NutriLens AI - Complete System State & Decisions Documentation

Document Version: 1.0
Last Updated: October 6, 2025
Purpose: Single source of truth for current implementation state and all architectural decisions

---

📋 TABLE OF CONTENTS

🎯 CURRENT SYSTEM STATE

Backend Implementation Status

✅ FULLY IMPLEMENTED Components

1. Database Schema (`backend/app/models/database.py`)

python

```
# Core Tables
✅ users - id, email, hashed_password, is_active, created_at, last_login
✅ user_profiles - name, age, height, weight, sex, activity_level, bmr, tdee, goal_calories
✅ user_goals - goal_type, target_weight, target_date, macro_targets
✅ user_paths - path_type, meals_per_day, meal_windows (JSON)
✅ user_preferences - dietary_type, allergies, disliked_ingredients, cuisines
✅ items - canonical_name, aliases (JSON), category, nutrition_per_100g (JSON)
✅ recipes - title, macros_per_serving (JSON), ingredients, goals (JSON)
✅ recipe_ingredients - recipe_id, item_id, quantity_grams
✅ meal_plans - user_id, plan_data (JSON), grocery_list (JSON), is_active
✅ meal_logs - user_id, recipe_id, meal_type, planned/consumed_datetime, was_skipped
✅ user_inventory - user_id, item_id, quantity_grams, expiry_date
✅ receipt_uploads - file_url, ocr_raw_text, parsed_items (JSON)
✅ agent_interactions - agent_type, interaction_type, input/response_text
```

2. Authentication System (`backend/app/`)

- [x] POST /auth/register - Create new user account
- [x] POST /auth/login - Login with email/password, returns JWT
- [x] POST /auth/refresh - Refresh access token
- [x] GET /auth/me - Get current user info
- [x] JWT token generation with expiry
- [x] Password hashing with bcrypt
- [x] OAuth2 password bearer authentication

3. Onboarding System (`backend/app/api/onboarding.py`, `backend/app/services/onboarding.py`)

- [x] POST /onboarding/basic-info - Save profile (age, height, weight, etc.)
- [x] POST /onboarding/goal-selection - Set goal and calculate macro targets
- [x] POST /onboarding/path-selection - Set eating path and meal windows
- [x] POST /onboarding/preferences - Set dietary preferences
- [x] GET /onboarding/calculated-targets - Get BMR, TDEE, goal calories

Service Functions:

- [x] calculate_bmr() - Mifflin-St Jeor equation
- [x] calculate_tdee() - BMR × activity multiplier
- [x] calculate_goal_calories() - TDEE ± goal adjustment
- [x] get_macro_targets() - Goal-based macro splits
- [x] get_meal_windows() - Path-based meal timing
- [x] complete_profile() - Save user profile with calculations
- [x] set_user_goal() - Save goal and update profile calories
- [x] set_user_path() - Save eating path and meal windows
- [x] set_user_preferences() - Save dietary preferences
- [x] get_calculated_targets() - Return all onboarding calculations

4. Meal Planning System (`backend/app/api/meal_plans.py`, `backend/app/agents/planning_agent.py`)

API Endpoints:

- ✅ POST /meal-plans/generate - Generate weekly meal plan
- ✅ GET /meal-plans/current - Get active meal plan
- ✅ GET /meal-plans/{id} - Get specific meal plan
- ✅ PUT /meal-plans/{id}/adjust - Update meal plan
- ✅ POST /meal-plans/{id}/swap-meal - Swap a meal with alternative
- ✅ GET /meal-plans/{id}/alternatives - Get recipe alternatives
- ✅ GET /meal-plans/{id}/grocery-list - Get grocery list
- ✅ POST /meal-plans/eating-out - Adjust plan for eating out
- ✅ POST /meal-plans/meal-prep-suggestions - Get prep suggestions
- ✅ POST /meal-plans/bulk-cooking-suggestions - Get bulk cooking tips
- ✅ POST /meal-plans/shopping-reminders - Get shopping reminders
- ✅ GET /meal-plans/history/meals - Get meal history

Planning Agent Tools:

- ✅ generate_weekly_meal_plan() - LP optimizer-based plan generation
- ✅ select_recipes_for_goal() - Goal-aligned recipe selection
- ✅ calculate_grocery_list() - Aggregate ingredients from plan
- ✅ find_recipe_alternatives() - Find similar macro alternatives
- ✅ suggest_meal_prep() - Weekend prep suggestions
- ✅ bulk_cooking_suggestions() - Batch cooking recommendations
- ✅ generate_shopping_reminders() - Smart shopping alerts
- ⚠️ _categorize_grocery_list() - RETURNS MOCK DATA (needs fix)
- ⚠️ _get_user_meal_windows() - RETURNS HARDCODED TIMES (needs fix)
- ⚠️ _get_storage_tip() - HAS 3 GENERIC TIPS (needs fix)

5. Tracking System (`backend/app/api/tracking.py`, `backend/app/agents/tracking_agent.py`)

API Endpoints:

- ✅ POST /tracking/log-meal - Log meal consumption with inventory deduction
- ✅ POST /tracking/skip-meal - Mark meal as skipped
- ✅ POST /tracking/update-inventory - Bulk inventory updates
- ✅ POST /tracking/manual-entry - Log off-plan food
- ✅ GET /tracking/today - Today's consumption summary
- ✅ GET /tracking/history - Historical consumption (7-90 days)
- ✅ GET /tracking/patterns - Consumption pattern analysis
- ✅ GET /tracking/inventory-status - Current inventory overview
- ✅ GET /tracking/expiring-items - Items expiring soon
- ✅ GET /tracking/restock-list - Low stock recommendations

Tracking Agent Tools:

- ✅ log_meal_consumption() - Record eaten meals, deduct inventory
- ✅ get_today_summary() - Daily macro totals and progress
- ✅ get_consumption_history() - Past meal logs with statistics
- ✅ analyze_patterns() - Meal timing, skip reasons, compliance
- ✅ calculate_macro_breakdown() - Detailed macro analysis
- ✅ calculate_inventory_status() - Categorize inventory by stock level
- ✅ check_expiring_items() - Find expiring items with recipe suggestions
- ✅ generate_restock_list() - Priority-ordered shopping list

