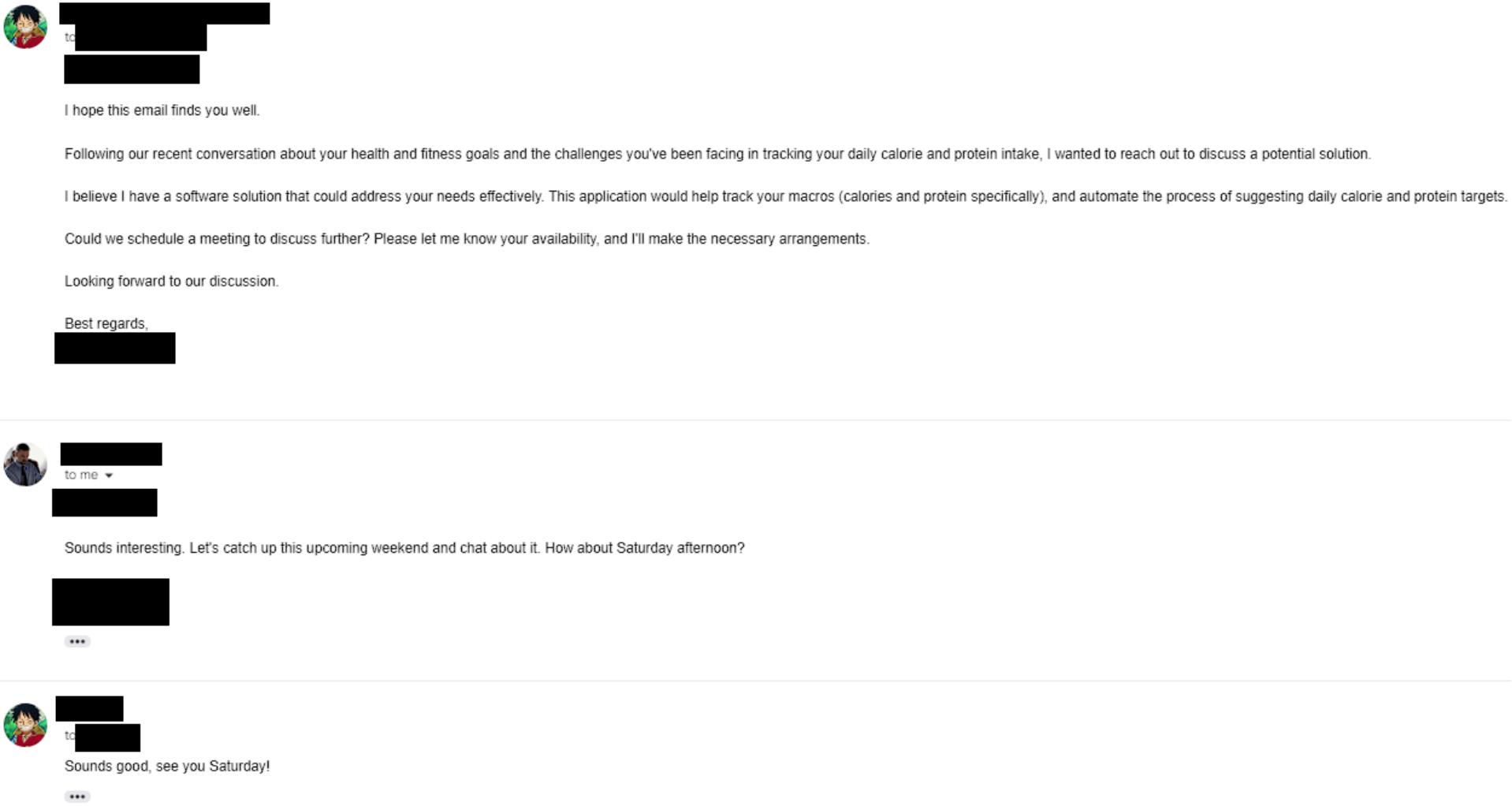
Appendix 1: Email contacting client



[redacted]
to [redacted]
[redacted]

I hope this email finds you well.

Following our recent conversation about your health and fitness goals and the challenges you've been facing in tracking your daily calorie and protein intake, I wanted to reach out to discuss a potential solution.

I believe I have a software solution that could address your needs effectively. This application would help track your macros (calories and protein specifically), and automate the process of suggesting daily calorie and protein targets.

Could we schedule a meeting to discuss further? Please let me know your availability, and I'll make the necessary arrangements.

Looking forward to our discussion.

