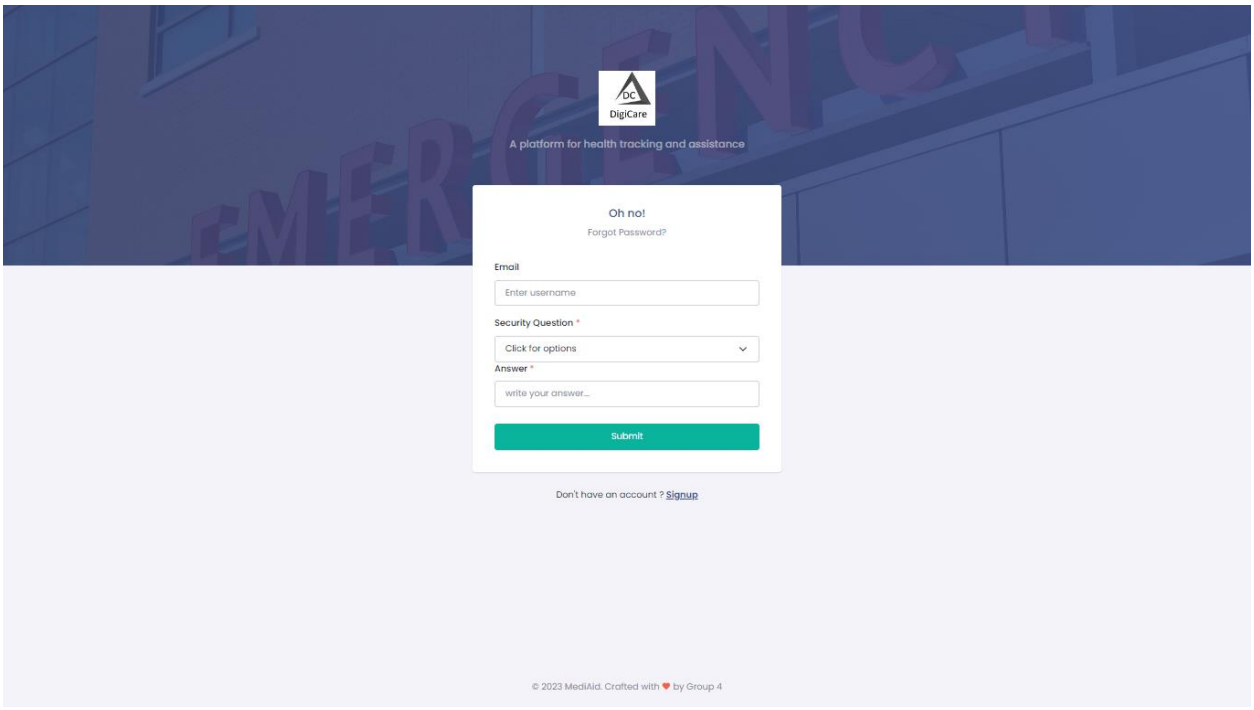


Figure 5: Forget Password page

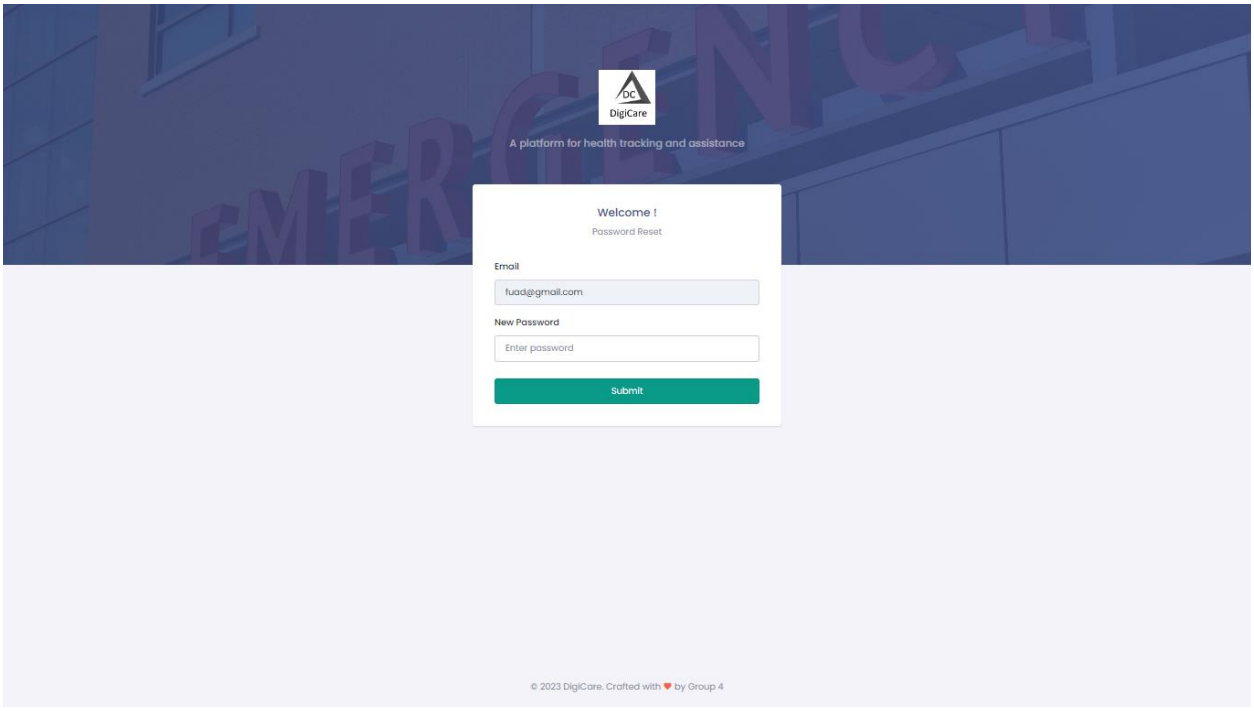


Figure 6: Reset Password page.

Profile Setup

Upon initial login, users will be automatically directed to the profile setup page. Subsequent logins will present an option labeled "Profile," which allows users to both view and input their information.

Figure 7: Profile button on the Home page

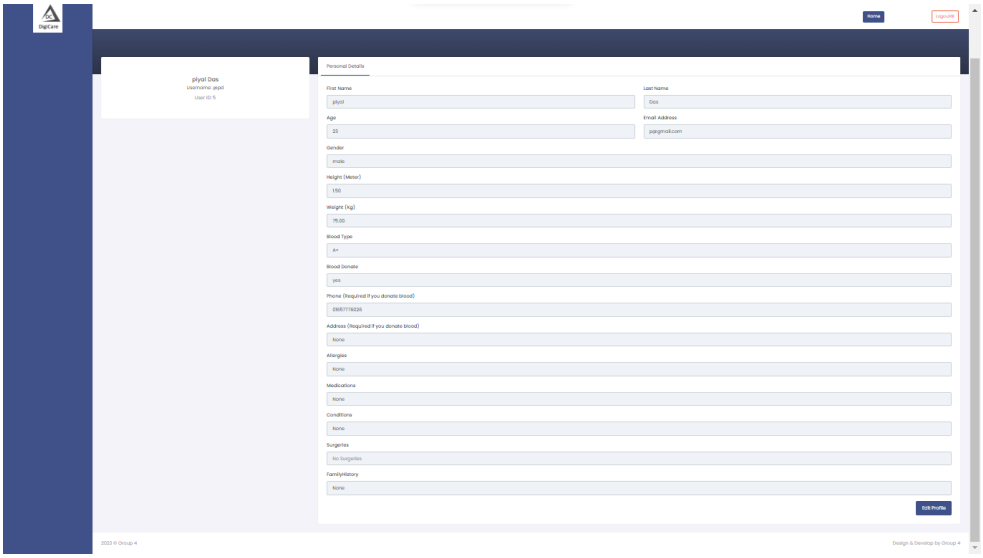


Figure 8: Profile Information setup page

Edit Profile

On the profile page, users have the option to edit their information. To initiate the editing process, locate the "Edit Profile" button situated beneath the profile details. Clicking this button will keep you on the same page, allowing you to modify your personal data at any time.

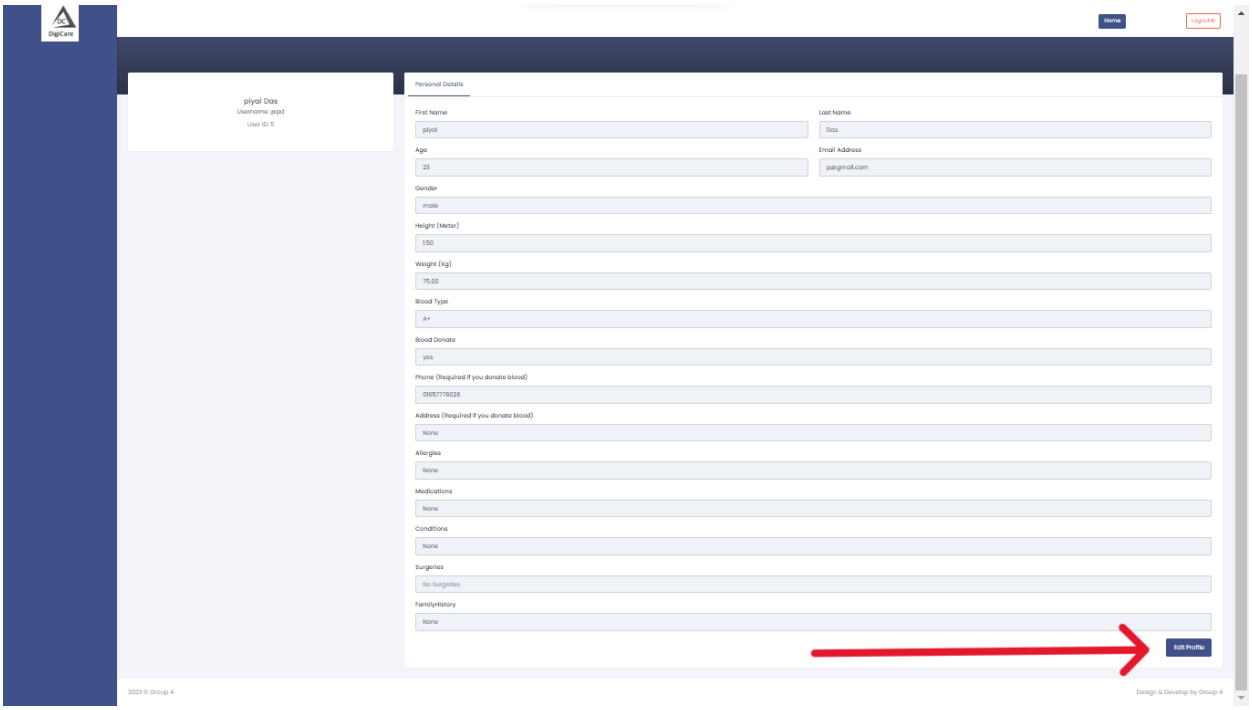


Figure 9: Edit Profile option.

Log out

To conclude their session, users can log out by clicking the "Log Out" button. On the Home page, a prominently placed "Log Out" button is available. Additionally, users can log out from any page within the app by accessing the logout option in the header menu section. Upon

selecting the logout option, users will be redirected to the home page.

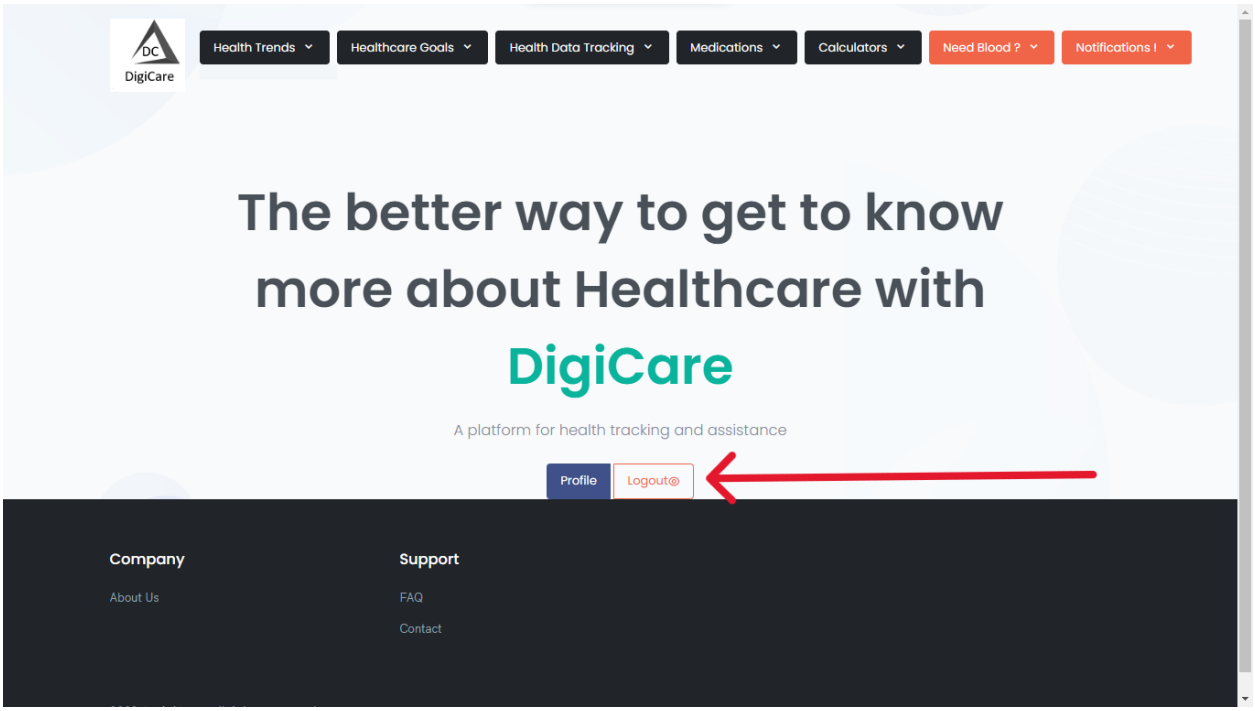


Figure 10: Log out from Home page

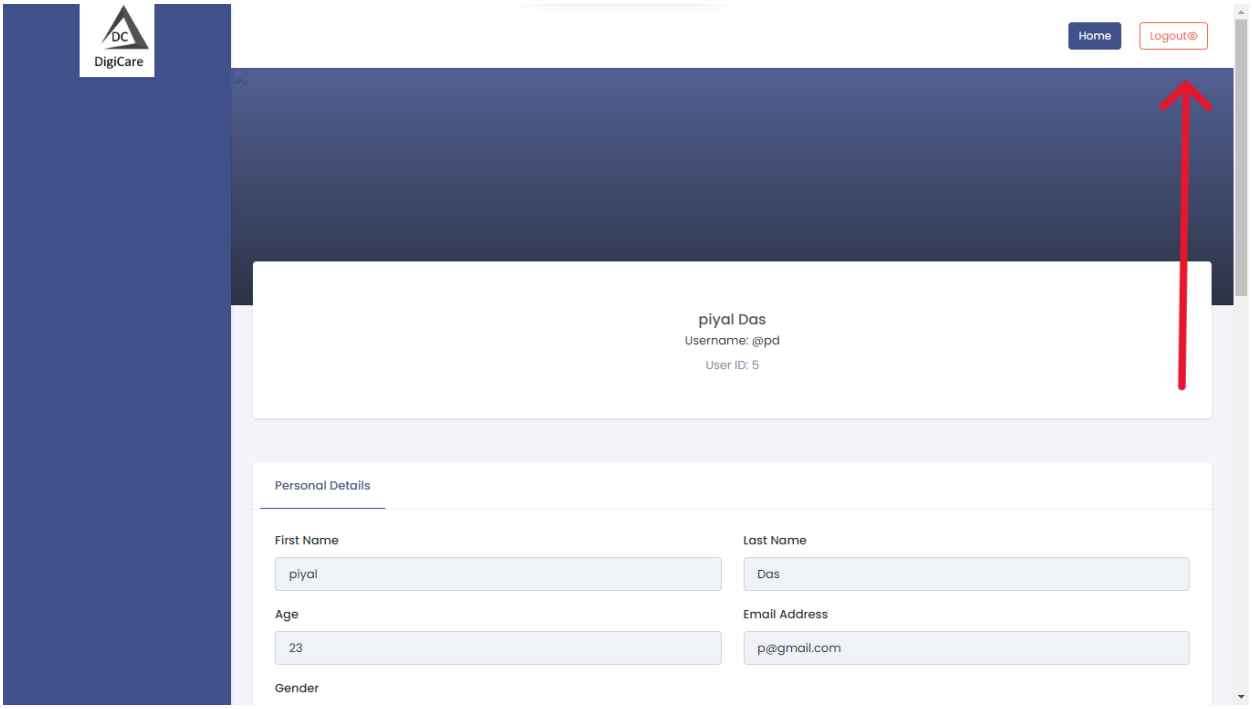


Figure 11: Log out from profile page.

