

# SWEN - Nutrition Manager App

Phase II

---

Esmari Louw, Dora Pehar-Ljoljic, Natalie Frank

# Overview:

**Goal:** Build a system to help users log their food intake, manage calorie limits, add exercises and track daily progress with graphical visuals.

## Key Features:

- Add and remove food items (basic food or recipes).
- View daily log of food intake.
- Set and track calorie limits and weight
- Add and remove exercises
- Display through GUI for users to view.

# Features:

- **Food Collection:**
  - Users can add basic food items and recipes.
  - Food data is saved in `foods.csv`
- **Daily Log:**
  - Log food consumption for each day.
  - Adjust and track daily calorie limits and weight in `log.csv`.
- **User Interaction:**
  - Command-line interface to interact with the program.
  - Menu-driven options for logging and viewing data
- **Exercises:**
  - Users can add exercises
  - Exercise data is saved in `exercise.csv`

# Structure:

## Class Overview

**Food & Subclasses:** Food is an abstract class. FoodItem stores basic nutrition info, while Recipe combines items.

**FoodCollection:** Manages Food Objects (items & recipes) and handles loading/saving from foods.csv

**DailyLog:** tracks daily food, exercise, weight, and calorie limits. Saves to log.csv

**Exercise & ExerciseManager:** Represents physical activities and manages exercise data.

**Model-View-Controller:** Model = holds app data, Controller = logic and updates handler, MainView, LogView, FoodView from GUI.

**App Entry:** Main and NutritionTrackerApp initialize and launch the application.

## Data Flow:

**User Input** → Add food item or log food/recipe. Log exercise and set weight and calories

**GUI** → User interacts through MainView, LogView and FoodView. Buttons and dropdowns, date pickers, text fields.

**Controller** → Handles user input events. Updates Model and triggers View refresh.

**Model** → Holds the current selected Date. Interfaces with DailyLog & FoodCollection.

**Nutrition Graph Update** → View updated with macros: Fat %, carbs %, Protein %

```

classDiagram
    class Main {
        main(args: String[]) void
    }
    class NutritionTrackerApp {
        Main: Main
    }
    class Controller {
        <<MVC: Controller>>
        model: Model
        mainView: MainView
        logView: LogView
        + initialize() void
        + loadDates(date: Date) void
        + handleDateChange(newDate: Date) void
        + refreshViews() void
        + updateNutritionGraph(hist: int, carbs: int, protein: int) void
    }
    class MainView {
        <<MVC: View>>
        - model: Model
        - controller: Controller
        + display() void
        + setController(controller: Controller) void
        + updateNutritionGraph(hist: int, carbs: int, protein: int) void
    }
    class LogView {
        - model: Model
        + updateLogDisplay() void
    }
    class Model {
        <<MVC: Model>>
        - histLog: DailyLog
        - currentDate: Date
        + setCurrentDate(date: Date) void
        + get(DailyLog): DailyLog
    }
    class DailyLog {
        logEntries: Map<Date, List<Food>>
        exerciseEntries: Map<Date, List<Exercise>>
        caloriesConsumed: Map<Date, Integer>
        weightLog: Map<Date, Double>
        foodCollection: FoodCollection
        + addFood(date: Date, food: Food) void
        + addExercise(date: Date, exercise: Exercise) void
        + changeDate(date: Date, newLimit: int, newWeight: double) void
        + accessHist(date: Date) void
        + deleteLog(date: Date, foodName: String) void
    }
    class View {
        + display() void
        + setController(controller: Controller) void
    }
    class ExerciseManager {
        exerciseMap: Map<String, Exercise>
        + loadFromFile() void
        + saveToFile() void
        + addExercise(name: String, caloriesPerHour: double, broken: boolean) void
        + getExercises(): Collection<Exercise>
        + getExercise(name: String): Exercise
    }
    class Exercise {
        name: String
        caloriesPerHour: double
        duration: int
        + getEnergy(): String
        + getCaloriesPerHour() double
        + getDuration() int
    }
    class FoodCollection {
        foodMap: Map<String, Food>
        + addFood(food: Food) void
        + getFood(name: String): Food
        + getMFoods(): Collection<Food>
    }
    class FoodItem {
        <<Composite: Leaf>>
        name: String
        nutrition: SimpleNutrition
        + display() void
    }
    class FoodDetailsDialog {
        + displayFoodDetails(food: Food) void
    }
    class Recipe {
        <<Composite: Composite>>
        ingredients: List<Food>
        + addIngredient(food: Food) void
        + getNutrition(nutrition: String) double
        + display() void
    }
    class FoodView {
        + displayFood(hist: Food, food: Food) void
    }
    Main --> NutritionTrackerApp
    NutritionTrackerApp --> Controller
    Controller --> MainView
    Controller --> LogView
    Controller --> Model
    MainView --> Controller
    MainView --> View
    LogView --> Controller
    Model --> Controller
    Model --> DailyLog
    Model --> View
    DailyLog --> Controller
    DailyLog --> FoodCollection
    DailyLog --> FoodItem
    DailyLog --> Recipe
    FoodCollection --> Controller
    FoodCollection --> FoodItem
    FoodCollection --> FoodDetailsDialog
    FoodItem --> Controller
    FoodItem --> FoodView
    FoodDetailsDialog --> Controller
    FoodDetailsDialog --> FoodView
    Recipe --> Controller
    Recipe --> FoodView
    ExerciseManager --> Controller
    ExerciseManager --> Exercise
    Exercise --> Controller
    Exercise --> FoodCollection
    Exercise --> FoodItem
    Exercise --> Recipe
    
```