Best regards,
[redacted]

[redacted]
to me
[redacted]

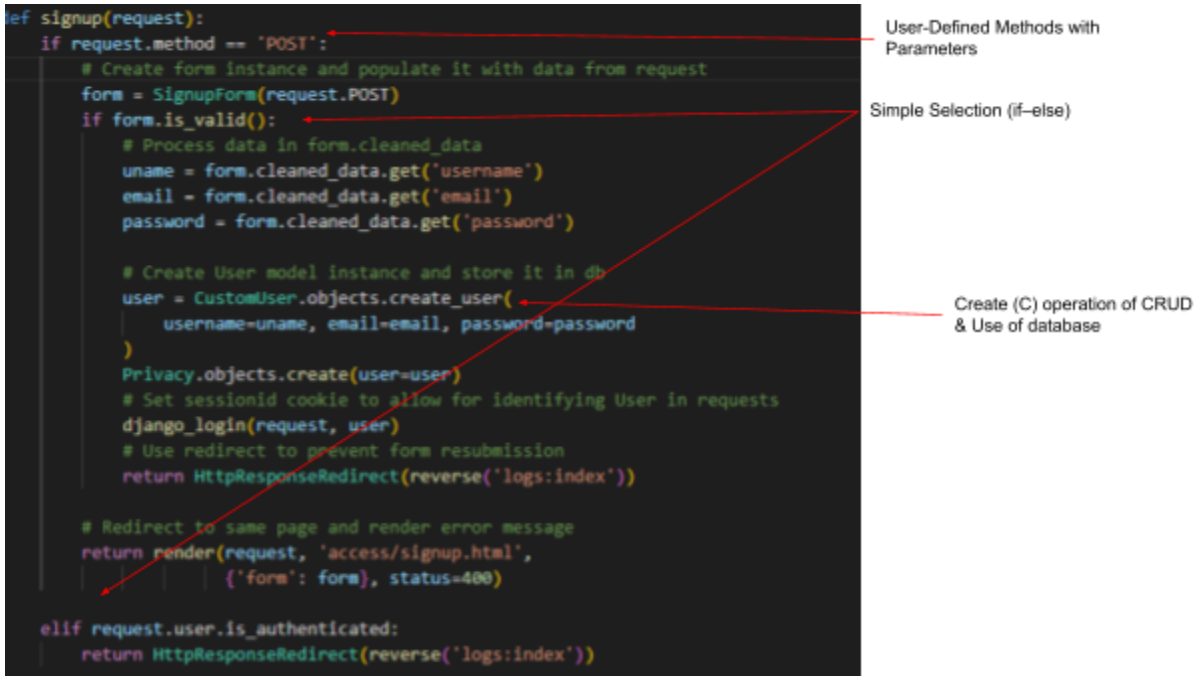
Sounds interesting. Let's catch up this upcoming weekend and chat about it. How about Saturday afternoon?

[redacted]

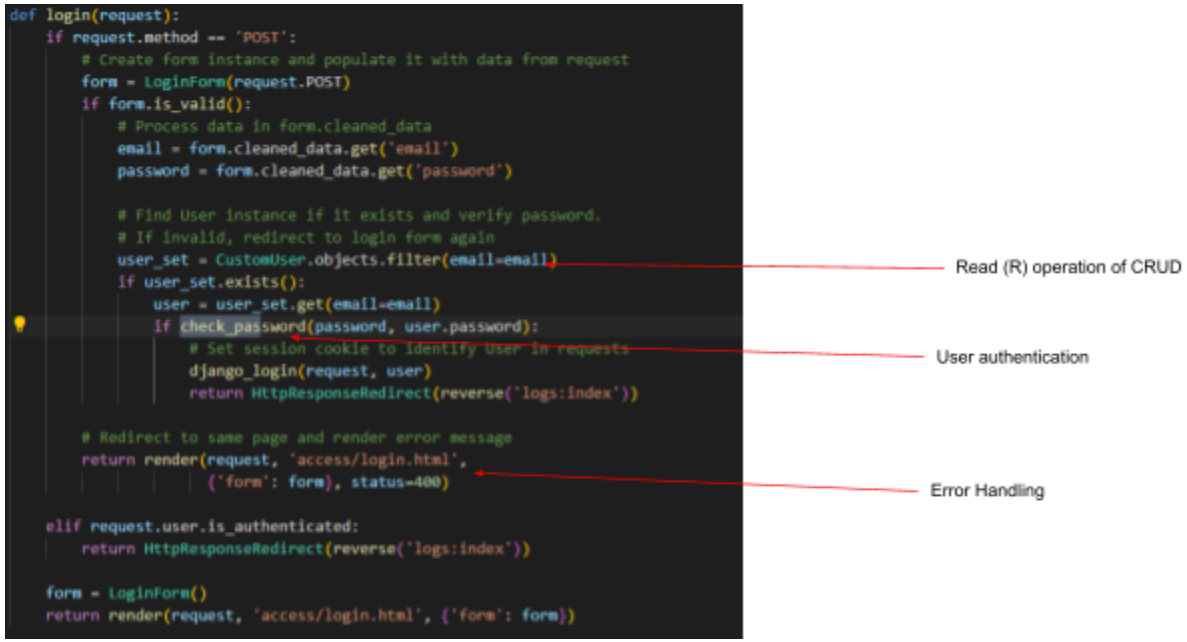
[redacted]
to [redacted]

Sounds good, see you Saturday!

