

## Appendix 1: Email contacting client



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]



to me ▼  
[REDACTED]

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

[REDACTED]  
...



to [REDACTED]

Sounds good, see you Saturday!

...

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

```

def signup(request):
    if request.method == 'POST':
        # Create form instance and populate it with data from request
        form = SignupForm(request.POST)
        if form.is_valid():
            # Process data in form.cleaned_data
            uname = form.cleaned_data.get('username')
            email = form.cleaned_data.get('email')
            password = form.cleaned_data.get('password')

            # Create User model instance and store it in db
            user = CustomUser.objects.create_user(
                username=uname, email=email, password=password
            )
            Privacy.objects.create(user=user)
            # Set sessionId cookie to allow for identifying User in requests
            django_login(request, user)
            # Use redirect to prevent form resubmission
            return HttpResponseRedirect(reverse('logs:index'))

        # Redirect to same page and render error message
        return render(request, 'access/signup.html',
            {'form': form}, status=400)

    elif request.user.is_authenticated:
        return HttpResponseRedirect(reverse('logs:index'))

```

User-Defined Methods with Parameters

Simple Selection (if-else)

Create (C) operation of CRUD & Use of database

### Appendix 3: Login form and User authentication,

```

def login(request):
    if request.method == 'POST':
        # Create form instance and populate it with data from request
        form = LoginForm(request.POST)
        if form.is_valid():
            # Process data in form.cleaned_data
            email = form.cleaned_data.get('email')
            password = form.cleaned_data.get('password')

            # Find User Instance if it exists and verify password.
            # If invalid, redirect to login form again
            user_set = CustomUser.objects.filter(email=email)
            if user_set.exists():
                user = user_set.get(email=email)
                if check_password(password, user.password):
                    # Set session cookie to identify User in requests
                    django_login(request, user)
                    return HttpResponseRedirect(reverse('logs:index'))

            # Redirect to same page and render error message
            return render(request, 'access/login.html',
                {'form': form}, status=400)

    elif request.user.is_authenticated:
        return HttpResponseRedirect(reverse('logs:index'))

    form = LoginForm()
    return render(request, 'access/login.html', {'form': form})

```

Read (R) operation of CRUD

User authentication

Error Handling

## Appendix 4: Definition of CustomUser and recording info to database

```
access > models.py > ...
1 from django.db import models
2 from django.contrib.auth.models import AbstractUser
3 from django.utils import timezone
4
5 default_avatar = "/default/default-avatar.jpg"
6
7
8 # Create your models here.
9 class CustomUser(AbstractUser):
10     profile_picture = models.ImageField(upload_to='avatars', default=default_avatar)
11     date_joined = models.DateTimeField(default=timezone.now)
12     calorie_goal = models.IntegerField(default=2000)
13     protein_goal = models.IntegerField(default=75)
14
15     def __str__(self):
16         return f"{self.username} joined on {self.date_joined}"
17
```