6. Consumption Service (`backend/app/services/consumption_services.py`)

- ✅ log_meal_consumption() - Main meal logging logic
- ✅ get_today_summary() - Aggregate today's consumption
- ✅ get_consumption_history() - Historical data with trends
- ✅ get_macro_breakdown() - Detailed macro distribution
- ✅ generate_consumption_analytics() - Pattern insights
- ✅ _calculate_meal_macros() - Macro calculation from recipe
- ✅ _calculate_remaining_targets() - Remaining macros for the day
- ✅ _analyze_trends() - Compliance and timing trends
- ✅ _calculate_time_consistency() - Meal timing variance
- ✅ _get_common_skip_reasons() - Skip pattern analysis

7. Inventory Service (`backend/app/services/inventory_service.py`)

- ✅ add_item() - Add items to inventory
- ✅ remove_item() - Remove items from inventory
- ✅ update_quantity() - Adjust item quantities
- ✅ get_inventory() - List all user inventory
- ✅ check_expiring() - Find items expiring within X days
- ✅ get_low_stock() - Find items below threshold
- ✅ normalize_item() - Fuzzy matching and alias resolution
- ✅ deduct_from_recipe() - Deduct recipe ingredients from inventory

8. Notification System (`backend/app/services/notification_service.py`, `backend/app/workers/notification_worker.py`)

- ✅ queue_notification() - Add notification to Redis queue
- ✅ process_queue() - Process queued notifications
- ✅ send_email() - Email delivery via SMTP
- ✅ send_sms() - SMS via Twilio
- ✅ send_whatsapp() - WhatsApp via Twilio
- ✅ send_push() - Push notifications
- ✅ get_user_notifications() - Retrieve notifications
- ✅ mark_as_read() - Update notification status
- ✅ get_preferences() - Get notification preferences
- ✅ update_preferences() - Update notification preferences
- ✅ Worker process - Background job for scheduled notifications

9. WebSocket System (`backend/app/services/websocket_manager.py`, `backend/app/api/websocket.py`)

- ✅ WS /ws/{user_id} - WebSocket connection endpoint
- ✅ Connection management - Multi-device support
- ✅ Heartbeat/keepalive - Connection health monitoring
- ✅ Redis pub/sub - Horizontal scaling support
- ✅ Broadcasting - User-specific and system-wide
- ✅ Real-time updates - Meal logging, macro updates, achievements

10. LP Meal Optimizer (`backend/app/services/final_meal_optimizer.py`)

- ✅ PuLP linear programming optimization
- ✅ Constraint satisfaction (calories, macros, variety)
- ✅ Inventory prioritization
- ✅ Meal timing constraints
- ✅ Fallback greedy algorithm
- ✅ Performance: <5 seconds for 7-day plan

❌ NOT IMPLEMENTED Components

1. WhatsApp Agent

- Mentioned in planning documents
- Database table exists (whatsapp_logs)
- NO CODE IMPLEMENTATION FOUND
- Status: Planned but not built

2. Nutrition Agent

- Critically needed by other components
- Required for:
  - Meal suggestions
  - Nutritional analysis
  - Progress insights
  - AI coaching
- Status: Day 7 work NOT DONE

3. Frontend Application

- NO PAGES IMPLEMENTED
- Only planning documents exist
- Required pages:
  - Dashboard (entry point)
  - Onboarding wizard
  - Meal planner
  - Inventory manager
  - Progress/analytics
  - Settings
- Status: 0% complete

4. Email Verification

- Registration exists but no email verification
- No verification token system
- No verification endpoints
- Status: Planned but not implemented

⚠️ ISSUES IN EXISTING CODE

Planning Agent Mock Implementations:


python

```python
# backend/app/agents/planning_agent.py

def _categorize_grocery_list(self, grocery_list):
    # ISSUE: Returns mock data instead of real categorization
    return {"other": grocery_list}  # ← Should categorize by food groups

def _get_user_meal_windows(self, user_id):
    # ISSUE: Returns hardcoded times instead of user preferences
    return [
        {"meal": "breakfast", "start": "08:00", "end": "09:00"},
        {"meal": "lunch", "start": "13:00", "end": "14:00"},
        {"meal": "dinner", "start": "19:00", "end": "20:00"}
    ]  # ← Should query UserPath.meal_windows from database

def _get_storage_tip(self, item_name):
    # ISSUE: Has 3 generic tips instead of food-specific advice
    tips = [
        "Store in a cool, dry place",
        "Keep refrigerated after opening",
        "Use within 3-5 days for best quality"
    ]
    return tips[hash(item_name) % 3]  # ← Should have item-specific tips
```

Endpoint Redundancy:

```
DUPLICATE/MISPLACED ENDPOINTS:
❌ POST /meal-plans/log-meal - Should be in /tracking (already exists there)
❌ GET /meal-plans/history/meals - Should be in /tracking (already exists there)
❌ POST /meal-plans/eating-out - Partially tracking, needs migration
```

🎯 JOURNEY 1: REGISTRATION & ONBOARDING

Current Implementation

What Works:

1. ✅ User can register with email/password
2. ✅ User can login and receive JWT token
3. ✅ User can complete all 4 onboarding steps
4. ✅ BMR, TDEE, goal calories are calculated
5. ✅ Macro targets are set based on goal

6. ✅ Meal windows are set based on eating path

Critical Gaps:

1. ❌ No field in User model to track onboarding completion
2. ❌ No logic to detect if user needs onboarding after login
3. ❌ No redirect logic after onboarding completion
4. ❌ No resume capability for interrupted onboarding
5. ❌ No email verification system

Decisions Made

Decision 1.1: Onboarding State Tracking

DECISION: Add comprehensive onboarding tracking to User model

Implementation:

python

```python
# backend/app/models/database.py - User model additions

class User(Base):
    # ... existing fields ...

    # Onboarding tracking
    onboarding_completed = Column(Boolean, default=False)
    onboarding_current_step = Column(Integer, default=1)
    basic_info_completed = Column(Boolean, default=False)
    goal_selection_completed = Column(Boolean, default=False)
    path_selection_completed = Column(Boolean, default=False)
    preferences_completed = Column(Boolean, default=False)
    onboarding_started_at = Column(DateTime, nullable=True)
    onboarding_completed_at = Column(DateTime, nullable=True)
```