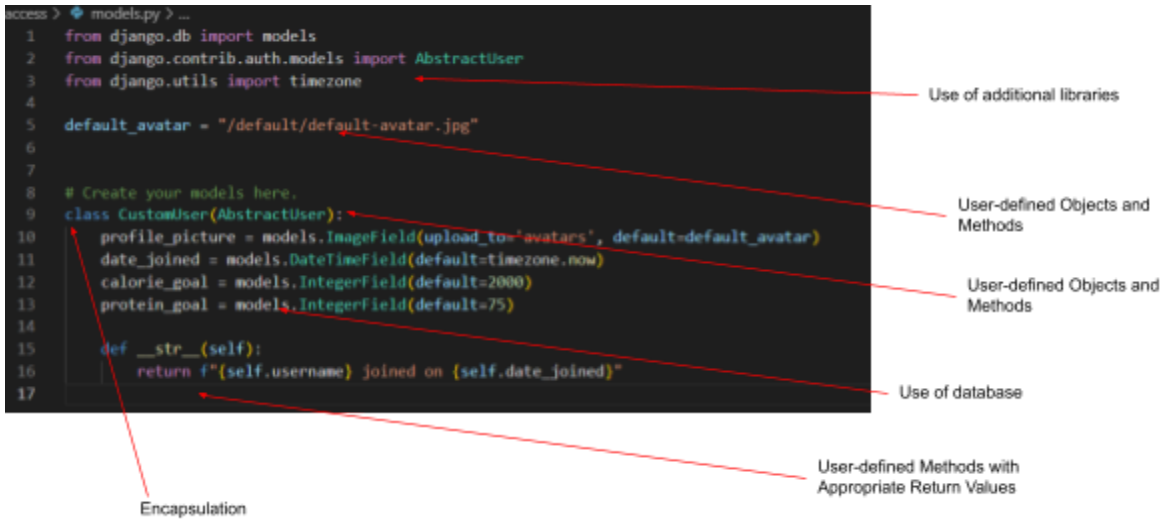
Appendix 2: Sign up form and registering user accounts to database.



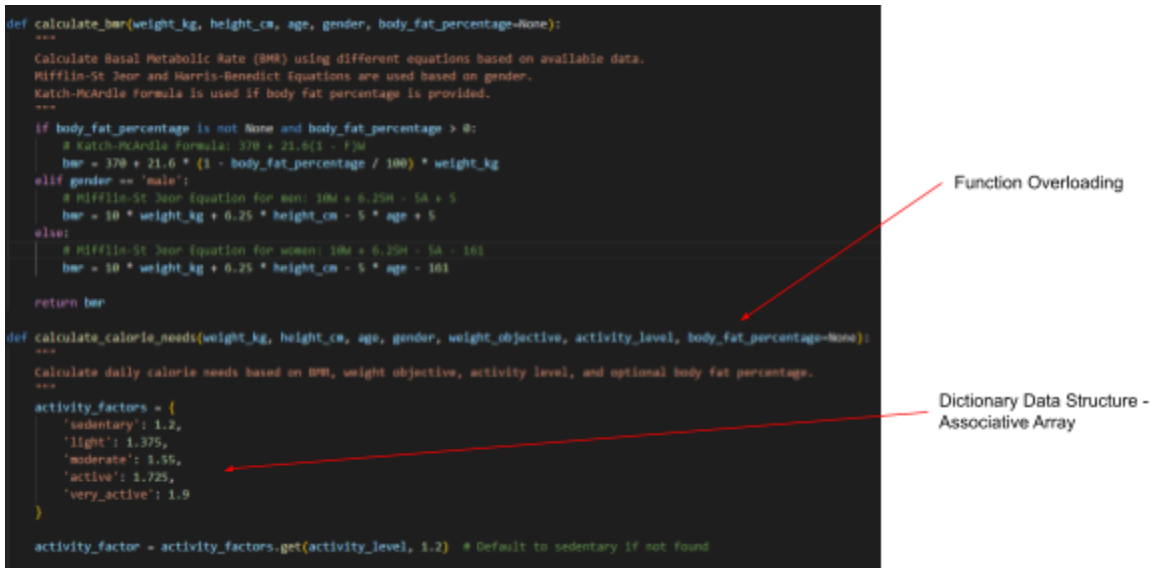
Appendix 3: Login form and User authentication,