Use of additional libraries

User-defined Objects and Methods

User-defined Objects and Methods

Use of database

Encapsulation

User-defined Methods with Appropriate Return Values

## Appendix 5: Calculating recommended Calorie and Protein Goals

```
def calculate_bmr(weight_kg, height_cm, age, gender, body_fat_percentage=None):
    """
    Calculate Basal Metabolic Rate (BMR) using different equations based on available data.
    Mifflin-St Jeor and Harris-Benedict Equations are used based on gender.
    Katch-McArdle Formula is used if body fat percentage is provided.
    """
    if body_fat_percentage is not None and body_fat_percentage > 0:
        # Katch-McArdle Formula: 370 + 21.6 * (1 - body_fat_percentage / 100) * weight_kg
        bmr = 370 + 21.6 * (1 - body_fat_percentage / 100) * weight_kg
    elif gender == 'male':
        # Mifflin-St Jeor Equation for men: 10 * weight_kg + 6.25 * height_cm - 5 * age + 5
        bmr = 10 * weight_kg + 6.25 * height_cm - 5 * age + 5
    else:
        # Mifflin-St Jeor Equation for women: 10 * weight_kg + 6.25 * height_cm - 5 * age - 161
        bmr = 10 * weight_kg + 6.25 * height_cm - 5 * age - 161

    return bmr

def calculate_calorie_needs(weight_kg, height_cm, age, gender, weight_objective, activity_level, body_fat_percentage=None):
    """
    Calculate daily calorie needs based on BMR, weight objective, activity level, and optional body fat percentage.
    """
    activity_factors = {
        'sedentary': 1.2,
        'light': 1.375,
        'moderate': 1.55,
        'active': 1.725,
        'very_active': 1.9
    }

    activity_factor = activity_factors.get(activity_level, 1.2) # Default to sedentary if not found
```

Function Overloading

Dictionary Data Structure - Associative Array

## Appendix 6: Questionnaire Form

```
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

```
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

```
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.

```
# Set user's goals from the profile
if request.user.is_authenticated:
    user_goals = CustomUser.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.combine(today, datetime.time.min))
midnight_tomorrow = timezone.make_aware(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