# Design Patterns Used:

1. MVC Pattern - separates concerns between model (Daily Log & Food Collection), View (all GUI classes - FoodView, LogView, MainView), and Controller
2. Composite Pattern - used in FoodCollection.java to allow treating individual FoodItems and Recipes uniformly
3. Adapter Pattern - To allow Recipe objects and FoodItem objects to be treated in the same way, even though they have different interfaces
4. Observer Pattern - the Controller manually notifies views, and the views update dynamically when the Model changes
5. Facade Pattern - MainView.java acts as a facade by providing a simplified interface to manage and coordinate multiple complex subsystems

# GUI Implementation

Nutrition Tracker

Select Date: 2025-04-29

Food ManagementDaily Log

Apple[Basic Food]

Milk[Basic Food]

Apple Pie[Recipe]

Add Basic Food

Add Recipe

View Details

Delete

Nutrition Tracker

Select Date: 2025-04-29

Food ManagementDaily Log

Total Calories: 56,0 / 2000 (2,8%)

Nutrition Breakdown: 38% Fat | 47% Carbs | 15% Protein

Food	Servings	Calories	Fat (g)	Carbs (g)	Protein (g)
Apple	1,0	12,0	22,0	21,0	11,0
Milk	2,0	44,0	86,0	88,0	22,0

Exercise Log

Exercise	Minutes	Calories Burned
Swimming	12,0	6,0

Exercise: Swimming + Minutes: 

Add ExerciseRemove Selected

Add Food: Milk Servings: 

Add to LogRemove Selected

Weight (lbs): 150,0 Calorie Limit: 2000 

Update Settings

Nutrition Breakdown

FatCarbsProtein

# Challenges:

## Extending Functionality :

- Integrating exercise logging alongside food entries
- Expanding support for recipes and serving sizes.

## Design Pattern Implementation: :

- Applying the patterns to unify food items and recipes along with the MVC and total access to full logs.
- Strengthening MVC architecture to support dynamic updates.

## Data Persistence :

- Updating log.csv to store both exercise and servings.
- Handling edge cases when reading malformed or incomplete entries.

## GUI Enhancements :

- Implementing nutrition graphs based on macro distribution
- Maintaining separation of concerns between Model and View.
- Ensuring real-time interaction for daily logs and goals.



# Conclusion

## Phase I Accomplishments:

- Successfully implemented core features: food logging, calorie limit management, weight tracking, and a GUI for easier interaction.
- Data is saved in CSV files, and the system allows users to track and update their daily logs efficiently.

## Phase II Accomplishments:

- Successfully implemented new features: exercise tracking, calorie goals, and weight logging
- Improvement of the Composite Pattern for recipes and strengthened MVC Pattern

## Next Steps:

- Complete the Bar Chart incorporation

# Thank you!

SWEN PROJECT: Phase II

Esmari Louw, Dora Phear-Ljoljic, Natalie Frank

30/04/2025