Appendix 4: Definition of CustomUser and recording info to database



Appendix 5: Calculating recommended Calorie and Protein Goals



Appendix 6: Questionnaire Form

```python
def questionnaire_view(request):
    if request.method == 'POST':
        form = QuestionnaireForm(request.POST)
        if form.is_valid():
            # Extracting form data
            weight_kg = form.cleaned_data['current_body_weight_kg']
            height_cm = form.cleaned_data['height_cm']
            age = form.cleaned_data['age']
            gender = form.cleaned_data['gender']
            weight_objective = form.cleaned_data['weight_objective']
            activity_level = form.cleaned_data['activity_level']
            body_fat_percentage = form.cleaned_data.get('body_fat_percentage')  # Optional, so use get

            # Call the calculation functions
            maintenance_calories, adjusted_calories = calculate_calorie_needs(
                weight_kg, height_cm, age, gender, weight_objective, activity_level, body_fat_percentage
            )
            daily_protein = calculate_protein_needs(weight_kg)

            context = {
                'form': form,
                'daily_calories': maintenance_calories,
                'adjusted_calories': adjusted_calories,
                'daily_protein': daily_protein
            }
            return render(request, 'questionnaire/questionnaire.html', context)
    else:
        form = QuestionnaireForm()

    return render(request, 'questionnaire/questionnaire.html', {'form': form})
```

Complex selection (nested if)

Data handling

Dictionary Data Structure - Associative Array

Simple selection (if–else)

User-defined methods with appropriate return values

Appendix 7: Create meal log

```python
def create_log(request):
    if not request.user.is_authenticated:
        return HttpResponseRedirect(reverse('access:signup'))

    if request.method == 'POST':
        form = FoodForm(request.POST, request.FILES)
        if form.is_valid():
            food_name = form.cleaned_data['name']
            desc = form.cleaned_data['desc']
            calories = form.cleaned_data['calories']
            protein = form.cleaned_data['protein']
            img = form.cleaned_data.get('image')

            food_obj = Food(creator=request.user, name=food_name, desc=desc,
                            calories=calories, protein=protein, image=img)
            food_obj.save()

            log = Log(creator=request.user, food=food_obj, pub_date=timezone.now())
            log.save()
            return HttpResponseRedirect(reverse('logs:index'))

    else:
        form = FoodForm()

    return render(request, 'logs/create-log.html', {'form': form})
```

User Authentication

Data Handling

Create (C) operation of CRUD & Use of database

Templates

Appendix 8: Edit meal log

```python
def edit_log(request, log_id=None):
    if log_id:
        log = get_object_or_404(Log, pk=log_id)
        if log.creator != request.user:
            # Redirect to an error page or handle as appropriate
            return redirect('some-error-page')
        initial_data = {'name': log.food.name, 'desc': log.food.desc, 'calories': log.food.calories, 'protein': log.food.protein}
        form = FoodForm(instance=log.food, initial=initial_data)
    else:
        log = None
        form = FoodForm()

    if request.method == 'POST':
        if log:
            form = FoodForm(request.POST, request.FILES, instance=log.food)
        else:
            form = FoodForm(request.POST, request.FILES)

        if form.is_valid():
            food = form.save(commit=False)
            if not log:
                food.creator = request.user
                food.save()
                log = Log(creator=request.user, food=food, pub_date=timezone.now())
            else:
                food.save()
                log.food = food
                log.save()
            return redirect('logs:detail', log.id)

    return render(request, 'logs/create-log.html', {'form': form, 'log': log})
```

Update (U) operation of CRUD & Use of database

Complex selection (nested if, if with multiple conditions or switch)

Create (C) operation of CRUD & Use of database

Appendix 9: index.html, contains delete form

```
    </div>
    {% if user.email == log.creator.email %}
    <form id="delete-form" action="{% url 'logs:detail' log.id %}" method="post">
            {% csrf_token %}
            <button type="submit"> Delete log </button>
        </form>
        <form id="edit-form" action="{% url 'logs:edit-log' log.id %}" method="get">
            <button type="submit">Edit log</button>
        </form>
    {% endif %}
</div>
```

Delete (D) operation of CRUD & Use of database

Appendix 10: Calculations and display of calories/protein remaining, current calories/protein and calories/protein goals.

```python
# Get user's goals from the profile
if request.user.is_authenticated:
    user_goals = Customer.objects.get(id=request.user.id)
    calorie_goal = user_goals.calorie_goal
    protein_goal = user_goals.protein_goal

    # Calculate current calories and protein intake from today's logs
    today = timezone.now().date()
    midnight_today = timezone.make_aware(datetime.datetime.combine(today, datetime.time.min))
    midnight_tomorrow = timezone.make_aware(datetime.datetime.combine(today + datetime.timedelta(days=1), datetime.time.min))

    today_logs = Log.objects.filter(creator=request.user, pub_date__gte=midnight_today, pub_date__lt=midnight_tomorrow)
    current_calories = sum(log.food.calories for log in today_logs)
    current_protein = sum(log.food.protein for log in today_logs)

    context['calorie_goal'] = calorie_goal
    context['current_calories'] = current_calories
    context['calories_remaining'] = calorie_goal - current_calories
    context['protein_goal'] = protein_goal
    context['current_protein'] = current_protein
    context['protein_remaining'] = protein_goal - current_protein
```