Set Goals:

Within the top header menu bar of DigiCare, a key feature is highlighted—Healthcare Goals. Clicking the "Healthcare Goals" button reveals a dropdown menu with two options: "Set a Goal" and "See All Goal Status." To establish a personal goal, users should choose "Set a Goal," leading them to the "Add Goal" page. Here, users can input their goal details, including the "Goal Name," "Goal Parameter," and "Goal Status." In "Goal Name," users describe their objective, while in "Goal Parameter," they specify the details of the goal. The "Goal Status" section allows users to indicate whether the goal is completed, in progress, or unsuccessful. The

fun fact is their user can not set up a health-related goal but also their daily life goals also.

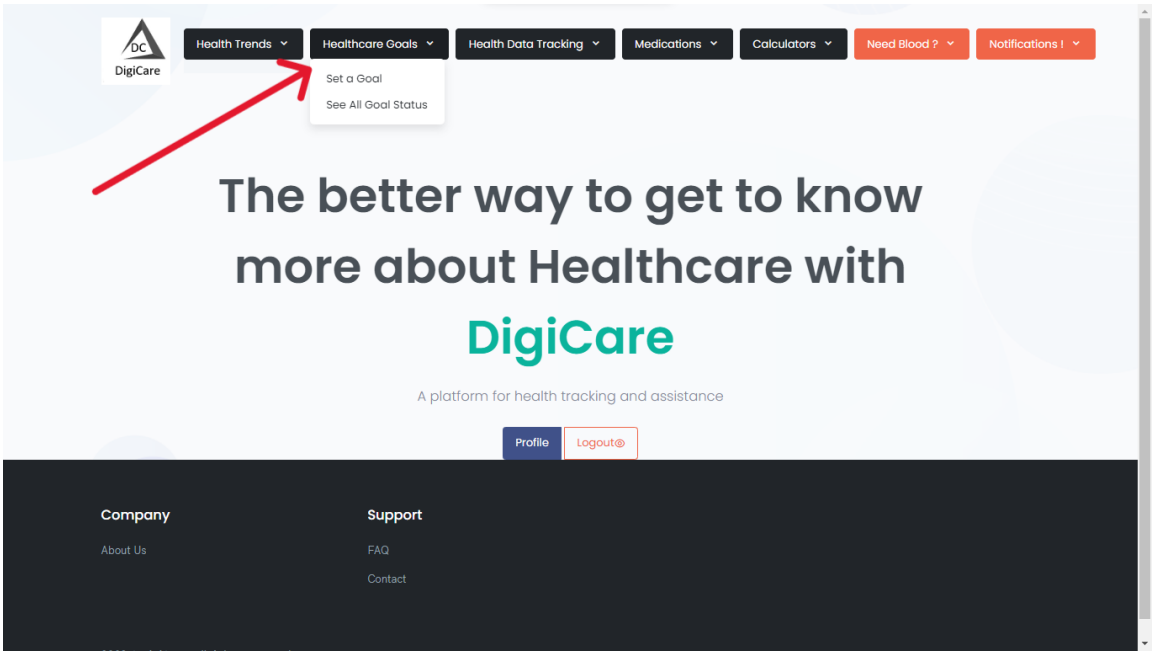


Figure 12: Health Goals in navigation bar

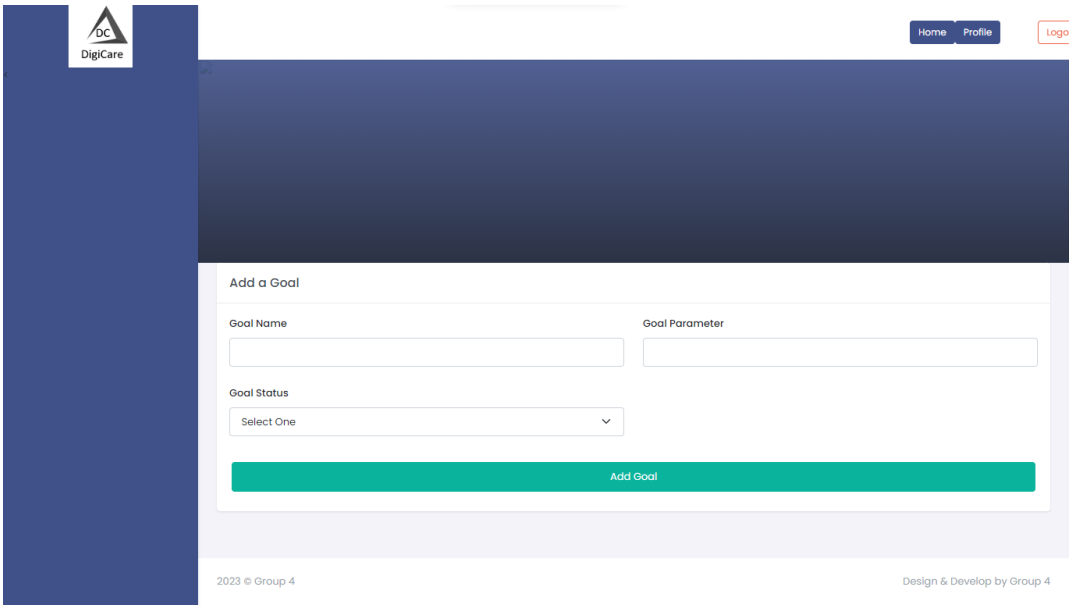


Figure 13: Add a Goal page

See All Goal Status:

Within DigiCare, users who have added goals can assess their progress by choosing "See All Goal Status" from the Healthcare Goals dropdown menu. This action directs them to the View Goals page, where they can review the status of their goals, including the number completed, in progress, or unsuccessful. Essentially, this page provides a historical overview of the user's goal achievements and outcomes. Moreover, the user can also select if the running goal is finished or failed to do that.

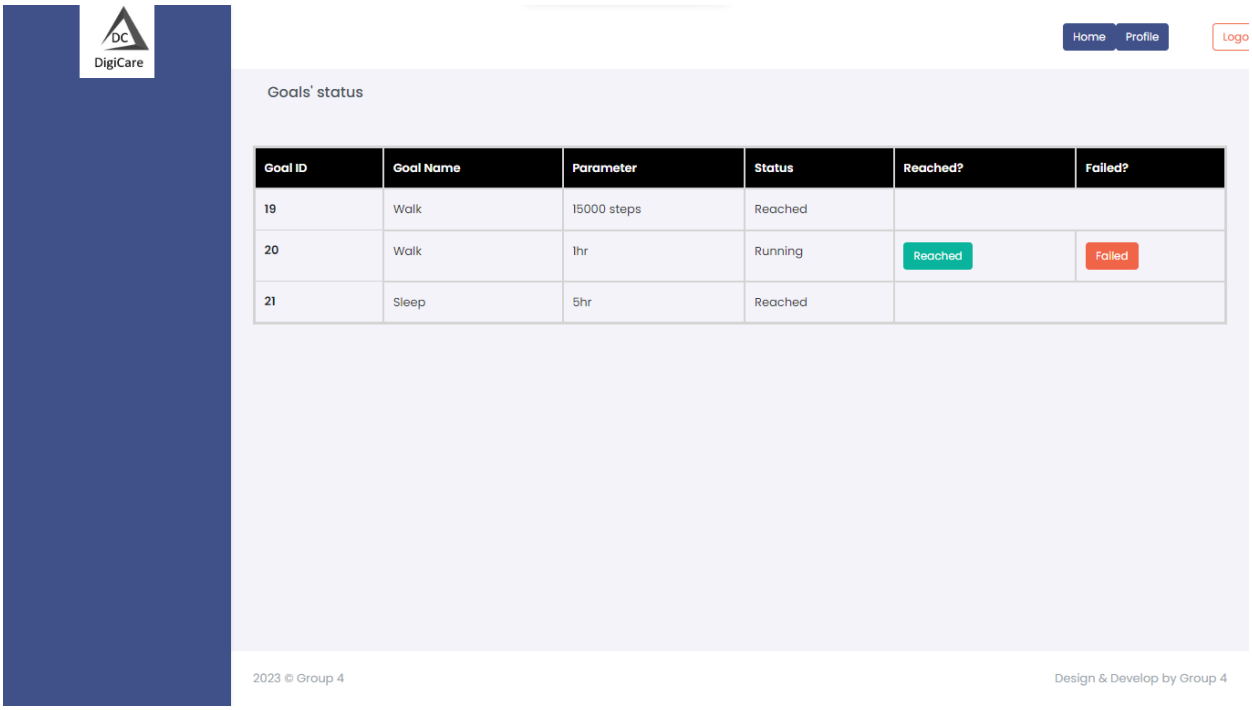


Figure 14: View Goal Page

Health Trends / Statical Overview

One of the most captivating features of DigiCare lies in its ability to motivate users and instill a sense of uniqueness and competition. Within the navigation bar, there is a dedicated button for "Health Trends." This section features two pie charts—one illustrating the user's own goal status and the other depicting the aggregated goal status of all users. The charts showcase the percentage of goals achieved, failed, and currently in progress for both the individual user and the broader user community. It provides statistical representation, fostering motivation and a

competitive mindset for upcoming goals. Additional charts and features may be introduced in this section in the future.

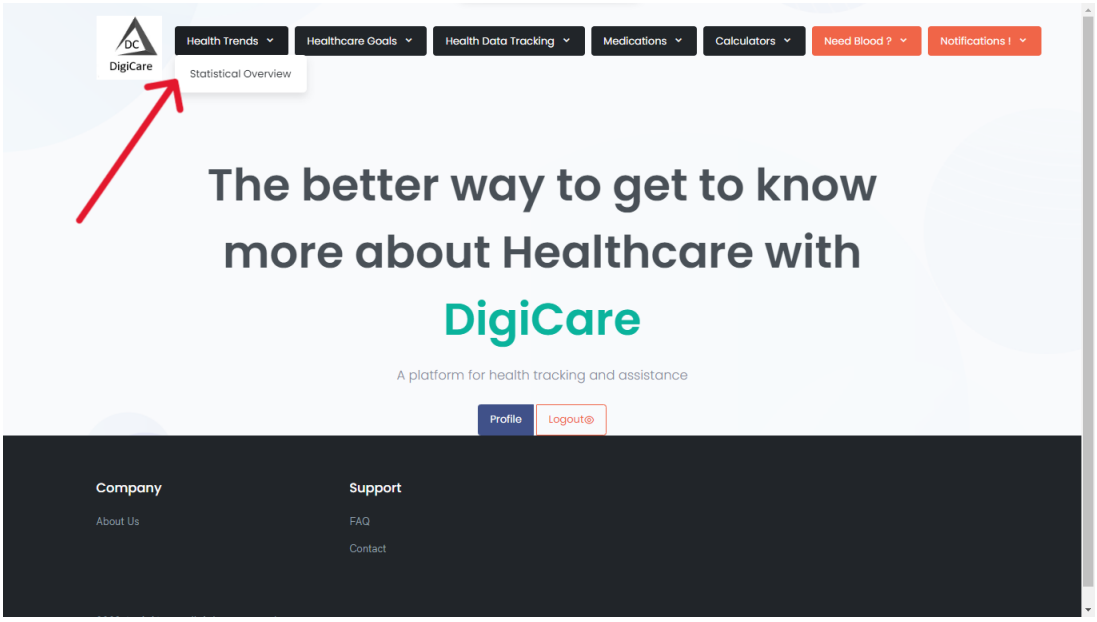


Figure 15: Statical Overview Button on Health Trends

Figure 16: Goal Statistic page

Health Data Tracking / Add Health Data

DigiCare incorporates a health data tracking feature, allowing users to manually input their current or past health statuses and monitor them effortlessly. Additionally, users can record workout or exercise sessions, and this data is stored in the database for future reference. To add health data or exercise information, users should choose "Add Health Data" from the "Health Data Tracking" dropdown menu. This action directs them to the "Add Health Data" page, where they can input relevant details. Future plans include the potential integration of health devices for automatic data input.

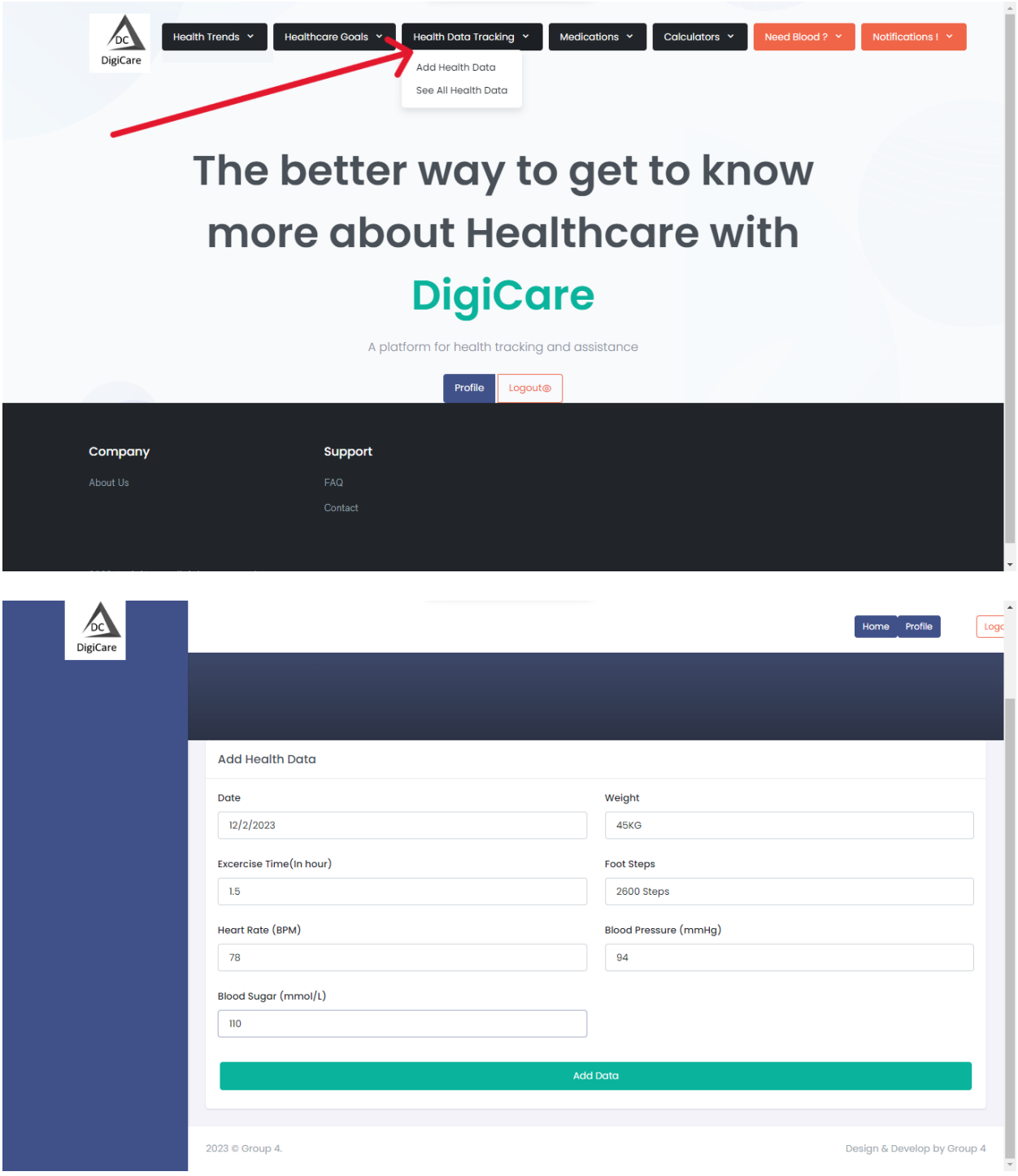


Figure 17: Add Health Data page

See All health data

Once a user has inputted health data, they can review it by selecting "See All Health Data" from the "Health Data Tracking" dropdown menu. This option provides a comprehensive history of the user's recorded health data inputs.

