Design Document

Sophros

A Health Planning App

ECE 49595SD – Software Section – Fall 2025

Team 2

Emerson Maddock, emaddock@purdue.edu

Evan Stonehurst, estonest@purdue.edu

Janis Mikits, jmikits@purdue.edu

Eduard Tanase, etanase@purdue.edu

Table of Contents

1. Design Diagrams 3
2. Figma 3
3. Design Trade 3
4. Deliverables 3

1. Design Diagrams

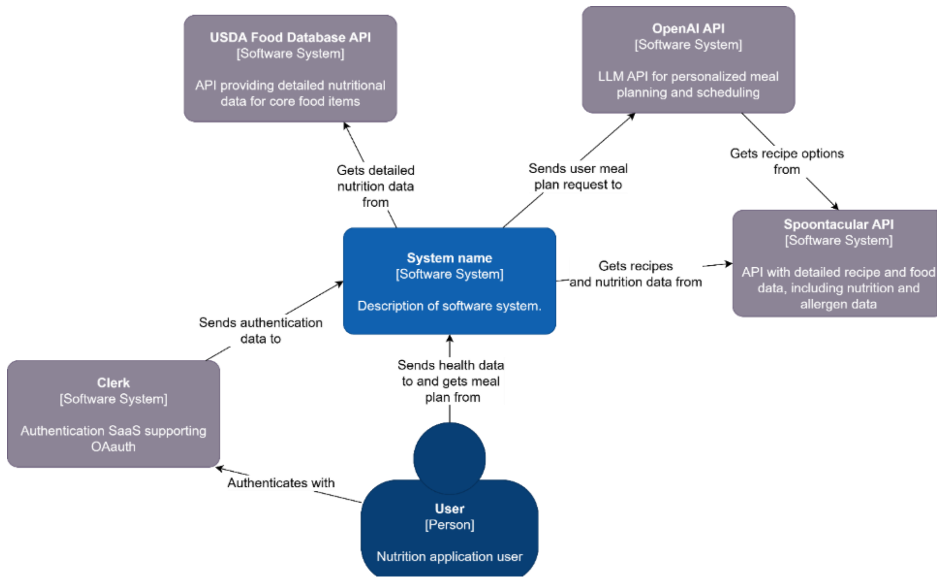
USDA Food Database API
[Software System]
API providing detailed nutritional data for core food items
OpenAI API
[Software System]
LLM API for personalized meal planning and scheduling
Gets detailed nutrition data from
Sends user meal plan request to
Gets recipe options from
Spoontacular API
[Software System]
API with detailed recipe and food data, including nutrition and allergen data
System name
[Software System]
Description of software system.
Gets recipes and nutrition data from
Sends authentication data to
Clerk
[Software System]
Authentication SaaS supporting OAauth
Sends health data to and gets meal plan from
Authenticates with
User
[Person]
Nutrition application user

Figure 1: System Context Diagram
This diagram shows the overall structure of the application and its high-level interactions.

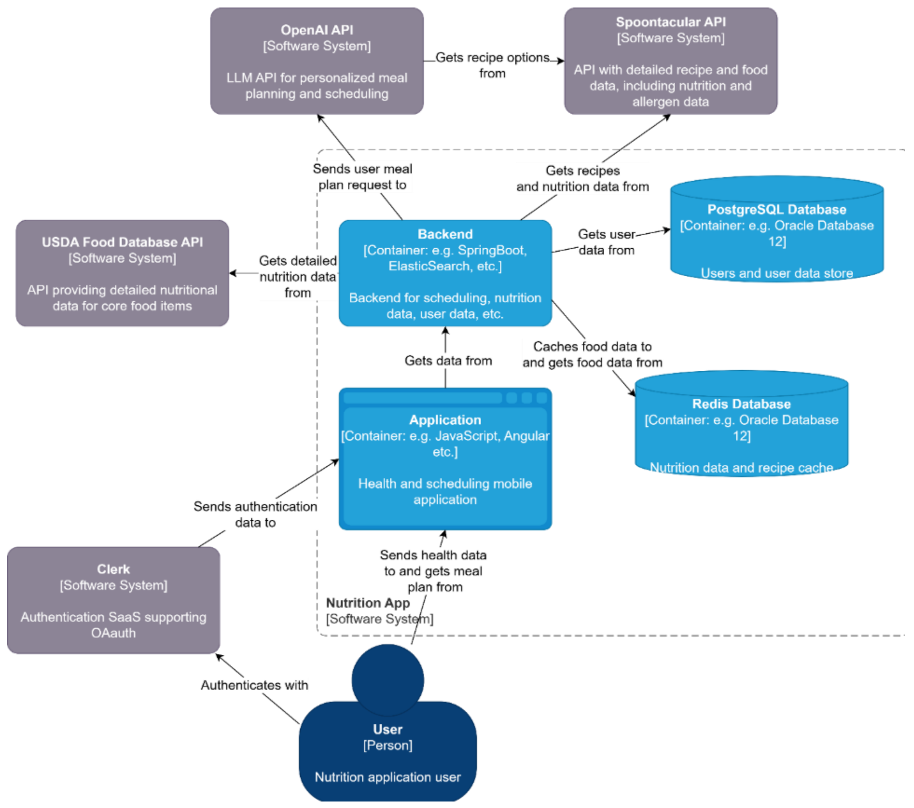
OpenAI API
[Software System]
LLM API for personalized meal planning and scheduling
Gets recipe options from
Spoontacular API
[Software System]
API with detailed recipe and food data, including nutrition and allergen data
Sends user meal plan request to
Gets recipes and nutrition data from
PostgreSQL Database
[Container: e.g. Oracle Database 12]
Users and user data store
USDA Food Database API
[Software System]
API providing detailed nutritional data for core food items
Gets detailed nutrition data from
Backend
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Backend for scheduling, nutrition data, user data, etc.
Gets user data from
Gets data from
Caches food data to and gets food data from
Redis Database
[Container: e.g. Oracle Database 12]
Nutrition data and recipe cache
Application
[Container: e.g. JavaScript, Angular etc.]
Health and scheduling mobile application
Sends authentication data to
Clerk
[Software System]
Authentication SaaS supporting OAauth
Sends health data to and gets meal plan from
Nutrition App
[Software System]
Authenticates with
User
[Person]
Nutrition application user

