CaloriQ

Nutritional Assistant

Problem

- Ineffective **generalized** advice
- Lack of **personalized** attention in other workout/nutrition apps
- Eating disorder prevalence

Target Customers

With the field of nutrition being convoluted with advice from every corner imaginable, it can be hard to find what works for you.

With an AI, we can remove the difficulty of taking your first step by providing a feature that will make your first step for you and get started on customized journey.

We have a few people that are interested in our product. Here are their statements.

Subject 1

I've always struggled to lose weight because it was hard to make or find a meal plan to follow. I can't wait to see a meal plan made from an AI that helps me stay on track with interesting foods that aren't the usual boring norm.

Subject 2

As a professional ice skater, I am trying to gain strength but stay lean through my training. It has been hard trying to find a meal plan that conforms to me. But with AI, I am looking forward to a customized meal plan tailored to me.

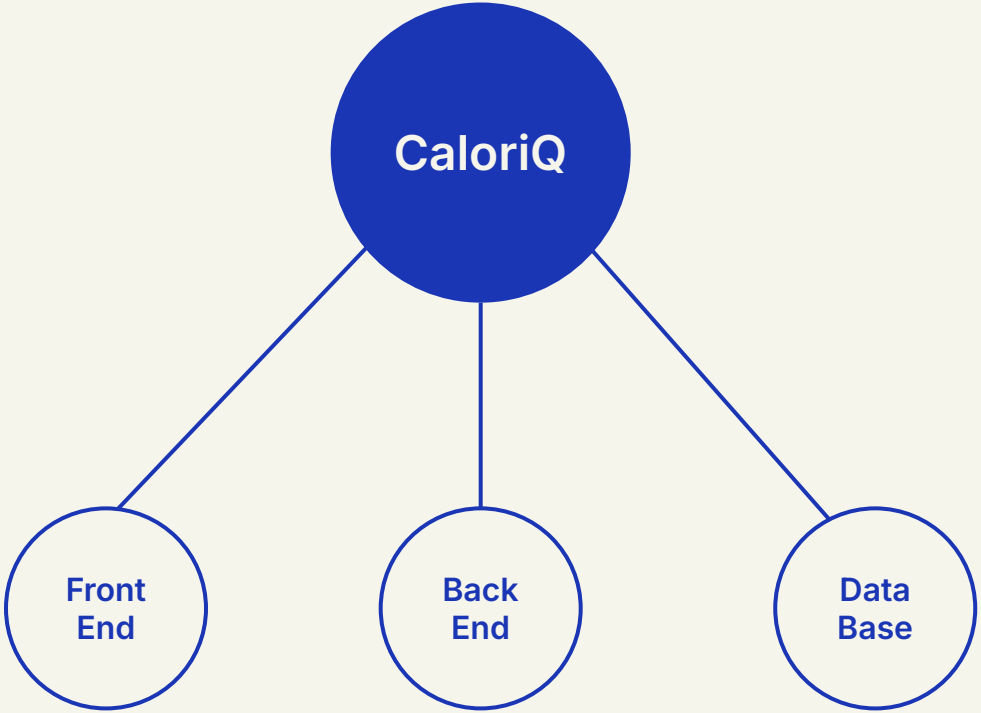
Cool features

- We use AI but behind the scenes
- Users can choose when to regenerate routines
- Support Chatbot* (Not Professional Help)

DEMO

Architecture

- **Front End**
  - Next.js
  - TailwindCSS / CSS
  - Typescript
- **Back End**
  - FastAPI
  - Python
  - DeepSeek
  - Ollama
- **Database**
  - PostgreSQL
- **Model**
  - DeepSeek



Front End

Front End Structure

Next.JS Framework

- Routing with file names
- Local storage for cookies, tokens, etc.
- Fast build times

Axios Fetching

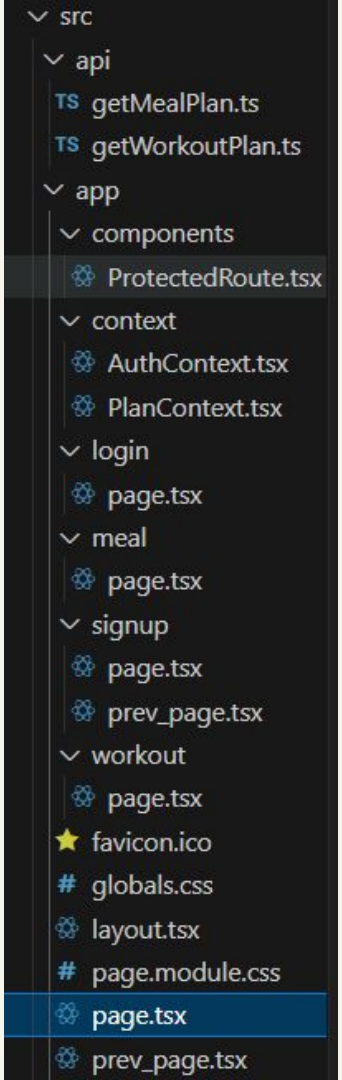
- Fetching made easy
- Stores authentication headers to make calls to backend easy