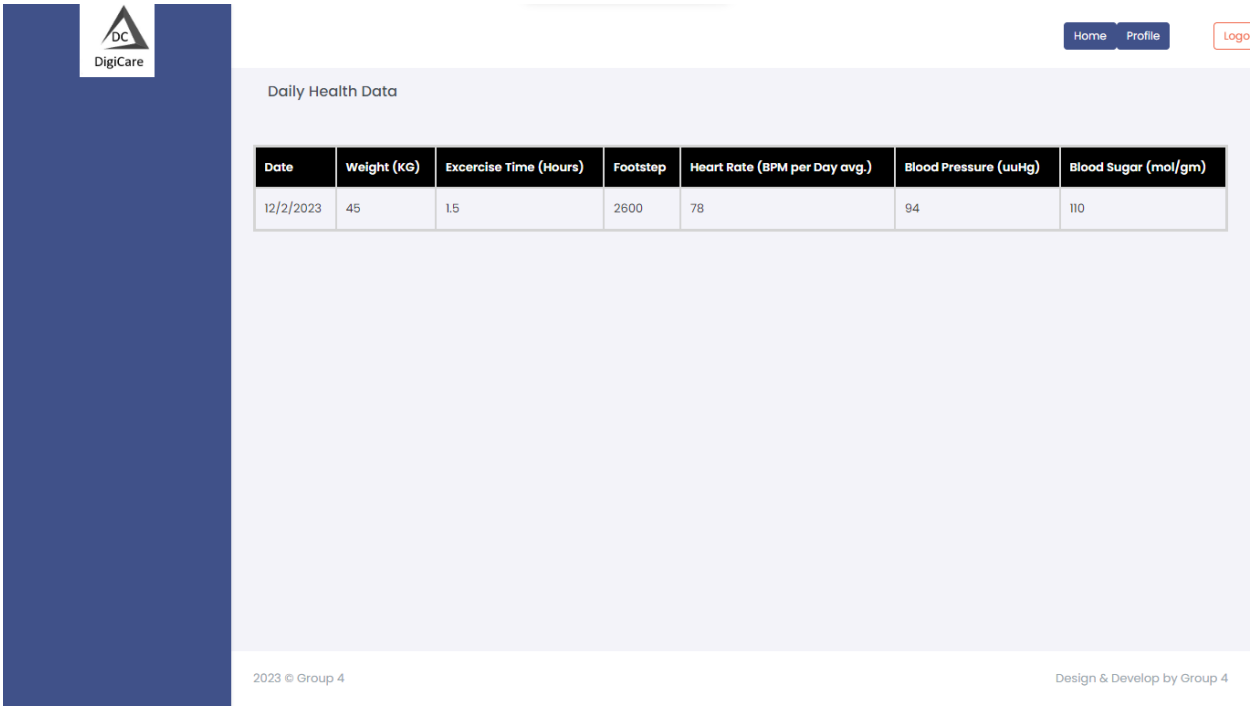
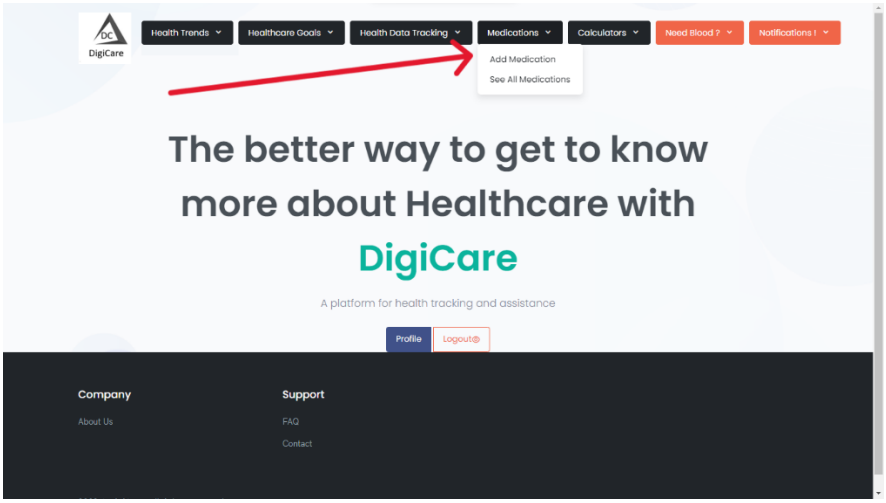


Figure 18: View Health Data Page

Medications / Add Medication

Within the DigiCare System, users can input their daily medication details to facilitate timely reminders for medication intake. To achieve this, users should choose "Add Medication" from the "Medications" dropdown menu. Here, users can provide all necessary information and set up reminders for their medications.



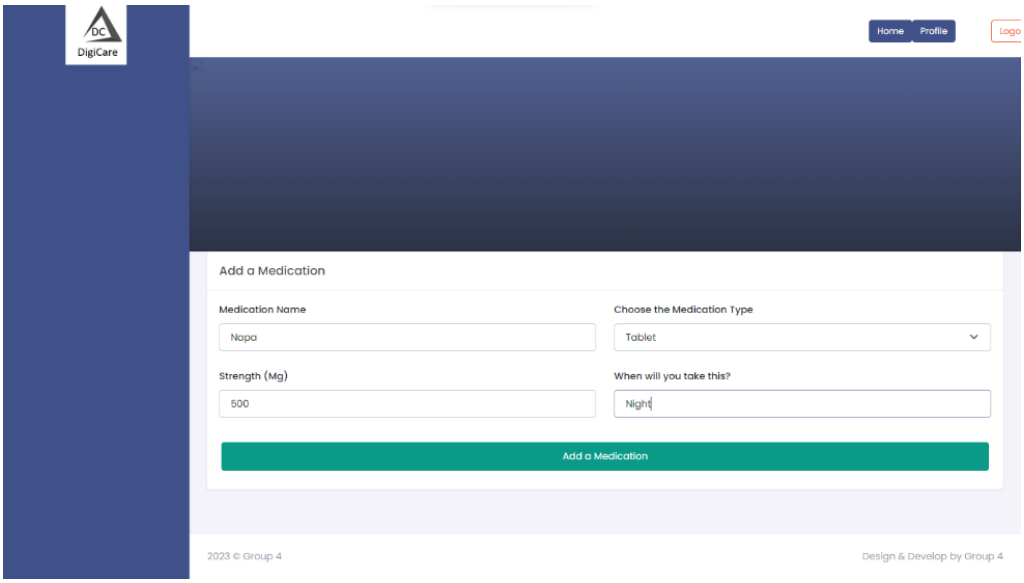


Figure 19: Add Medication Page

See all medications.

After a user has entered their medication details, they can access a comprehensive history of their recorded medications by selecting "See All Medications" from the "Medications" dropdown menu. This option allows users to review a detailed log of their medication inputs over time. There user can select whether their medication course is completed or not also.

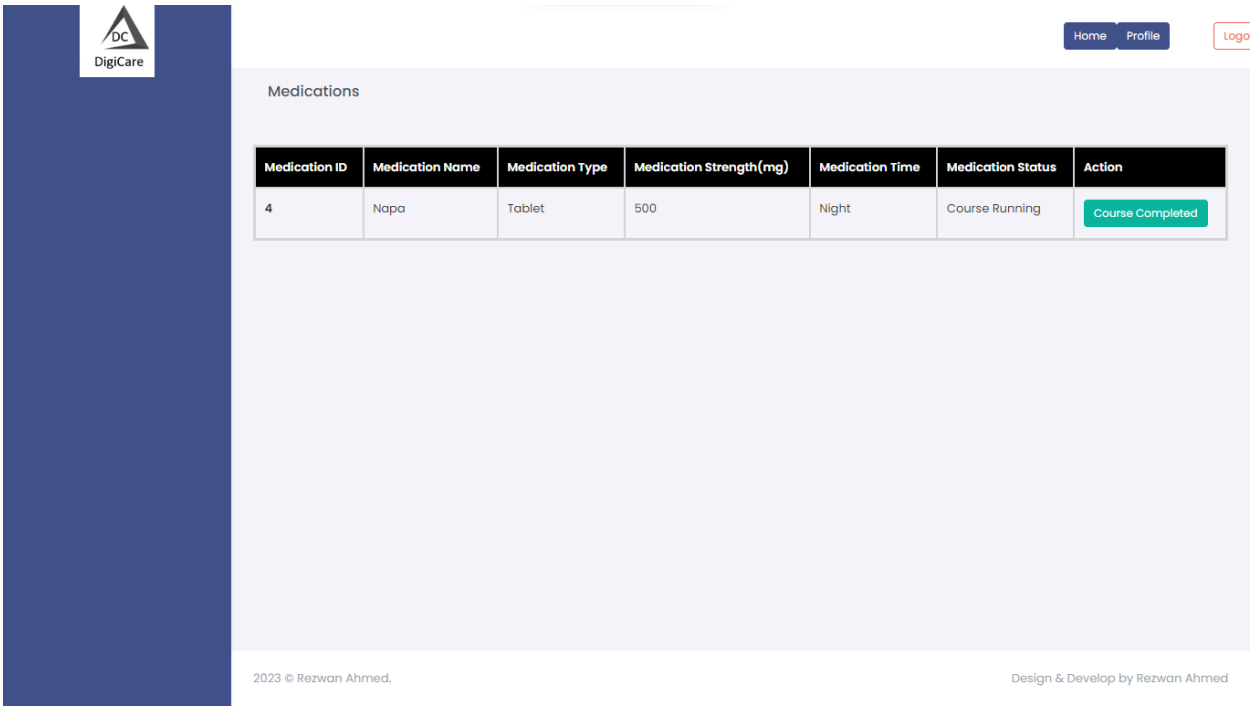
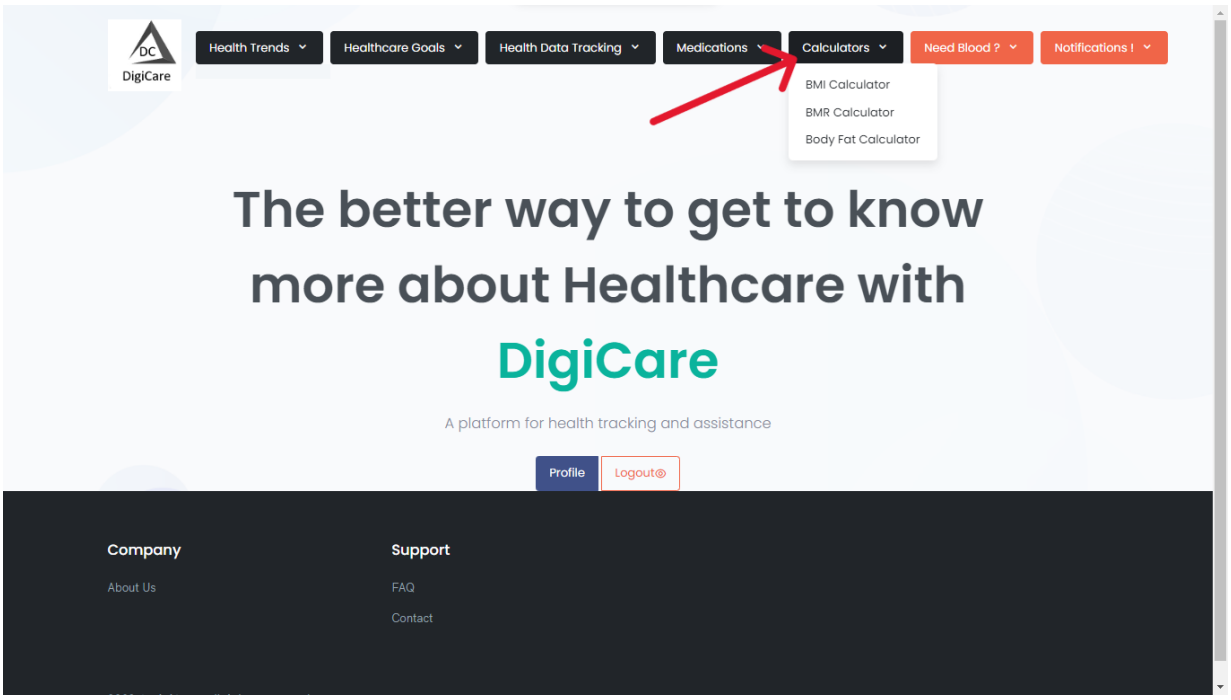


Figure 20: View Medication page

Calculators / BMI Calculators



Within the "Calculator" dropdown menu of DigiCare, users can utilize the BMI calculator. This tool not only computes the Body Mass Index (BMI) based on user inputs but also presents a comprehensive chart indicating health conditions associated with different BMI ranges,

including classifications such as good, bad, etc. The BMI calculator assists users in assessing their health status and provides valuable information on their body composition.