Rationale:

- Industry standard approach (LinkedIn, Twitter, Stripe use this)
- Allows resume from any step
- Provides progress tracking
- Enables analytics on onboarding completion rates

Decision 1.2: API Response Format

DECISION: Use standard REST response format

Success Response:


json

```
{
  "success": true,
  "data": { /* actual data */ },
  "message": "Operation successful"
}
```

Error Response:


json

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "User-friendly message"
  }
}
```

Onboarding Step Response:


json

```
{
  "success": true,
  "data": {
    "user_id": 123,
    "profile": { /* saved data */ }
  },
  "message": "Basic information saved successfully",
  "next_step": "/onboarding/goal-selection"
}
```

Rationale:

- Used by Auth0, Firebase, Stripe, most modern APIs
- Consistent across all endpoints
- Easy to parse on frontend
- Clear success/failure indication

Decision 1.3: Login Flow

DECISION: Separate auth token from user state (OAuth2 standard pattern)

Step 1 - Login:

```
POST /auth/login
Response:
{
 "access_token": "eyJhbGc...",
 "token_type": "bearer",
 "expires_in": 3600,
 "user": {
  "id": 123,
  "email": "user@example.com"
 }
}
```

Step 2 - Get User State:

```
GET /auth/me (with Bearer token)
Response:
{
 "success": true,
 "data": {
  "id": 123,
  "email": "user@example.com",
  "onboarding_completed": false,
  "onboarding_current_step": 3,
  "redirect_to": "/onboarding/path-selection"
 }
}
```

Rationale:

- OAuth2/JWT industry standard (Google, GitHub, Stripe)
- Separates authentication from application state
- Allows frontend to make decisions
- Clean architecture separation

Decision 1.4: Step Order Enforcement

DECISION: Strict linear flow with backend validation

Implementation Pattern:

python

```python
@router.post("/onboarding/goal-selection")
def select_goal(data, user, db):
    # Validate prerequisite
    if not user.basic_info_completed:
        raise HTTPException(
            status_code=400,
            detail={
                "code": "PREREQUISITE_NOT_MET",
                "message": "Please complete basic information first",
                "redirect_to": "/onboarding/basic-info"
            }
        )
    # ... save goal data ...
    user.goal_selection_completed = True
    user.onboarding_current_step = 3
    db.commit()

    return {
        "success": true,
        "message": "Goal set successfully",
        "redirect_to": "/onboarding/path-selection"
    }
```

Frontend Implementation:

- Step 2 button DISABLED until step 1 complete
- URL navigation validated by backend
- No manual step skipping allowed

Rationale:

- LinkedIn/Twitter/Slack onboarding pattern
- Data consistency (can't set goal without profile)
- Better user experience (guided flow)
- Prevents data corruption

Decision 1.5: Navigation Control

DECISION: Backend directs navigation flow

Backend Returns Explicit Instructions:

json

```
{
 "success": true,
 "data": { /* ... */ },
 "redirect_to": "/dashboard"  // Frontend follows
}
```

Rationale:

- Shopify, Notion, Slack use this pattern
- Backend has full business logic context
- Easier to change flow without frontend updates
- Consistent across web and mobile

Decision 1.6: Post-Onboarding Behavior

DECISION:

- User lands on Entry Point Dashboard
- NO automatic meal plan generation
- User chooses when to generate first plan

Flow:

```
User completes Step 4 (preferences)
→ All flags set to true
→ onboarding_completed = true
→ onboarding_completed_at = now()
→ Response: { "redirect_to": "/dashboard" }
→ Dashboard shows "Generate Your First Meal Plan" CTA
→ User clicks when ready
→ Plan generated on demand
```

Rationale:

- Meal plan generation takes 3-5 seconds
- User should control timing
- Sets clear expectations
- Prevents timeout issues
- Better user experience (no waiting)

Decision 1.7: Post-Onboarding Edits

DECISION: Full edit access + reset option

User Settings Page:

```
Settings
├── Profile
│   ├── Edit Basic Info (individual fields)
│   ├── Edit Goals (change anytime, recalculates macros)
│   ├── Edit Eating Path (change anytime, updates meal windows)
│   └── Edit Preferences (cuisines, restrictions, etc.)
└── Advanced
    └── Reset Profile (wipe data, restart onboarding)
```

API Endpoints:

```
PUT /user/profile/basic-info
PUT /user/profile/goals
PUT /user/profile/path
PUT /user/profile/preferences
POST /user/profile/reset
```

Rationale:

- Fitbit, MyFitnessPal, Noom pattern
- Users' needs change over time
- Edit anything individually OR
- Reset everything and start over
- Flexibility improves retention

Decision 1.8: Email Verification

DECISION: Add email verification post-launch (not blocker)

Future Implementation:

- Verify email after onboarding
- Allow full access while unverified
- Periodic reminder to verify
- No functionality restrictions

Rationale:

- Not critical for MVP launch
- Most modern apps allow usage before verification
- Can be added incrementally
- Focus on core functionality first

Implementation Checklist - Journey 1

Phase 1: Database Schema Update

- [ ] Add onboarding tracking fields to User model
- [ ] Create database migration
- [ ] Run migration on dev database
- [ ] Verify fields exist

Phase 2: Update Onboarding Endpoints

- [ ] Update POST /onboarding/basic-info
  - Set basic_info_completed = True
  - Set onboarding_current_step = 2
  - Set onboarding_started_at if first step
  - Return next_step in response

- [ ] Update POST /onboarding/goal-selection
  - Validate basic_info_completed = True
  - Set goal_selection_completed = True
  - Set onboarding_current_step = 3
  - Return next_step in response

- [ ] Update POST /onboarding/path-selection
  - Validate goal_selection_completed = True
  - Set path_selection_completed = True
  - Set onboarding_current_step = 4
  - Return next_step in response

- [ ] Update POST /onboarding/preferences
  - Validate path_selection_completed = True
  - Set preferences_completed = True
  - Set onboarding_completed = True
  - Set onboarding_completed_at = now()
  - Return redirect_to: "/dashboard"

Phase 3: Update Auth Endpoints

[ ] Update GET /auth/me
- Return onboarding_completed
- Return onboarding_current_step
- Return completed_steps array
- Calculate redirect_to based on state:
  * If onboarding_completed → "/dashboard"
  * If not completed → "/onboarding/{current_step_name}"

Phase 4: Create User Settings Endpoints

[ ] Create PUT /user/profile/basic-info
[ ] Create PUT /user/profile/goals (with macro recalculation)
[ ] Create PUT /user/profile/path (with meal window update)
[ ] Create PUT /user/profile/preferences
[ ] Create POST /user/profile/reset (with confirmation)

🎯 JOURNEY 2: ENTRY POINT DASHBOARD

Current State

Implementation Status: 0% - Nothing implemented

What Needs Building:

1. Frontend dashboard page
2. Dashboard component structure
3. API integrations
4. Real-time WebSocket connections
5. Navigation system

Decisions Made

Decision 2.1: Dashboard Entry Point

DECISION: Universal Home Dashboard (not direct to specific dashboard)

Dashboard Structure:

Home Dashboard (Entry Point after onboarding)

- Header
  - User avatar + name
  - Notifications icon (with count)
  - Settings icon
  - Navigation menu
- Quick Summary Cards (4 cards)
  - Today's Meals Card
    - "3 / 4 meals logged"
    - Next meal: "Dinner at 7 PM"
    - CTA: → Go to Meal Dashboard
  - Macros Progress Card
    - Circular progress bars (Calories, Protein, Carbs, Fats)
    - "68% of daily target"
    - CTA: → Go to Nutrition Dashboard
  - Inventory Status Card
    - "3 items expiring soon"
    - "2 items low stock"
    - CTA: → Go to Inventory Dashboard
  - Goal Progress Card
    - "2kg lost / 5kg target"
    - "7-day streak 🔥"
    - CTA: → Go to User Dashboard
- Quick Actions (4 buttons)
  - 🍽️ Log a Meal
  - 📋 Generate New Plan
  - 📦 Add to Inventory
  - 📊 View Progress
- AI Coach Panel
  - Daily insight/tip
  - "Great job! You're on track this week..."
  - CTA: Chat with AI
- Recent Activity Feed
  - "Breakfast logged - 08:15 AM"
  - "Achieved 7-day streak"

```
├── "Grocery list updated"
└── View All →
```

Rationale:

- Shows everything at a glance
- Clear entry points to all features
- Quick actions for common tasks
- AI coaching builds engagement
- Similar to Notion, Todoist, Asana dashboards

Decision 2.2: Navigation Pattern

DECISION: Persistent sidebar navigation

Navigation Structure:

```
Sidebar (Always Visible)
├── 🏠 Home (Entry Point Dashboard)
├── 👤 User Dashboard
├── 🍽️ Meal Dashboard
├── 📦 Inventory Dashboard
├── 📊 Nutrition Dashboard
├── ⚙️ Settings
└── 💬 AI Chat (floating button)
```

Rationale:

- Industry standard (Notion, Slack, Asana)
- Always accessible
- Clear hierarchy
- Mobile: Hamburger menu

Implementation Checklist - Journey 2

- [ ] Design entry point dashboard layout
- [ ] Create dashboard API endpoints
  - [ ] GET /dashboard/summary (aggregate all card data)
  - [ ] GET /dashboard/recent-activity
- [ ] Build frontend dashboard page
- [ ] Integrate WebSocket for real-time updates
- [ ] Create navigation component
- [ ] Add quick action handlers

🎯 JOURNEY 3: MEAL PLANNING

Current State

Implementation: 80% complete - Endpoints exist but need cleanup

Issues Identified

Issue 3.1: Endpoint Redundancy

DUPLICATE ENDPOINTS:

❌ POST /meal-plans/log-meal
- Should NOT exist in meal planning
- Already exists: POST /tracking/log-meal ✅
- Action: DELETE from meal planning API

❌ GET /meal-plans/history/meals
- Should NOT exist in meal planning
- Already exists: GET /tracking/history ✅
- Action: DELETE from meal planning API

Issue 3.2: Misplaced Functionality

"Eating Out" Feature:

```
Current: POST /meal-plans/eating-out
Problem: Partially tracking, partially planning
Decision: NEEDS ANALYSIS

When user says "eating out tonight":
Option A: Remove meal from plan (Planning concern)
Option B: Log as external meal (Tracking concern)
Option C: Both - adjust plan AND log differently

DECISION NEEDED: Which option?
```

Issue 3.3: Planning Agent Mock Methods

3 Methods Need Real Implementation:

python

```
⚠️ _categorize_grocery_list()
Current: Returns {"other": [...]}
Needed: Categorize by food groups
    - Proteins, Vegetables, Fruits
    - Grains, Dairy, Pantry, Other

⚠️ _get_user_meal_windows()
Current: Returns hardcoded times
Needed: Query UserPath.meal_windows from database

⚠️ _get_storage_tip()
Current: 3 generic tips
Needed: Item-specific storage advice database
```

Decisions Made

Decision 3.1: Planning vs Tracking Boundary

DECISION: Clear separation of responsibilities

Planning Agent Scope (Future-Oriented):

- ✅ Generate weekly meal plan
- ✅ Find recipe alternatives
- ✅ Calculate grocery lists
- ✅ Meal prep suggestions
- ✅ Bulk cooking suggestions
- ✅ Swap meals in plan
- ✅ Shopping reminders
- ✅ Optimize for inventory usage

Tracking Agent Scope (Present/Past-Oriented):

- ✅ Log meal consumption
- ✅ Get consumption history
- ✅ Analyze eating patterns
- ✅ Skip meal logging
- ✅ Daily/weekly summaries
- ✅ Adherence metrics
- ✅ Eating out adjustments (MOVE HERE)

Rationale:

- Clear mental model
- No overlapping responsibilities
- Easy to understand which endpoint to use
- Maintainable codebase

Decision 3.2: Eating Out Feature Placement

DECISION: Move to Tracking Agent

Why Tracking:

- User is CONSUMING food (present action)
- Needs to be logged in meal history
- Should affect today's macro totals
- Similar to logging any other meal

Implementation:

```
NEW: POST /tracking/eating-out
{
 "meal_type": "dinner",
 "restaurant_name": "Pizza Place",
 "estimated_calories": 800,
 "estimated_macros": {
  "protein_g": 30,
  "carbs_g": 90,
  "fat_g": 35
 },
 "notes": "Company dinner"
}

Backend:
1. Create meal_log with external_meal JSON
2. Update today's consumption totals
3. Mark planned meal as replaced
4. Broadcast WebSocket update
5. Return updated day summary
```

Decision 3.3: Duplicate Endpoint Cleanup

DECISION: Remove all duplicates from meal planning API

Endpoints to DELETE:

❌ POST /meal-plans/log-meal
❌ GET /meal-plans/history/meals

Keep in Tracking API:

✅ POST /tracking/log-meal
✅ GET /tracking/history
✅ POST /tracking/eating-out (new)

Decision 3.4: Unused Functions Audit

DECISION: Audit all tracking and consumption service functions

Process:

1. List ALL tracking agent tools → Mark which have endpoints
2. List ALL consumption service functions → Mark where they're used
3. For each function without endpoint OR usage:
    - Option A: Create endpoint (if valuable to expose)
    - Option B: Keep internal only (if used by other functions)
    - Option C: Delete (if truly unused)

This audit will be done separately after journey decisions are complete

Implementation Checklist - Journey 3

Phase 1: Fix Planning Agent Mock Methods

[ ] Fix _categorize_grocery_list()
- Create food category mapping
- Implement real categorization logic
- Test with various grocery lists

[ ] Fix _get_user_meal_windows()
- Query UserPath table for user_id
- Return actual meal_windows from database
- Handle case where user has no path set

[ ] Fix _get_storage_tip()
- Create storage tips database
- Map items to storage advice
- Implement lookup function

Phase 2: Move Eating Out Feature

[ ] Create POST /tracking/eating-out endpoint
[ ] Implement eating out logging in TrackingAgent
[ ] Update consumption totals calculation
[ ] Test with various scenarios
[ ] Delete POST /meal-plans/eating-out

Phase 3: Remove Duplicate Endpoints

- [ ] Delete POST /meal-plans/log-meal
- [ ] Delete GET /meal-plans/history/meals
- [ ] Update API documentation
- [ ] Update frontend to use /tracking endpoints

Phase 4: Function Audit (Separate Task)

- [ ] Audit tracking agent tools (8 tools)
- [ ] Audit consumption service functions (15+ functions)
- [ ] Document findings
- [ ] Create action items per function

🎯 JOURNEY 4: DASHBOARD ARCHITECTURE

Decisions Made

Decision 4.1: Dashboard Structure

DECISION: 4 Main Dashboards + 1 Entry Point

Dashboard Hierarchy:

- [ ]

🏠 Home Dashboard (Entry Point)
↓
├→ 👤 User Dashboard
├→ 🍽️ Meal Dashboard
├→ 📦 Inventory Dashboard
└→ 📊 Nutrition Dashboard

Individual Dashboard Specifications

Dashboard 1: User Dashboard 👤

Purpose: Profile, preferences, goals, settings, account management

Components:

```
User Dashboard
├── Profile Section
│   ├── Avatar + Name
│   ├── Email
│   ├── Age, Height, Weight, Activity Level
│   ├── BMR, TDEE, Goal Calories (display only)
│   └── Edit Profile button
│
├── Goals Section
│   ├── Current Goal (Weight Loss / Muscle Gain / etc.)
│   ├── Target Weight
│   ├── Target Date
│   ├── Progress Bar
│   ├── Macro Targets (Protein, Carbs, Fats %)
│   └── Edit Goals button
│
├── Eating Path Section
│   ├── Current Path (IF 16:8 / Traditional / etc.)
│   ├── Meals Per Day
│   ├── Meal Windows (visual timeline)
│   └── Change Path button
│
├── Preferences Section
│   ├── Dietary Type (Vegetarian / Non-Veg / etc.)
│   ├── Allergies List
│   ├── Disliked Ingredients
│   ├── Cuisine Preferences
│   └── Edit Preferences button
│
├── Account Activity
│   ├── Member Since
│   ├── Meal Plans Generated
│   ├── Meals Logged
│   ├── Current Streak
│   └── Achievements
│
└── Danger Zone
    └── Reset Profile (restart onboarding)
```

Key Features:

- Edit any section individually

- Visual representation of meal windows
- Progress toward goal weight
- Activity statistics

Dashboard 2: Meal Dashboard 🍽️ (MOST IMPORTANT)

Purpose: Meal planning, tracking, recipe browsing, meal history

Components:

Meal Dashboard
├── Tab 1: This Week's Plan
│ ├── 7-Day Calendar View
│ │ ├── Day Cards (Sun-Sat)
│ │ ├── Each Day Shows:
│ │ │ ├── Breakfast Card (recipe name, macros, thumbnail)
│ │ │ ├── Lunch Card
│ │ │ ├── Dinner Card
│ │ │ └── Snack Cards
│ │ └── Day Status (planned/logged/skipped)
│ │
│ ├── Actions
│ │ ├── Generate New Plan button
│ │ ├── Export Plan button
│ │ └── View Grocery List button
│ │
│ └── Current Meal Section (if within meal window)
│ ├── "Next: Dinner at 7 PM"
│ ├── Recipe card with details
│ ├── Quick Log Meal button
│ └── Swap Meal button
│
├── Tab 2: Today's Meals (Focused View)
│ ├── Meal Timeline (breakfast → lunch → dinner)
│ ├── Each Meal Shows:
│ │ ├── Recipe name + image
│ │ ├── Macros (protein, carbs, fats)
│ │ ├── Logged Status (✅ / ⏰ / ⭕)
│ │ ├── Time (planned vs actual)
│ │ └── Actions (Log / Skip / Swap)
│ │
│ ├── Today's Macro Progress
│ │ ├── Calories: 1486 / 2200 (68%)
│ │ ├── Protein: 124g / 165g (75%)
│ │ ├── Carbs: 148g / 220g (67%)
│ │ └── Fats: 52g / 73g (71%)
│ │
│ └── Quick Actions
│ ├── Log Meal button
│ ├── Log Eating Out button
│ └── Skip Meal button
│
├── Tab 3: Recipe Browser
│ ├── Search & Filters

Key Decisions:

- **Combine planning and tracking in one dashboard** (UX decision)
- Quick log meal action always visible
- Calendar + today view for different use cases
- Recipe browser integrated (no separate page)
- History for reflection and patterns

Dashboard 3: Inventory Dashboard 📦

Purpose: Pantry management, grocery lists, expiry tracking, receipt scanning

Components:

Inventory Dashboard
├── Alert Banner (if applicable)
│ ├── 🔴 3 Items Expiring Soon
│ ├── 🟡 2 Items Low Stock
│ └── Quick Actions (View / Generate Restock List)
│
├── Current Inventory Section
│ ├── Search & Filter Bar
│ │ ├── Search by name
│ │ ├── Filter by category
│ │ ├── Filter by stock level
│ │ └── Filter by expiry
│ │
│ ├── View Toggle (Grid / List)
│ │
│ ├── Item Display (Grid View)
│ │ ├── Item Cards
│ │ │ ├── Image/Icon
│ │ │ ├── Item Name
│ │ │ ├── Quantity (with bar)
│ │ │ ├── Expiry Date
│ │ │ ├── Days Remaining
│ │ │ ├── Status Indicator (🟢 / 🟡 / 🔴)
│ │ │ └── Quick Actions (+/-/Edit/Delete)
│ │ └── Categories: Proteins, Grains, Vegetables, etc.
│ │
│ └── Actions
│ ├── Add Item Manually
│ ├── Scan Receipt
│ └── Bulk Add
│
├── Restock List Section
│ ├── Generated Restock List
│ │ ├── High Priority (out of stock)
│ │ ├── Medium Priority (low stock)
│ │ └── Low Priority (getting low)
│ │
│ └── Actions
│ ├── Generate List button
│ ├── Export to Shopping App
│ └── Add to Grocery List
│
├── Recipe Suggestions (Expiring Items)
│ ├── "Use these items before they expire:"

Key Features:

- Expiring items prominently displayed
- Recipe suggestions for expiring items
- Restock list generation
- Receipt scanning for easy input
- Visual stock level indicators

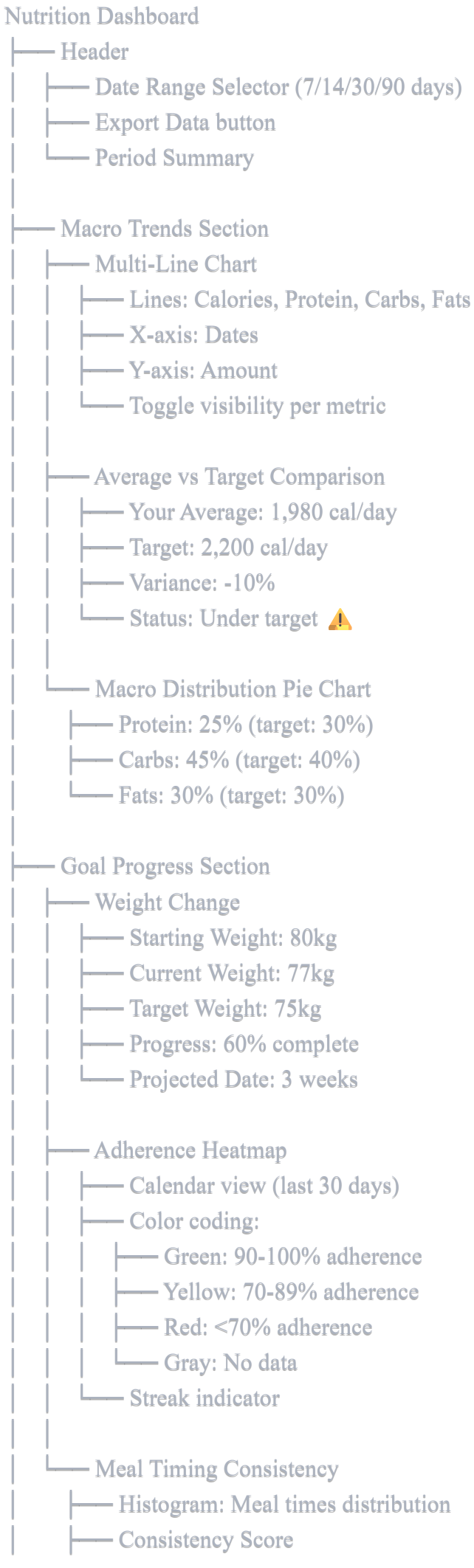
Question for Discussion:

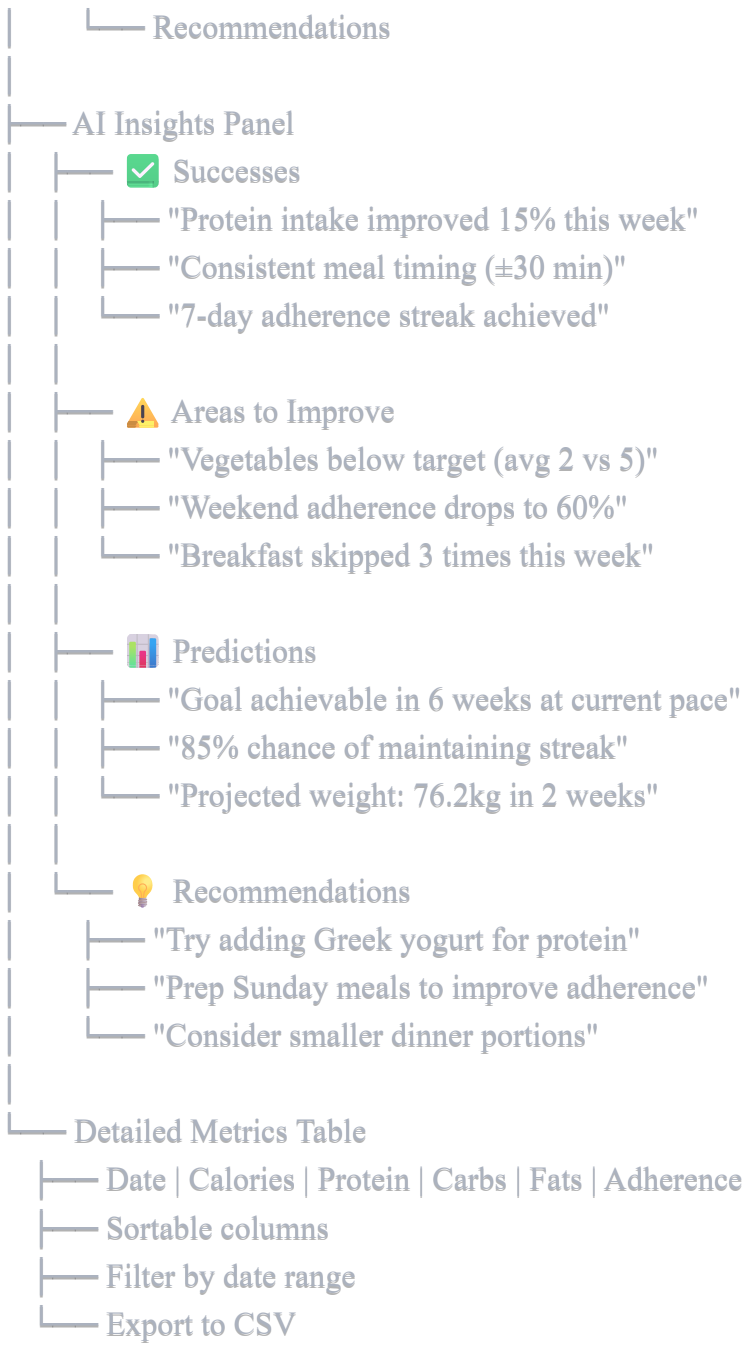
- Should grocery list be here OR in Meal Dashboard?
- Current thought: Grocery list derived from meal plan → stays in Meal Dashboard
- Restock list (based on inventory levels) → stays in Inventory Dashboard

Dashboard 4: Nutrition Dashboard 📊

Purpose: Nutrition tracking, progress analytics, goal adherence, insights

Components:


Nutrition Dashboard
Header
Date Range Selector (7/14/30/90 days)
Export Data button
Period Summary
Macro Trends Section
Multi-Line Chart
Lines: Calories, Protein, Carbs, Fats
X-axis: Dates
Y-axis: Amount
Toggle visibility per metric
Average vs Target Comparison
Your Average: 1,980 cal/day
Target: 2,200 cal/day
Variance: -10%
Status: Under target ⚠️
Macro Distribution Pie Chart