Easy to Learn

- React
- TailwindCSS, Bootstrap, etc.
- Typescript
  - Helps with typing issues
  - Catch errors fast

Next.JS Routing

```
// redirect the user to the home page after responses are received
router.push("/");
```



Axios Fetching

```
// fire both the meal and workout plan API requests in parallel
const [meal_response, workout_response] = await Promise.all([
    axios.post(
        "http://localhost:8000/user/meals",
        { headers: { Authorization: `Bearer ${auth.user?.access_token}` } }
    ),
    axios.post(
        "http://localhost:8000/user/workouts",
        { headers: { Authorization: `Bearer ${auth.user?.access_token}` } }
    ),
]);
```

```
// use axios to send a POST request to specified URL
// include formData as body of request
// headers tells server how to interpret data
const response = await axios.post("http://localhost:8000/auth/token", formData, {
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
});

// once response is recieved, set default Authorization header for all subsequent axios requests
axios.defaults.headers.common["Authorization"] = `Bearer ${response.data.access_token}`;

// save access token in browser's local storage to persist across sessions or page refreshes
localStorage.setItem("token", response.data.access_token);
```

Backend

Backend Routes

Authentication

- Created register and login routes.
- Generates a cookie that is attached to the user.
- Allows user to access protected routes.

User Preferences

- Decode cookie sent from frontend
- Change data types from survey to database types
- Store User Preferences from initial survey in the database

Workout / Meal Plans

- Call models that create a routine based on user preferences.
- Call functions to extract keywords from surveys

Authentication Cookie

```python
@router.post('/token', response_model=Token)
async def access_token_login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()], db: db_dependency):
    print(f"Login attempt - Username: {form_data.username}, Password: {form_data.password}")
    user = auth_user(form_data.username, form_data.password, db)
    if not user:
        print("Authentication failed for user:", form_data.username)
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Problem Validating User"
        )

    print("Authentication successful for user:", user.username)
    token = create_access_token(user.username, user.id, timedelta(minutes=60))
    return {'access_token': token, 'token_type': 'bearer'}
```

```python
def create_access_token(username: str, user_id: int, expires_delta: timedelta):
    encrypt = {'sub': username, 'id': user_id}
    token_expires = datetime.now(timezone.utc) + expires_delta
    encrypt.update({'exp': token_expires})
    return jwt.encode(encrypt, SECRET_KEY, algorithm=ALGORITHM)
```

User Preferences

```python
@router.post('/preferences', status_code=status.HTTP_201_CREATED)
async def get_data(db: db_dependency, request: Request):
    user = await get_user(db, request)
    user_data = await request.json()

    #turn inputs into data types of database, some inputs lists, have to turn into comma separated strings
    user.age = int(user_data.get("age")) or user.age #looks for age in Json data, turns into int, else sets to user.age
    user.height_cm = float(user_data.get("height")) or user.height_cm
    user.weight_kg = float(user_data.get("weight")) or user.weight_kg
    user.gender = Gender(user_data.get("gender")) or user.gender
    user.activity_level = ActivityLevel(user_data.get("activityLevel")) or user.activity_level
    user.diet_preference = user_data.get("dietPreference") or user.diet_preference
    user.allergies = ",".join(user_data.get("allergyArray", [])) or user.allergies


    user.exercise_preferences = [ExercisePreferences(item) for item in user_data.get("exercisePreference", [])] or user.exercise_preferences
    user.fitness_goals = [FitnessGoals(item) for item in user_data.get("fitnessGoals", [])] or user.fitness_goals
    user.meal_prep_availability = [DayOfWeek(item) for item in user_data.get("mealPrepAvailability", [])] or user.meal_prep_availability
    user.exercise_availability = [DayOfWeek(item) for item in user_data.get("exerciseAvailability", [])] or user.exercise_availability

    db.commit()
    db.refresh(user)
```