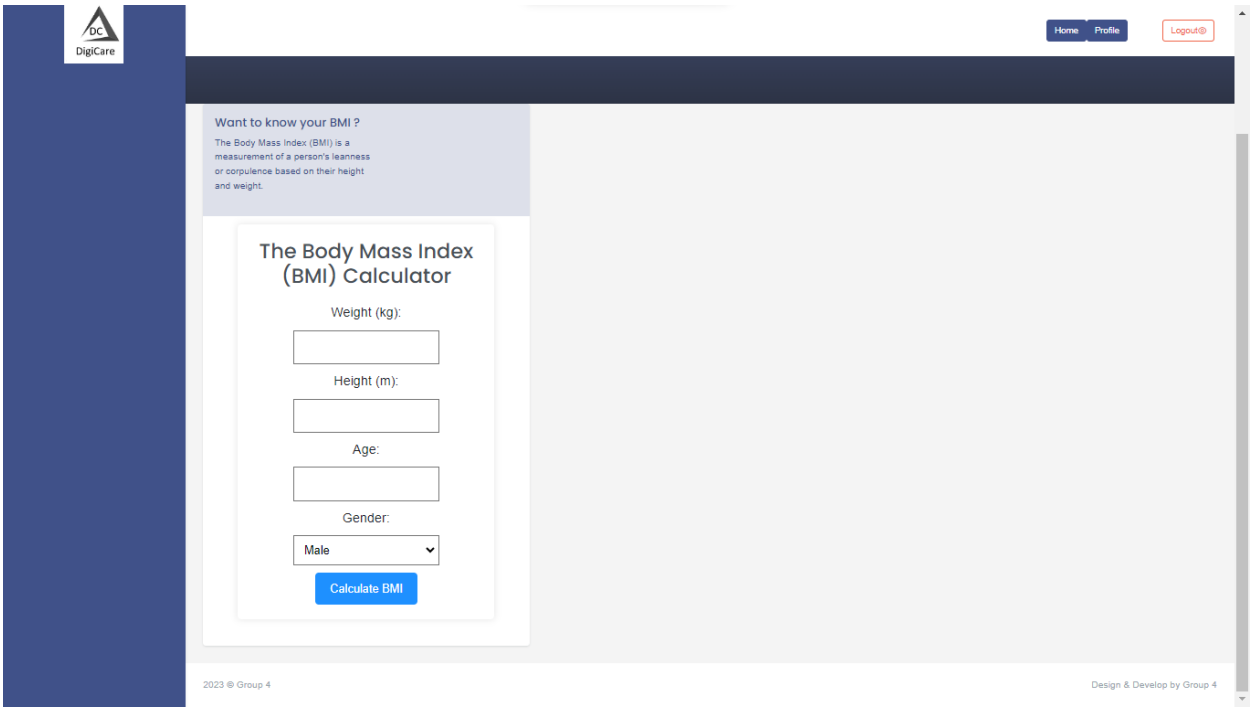


Figure 21: BMI Calculator

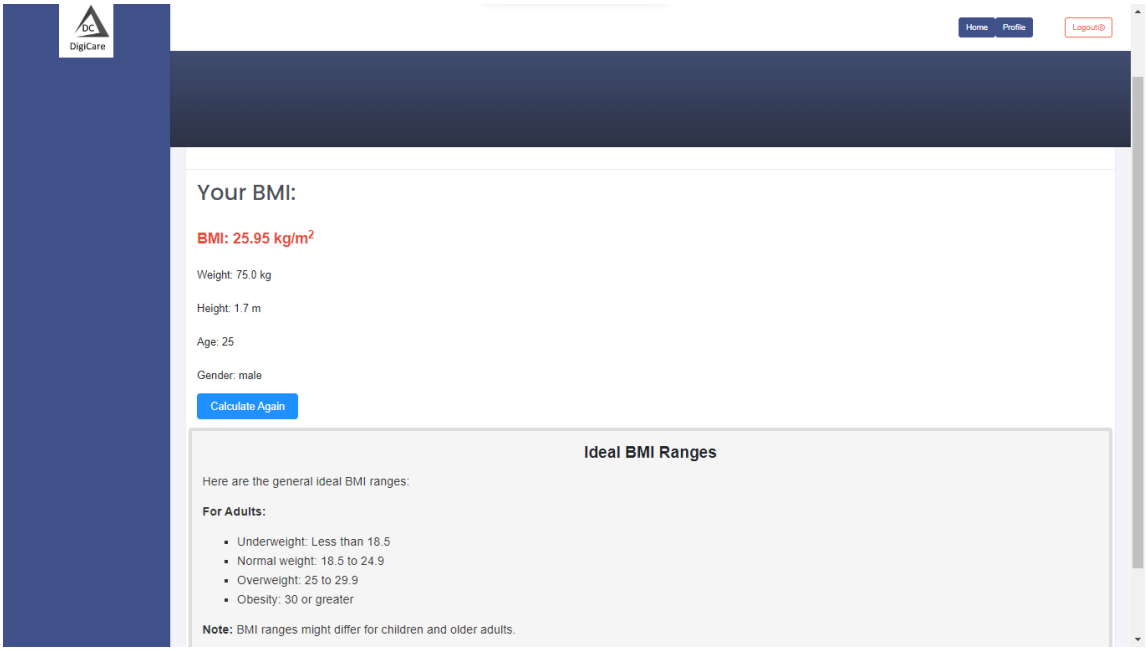
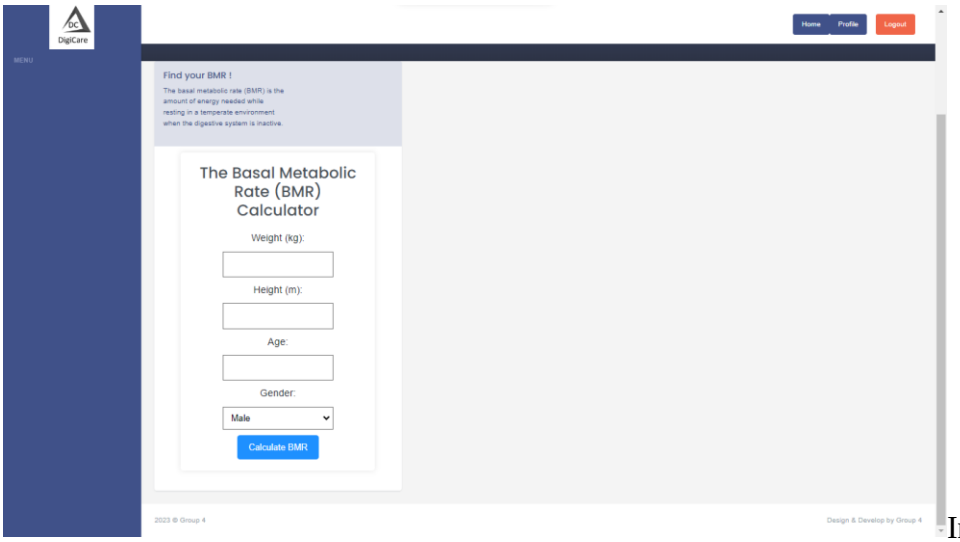


Figure 22: After Calculations of BMI showing the results

Calculator / BMR Calculator



In the "Calculator" dropdown menu of DigiCare, users can access the Basal Metabolic Rate (BMR) calculator. This tool calculates the estimated number of calories a person's body requires at rest. It's important to note that the data generated by the BMR calculator is not stored in the database; it serves as an on-the-fly calculation to provide users with real-time insights into their basal metabolic rate.

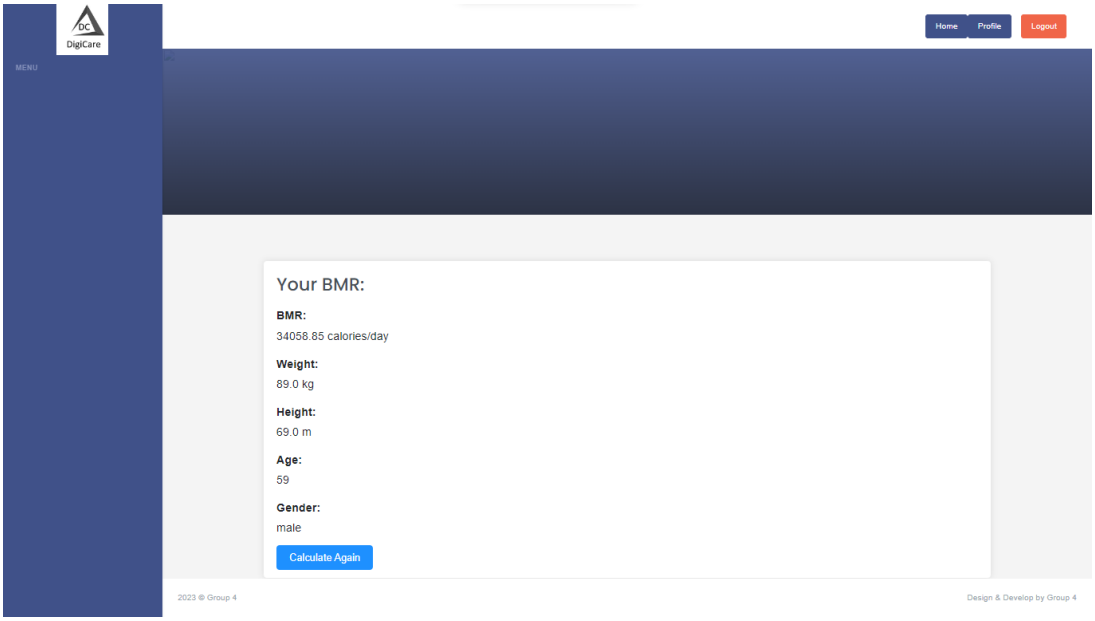


Figure 23: BMR Calculation

Calculator / Body Fat Calculator

DigiCare offers a Body Fat Calculator within its "Calculator" dropdown menu. This feature enables users to estimate their body fat percentage based on input data. The calculator utilizes various factors to provide a comprehensive analysis of body composition. Notably, it's important to mention that the data generated by the Body Fat Calculator is not stored in the database, ensuring privacy. This tool is designed to offer users immediate insights into their body fat

composition, aiding them in their health and fitness assessments.

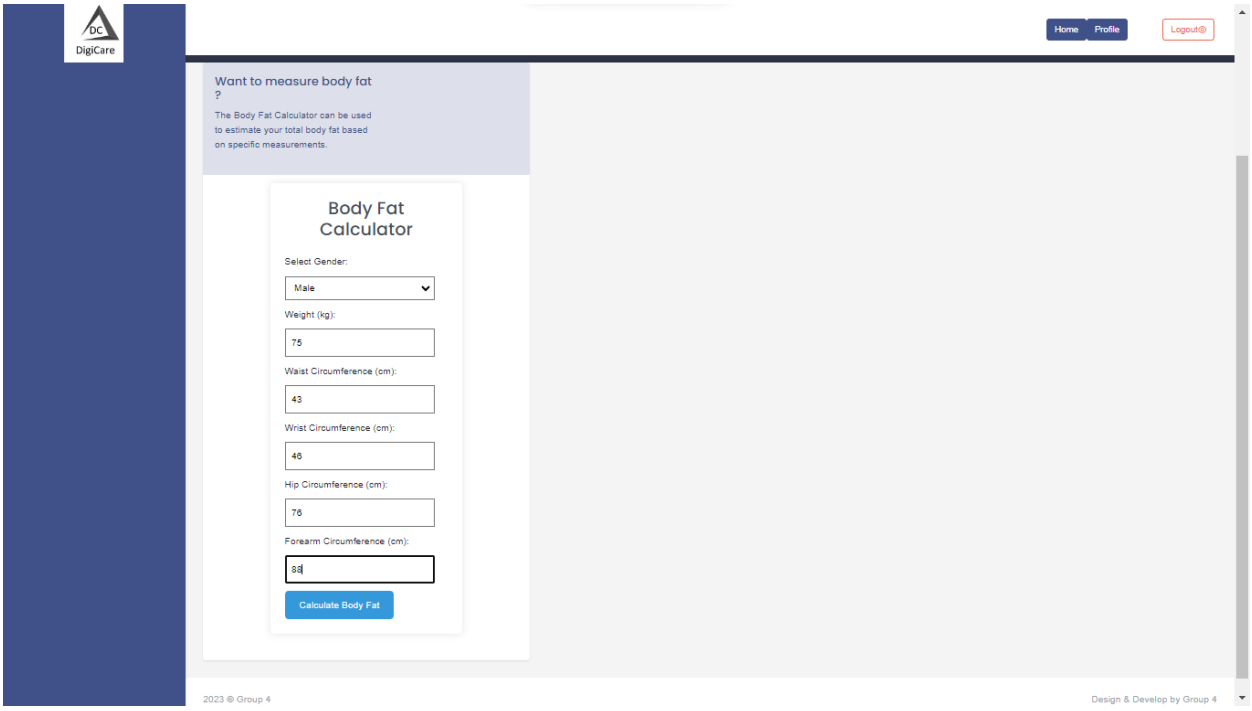


Figure 24: Body Fat Calculation

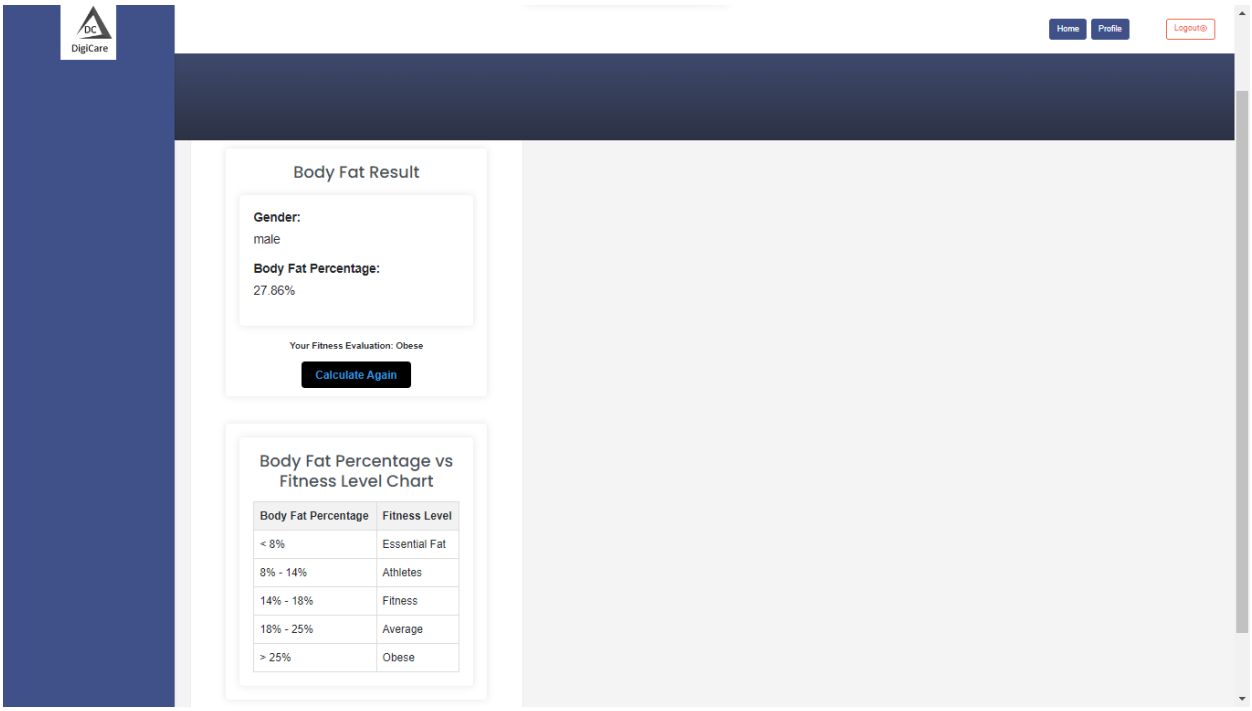
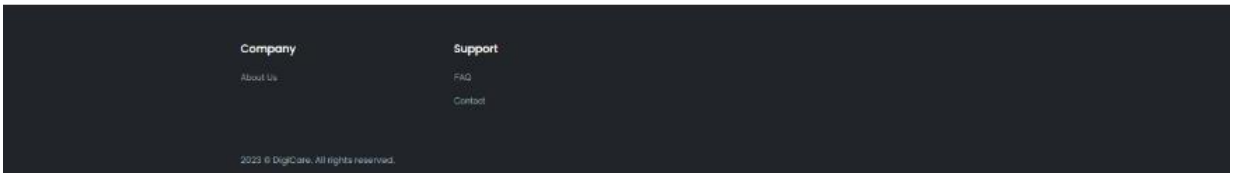


Figure 25Body Fat Calculation Result

Need Blood / Blood Donation / Blood Bank



DigiCare prioritizes the well-being of its users through the Blood Bank feature, fostering a sense of community and compassion. Blood donation is a noble act that contributes to the welfare of society, saving lives in critical situations. In the Blood Bank section, users in need of blood can access a dedicated button guiding them to potential donors within the DigiCare community. This unique system ensures privacy by facilitating communication between users without disclosing personal details. Users seeking blood can send direct messages to potential donors, and if the donor accepts the request, they can proceed to coordinate and support each other. DigiCare's Blood Bank serves as a platform not only for urgent blood needs but also as a testament to the

spirit of altruism, where users can come together to make a meaningful impact on each other's lives. So, to make it simplify for a successful log in, these are the steps to follow:

1. Navigate to the Blood Bank section through the dedicated button in DigiCare.
2. In this section, users in need of blood can review profiles or user number of potential donors from the DigiCare community.
3. Users can send direct messages to potential donors expressing their need for blood. This communication system ensures privacy, as no personal details are shared at this stage.
4. If a potential donor accepts the user's request, they can proceed to coordinate the blood donation without revealing additional personal information.
5. Once the donor accepts the request, users can coordinate the logistics of the blood donation, fostering a sense of community and support within the DigiCare platform.

