Product Requirements Document (PRD)

Family Meal Planner & Shopping List Application

Version: 1.0
Date: December 5, 2025
Document Owner: Product Management
Target Audience: AI Coding Tool / Development Team

---

1. Executive Summary

1.1 Product Overview

The Family Meal Planner & Shopping List application is a comprehensive meal planning solution designed to help families organize weekly meals, manage nutritional goals, track inventory, and generate intelligent shopping lists. The application leverages AI capabilities to automate meal plan generation, provide nutritional guidance, and simplify recipe management.

1.2 Product Goals

- Reduce meal planning time and decision fatigue for families
- Support individual nutritional goals and dietary preferences
- Minimize food waste through inventory tracking and smart shopping lists
- Provide AI-assisted meal planning and nutritional guidance
- Maintain recipe library with ratings and variety optimization

1.3 Success Metrics

- User completes full weekly meal plan in < 15 minutes
- 80%+ of generated meal plans accepted without major modifications
- Reduction in duplicate ingredient purchases
- User retention rate > 70% after 4 weeks
- Average recipe rating > 7/10

---

2. Technical Specifications

2.1 Platform & Architecture

- **Platform Type:** Cross-platform application (Web + Mobile)
- **Recommended Framework:** React Native with Expo for mobile (iOS/Android) and React for web
- **Backend:** Node.js with Express or Next.js API routes
- **Database:** PostgreSQL for relational data integrity
- **Cloud Storage:** AWS S3 or similar for recipe images
- **Authentication:** Firebase Auth or Auth0 for user management
- **AI Integration:** Anthropic Claude API for meal planning, nutritional analysis, and photo recognition
- **Hosting:** Vercel/Netlify (frontend) + AWS/Railway (backend and database)

2.2 Technology Justification

This stack is optimal for Claude Code because:

- TypeScript support throughout for type safety
- Component-based architecture aligns with modern development practices
- Strong ecosystem for database migrations and API development
- Easy deployment and CI/CD integration
- Cost-effective scaling options

2.3 Data Storage Strategy

- **Cloud-First:** All user data stored in cloud database with real-time sync
- **Local Caching:** Implement offline-first patterns for mobile with sync when online
- **Backup Strategy:** Automated daily backups of database
- **Data Retention:** User data retained indefinitely until account deletion

---

3. User Roles & Authentication

3.1 User Roles

- **Family Account:** Single login per household with access to all features
- **Family Profiles:** Multiple person profiles under one account (maximum 6 profiles)

3.2 Authentication Requirements

- Email/password authentication
- Password reset functionality
- Optional social login (Google, Apple)
- Session management with secure token storage
- Account deletion with data purge capability

---

4. Core Features & Functional Requirements

4.1 Family Profile Management

4.1.1 Profile Structure

Each family profile must store:

Required Fields:

- Profile Name (string, max 50 characters)
- Age (integer, 0-120)
- Profile Avatar (optional image upload or default icon)

Dietary Preferences:

- Food Likes (array of strings, searchable/taggable)
- Food Dislikes (array of strings, searchable/taggable)
- Allergies (array of strings with severity indicator: mild, moderate, severe)

Availability:

- Weekly meal availability grid:
  - Days: Monday - Sunday
  - Meals: Breakfast, Lunch, Dinner, Snacks
  - Each cell: boolean (available/not available)
- Default availability template (e.g., "Kids not home for lunch Mon-Fri")

Nutritional Goals (Macros):

- Daily Calorie Target (integer, optional)
- Daily Protein Target (grams, optional)

- Daily Carbohydrates Target (grams, optional)
- Daily Fat Target (grams, optional)
- Daily Fiber Target (grams, optional)
- Custom macro tracking enabled/disabled toggle
- Macro tracking level: per meal or per day

Activity Level:

- Dropdown: Sedentary, Lightly Active, Moderately Active, Very Active, Extremely Active
- Used by AI Nutritionist for recommendations

4.1.2 Profile Management UI Requirements

- Add/Edit/Delete profiles
- Duplicate profile functionality (for similar family members)
- Reorder profiles in display list
- Profile quick-view cards showing key info (name, age, key restrictions)
- Bulk edit meal availability across multiple profiles

4.1.3 Validation Rules

- Minimum 1 profile required
- Maximum 6 profiles per family account
- Profile name must be unique within family
- Age must be numeric and within valid range
- At least one meal per week must be marked as available per profile

---

4.2 Recipe Management System

4.2.1 Recipe Data Model

Core Recipe Fields:

- Recipe ID (UUID, auto-generated)
- Recipe Name (string, max 100 characters, required)
- Description (text, max 500 characters, optional)
- Recipe Image (URL to uploaded image or default placeholder)

- Servings (integer, default 4, required)
- Prep Time (minutes, integer, optional)
- Cook Time (minutes, integer, optional)
- Total Time (calculated: prep + cook, integer)
- Cuisine Type (dropdown: Italian, Mexican, Asian, American, Mediterranean, etc.)
- Meal Category (multi-select: Breakfast, Lunch, Dinner, Snack)
- Difficulty Level (dropdown: Easy, Medium, Hard)
- Recipe Source (dropdown: Manual Entry, URL Import, Custom)
- Source URL (string, URL format, optional)
- Date Added (timestamp, auto-generated)
- Last Modified (timestamp, auto-updated)
- Times Used (integer, auto-incremented when added to meal plan)
- Last Used Date (timestamp, updated when added to meal plan)

Ingredients List:

- Array of ingredient objects:

```typescript
{
  ingredientId: string (UUID),
  name: string (required),
  quantity: number (required),
  unit: string (dropdown: g, kg, ml, L, cup, tbsp, tsp, oz, lb, unit, etc.),
  category: string (dropdown: Produce, Dairy, Meat, Pantry, Frozen, Bakery, etc.),
  notes: string (optional, e.g., "finely chopped", "optional")
}
```

Instructions:

- Array of instruction steps:

```typescript
```

```
{
  stepNumber: integer (auto-incremented),
  instruction: string (max 500 characters per step),
  timerMinutes: integer (optional, for steps requiring timing)
}
```

Nutritional Information (per serving):

- Calories (integer, optional but recommended)
- Protein (grams, decimal)
- Carbohydrates (grams, decimal)
- Fat (grams, decimal)
- Fiber (grams, decimal)
- Sugar (grams, decimal)
- Sodium (mg, integer)
- Auto-calculated flag (boolean: true if AI-estimated, false if manually entered)

Rating & Tracking:

- Family Rating (integer, 1-10 scale, optional)
- Rating Date (timestamp, when last rated)
- Notes/Reviews (text, max 1000 characters, optional)
- Tags (array of strings for searchability: "quick", "kid-friendly", "meal-prep", etc.)

Meal Prep & Yield:

- Yields Multiple Meals (boolean)
- Number of Meals Yielded (integer, if above is true)
- Leftover Storage Instructions (text, optional)
- Freezable (boolean)
- Reheating Instructions (text, optional)

Profile Compatibility:

- Compatible Profiles (array of profile IDs that can eat this recipe)
- Incompatible Ingredients by Profile (array mapping profile ID to conflicting ingredients)

4.2.2 Recipe Management Features

Add Recipe Manually:

- Form-based input with all fields from data model
- Image upload with crop/resize functionality
- Ingredient auto-complete based on existing inventory
- Step-by-step instruction builder with drag-to-reorder
- Save as draft functionality
- Duplicate recipe functionality for variations

Import Recipe from URL:

- URL input field
- AI parsing using Claude API to extract:
  - Recipe name
  - Ingredients with quantities and units
  - Instructions
  - Cook/prep times
  - Servings
  - Nutritional info (if available)
- Display parsed content for user review and editing
- Allow manual correction of any auto-parsed field
- Save source URL for reference

Recipe Photo Recognition:

- Camera/photo upload for recipe images
- Claude Vision API to identify dish type
- Suggest recipe name and ingredients
- User confirmation required before saving

Bulk Recipe Import:

- CSV import template for multiple recipes
- Validation and error reporting

- Preview imported recipes before saving

Recipe Search & Filter:

- Search by: name, ingredients, tags, cuisine type
- Filter by: meal category, difficulty, prep time range, rating range, dietary compatibility
- Sort by: rating (high to low), times used (frequency), date added, prep time, last used date
- Quick filters: "Quick meals (<30 min)", "Highly rated (8+)", "Never tried", "Favorites"

Recipe Detail View:

- Full recipe display with image
- Editable fields (inline editing or modal)
- Portion calculator (adjust servings, auto-scale ingredients)
- Share recipe (export as PDF or text)
- Print-friendly view
- Related recipes suggestions (AI-based on similar ingredients/cuisine)

Recipe Rating System:

- 1-10 scale with half-point increments (e.g., 7.5)
- Rating prompted after recipe is used in meal plan
- Display average rating prominently
- Show rating history/trend over time
- Option to add notes with rating

Recipe Favorites:

- Bookmark/favorite toggle on recipe cards
- Quick access to favorites list
- Favorites shown first in meal plan generation