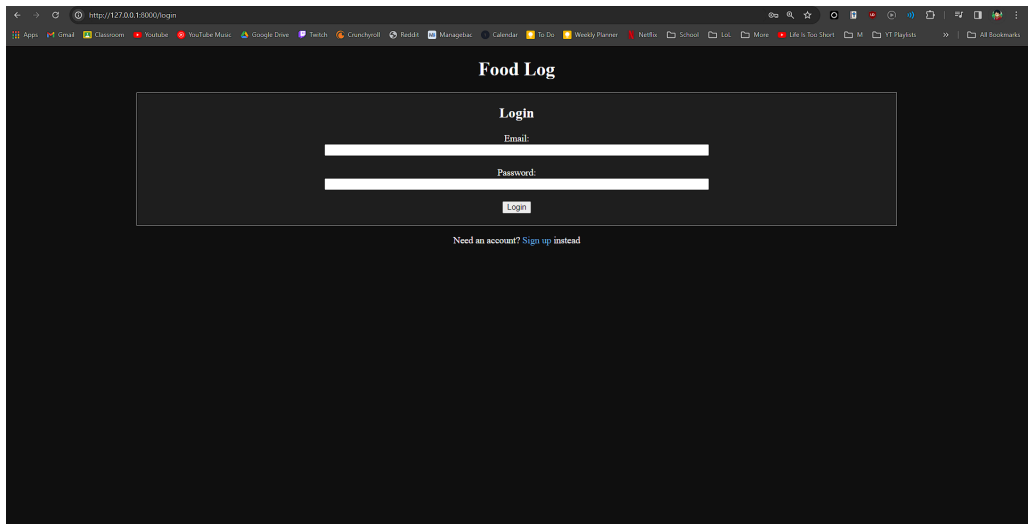
Object Oriented Programming

Querying and Date/Time  
Handling

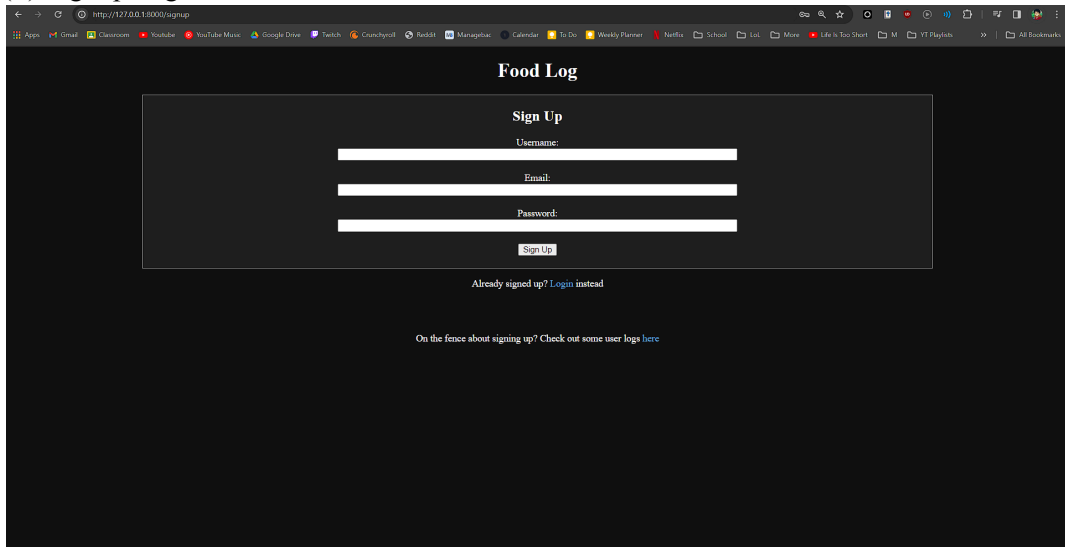
Templates and Context  