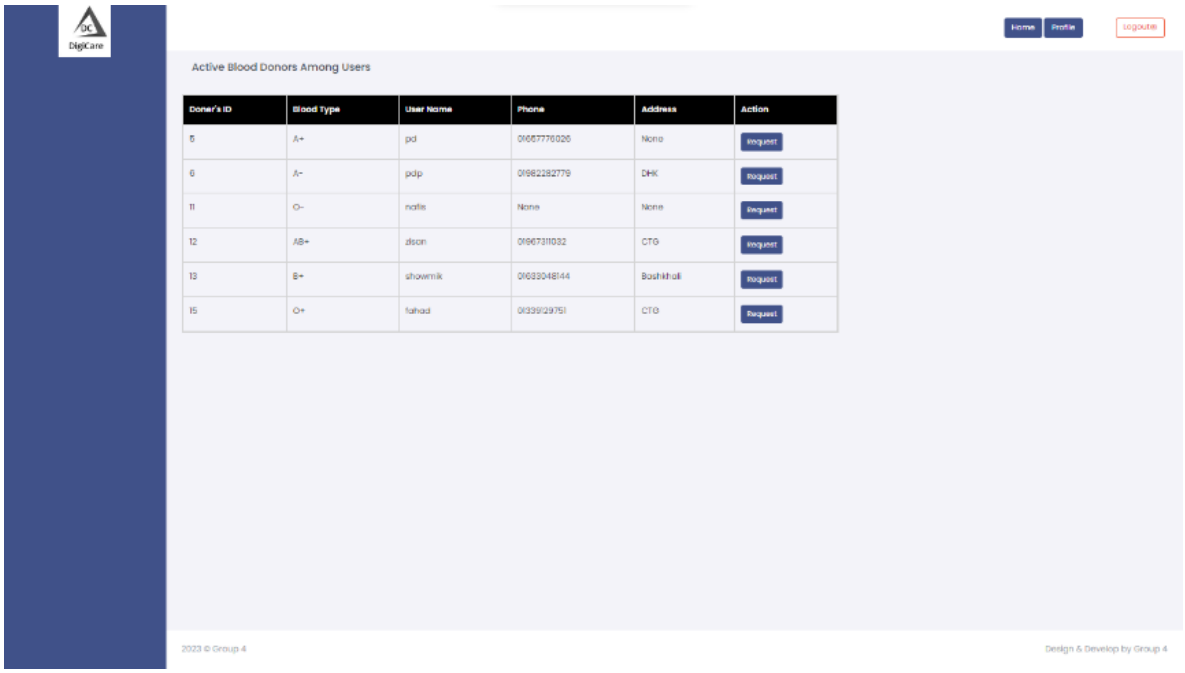


Figure 26: Active Blood Donar's page

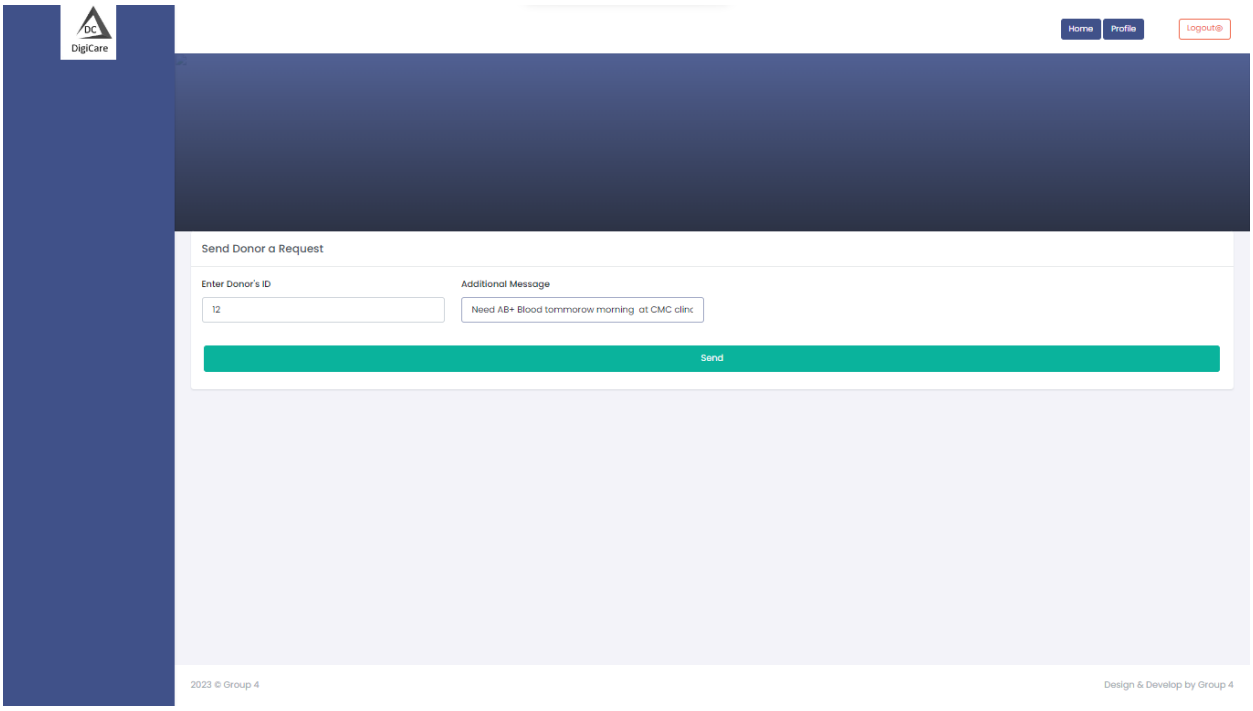


Figure 27: Requesting Donar for blood.

Notifications

DigiCare's notification system ensures seamless communication between users in need of blood and potential donors. Here's a detailed breakdown:

1. When a user sends a blood request, the recipient's notification section displays the request as pending, providing real-time updates on the status of their request.
2. Simultaneously, potential donors receive a notification detailing the blood request, including essential information about the user in need. This allows donors to make informed decisions about whether they want to contribute.
3. In the donor's notification, they have the option to accept or reject the blood donation proposal. If they choose to accept, the notification guides them on how to connect with the requester while maintaining privacy. If they decline, the system automatically communicates this decision to the user in need.
4. The recipient is promptly notified of the donor's decision, providing transparency and enabling quick and informed action based on the donor's choice.
5. DigiCare is committed to enhancing its notification system to include additional features and functionalities, ensuring users stay informed about various aspects of their health and well-being. Future updates may introduce notifications for health data tracking, goal achievements, and other pertinent information, contributing to a comprehensive and user-

friendly experience.

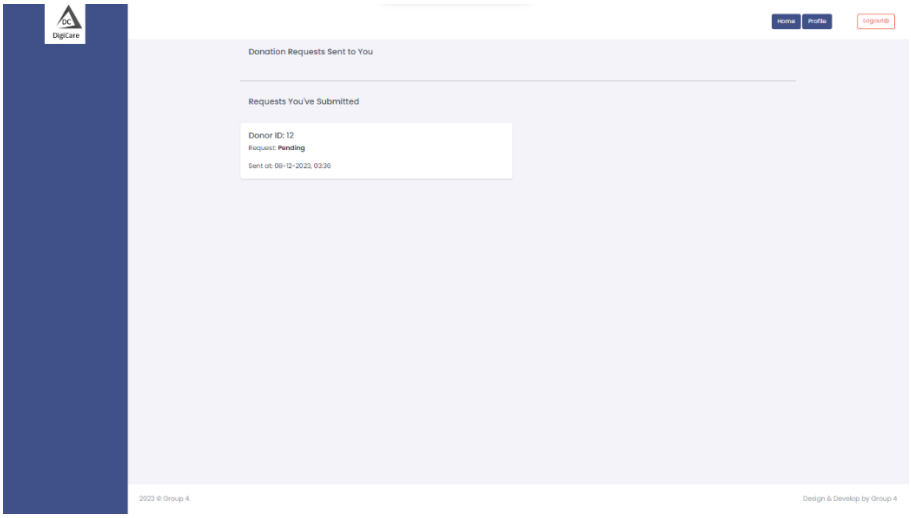


Figure 28:Receiver side notification pane- Receiver after sending request to the Blood Donar

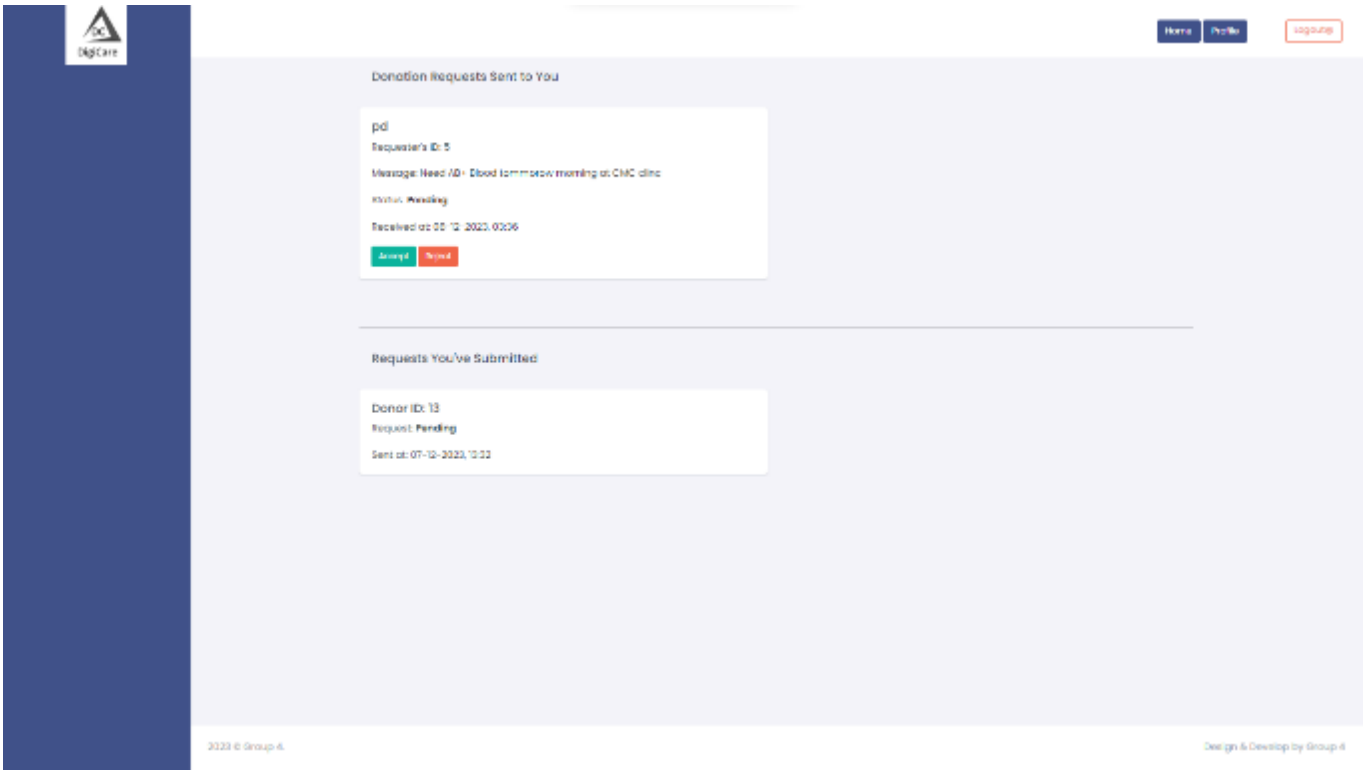


Figure 29: Donar side notification pane- after receiving the request from the needy

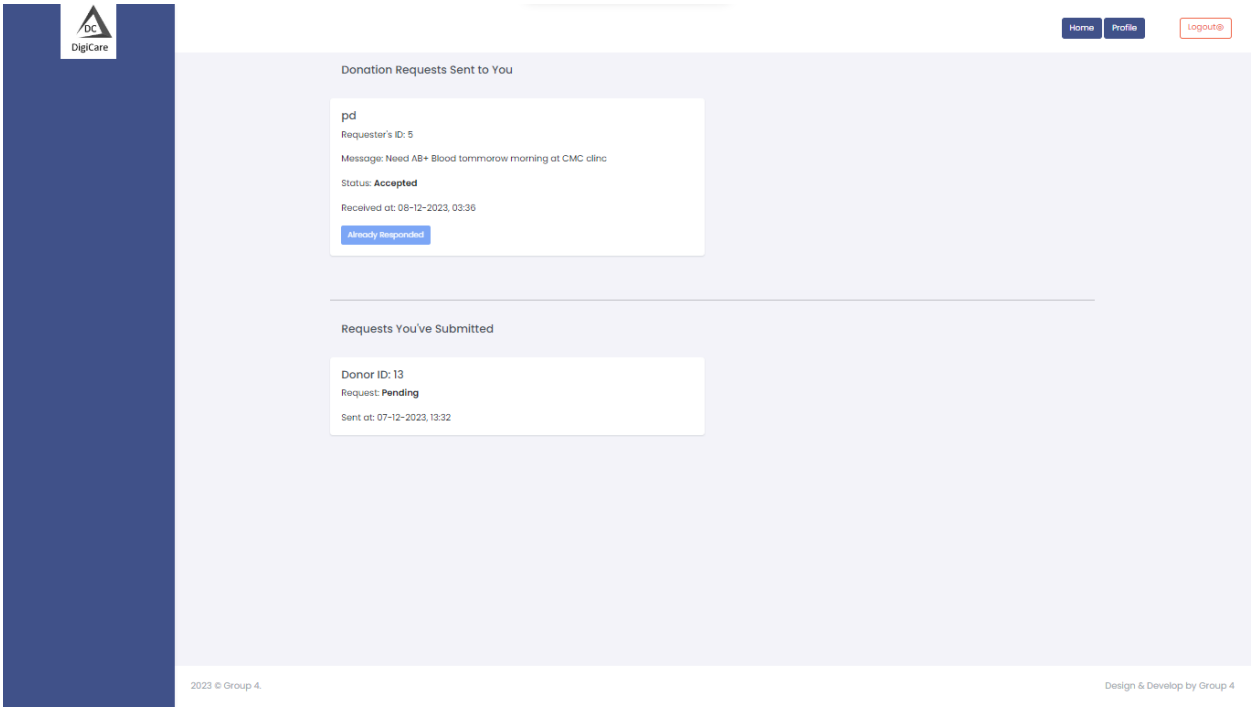


Figure 30: Donar Accepted the request.

Ban / Unban from Admin Pannel

The administration panel of this project is meticulously crafted using phpMyAdmin, providing administrators with a comprehensive toolset for system oversight. The admin dashboard, a focal point of control, offers a unique and powerful feature—the capability to ban and unban users. This feature serves as a crucial security measure, allowing administrators to regulate user access to the platform. Banning a user restricts their ability to log in, offering an effective means to manage and address any issues that may arise. The manual creation of the admin panel through phpMyAdmin ensures a tailored and secure environment, empowering administrators to maintain the integrity and security of the health management system. The ban/unban feature enhances the overall governance of user access, contributing to the robustness and reliability of the platform.