Figure 2: Container Diagram
This diagram shows the low-level interactions of the application.

Technology Stack

Tool	System Scope	Reasoning
React Native	Application	JavaScript frontend mobile development framework that uses native OS elements
Redux	Application	For maintaining app states in React Native
TypeScript	Application	JavaScript with strict type checking
FastAPI	Backend	Python backend framework
Python	Backend	Language used by FastAPI
Swagger	Backend	API documentation tool supported by FastAPI
Clerk	Shared	Authentication SaaS supporting modern OAuth
HealthKit	Application	iOS Health data which we will interact with
Google Fit	Application	Android Health data which we will interact with
OpenAI API	Backend	For LLM scheduling and recipes
AWS Lambda/API/	Backend	For backend hosting
PostgreSQL	Backend	Managed relational user database

2. Figma

High-fidelity page designs:
https://www.figma.com/design/T5cNSwi7W04vdAXEcK1yvS/CDR-Designs?node-id=0-1&t=6r5EUL4L1tPIEMCO-1

User flow brainstorms and designs:
https://www.figma.com/design/OKTwg3XvMwLDMTjiN9vzRr/Brainstorms?node-id=0-1&t=UKdwMxceDH5xChry-1

3. Design Trade

System Element 1: User Profile

Design Alternative Summaries

1. Structured Profile with Fixed Attributes: Users will fill out predefined fields during onboarding (which they can edit later). The backend will store one profile row per user. A deterministic function will compute calorie and macro targets
2. Adaptive, Goal-oriented Profile: User sets high-level goals (i.e. “cut fat”). Profile state will be continuously updated using behavior signals from user data. Target, nudges, and meal suggestions will adapt over time.

3. Modular Multi-profile System: Users can own multiple profiles/contexts, where each profile maintains its own targets, constraints, and analytics. A registry will mark the active profile, switching update recommendations as required.

Evaluation Table:

Design ID	UI Ease of Interpretation [20%]	Data Relevance [40%]	Data Security [30%]	User Database Scalability [10%]	Weighted Sum [%/1000]
UP1	7 | 140	5 | 200	7 | 210	9 | 90	640
UP2	8 | 160	9 | 360	5 | 150	7 | 70	740
UP3	4 | 80	8 | 320	4 | 120	5 | 50	570

Conclusion: Our final choice is the UP2 design (the adaptive, goal-oriented profile). It combines all the core components of our solution, allowing users to input both their data and their fitness goals. The dynamic nature of this design allows for personalized insights that update as the user logs their progress. This is reflected in the design score, which shows high ease of interpretation and data relevance and moderate potential for scalability. UP2 outperforms the other designs in this regard, as UP3 struggles with UI interpretation and UP1 with data relevance.