Management

Appendix 11: Website Pages

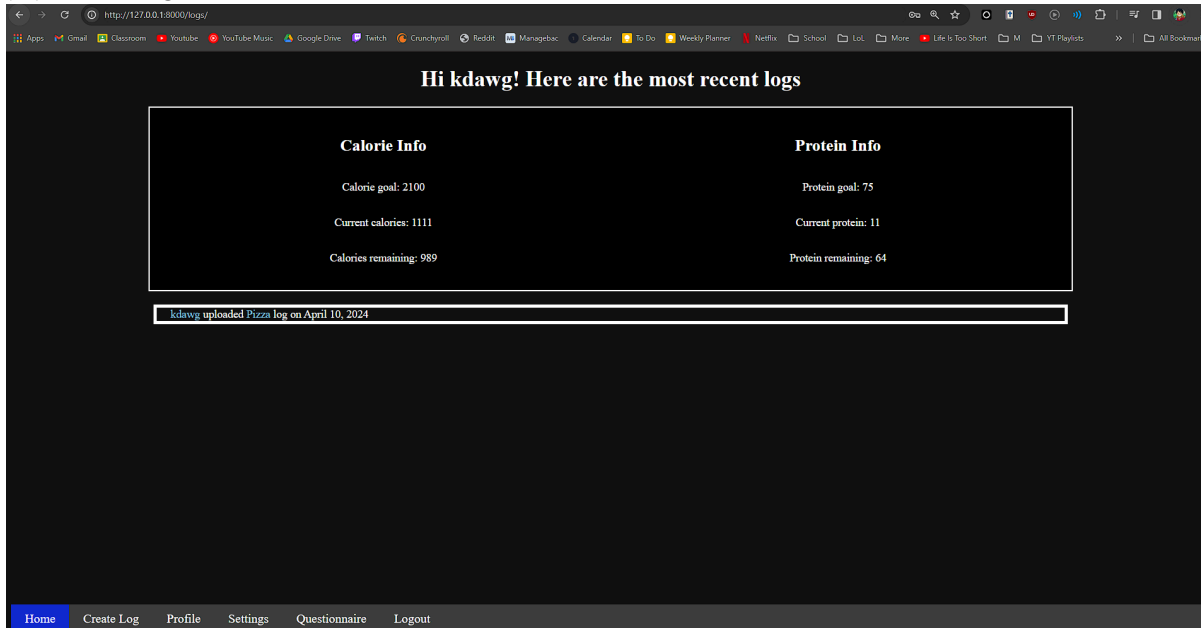
(i) Login Page



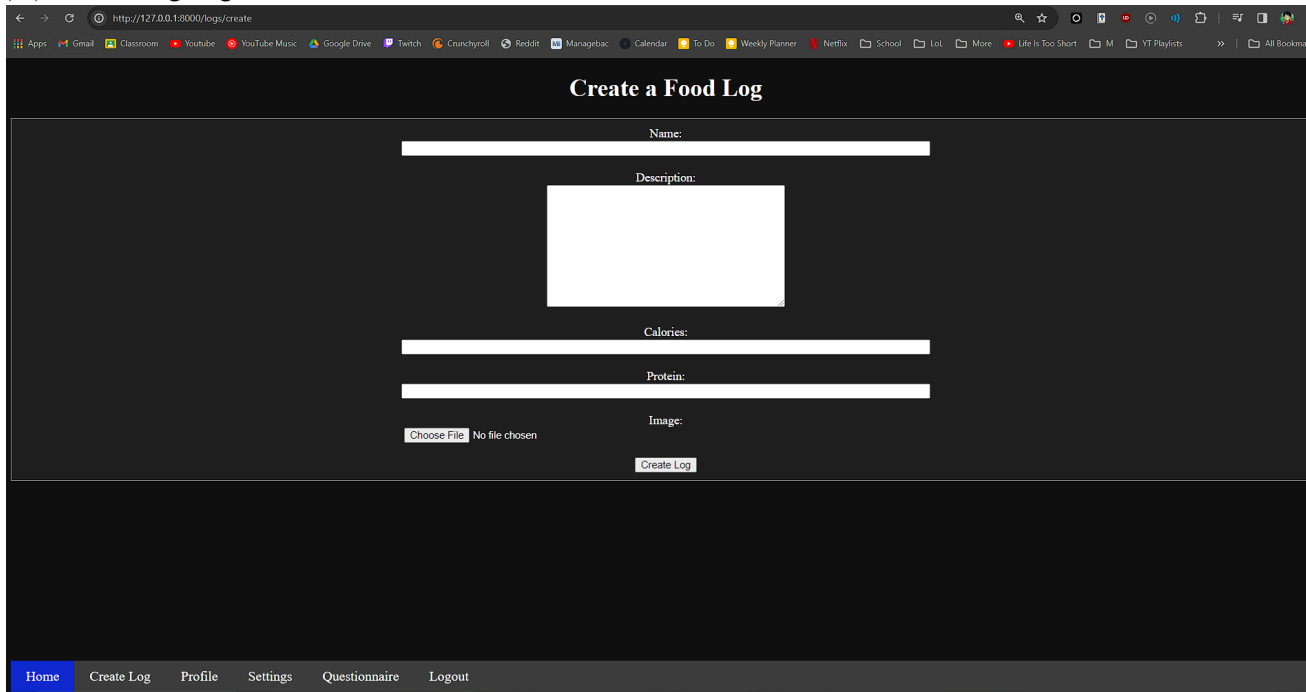
(ii) Signup Page