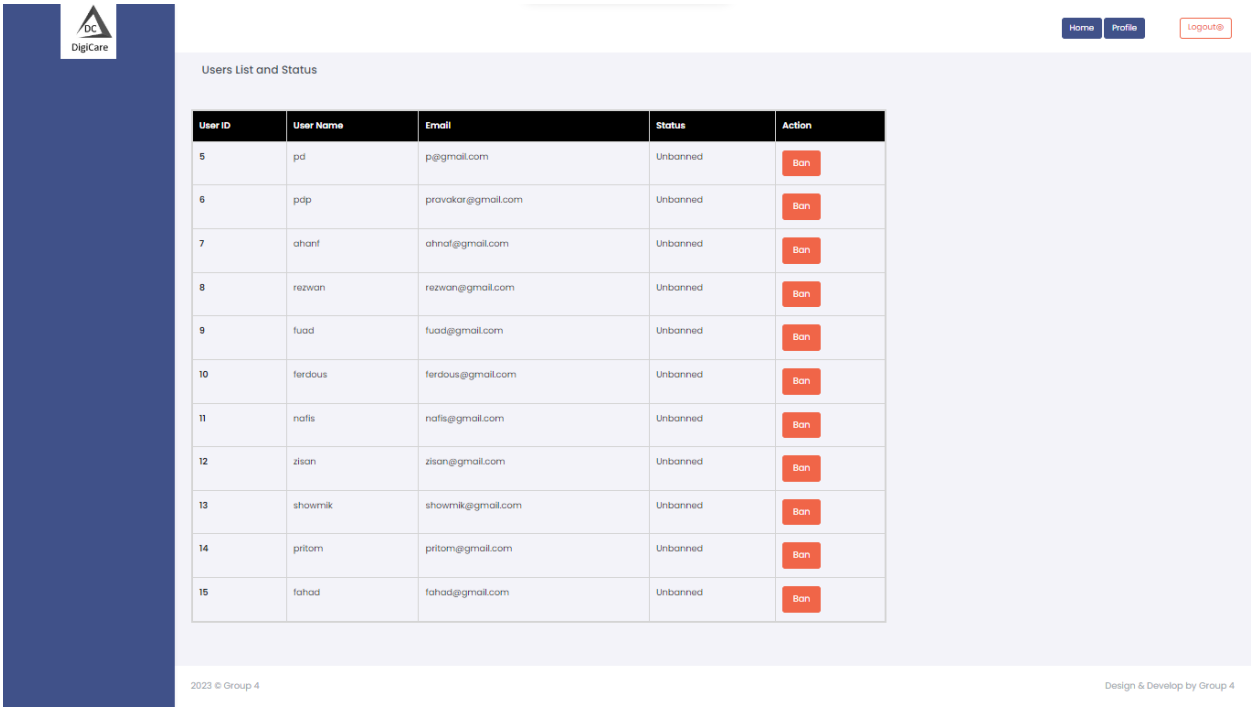
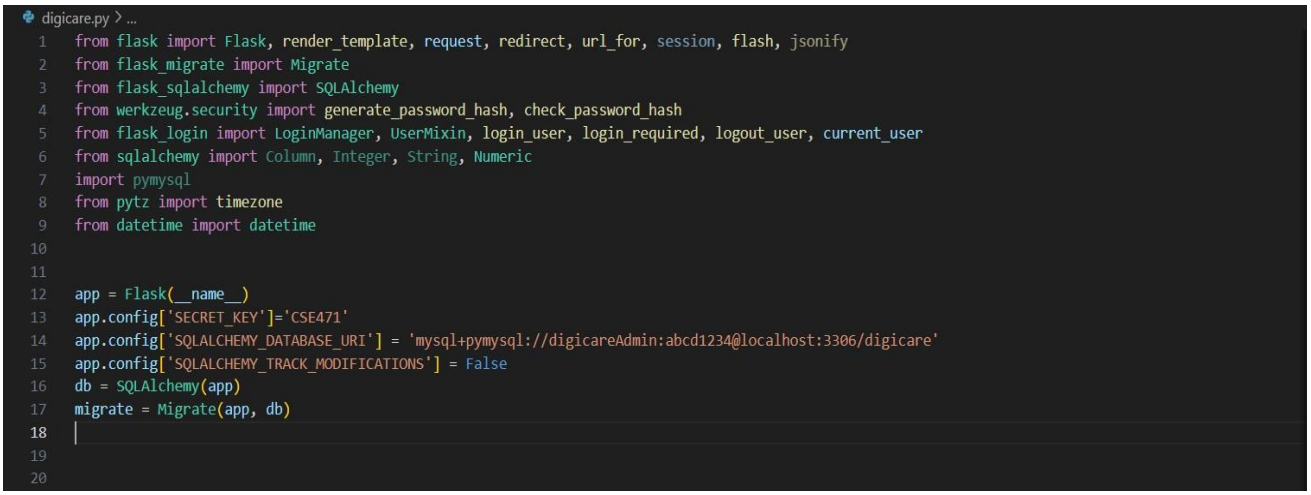


Figure 31: User list and status from the admin panel

Backend Development

The backend of our project is developed using Flask, a lightweight web framework for Python. It handles data, manages user authentication, and serves as the core logic for our application. It is utilized for routing, handling HTTP requests, and managing our application's logic. We use SQLAlchemy to interact with a MySQL database, allowing us to define models and perform database operations seamlessly.



```
digicare.py > ...
1  from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
2  from flask_migrate import Migrate
3  from flask_sqlalchemy import SQLAlchemy
4  from werkzeug.security import generate_password_hash, check_password_hash
5  from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
6  from sqlalchemy import Column, Integer, String, Numeric
7  import pymysql
8  from pytz import timezone
9  from datetime import datetime
10
11
12 app = Flask(__name__)
13 app.config['SECRET_KEY']='CSE471'
14 app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://digicareAdmin:abcd1234@localhost:3306/digicare'
15 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
16 db = SQLAlchemy(app)
17 migrate = Migrate(app, db)
18
19
20
```

Flask-Login is employed for user authentication. Users can register, log in, and log out of the system. Additionally, the system includes a setup process to gather essential user information after registration. Also in case the password for logging in is forgotten by the user there is a forget password and reset password option. The application includes various routes and views for all the features mentioned above. There is also a goal model which is

used to track user goals, a med model which stores information about medications users are taking and a HealthData model captures health-related information such as weight.

Register

```
136 @app.route('/')
137 def index():
138     return render_template('index.html')
139 
140 @app.route("/register", methods=['GET', 'POST'])
141 def register():
142     if request.method == 'POST':
143         email = request.form.get('useremail')
144         username = request.form.get('username')
145         password = request.form.get('userpassword')
146         seq_q = request.form.get('seq_q')
147         seq_a = request.form.get('seq_a')
148         #aType = request.form.get('aType')
149         #print(aType)
150         user = User.query.filter_by(email=email).first()
151         if user:
152             flash('Email already exists.', category='error')
153         else:
154             new_user = User(email=email, username=username, password=generate_password_hash(password, method='sha256'), seq_q=seq_q, seq_a=seq_a, aT
155             db.session.add(new_user)
156             db.session.commit()
157             return redirect(url_for('index'))
158 
159     return render_template("signup.html", user=current_user)
160 
```

Login, Logout

```
120 
121 login_manager = LoginManager()
122 login_manager.login_view = 'login'
123 login_manager.init_app(app)
124 
125 @login_manager.user_loader
126 def load_user(id):
127     return User.query.get(int(id))
128 
```

```python
@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('useremail')
        password = request.form.get('userpassword')
        user = User.query.filter_by(email=email).first()
        if user:
            if check_password_hash(user.password, password):
                if user.flag == "Banned":
                    return redirect(url_for('AccountBanned'))
                flash('Logged in successfully.', category='success')
                login_user(user, remember=True)
                if user.Fname==None or user.Lname==None or user.age==None or user.gender==None or user.height==None or user.weight==None or user.blo
                    return redirect(url_for('setup'))

                return redirect(url_for('index'))

            else:
                flash('Incorrect password, try again.', category='error')
        else:
            flash('Email does not exist.', category='error')
    return render_template("login.html", user=current_user)
```

```python
@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))
```

Setup Profile

```python
@app.route("/setup", methods=['GET', 'POST'])
@login_required
def setup():
    user=current_user
    if request.method == 'POST':
        user.Fname = request.form.get('Fname')
        user.Lname = request.form.get('Lname')
        user.age = request.form.get('age')
        user.gender = request.form.get('gender')
        user.height = request.form.get('height')
        user.weight = request.form.get('weight')
        user.bloodType = request.form.get('bloodType')
        user.bloodDonate = request.form.get('bloodDonate')
        user.phone = request.form.get('phone')
        user.address = request.form.get('address')
        user.allergies = request.form.get('allergies')
        user.medications = request.form.get('medications')
        user.conditions = request.form.get('conditions')
        user.surgeries = request.form.get('surgeries')
        user.familyHistory = request.form.get('familyHistory')
        db.session.commit()
        return redirect(url_for('index'))
    return render_template("setup.html", user=current_user)
```

Forget Password, Reset Password

```python

@app.route("/forgotpassword", methods=['GET', 'POST'])
def forgotpassword():
    if request.method == 'POST':
        email = request.form.get('useremail')
        seq_q = request.form.get('seq_q')
        seq_a = request.form.get('seq_a')
        user = User.query.filter_by(email=email).first()
        if user:
            if user.seq_q == seq_q and user.seq_a == seq_a:
                return render_template("resetpassword.html", email=user.email)
            else:
                return render_template("index.html", user=current_user)
        else:
            return render_template("index.html", user=current_user)
    return render_template("forgotpassword.html", user=current_user)

```

```python

@app.route("/resetpassword", methods=['GET', 'POST'])
def resetpassword():
    if request.method == 'POST':
        email = request.form.get('useremail')
        password = request.form.get('userpassword')
        user = User.query.filter_by(email=email).first()
        print(email)
        print(password)
        if user:
            user.password = generate_password_hash(password, method='sha256')
            db.session.commit()
            return redirect(url_for('login'))
        else:
            return render_template("index.html", user=current_user)
    return render_template("resetpassword.html", user=current_user)

```

Goal Model

```
267  @app.route("/AddGoal", methods=['GET', 'POST'])
268  @login_required
269  def AddGoal():
270      if request.method == 'POST':
271          name = request.form.get('gname')
272          description = request.form.get('gdesc')
273          status = request.form.get('gs')
274          user_id = current_user.id
275          new_goal = Goal(name=name, description=description, status=status, user=user_id)
276          db.session.add(new_goal)
277          db.session.commit()
278          return redirect(url_for('ViewGoals'))
279      return render_template("AddGoal.html", user=current_user)
280
281
282
283  @app.route("/ViewGoals", methods=['GET', 'POST'])
284  @login_required
285  def ViewGoals():
286      goals = Goal.query.filter_by(user=current_user.id).all()
287      return render_template("ViewGoals.html", user=current_user, goals=goals)
```

```
290  @app.route("/GoalReached/<int:id>", methods=['GET', 'POST'])
291  @login_required
292  def GoalReached(id):
293      goal = Goal.query.get_or_404(id)
294
295
296      goal.status = "Reached"
297      db.session.commit()
298      return redirect(url_for('ViewGoals'))
299
300
301
302  @app.route("/GoalFailed/<int:id>", methods=['GET', 'POST'])
303  @login_required
304  def GoalFailed(id):
305      goal = Goal.query.get_or_404(id)
306      goal.status = "Failed"
307      db.session.commit()
308      return redirect(url_for('ViewGoals'))
```

Med Model

```python
@app.route("/AddMed", methods=['GET', 'POST'])
@login_required
def AddMed():
    if request.method == 'POST':
        name= request.form.get('mname')
        type = request.form.get('mtype')
        strength = request.form.get('mstrength')
        time = request.form.get('mtime')
        user_id = current_user.id
        new_med = Med(name=name, type=type, strength=strength, time=time, user=user_id)
        db.session.add(new_med)
        db.session.commit()
        return redirect(url_for('ViewMeds'))
    return render_template("AddMed.html", user=current_user)



@app.route("/ViewMeds", methods=['GET', 'POST'])
@login_required
def ViewMeds():
    meds = Med.query.filter_by(user=current_user.id).all()
    return render_template("ViewMeds.html", user=current_user, meds=meds)


@app.route("/MedDone/<int:id>", methods=['GET', 'POST'])
@login_required
def MedDone(id):
    med = Med.query.get_or_404(id)
    med.status = "Course Completed"
    db.session.commit()
    return redirect(url_for('ViewMeds'))
```

HealthData Model

```python
@app.route("/AddHealthData", methods=['GET', 'POST'])
@login_required
def AddHealthData():
    if request.method == 'POST':
        date = request.form.get('date')
        weight = request.form.get('weight')
        etime = request.form.get('etime')
        fstep = request.form.get('fstep')
        heartRate = request.form.get('heartRate')
        bloodPressure = request.form.get('bloodPressure')
        bloodSugar = request.form.get('bloodSugar')
        user_id = current_user.id
        new_healthdata = HealthData(date=date, weight=weight, etime=etime, fstep=fstep, heartRate=heartRate, bloodPressure=bloodPressure, bloodSug
        db.session.add(new_healthdata)
        db.session.commit()
        return redirect(url_for('ViewHealthData'))
    return render_template("AddHealthData.html", user=current_user)


@app.route("/ViewHealthData", methods=['GET', 'POST'])
@login_required
def ViewHealthData():
    healthdata = HealthData.query.filter_by(user=current_user.id).all()

    return render_template("ViewHealthData.html", user=current_user, healthdata=healthdata)
```

There are also database models of User

```python
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    username = db.Column(db.String(100), nullable = False)
    email = db.Column(db.String(100), nullable = False)
    password = db.Column(db.String(100), nullable = False)
    seq_q = db.Column(db.String(100))
    seq_a = db.Column(db.String(100))
    Fname = db.Column(db.String(100))
    Lname = db.Column(db.String(100))
    age = db.Column(db.Integer)
    gender = db.Column(db.String(100))
    height = db.Column(Numeric(5,2))
    weight = db.Column(Numeric(5,2))
    bloodType = db.Column(db.String(100))
    bloodDonate = db.Column(db.String(100))
    phone = db.Column(db.String(100))
    address = db.Column(db.String(100))
    allergies = db.Column(db.String(100))
    medications = db.Column(db.String(100))
    conditions = db.Column(db.String(100))
    surgeries = db.Column(db.String(100))
    familyHistory = db.Column(db.String(100))
    aType = db.Column(db.String(100))
    flag = db.Column(db.String(100))

    def __repr__(self) -> str:
        return f'{self.id} - {self.email} - {self.username} - {self.password}'
```

Goal

```python
class Goal(db.Model):

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(100), nullable = False)
    description = db.Column(db.String(100), nullable = False)
    status = db.Column(db.String(100), nullable = False)
    user = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)


    def __repr__(self) -> str:
        return f'{self.id} - {self.name} - {self.description}'

```

Med

```python
class Med(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(100), nullable = False)
    type = db.Column(db.String(100), nullable = False)
    strength = db.Column(db.String(100), nullable = False)
    time = db.Column(db.String(100), nullable = False)
    user = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    status = db.Column(db.String(100), default="Course Running")


    def __repr__(self) -> str:
        return f'{self.id} - {self.name} - {self.type} - {self.strength} - {self.time}'

```

HealthData

```
class HealthData(db.Model):

        id = db.Column(db.Integer, primary_key=True, autoincrement=True)
        date = db.Column(db.String(100), nullable = False)
        weight = db.Column(db.Integer, nullable = False)
        etime = db.Column(db.String(100), nullable = False)
        fstep = db.Column(db.Integer, nullable = False)
        heartRate = db.Column(db.Integer, nullable = False)
        bloodPressure = db.Column(db.String(100), nullable = False)
        bloodSugar = db.Column(db.Integer, nullable = False)
        user = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)


        def __repr__(self) -> str:
            return f'{self.id} - {self.date} - {self.etime} - {self.heartRate} - {self.bloodPressure} - {self.bloodSugar}'
```

DonationRequest

```
class DonationRequest(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    user_name= db.Column(db.String(100), nullable=False)
    donor_id = db.Column(db.Integer, nullable=False)
    message = db.Column(db.String(100), nullable=False)
    is_seen = db.Column(db.String(100), default=None)
    accepted = db.Column(db.String(100), default=None)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)



    def __repr__(self) -> str:
        return f'{self.id} - {self.donor_id} - {self.donor_id} - {self.message} - {self.is_seen} - {self.accepted}'
```

In the profile setup model if the users are potential blood donors (bloodDonate=`yes`) their info is displayed in the list of potential blood donors.