Protein: 25% (target: 30%)
Carbs: 45% (target: 40%)
Fats: 30% (target: 30%)
Goal Progress Section
Weight Change
Starting Weight: 80kg
Current Weight: 77kg
Target Weight: 75kg
Progress: 60% complete
Projected Date: 3 weeks
Adherence Heatmap
Calendar view (last 30 days)
Color coding:
Green: 90-100% adherence
Yellow: 70-89% adherence
Red: <70% adherence
Gray: No data
Streak indicator
Meal Timing Consistency
Histogram: Meal times distribution
Consistency Score



Key Features:

- Visual trend analysis
- Goal progress tracking
- AI-powered insights (requires Nutrition Agent)
- Adherence heatmap
- Predictive analytics

Note on Overlap:

- Meal Dashboard has "Today's Macro Progress" (present/real-time)
- Nutrition Dashboard has trend analysis (past/patterns)
- Clear separation: Today vs Historical

Dashboard Navigation Flow

User Journey Example:

Morning:
1. Opens app → Home Dashboard
2. Sees "Next Meal: Breakfast at 8 AM" card
3. Clicks card → Meal Dashboard (Today tab)
4. Clicks "Log Meal" → Meal logged
5. Sees macro progress bars update in real-time

Planning Time:
1. Home Dashboard → Meal Dashboard
2. Click "This Week" tab
3. See current plan
4. Click "Swap Meal" on Thursday dinner
5. Browse alternatives, select new recipe
6. Plan updated instantly

Inventory Management:
1. Home Dashboard → "3 expiring items" alert
2. Click alert → Inventory Dashboard
3. See expiring items highlighted
4. View suggested recipes using those items
5. Click "Add to Meal Plan"
6. Recipe added, inventory will auto-deduct when logged

Progress Checking:
1. Home Dashboard → "2kg lost" card
2. Click card → Nutrition Dashboard
3. See weight trend chart
4. See adherence heatmap
5. Read AI insights
6. Feel motivated, continue plan

Implementation Checklist - Journey 4

Phase 1: Design & Mockups

- [ ] Create wireframes for all 5 dashboards
- [ ] Design component library (cards, charts, buttons)
- [ ] Create color scheme and styling guide
- [ ] Review and approve designs

Phase 2: Backend API Preparation

- [ ] Create GET /dashboard/summary endpoint
- [ ] Create GET /user/dashboard/profile endpoint
- [ ] Create GET /meal/dashboard/today endpoint
- [ ] Create GET /meal/dashboard/week endpoint
- [ ] Create GET /inventory/dashboard/status endpoint
- [ ] Create GET /nutrition/dashboard/analytics endpoint

Phase 3: Frontend Implementation

- [ ] Build shared components (cards, charts, navigation)
- [ ] Implement Home Dashboard (entry point)
- [ ] Implement User Dashboard
- [ ] Implement Meal Dashboard (4 tabs)
- [ ] Implement Inventory Dashboard
- [ ] Implement Nutrition Dashboard
- [ ] Integrate WebSocket for real-time updates

🚀 IMPLEMENTATION ROADMAP

Priority 1: Fix Journey 1 (Onboarding) - 1 Day

Critical for all other work

Day 1: Database & Onboarding Updates
├── Morning (4 hours)
│ ├── Add onboarding fields to User model
│ ├── Create and run database migration
│ ├── Update all 4 onboarding endpoints
│ └── Update GET /auth/me endpoint
│
└── Afternoon (4 hours)
├── Create user settings endpoints (5 endpoints)
├── Test complete flow (register → onboard → resume → complete)
├── Test settings edit flow
└── Documentation update

Priority 2: Fix Journey 3 (Meal Planning Cleanup) - 1 Day

Remove blockers and technical debt

Day 2: Planning & Tracking Cleanup
├── Morning (4 hours)
│ ├── Fix _categorize_grocery_list() in PlanningAgent
│ ├── Fix _get_user_meal_windows() in PlanningAgent
│ ├── Fix _get_storage_tip() in PlanningAgent
│ └── Test all planning agent tools
│
└── Afternoon (4 hours)
├── Create POST /tracking/eating-out endpoint
├── Delete duplicate meal planning endpoints
├── Update API documentation
└── Test all tracking flows

Priority 3: Build Entry Point Dashboard - 2-3 Days

First user-facing interface

Day 3-4: Frontend Foundation + Home Dashboard
├── Day 3: Setup & Basic Structure
│   ├── Frontend framework setup
│   ├── Component library
│   ├── API client service
│   ├── Authentication flow
│   └── Navigation system
│
└── Day 4-5: Home Dashboard Implementation
    ├── Create dashboard layout
    ├── Build summary cards (4 cards)
    ├── Implement quick actions
    ├── Add AI coach panel
    ├── Add recent activity feed
    ├── WebSocket integration
    └── Test and polish

Priority 4: Individual Dashboards - 1-2 Days Each

Order by importance

Week 2:
Day 6-7: Meal Dashboard (Most Important)
├── Tab 1: This Week's Plan
├── Tab 2: Today's Meals
├── Tab 3: Recipe Browser
└── Tab 4: Meal History

Day 8: User Dashboard
├── Profile section
├── Goals section
├── Path section
└── Preferences section

Day 9: Inventory Dashboard
├── Current inventory view
├── Add items interface
├── Restock list
└── Recipe suggestions

Day 10: Nutrition Dashboard
├── Macro trends charts
├── Goal progress
├── Adherence heatmap
└── AI insights (requires Nutrition Agent)

Priority 5: Build Nutrition Agent - 1 Day

Critical for AI features