System Element 2: Nutrition Algorithm

Design Alternative Summaries

1. Constraint Satisfaction: Meal nutrient intake will be computed through a linear constraint satisfaction regression. Taking in a variety of features from the user data, we maximize the nutritional value.
2. Neural Network: Meal nutrient intake is determined by a trained Feed-Forward Neural Network. We train a model with a variety of meal plans, user information, and nutrition scores and have it predict a meal set with a high nutrition score from a set of user information.
3. Genetic Mixed Model: The nutrient value of a day is broken up over the course of the day using a linear model. A neural network is used to limit the recipe search space for the regression, repeatedly generating meal suggestions which we calculate the nutrition score of linearly. We then genetically tweak the inputs into the neural network to arrive at more optimal meal suggestions.

Evaluation Table:

Design ID	Accuracy [40%]	Fault-Tolerance [30%]	Scalability [20%]	Training Ease [10%]	Weighted Sum [%/1000]
NA1	8 | 320	7 | 210	2 | 40	10 | 100	670
NA2	6 | 240	4 | 120	10 | 200	3 | 30	590
NA3	9| 360	8 | 240	7 | 140	6 | 60	800

Conclusion: Our final choice is the NA3 design, a mixed genetic model. It takes the scalability of neural networks and gives it accuracy of a linear regression system. It is also more fault-tolerant than both systems since it has a multi-model guardrail that limits outliers determining the final result. This is reflected in the design score, which shows high accuracy and fault-tolerance with acceptable scalability and training requirements. NA3 outperforms the other designs in this mix, where the computationally intense linear model is accurate and fault tolerant, but unable to scale and the machine learning model lacks the reliability for this core health feature.

System Element 3: User Database

Design Alternative Summaries

1. AWS RDS PostgreSQL: This design uses AWS's RDS PostgreSQL managed relational database, an industry standard for scalable and reliable databases, for persistent data storage connected via FastAPI.
2. Neon: This design uses Neon's serverless PostgreSQL SaaS relational database for persistent data storage, also connected via FastAPI to our backend.
3. Supabase: This design uses Supabase's PosgreSQL SaaS database for persistent data storage similarly to Neon.

Evaluation Table:

Criterion (weight)	D1: AWS RDS	D2: Neon	D3: Supabase

Consistency (20)	10	10	10
Security (15)	9	9	8
Scalability (15)	8	9	8
Operability (10)	9	9	9
Query Flexibility (10)	10	10	10
Latency (10)	8	8	8
Cost (10)	7	9	8
Integration (10)	9	8	7
Total	885	910	860

Conclusion: Our final choice uses Neon as our database provider. All databases preserve SQL/ACID therefore receive 10 on consistency and query flexibility. Neon is the most scalable, and while Supabase and Neon both have good DX, Supabase's auth/API's overlap with existing requirements. Although RDS fits an AWS stack the best, it falls behind in other categories.

System Element 4: UI/UX Menu Layout

Design Alternative Summaries

1. Bottom Navigation Bar: At the bottom of the screen, indexing a home page, a detailed schedule page, a progress page, and a profile page. This classic layout makes it easy to navigate between a small set of core pages and presents these pages as relatively equally important.
2. Integrated Buttons: The pages could be remade so that the home page was the undisputed primary page with integrated links to the other pages. The profile could be accessed through an icon on the top right of the page, as well as another button to access the progress page next to it. This would end up looking similar to Instagram's messaging and likes page, where they have buttons at the top of the screen that disappear when you scroll down. Items that require the schedule could link to the full schedule view.
3. Mono-page: Instead of breaking down the information across multiple pages, we could have all the information on the home page, visually breaking down the page into segments that could be expanded by clicking on them. Reframing the profile page to focus more on your measurements could make this more intuitive. Different quick-infomation boxes on the home page expand to detailed schedule or social information after clicking on them.

Evaluation Table:

Design ID	Clarity [40%]	Functionality [30%]	Expansion Ease [20%]	Implementation Ease [10%]	Weighted Sum [%/1000]
UI1	9 | 360	8 | 240	6 | 120	7 | 70	790

UI2	7 | 280	7 | 240	5 | 100	7 | 70	690
UI3	4| 160	5 | 150	7 | 140	10 | 100	550

Conclusion: Our final choice is the UI1 design, implementing a bottom navigation bar that allows movement between the four main pages of the application. This solution is the clearest solution to show the user the main areas of the application. The other alternatives lacked the significant clarity to compete with this tried-and-tested UI solution. UI2, buttons at the top of the page, is also a reasonable strategy, but it did not perform as well due to the inherently less fixed nature of these buttons. Incorporating these disappearing type buttons when further pages are required could be an effective way to expand the application without compromising screen space and distracting the user. UI3 presents an interesting model of expansion, but this model is better suited to minor pages and information display instead of major pages. In conclusion, UI1 best fits the purpose of the pages, which is to be near-equal strength components of the app to the home page, compared to the other design styles.