Data handling

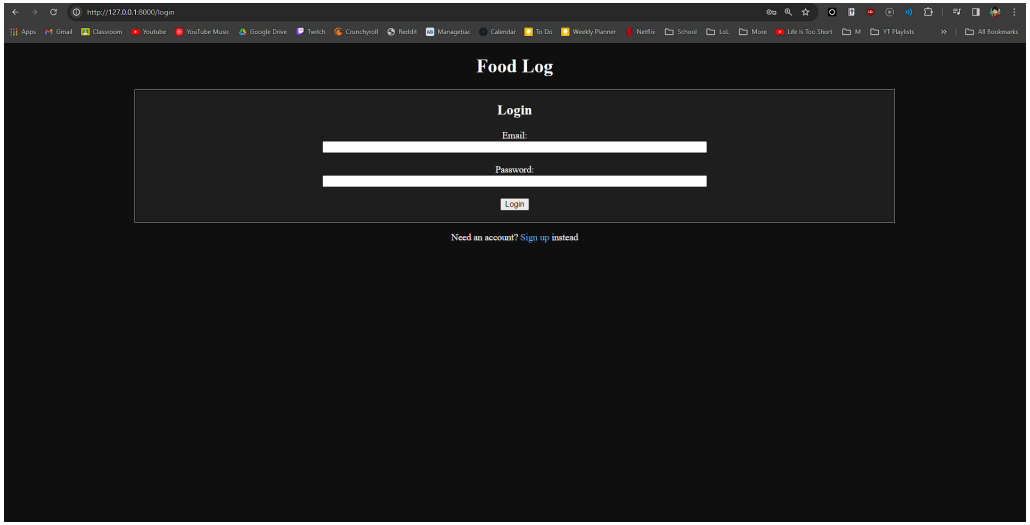
Object Oriented Programming

Querying and Date/Time Handling

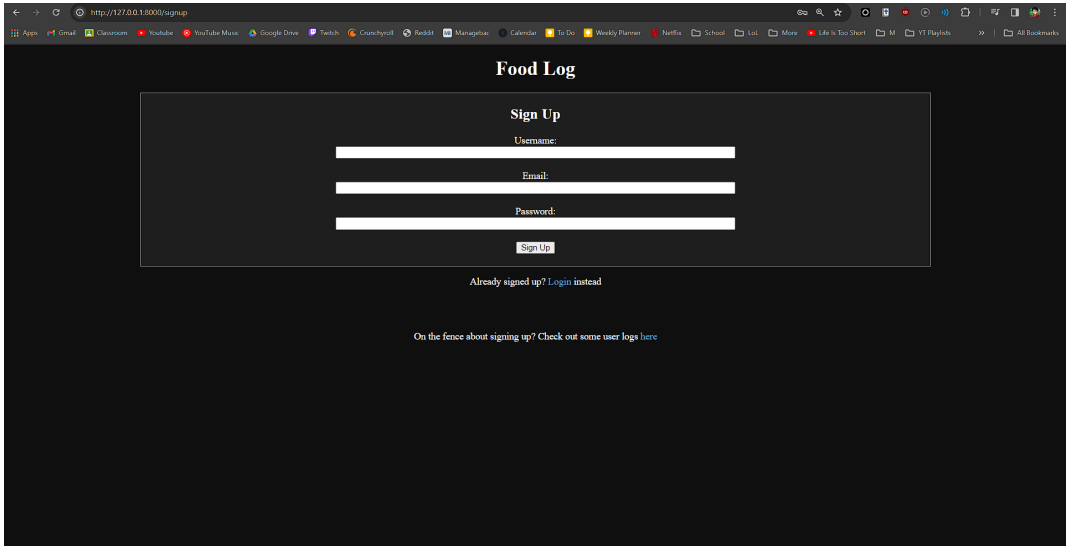