```
Day 11: Nutrition Agent Implementation
├── Morning (4 hours)
│   ├── Create NutritionAgent class
│   ├── Implement suggest_next_meal() tool
│   ├── Implement analyze_meal_macros() tool
│   └── Implement provide_nutrition_education() tool
│
└── Afternoon (4 hours)
    ├── Create nutrition API endpoints
    ├── Integrate with Planning Agent
    ├── Integrate with Tracking Agent
    ├── Test AI insights generation
    └── Update Nutrition Dashboard with real insights
```

Priority 6: Testing & Polish - 2-3 Days

```
Days 12-14: Complete Testing & Launch Prep
├── Unit tests for new code
├── Integration tests for user journeys
├── End-to-end tests for critical paths
├── Performance testing
├── Bug fixes
├── UI/UX polish
└── Documentation finalization
```

📊 SUMMARY

Current State

Completed (80%):

- ✅ Database schema
- ✅ Authentication system
- ✅ Onboarding backend (needs state tracking)
- ✅ Meal planning backend (needs cleanup)
- ✅ Tracking backend
- ✅ Inventory backend
- ✅ Notification system
- ✅ WebSocket infrastructure

Not Started (20%):

- ❌ Frontend (0%)
- ❌ Nutrition Agent
- ❌ WhatsApp Agent (planned but not critical)
- ❌ Email verification

Critical Path to MVP

1. Fix Onboarding (1 day) → Enables user registration flow
2. Clean Up Meal Planning (1 day) → Removes technical debt
3. Build Home Dashboard (2-3 days) → First user interface
4. Build Meal Dashboard (2 days) → Core feature
5. Build Other Dashboards (3 days) → Complete feature set
6. Build Nutrition Agent (1 day) → AI intelligence
7. Testing & Polish (2-3 days) → Production ready

Total: 12-15 days to MVP

Key Decisions Made

1. ✅ Onboarding state tracking approach (industry standard)
2. ✅ API response format (REST standard)
3. ✅ Login flow pattern (OAuth2 standard)
4. ✅ Step order enforcement (linear with validation)
5. ✅ Navigation control (backend-directed)
6. ✅ Post-onboarding behavior (land on dashboard, user generates plan)
7. ✅ Edit access (full edit + reset option)
8. ✅ Email verification (post-launch, not blocker)
9. ✅ Dashboard architecture (5 dashboards: 1 entry + 4 specialized)
10. ✅ Planning vs Tracking boundary (clear separation)
11. ✅ Eating out feature placement (tracking agent)
12. ✅ Duplicate endpoint cleanup (remove from planning)

Next Immediate Actions

Start with:

1. Add onboarding tracking fields to User model
2. Update onboarding endpoints to track progress
3. Update GET /auth/me to return redirect_to
4. Test complete registration → onboarding → dashboard flow

Then: 5. Fix 3 planning agent mock methods 6. Move eating out feature to tracking 7. Delete duplicate endpoints

Then: 8. Begin frontend development starting with Home Dashboard

---

📞 OPEN QUESTIONS (Require Discussion)

Question 1: Eating Out Feature Behavior

When user says "I'm eating out tonight", what should happen?

Option A: Remove from plan + log as external **Option B:** Just log as external (keep plan unchanged) **Option C:** Mark as "eating out" in plan, update plan view

Your Decision: _____

Question 2: Grocery List Location

Should grocery list (from meal plan) appear in:

Option A: Meal Dashboard (since it's derived from meal plan) **Option B:** Inventory Dashboard (since it's about shopping) **Option C:** Both (readonly in Inventory, editable in Meal)

Your Decision: _____

Question 3: Function Audit Priority

When should we do the complete audit of unused functions?

Option A: Now, before building frontend (clean codebase first) **Option B:** After frontend (focus on user-facing features first) **Option C:** During frontend (clean up as we discover what's needed)

Your Decision: _____

END OF DOCUMENT

This document represents the complete understanding of NutriLens AI's current state and all architectural decisions as of October 6, 2025. Any future work should reference this document as the single source of truth.