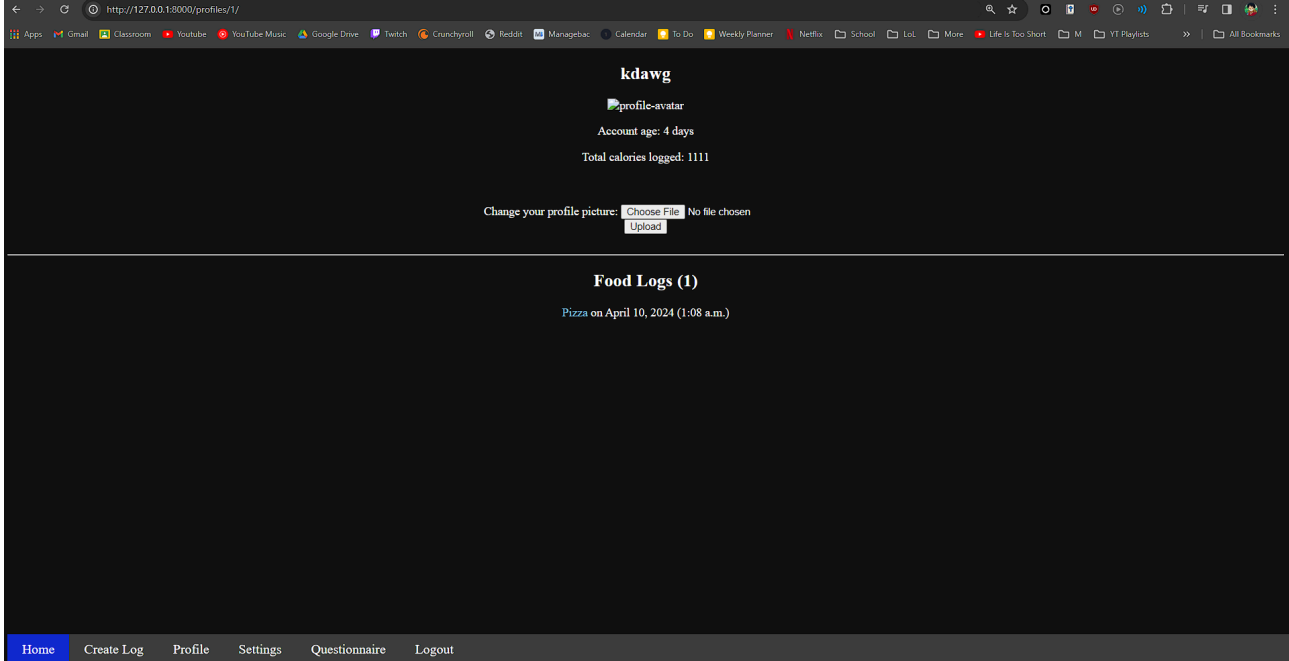
### (iii) Home Page



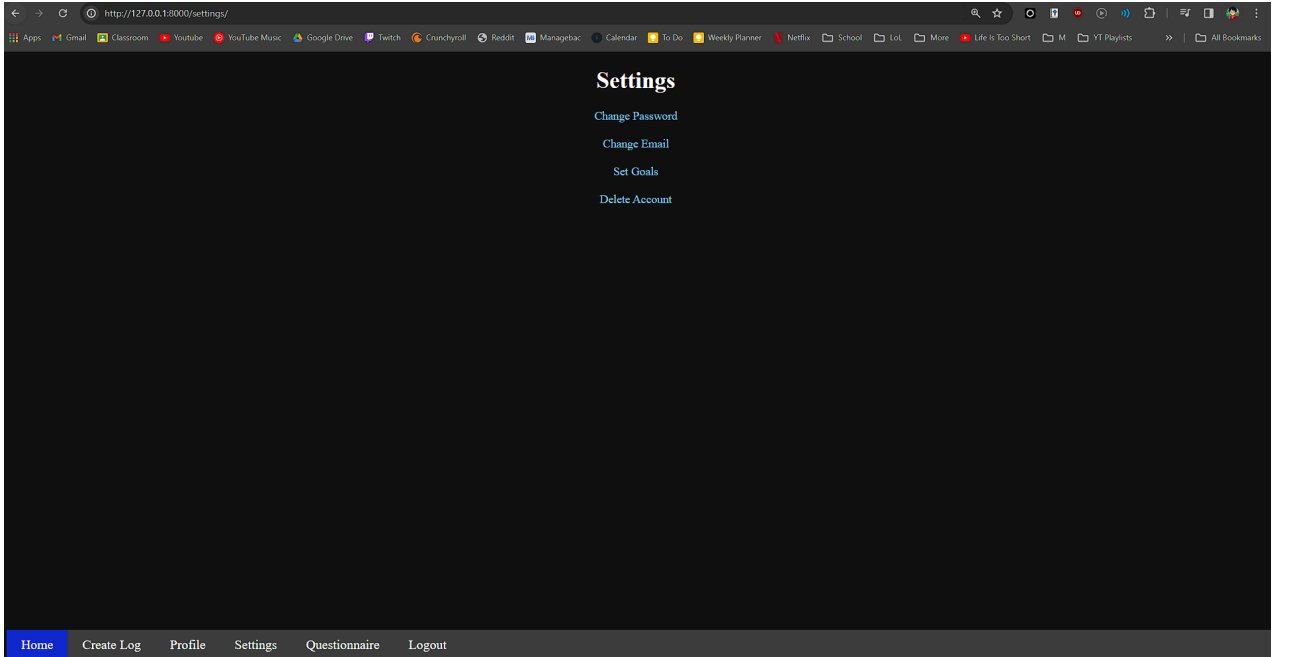
### (iv) Create Log Page



(v) Profile Page