Templates and Context Management

Appendix 11: Website Pages

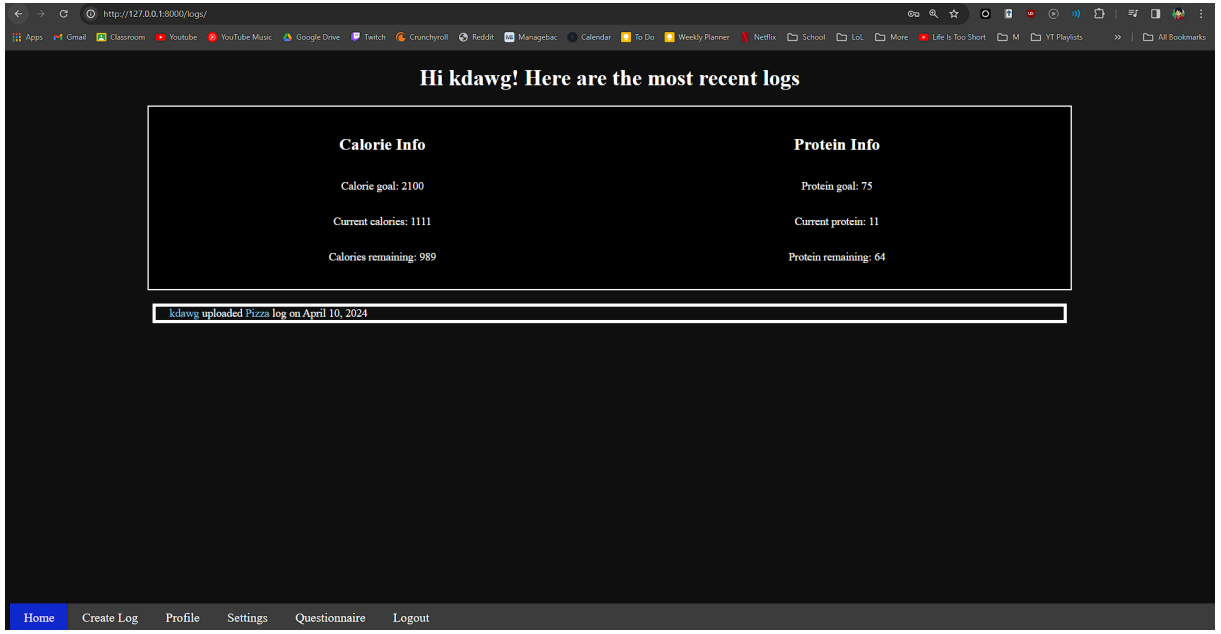
(i) Login Page



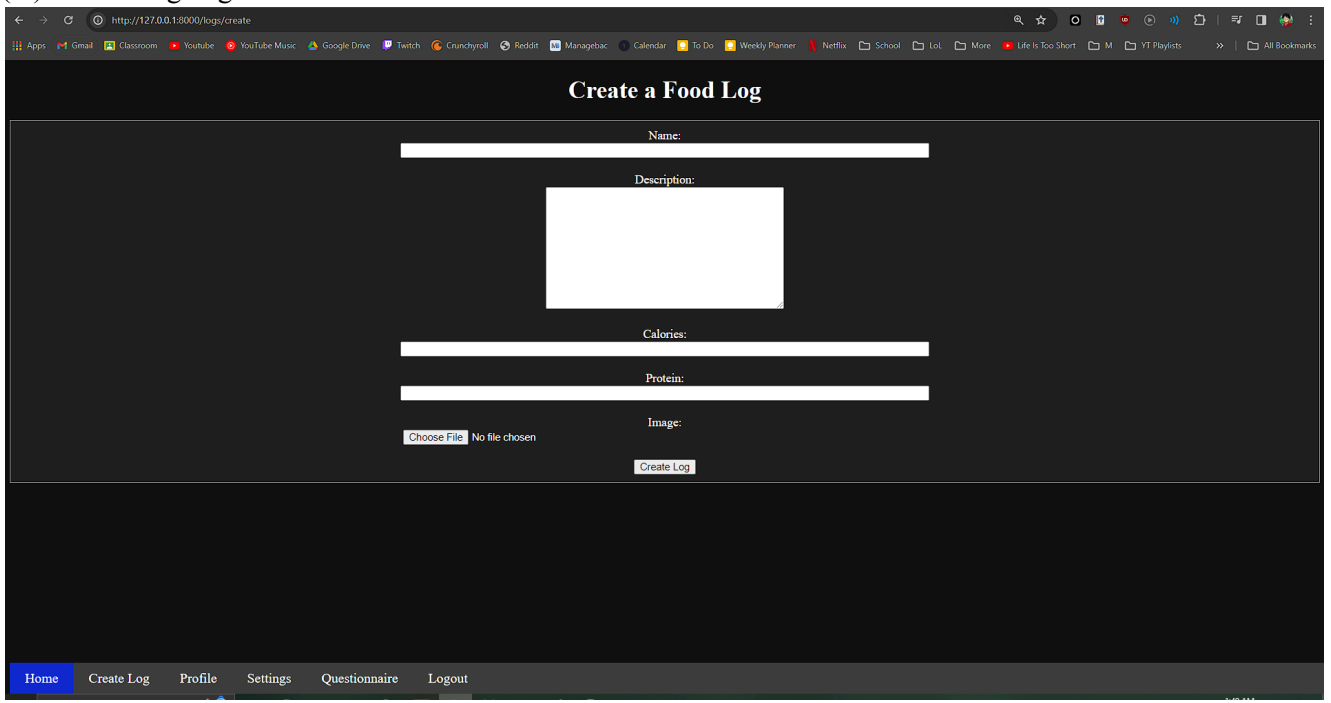
(ii) Signup Page