Workout and Meal Plans

```python
@router.post('/meals', status_code=status.HTTP_201_CREATED)
async def gen_meal_plan(db: db_dependency, request: Request):
    user = await get_user(db, request)

    try:
        raw_meal_plan = create_meal_plan(user.id)
        meal_plan_dict = parse_meal_plan_to_dict(raw_meal_plan)
        print(meal_plan_dict)

        user.meal_plan = meal_plan_dict
        db.add(user)
        db.commit()
        db.refresh(user)

        return {"meal_plan": meal_plan_dict}

    except Exception as e:
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail=f"Error generating meal plan: {str(e)}")
```

Database / Model

Database

Personal Information	id, email, password, first name, last name, age, gender, height, weight
Activity Level	Sedentary, Lightly Active, Moderately Active, Very Active
Fitness goals	Weight Loss, Gain Muscle, Increase Endurance, Improve Flexibility, General Fitness, Sports Performance, Maintain Weight
Exercise preferences	Cardio, Strength Training, Flexibility & Mobility, High-Intensity Interval Training
Diet preferences & Allergies (open)	Vegan, Keto, etc. / peanuts, shellfish, etc.
Availability	Exercise & Meal Prep Availability (days of the week)
Generated plans	Meal Plan, Workout Plan (dictionaries)
Preferences	liked_meals, disliked_meals, liked_workouts, disliked_workouts (lists of keywords)

Generating Plans

Step 1: query the database to get the user's info → build a context paragraph

```
user_context = (
    f"{user.age}-year-old {user.gender if user.gender else 'person'} "
    f"weighing {user.weight_kg} kg and {user.height_cm} cm tall, "
    f"with {user.activity_level if user.activity_level else 'unspecified'} activity level, "
    f"aiming for {', '.join(goal for goal in user.fitness_goals)}, "
    f"following a {user.diet_preference if user.diet_preference else 'flexible'} diet, "
    f"liking foods such as {user.liked_meals if user.liked_meals else 'unspecified'}, "
    f"disliking foods such as {user.disliked_meals if user.disliked_meals else 'unspecified'}, "
    f"alergic to (DO NOT INCLUDE THESE FOODS) {user.allergies if user.allergies else 'no specific allergies'}, "
)
```

Step 2: query DeepSeek with the user's context → returns the plans as a string

```
data = {
    "model": "deepseek/deepseek-chat:free",
    "messages": [{
        "role": "user",
        "content": (
            f"I am [{user_context}]. Help me create a 7-day meal plan with breakfast, lunch, and dinner. "
            f"The meals should be balanced, nutritious, and supportive of my goals, with flexibility in portion sizes "
            f"to avoid strict calorie restrictions. Prioritize whole foods and avoid unhealthy options like excessive "
            f"processed foods or sugary snacks. If muscle gain is a goal, emphasize protein-rich options. "
            f"Meal prep should take no more than 30 minutes for each meal. "
            f"Return the output in this format: day_breakfast_lunch_dinner. "
            f"For example: Monday_eggs with ham_grilled chicken with veggies_steak with rice. "
            f"Only return this parsed output, no extra text."
            )
    }]
}
```

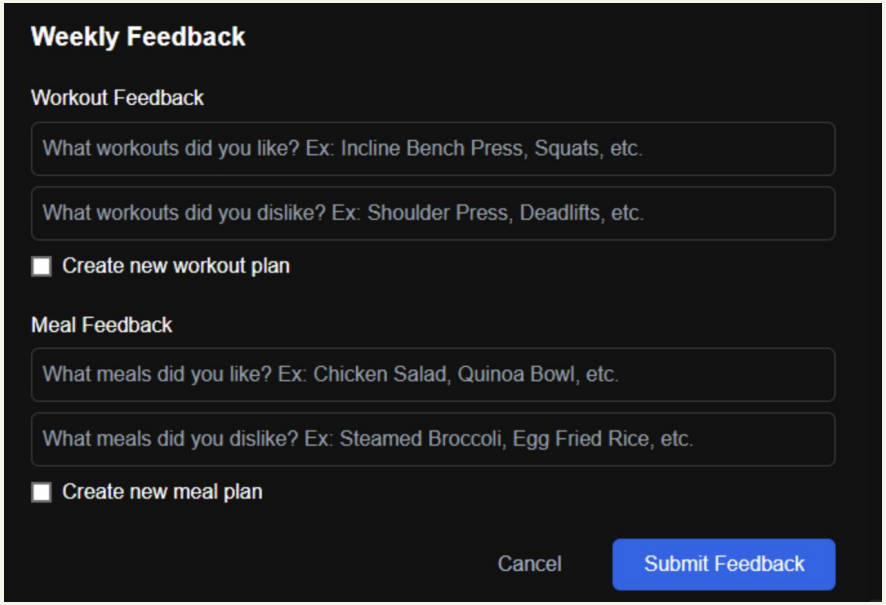
Step 3: parse the string into a dictionary

```
{
    "Monday": {
        "breakfast": "oatmeal with berries and almond butter",
        "lunch": "lentil curry with brown rice",
        "dinner": "quinoa stir-fry with tofu and vegetables  "
    },
    "Tuesday": {
        "breakfast": "chia pudding with coconut milk and fruit",
        "lunch": "chickpea salad with spinach and tahini dressing",
        "dinner": "black bean and sweet potato tacos  "
    },
    "Wednesday": {
        "breakfast": "smoothie with spinach, banana, and plant-based protein powder",
        "lunch": "red lentil soup with whole grain bread",
        "dinner": "veggie stir-fry with tempeh and quinoa  "
    },
    "Thursday": {
        "breakfast": "tofu scramble with vegetables",
        "lunch": "quinoa and black bean bowl with avocado",
        "dinner": "roasted vegetable and lentil stew  "
    },
```

```
{
    "Monday": [
        {
            "sets": 3,
            "reps": 12,
            "exercise": "bicep curl"
        },
        {
            "sets": 3,
            "reps": 10,
            "exercise": "triceps extension"
        },
        {
            "sets": 4,
            "reps": 15,
            "exercise": "dumbbell shoulder press"
        },
        {
            "sets": 3,
            "reps": 12,
            "exercise": "dumbbell lateral raise"
        }
    ],
    "Wednesday": [
```