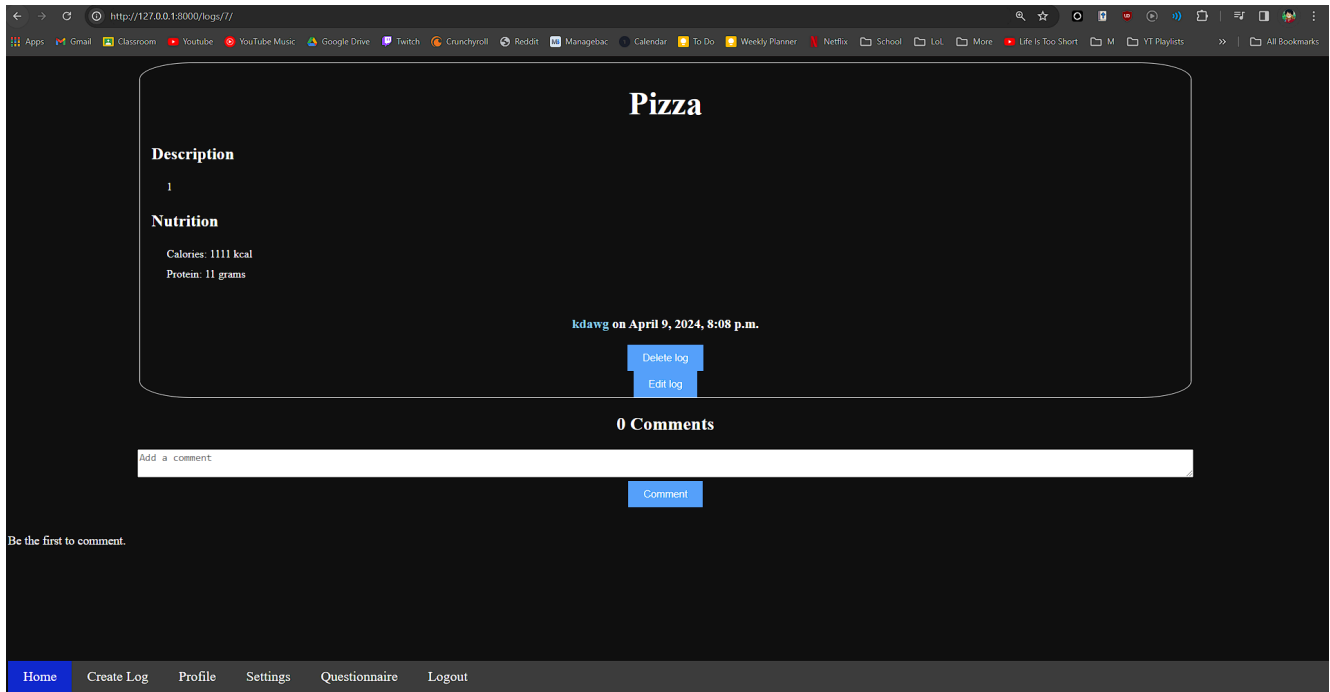


(vi) Settings Page





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



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