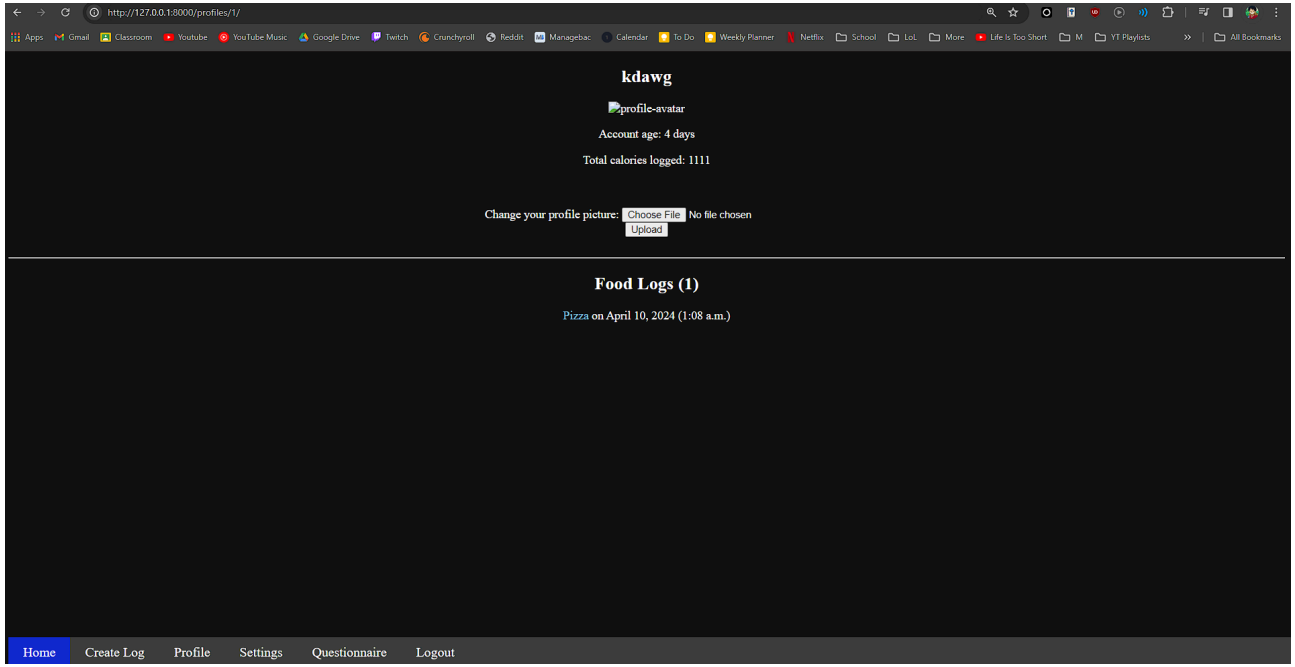
(iii) Home Page



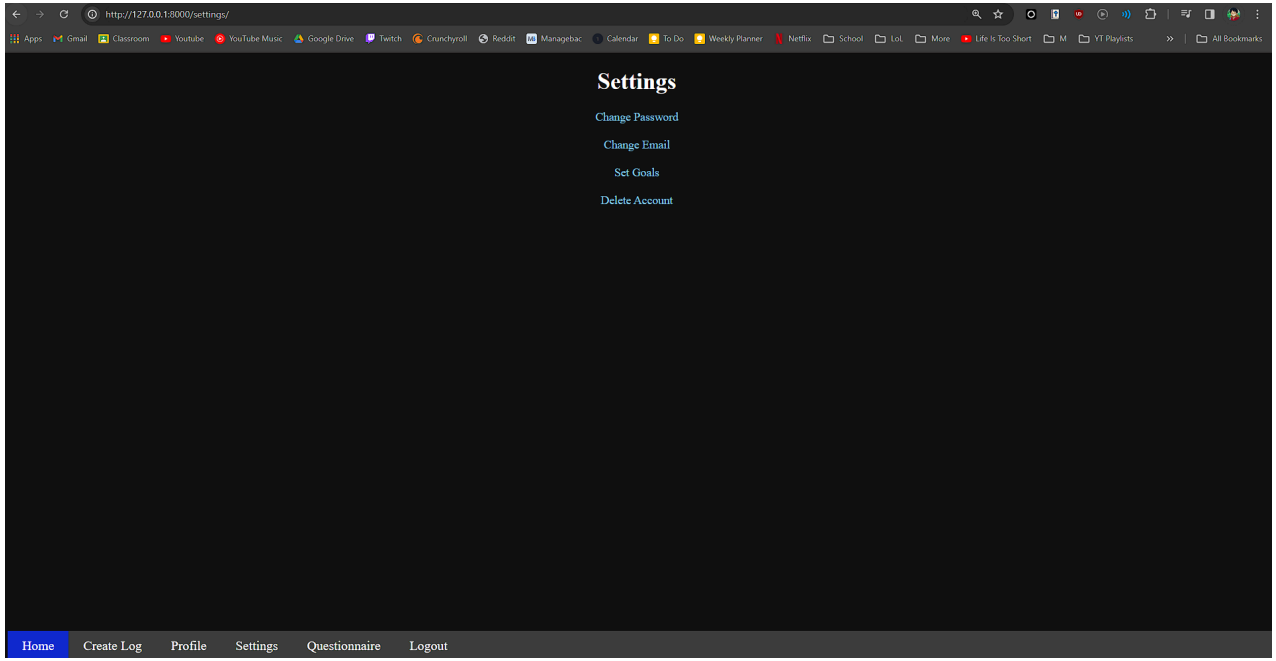
(iv) Create Log Page



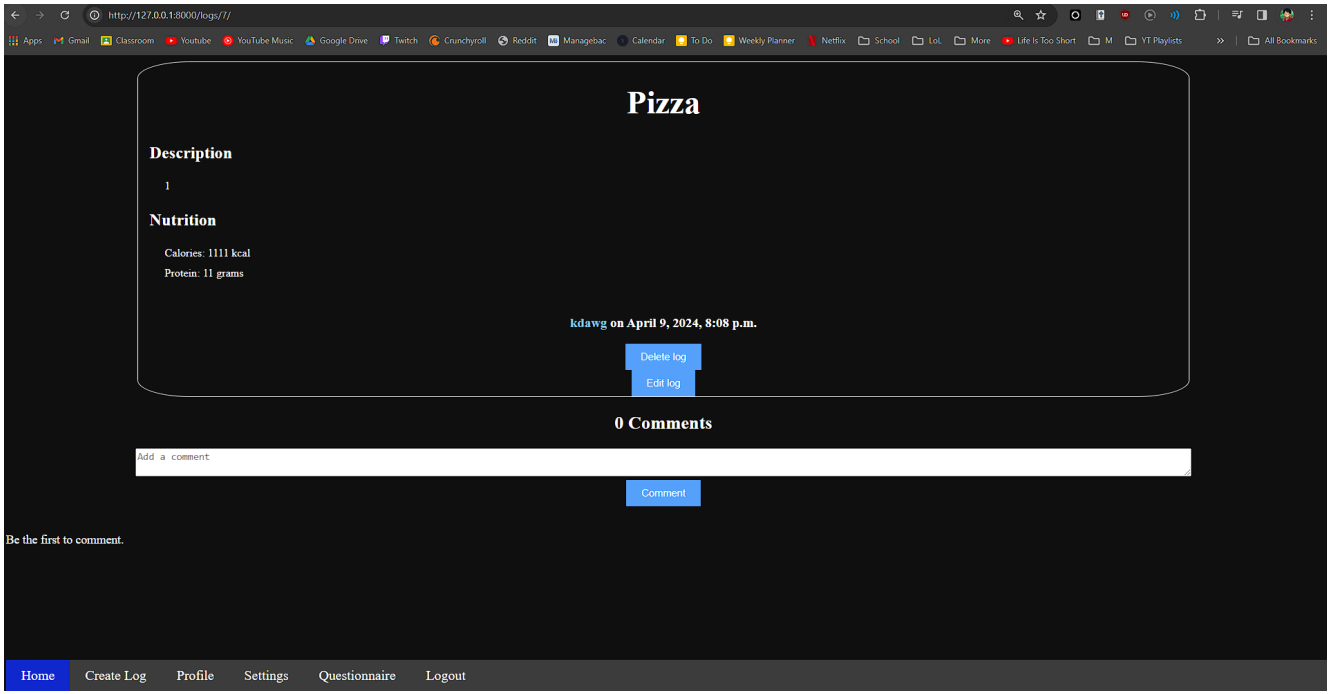
(v) Profile Page



(vi) Settings Page



(vii) Food Log (accessible by clicking on a food log on the Home Page)



Appendix 12: Final interview with the client. 22 Mar. 2024