Blood Donor Routes

```python
@app.route("/BloodDonorList", methods=['GET', 'POST'])
@login_required
def BloodDonorList():
    users = User.query.filter_by(aType="User",flag='Unbanned', bloodDonate='yes' ).all()
    return render_template("BloodDonorList.html", user=current_user, users=users)


@app.route('/request_blood', methods=['GET','POST'])
def request_blood():
    if request.method == 'POST':
        requester_id = current_user.id
        donor_id = request.form.get('donor_id')
        user_name = current_user.username
        message = request.form.get('message')

        # timezone
        bd_tz = timezone('Asia/Dhaka')
        current_time = datetime.now().astimezone(bd_tz)


        new_request = DonationRequest(user_id=requester_id, donor_id=donor_id, user_name=user_name, message=message, accepted="Pending", timestamp
        db.session.add(new_request)
        db.session.commit()
        return redirect(url_for('index'))
    return render_template("request_blood.html", user=current_user)
```

Notifications

The application implements a notification system for blood donation requests. Users can send and receive donation requests, with options to accept or reject them.

```python
@app.route("/BloodDonationNotifications", methods=['GET', 'POST'])
@login_required
def BloodDonationNotifications():
    current_user_id = current_user.id

    # donation requests made by the current user (requester)
    requester_requests = DonationRequest.query.filter_by(user_id=current_user_id).all()

    # donation requests received by the current user (donor)
    donor_requests = DonationRequest.query.filter_by(donor_id=current_user_id).all()

    # additional information of donors based on donor_id (Not Using this anymore)
    donor_ids = [request.donor_id for request in requester_requests]
    donors_info = User.query.filter(User.id.in_(donor_ids)).all()

    return render_template("BloodDonationNotifications.html",
                           user=current_user,
                           requester_requests=requester_requests,
                           donor_requests=donor_requests,
                           donors_info=donors_info)
```

```python
@app.route("/NotificationsAccept/<int:id>", methods=['GET', 'POST'])
@login_required
def NotificationsAccept(id):
    request = DonationRequest.query.get_or_404(id)
    if request.accepted == "Pending":
        request.accepted = "Accepted"
        db.session.commit()
    return redirect(url_for('BloodDonationNotifications'))

@app.route("/NotificationsReject/<int:id>", methods=['GET', 'POST'])
@login_required
def NotificationsReject(id):
    request = DonationRequest.query.get_or_404(id)
    if request.accepted == "Pending":
        request.accepted = "Rejected"
        db.session.commit()
    return redirect(url_for('BloodDonationNotifications'))
```

Statistics

A statistics route provides an overview of goal statistics for all users and personalized statistics for the logged-in user.

```python
# Route to get goal statistics for all users
@app.route('/statistics', methods=['GET', 'POST'])
@login_required
def get_statistics():
    #with app.app_context():
    total_users = Goal.query.distinct(Goal.id).count()

    running_count = Goal.query.filter_by(status='Running').count()
    reached_count = Goal.query.filter_by(status='Reached').count()
    failed_count = Goal.query.filter_by(status='Failed').count()
    running_percentage = round((running_count / total_users) * 100,2) if total_users > 0 else 0
    reached_percentage = round((reached_count / total_users) * 100,2) if total_users > 0 else 0
    failed_percentage = round((failed_count / total_users) * 100,2) if total_users > 0 else 0
    current_user_id = current_user.id
    c_running_count = Goal.query.filter_by(user=current_user_id, status='Running').count()
    c_reached_count = Goal.query.filter_by(user=current_user_id, status='Reached').count()
    c_failed_count = Goal.query.filter_by(user=current_user_id, status='Failed').count()
    c_total_count = c_running_count + c_reached_count + c_failed_count
    c_running_percentage = round((c_running_count / c_total_count) * 100, 2) if c_total_count > 0 else 0
    c_reached_percentage = round((c_reached_count / c_total_count) * 100, 2) if c_total_count > 0 else 0
    c_failed_percentage = round((c_failed_count / c_total_count) * 100, 2) if c_total_count > 0 else 0

    return render_template('statistics.html',
            running_percentage= running_percentage,
            reached_percentage= reached_percentage,
            failed_percentage= failed_percentage,
            c_running_percentage= c_running_percentage,
            c_reached_percentage= c_reached_percentage,
            c_failed_percentage= c_failed_percentage)
```

Calculators

The application features calculators for BMI, BMR, and body fat index, providing users with health-related insights.

BMI

```python
@app.route("/BmiIndex", methods=['GET', 'POST'])
@login_required
def BmiIndex():
    return render_template('BmiIndex.html')


@app.route('/calculate', methods=['POST'])
def calculate():
    if request.method == 'POST':
        weight = float(request.form['weight'])
        height = float(request.form['height'])
        age = int(request.form['age'])
        gender = request.form['gender']

        bmi = calculate_bmi(weight, height)
        return render_template('BmiResult.html', weight=weight, height=height, bmi=bmi, age=age, gender=gender)

def calculate_bmi(weight, height):
    bmi = weight / (height * height)
    return round(bmi, 2)
```

BMR

```python
@app.route("/BmrIndex", methods=['GET', 'POST'])
@login_required
def BmrIndex():
    return render_template('BmrIndex.html')

@app.route('/calculate_bmr', methods=['POST'])
def calculate_bmr():
    if request.method == 'POST':
        weight = float(request.form['weight'])
        height = float(request.form['height'])
        age = int(request.form['age'])
        gender = request.form['gender']

        bmr = calculate_bmr_value(weight, height, age, gender)
        return render_template('BmrResult.html', weight=weight, height=height, age=age, gender=gender, bmr=bmr)

def calculate_bmr_value(weight, height, age, gender):
    if gender == 'male':
        bmr = 88.362 + (13.397 * weight) + (4.799 * height * 100) - (5.677 * age)
    else:  # female
        bmr = 447.593 + (9.247 * weight) + (3.098 * height * 100) - (4.330 * age)

    return round(bmr,2)
```

Body Fat Index

```python
@app.route('/calculate_bodyfat', methods=['POST'])
def calculate_bodyfat():
    if request.method == 'POST':
        gender = request.form['gender']
        weight = float(request.form['weight'])
        waist = float(request.form['waist'])
        wrist = float(request.form['wrist'])
        hip = float(request.form['hip'])
        forearm = float(request.form['forearm'])

        body_fat = calculate_body_fat_percentage(gender, weight, waist, wrist, hip, forearm)
        fitness_evaluation = evaluate_fitness(body_fat, gender)

        return render_template('bodyfat_result.html', gender=gender, body_fat=body_fat, fitness_evaluation=fitness_evaluation)

def calculate_body_fat_percentage(gender, weight, waist, wrist, hip, forearm):
    if gender == 'male':
        body_fat = (weight * 0.732) + (waist / 3.785) - (wrist * 0.393) - (hip * 0.394) + (forearm * 0.434) - 28.57
    else:  # female
        body_fat = (weight * 0.732) + (waist / 3.786) - (wrist * 0.393) - (hip * 0.394) + (forearm * 0.434) - 28.57
    return round(body_fat, 2)
```

Fitness Evaluation

```python
def evaluate_fitness(body_fat, gender):
    if gender == 'male':
        if body_fat < 8:
            return "Essential Fat"
        elif 8 <= body_fat < 14:
            return "Athletes"
        elif 14 <= body_fat < 18:
            return "Fitness"
        elif 18 <= body_fat < 25:
            return "Average"
        else:
            return "Obese"
    else:  # female
        if body_fat < 21:
            return "Essential Fat"
        elif 21 <= body_fat < 25:
            return "Athletes"
        elif 25 <= body_fat < 31:
            return "Fitness"
        elif 31 <= body_fat < 38:
            return "Average"
        else:
            return "Obese"
```

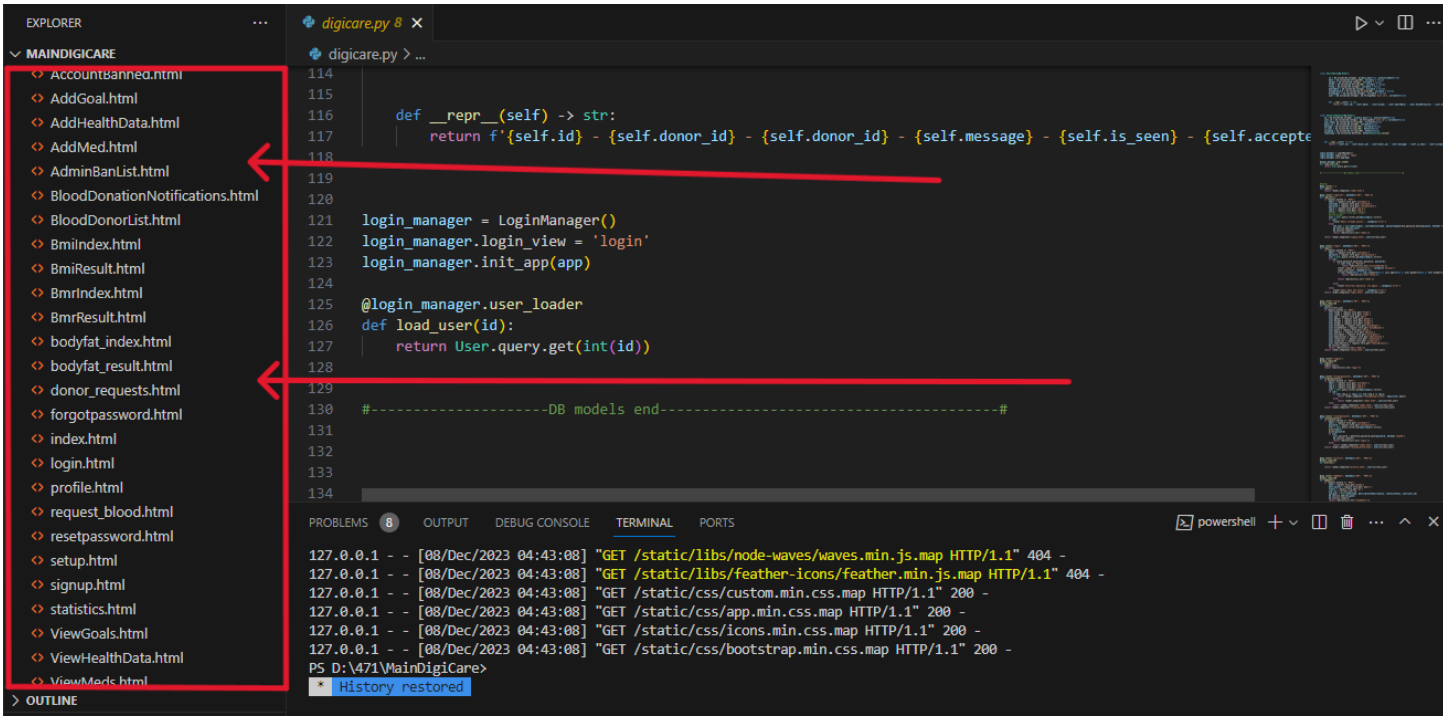
Admin Panel

An admin panel is available for administrative purposes. Admins can ban and unban users as needed.

```
@app.route("/AdminBanList", methods=['GET', 'POST'])
@login_required
def AdminBanList():
    users = User.query.filter_by(aType="User").all()
    return render_template("AdminBanList.html", user=current_user, users=users)


@app.route("/AdminBan/<int:id>", methods=['GET', 'POST'])
@login_required
def AdminBan(id):
    user = User.query.get_or_404(id)


    user.flag = "Banned"
    db.session.commit()
    return redirect(url_for('AdminBanList'))

@app.route("/AdminUnban/<int:id>", methods=['GET', 'POST'])
@login_required
def AdminUnban(id):
    user = User.query.get_or_404(id)


    user.flag = "Unbanned"
    db.session.commit()
    return redirect(url_for('AdminBanList'))

@app.route("/AccountBanned", methods=['GET', 'POST'])
def AccountBanned():
    return render_template("AccountBanned.html", user=current_user)
```

Frontend Development

The front end of the DigiCare project is meticulously designed to provide a seamless and user-friendly experience. All pages share a common foundation built with HTML, CSS, and JavaScript, incorporating elements from Bootstrap for enhanced design aesthetics. The front-end files are systematically organized within the templates folder. Through efficient routing, each page establishes a connection with the database, ensuring the cohesive functioning of the entire project. This structured approach to front-end development not only ensures consistency in design but also facilitates a dynamic and responsive user interface. The integration of HTML, CSS, JavaScript, and Bootstrap elements creates an aesthetically pleasing and intuitively navigable interface, contributing to the overall success and user satisfaction of the DigiCare platform.



Index.html

The provided `index.html` file is the frontend code for the main page of DigiCare. It is designed using a combination of HTML, CSS, and JavaScript, with additional styling elements borrowed from Bootstrap for a visually appealing and responsive layout. The document includes metadata for defining the page title, description, and author. It references various external libraries and resources for icons, styling, and functionality. The navigation bar is thoughtfully organized with dropdown menus, directing users to different sections of the platform related to health tracking and assistance. The hero section showcases a call-to-action with login and signup buttons. The footer contains links for company information, support, and copyright details. The JavaScript

files at the end handle dynamic page elements and functionality. Overall, the code reflects a well-structured and aesthetically pleasing homepage for the DigiCare platform.

```
<section class="section pb-0 hero-section" id="hero">
    <div class="bg-overlay bg-overlay-pattern"></div>
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-lg-10 col-sm-8">
                <div class="text-center mt-lg-5 pt-5">
                    <h1 class="display-4 fw-semibold mb-3 lh-base">The better way to get to know more about Healthcare with <span
                            class="text-success">DigiCare </span></h1>
                    <p class="lead text-muted lh-base">A platform for health tracking and assistance</p>

                    <div class="d-flex gap-2 justify-content-center mt-4">
                        {% if current_user.is_authenticated %}
                        <div class="btn-group me-0">
                            <a href="/profile" class="btn btn-primary me-1">Profile</a>
                            <a href="/logout" class="btn btn-outline-danger">Logout<i class="ri-eye-line align-middle ms-0"></i></a>
                        </div>
                        {% else %}
                        <div class="me-0">
                            <a href="/login" class="btn btn-danger me-0">Login<i class="ri-eye-line align-middle ms-0"></i></a>
                            <a href="/register" class="btn btn-primary">Sign Up</a>
                        </div>
                        {% endif %}

                    </div>
                </div>


            </div>
        </div>
        <!-- end row -->
    </div>
</div>
```

This part is basically for the main viewpoint which the users going to view in the first page.

In every static page, the same thing is running repeatedly. But according to the features, there is some modifications such as:

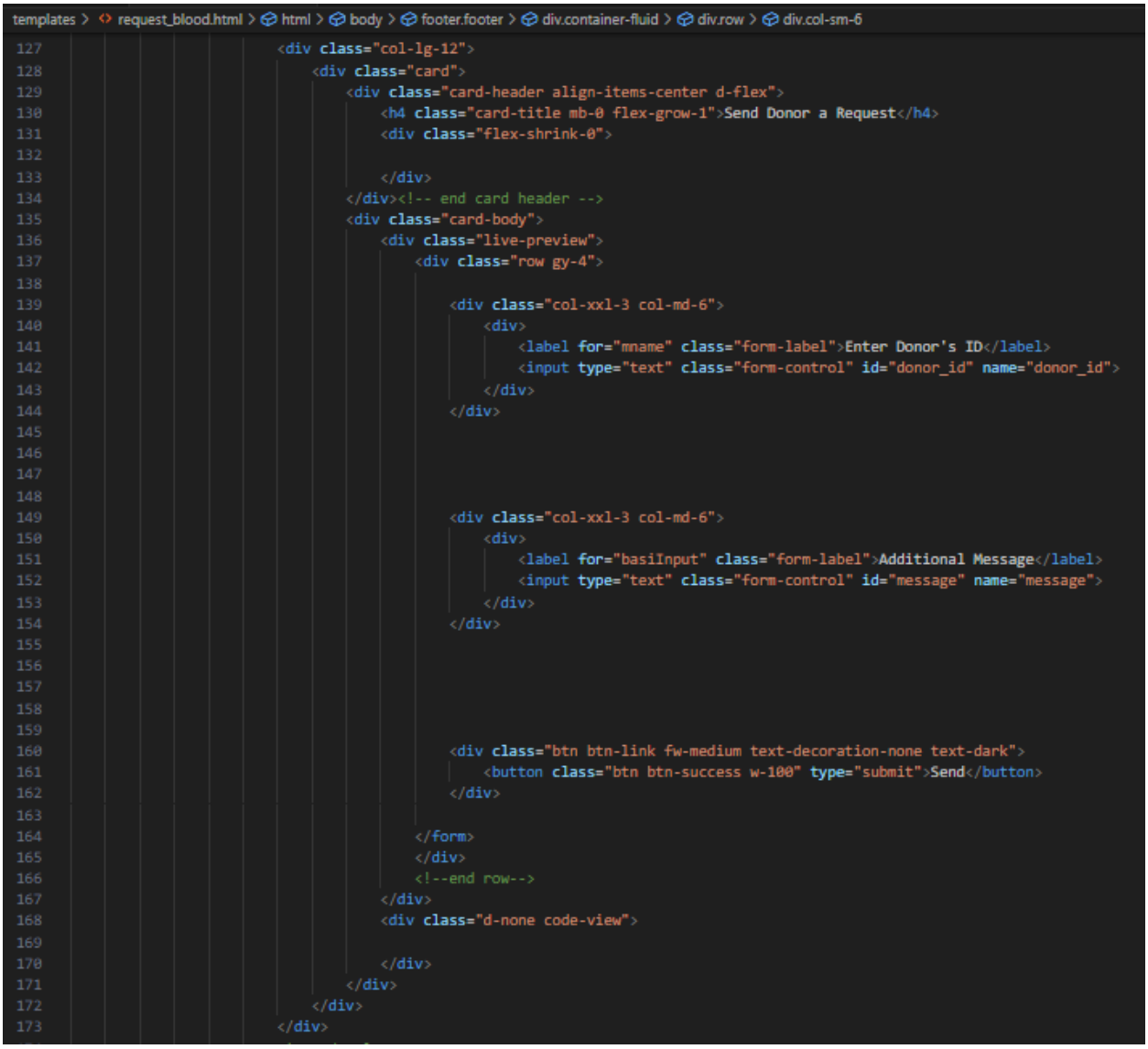


Figure 32: Blood Donar page

templates > BloodDonorList.html > html > body > div.main-content > div.page-content > div.container-fluid > footer.footer > div.container-fluid

```
                }

                /* Alternate Row Colors */
                .table-striped tbody tr:nth-of-type(odd) {
                    background-color: #fafafa; /* Alternate row color */
                }

                /* Table Header Styling */
                .table thead th {
                    background-color: #007bff; /* Table header background color */
                    color: #ffffff; /* Table header text color */
                    border-color: #007bff; /* Table header border color */
                }

                /* Table Body Text Color */
                .table tbody td {
                    color: #333333; /* Table body text color */
                }
            </style>
        </head>
        <body>
            <div class="col-xxl-8">
                <h5 class="mb-3 p-3">Active Blood Donors Among Users</h5>
                <!-- Table with Styling -->
                <div class="table-responsive">
                    <table class="table table-bordered table-hover">
                        <thead class="table-light">
                            <tr>
                                <th scope="col">Doner's ID</th>
                                <th scope="col">Blood Type</th>
                                <th scope="col">User Name</th>
                                <th scope="col">Phone</th>
                                <th scope="col">Address</th>
                                <th scope="col">Action</th>

                            </tr>
                        </thead>
                        <tbody>
                            {% for u in users %}
                            <tr>
                                <td>{{ u.id }}</td>
                                <td>{{ u.bloodType }}</td>
                                <td>{{ u.username }}</td>
                                <td>{{ u.phone }}</td>
                                <td>{{ u.address }}</td>
                                <td><a href="/request_blood" class="btn btn-primary btn-sm">Request</a></td>
                            </tr>
                            {% endfor %}
```

Figure 33: Blood Donar List

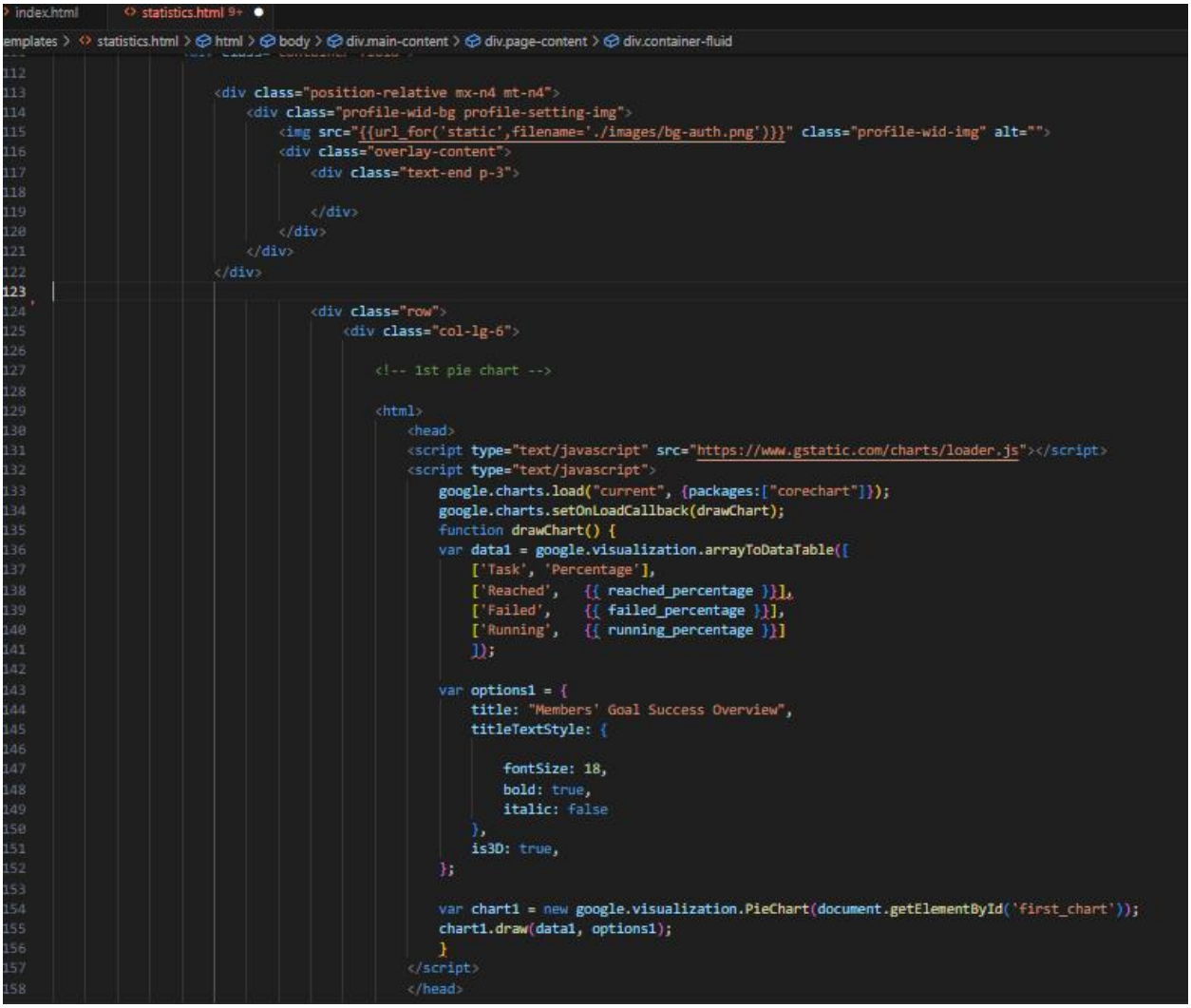


Figure 34: Statastic page

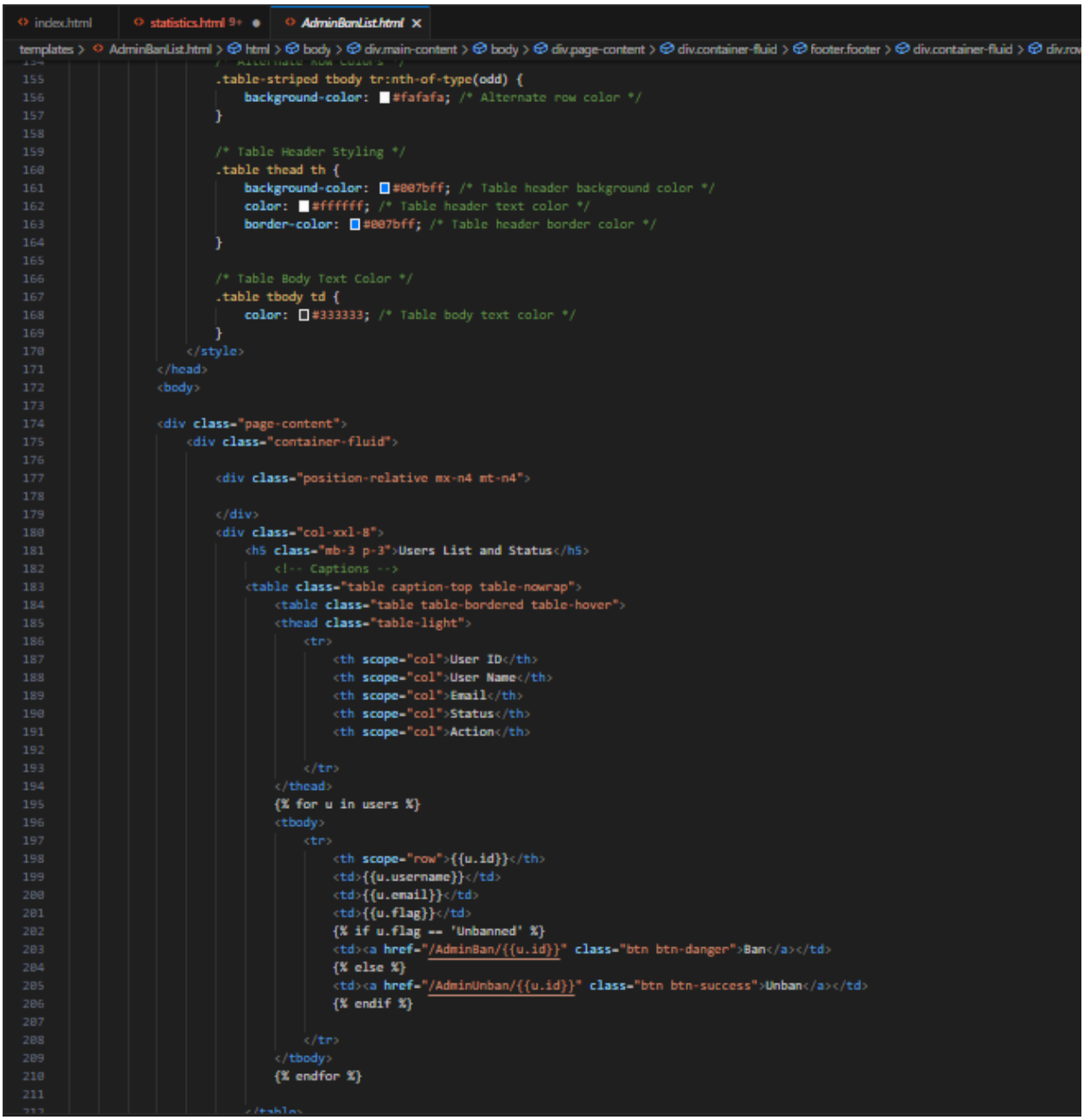
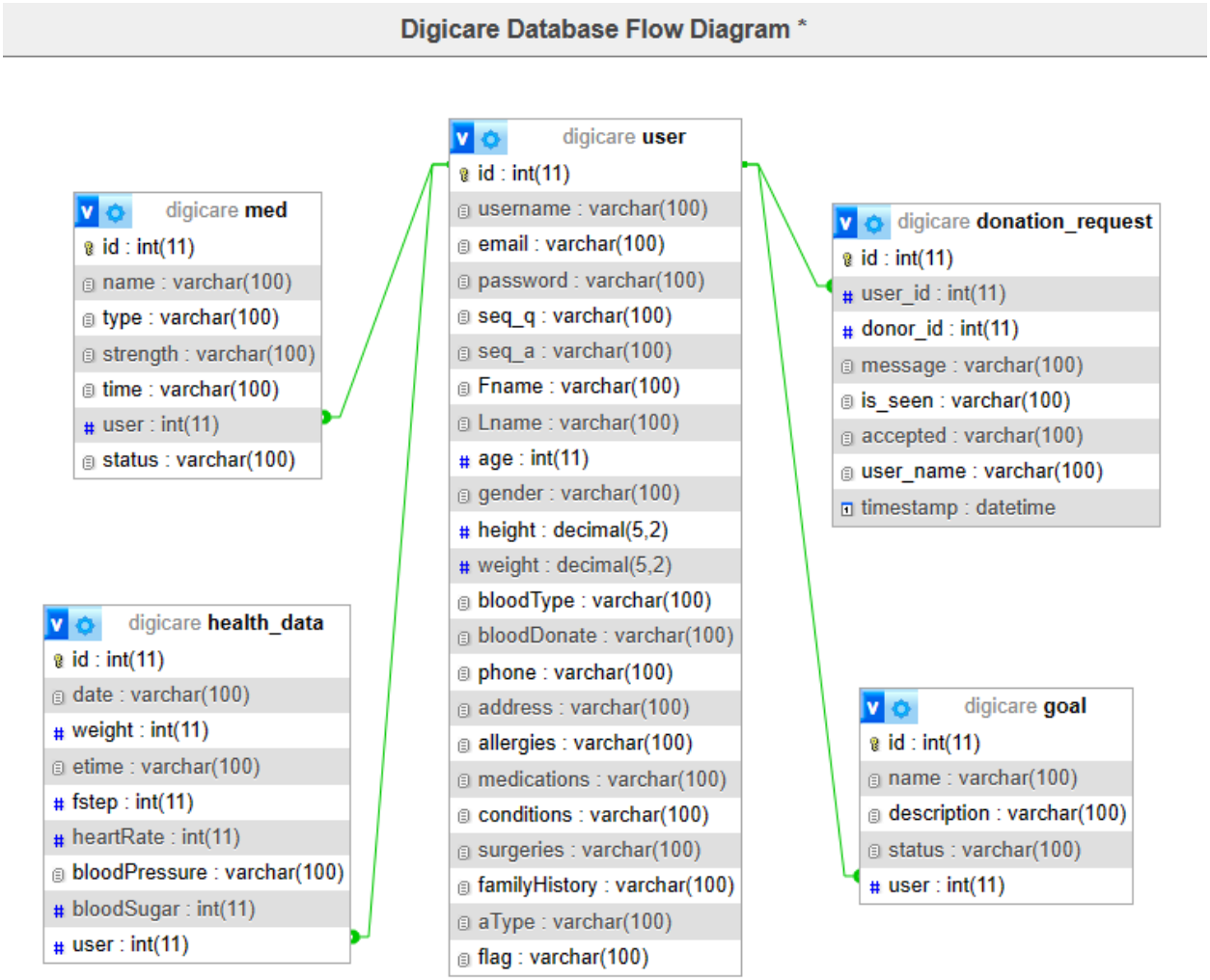


Figure 35: admin ban list

The frontend development of DigiCare represents a harmonious blend of HTML, CSS, and JavaScript, with additional styling contributions from Bootstrap for a polished and responsive design. The carefully crafted layout ensures a seamless and visually appealing user experience across various devices. Each page, using a shared base, is organized within the templates folder, enabling efficient routing that establishes a connection with the database, allowing the entire

project to function cohesively. The navigation bar elegantly guides users to different sections of the platform, fostering an intuitive journey through health tracking and assistance features. The strategic use of dropdown menus enhances user interaction. The hero section, complemented by call-to-action buttons, invites users to engage with the platform. The overall design, as evidenced by the attached screenshots, reflects a commitment to user-friendly aesthetics and navigability, encapsulating the essence of DigiCare's mission for comprehensive health management.

Database Schema



Technology (Framework, Languages)

The project is built on a Flask backend, seamlessly integrated with HTML, CSS, and JavaScript for the frontend. This combination of technologies forms a robust platform that encompasses a wide range of health management features. The database functionality is supported by MySQL or phpMyAdmin, ensuring efficient data storage and retrieval. This technological foundation provides a solid framework for the development and implementation of diverse functionalities within the health management and assistance platform.

Conclusion

In summary, Digicare is a one-stop platform for managing health and getting assistance. It offers a range of user-friendly features like easy sign-up, secure log-in, and customizable profiles. With tools to track health data, set goals, and connect with potential blood donors, Digicare is all about making health management simple. Users can input and manage their health metrics, set goals, and track progress in real-time. The platform also includes handy calculators for BMI, BMR, and body fat. In the Blood Donation section, users can request blood donations and receive notifications, creating a supportive community. Digicare's admin features ensure a safe and ethical environment. Administrators can manage blood donation requests, accept, or reject them, and oversee user activities. The platform's goal is to revolutionize health monitoring with an intuitive interface and practical features. In a nutshell, Digicare is reshaping how individuals approach health management, emphasizing simplicity, connectivity, and innovation.