Keyword extraction

Step 1: user decides they want a new plan → fills out feedback survey on last plan



Step 2: Query DeepSeek to extract keywords from each review (4 times) → returns keywords as a list

```
data = {
    "model": "deepseek/deepseek-chat:free",
    "messages": [{
        "role": "user",
        "content": (
            f"Extract the key words or phrases from the following text: '{text}'. "
            f"Focus on specific foods that they liked and adjectives describing the food. "
            f"Return the keywords as a comma-separated list, e.g., 'Indian, chicken, rice, easy'. "
            f"Only return the list, no extra text."
        )
    }]
}
```

Step 3: parse comma separated list → append keywords to the user's corresponding preferences list

What did you enjoy about the meal plan?

“I really liked that you included Indian cuisine, I love spicy food! The meals were also very easy to make which fit perfectly into my schedule”

→ [Indian cuisine, spicy food, easy to make]

↓

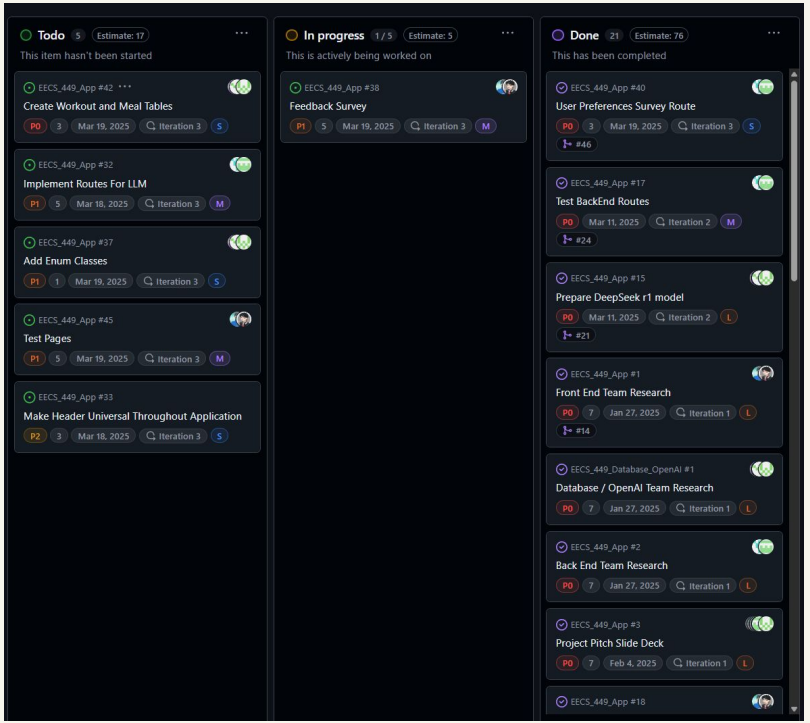
liked_meals: {.... + Indian cuisine, spicy food, easy to make}

SDK

GitHub

Cloud based version control and collaboration platform for software development built around Git.

Version Control with Git, Repositories, Branching and Merging, Issues & Project Management, GitHub Actions (CI/CD), Code Review & Collaboration, Security & Code Scanning, Hosting & Pages (Documentation)



Thank you!

Questions?