Recipe Deletion & Archive:

- Soft delete (archive) functionality to preserve meal plan history
- Confirmation prompt before deletion
- Restore archived recipes

4.2.3 Recipe Validation Rules

- Recipe name required and unique within family account
- At least one ingredient required
- At least one instruction step required
- Servings must be > 0
- If nutritional info provided, all macros must be non-negative
- If yields multiple meals is true, number of meals must be ≥ 2
- Rating must be 1-10 if provided
- Image file size < 5MB, formats: JPG, PNG, WEBP

4.3 Weekly Staples Management

4.3.1 Staples Data Model

```typescript
{
  stapleId: string (UUID),
  itemName: string (required),
  quantity: number (required),
  unit: string (required),
  category: string (dropdown: same as recipe ingredient categories),
  autoAddToList: boolean (default true),
  notes: string (optional)
}
```

4.3.2 Staples Features

- Add/Edit/Delete staples
- Enable/disable auto-add to weekly shopping list
- Categorize staples for shopping list organization
- Bulk add common staples (preset templates: "Basic Pantry", "Weekly Dairy", etc.)
- Temporary disable staples (e.g., "already stocked this week")

4.3.3 Staples UI Requirements

- Dedicated "Weekly Staples" section in app navigation
- Simple list view with checkboxes for enable/disable

- Quick add button with common staples suggestions
- Visual indicator showing which staples are active for current week
- Edit quantity inline for one-time adjustments without changing default

---

4.4 Inventory Management System

4.4.1 Inventory Data Model

```
typescript

{
  inventoryId: string (UUID),
  itemName: string (required),
  quantity: number (required),
  unit: string (required),
  category: string (required),
  location: string (dropdown: Fridge, Freezer, Pantry, Custom),
  expiryDate: date (optional),
  autoPopulatedExpiry: boolean (true if AI-estimated),
  dateAdded: timestamp,
  addedBy: string (Manual, Photo Recognition, Shopping List),
  notes: string (optional),
  isUsedInPlannedMeal: boolean (calculated field)
}
```

4.4.2 Inventory Features

Add Item Manually:

- Quick add form with name, quantity, unit, location
- Expiry date picker with smart defaults based on item type
- Category auto-suggestion based on item name

Add via Photo Recognition:

- Camera access or photo upload
- Claude Vision API to identify items in photo
- Return structured list of items with estimated quantities
- AI-estimated expiry dates based on item type and typical shelf life
- User review and adjustment screen before adding to inventory

- Handle multiple items in single photo

Photo Recognition Specifications:

- Recognize individual grocery items (e.g., "3 carrots", "1L milk", "chicken breast")
- Identify packaged goods by label when possible
- Estimate quantities for loose items (e.g., "approximately 5 apples")
- Confidence score display for user verification
- "Unable to identify" option with manual entry fallback

Expiry Date Intelligence:

- AI-populated expiry dates based on item type:
  - Dairy: 7 days default
  - Produce: varies by type (leafy greens 3-5 days, root vegetables 1-2 weeks)
  - Meat: 2-3 days (fridge), 3-6 months (freezer)
  - Pantry items: 6-12 months
- User can always override AI suggestion
- Mark items with approaching expiry (< 3 days) with warning icon
- Mark expired items in red

Inventory Management:

- List view with filters: all items, by location, by category, expiring soon, expired
- Search by item name
- Sort by: expiry date, date added, category, quantity
- Edit quantity and details inline
- Use/consume button to decrement quantity or remove item
- Quick actions: Move to different location, update expiry date
- Bulk delete expired items

Smart Depletion:

- When meal plan is finalized, automatically mark inventory items used in recipes
- Option to manually mark items as used
- Adjust quantities when recipes use portions of inventory items

Inventory Alerts:

- Low stock alerts for staples (configurable threshold)
- Expiring soon notifications (3 days before expiry)
- Expired items banner on inventory screen

Inventory History:

- Track item usage over time
- Most frequently used items report
- Waste tracking (items expired before use)

4.4.3 Validation Rules

- Item name required
- Quantity must be > 0
- Expiry date cannot be in the past when adding new items
- Duplicate items with same name/location should prompt to update existing quantity rather than create new entry

---

4.5 Meal Planning System

4.5.1 Meal Plan Data Model

```typescript
```

```
{
  mealPlanId: string (UUID),
  weekStartDate: date (Monday of the week),
  weekEndDate: date (Sunday of the week),
  status: string (Draft, Finalized, Archived),
  createdDate: timestamp,
  finalizedDate: timestamp (nullable),

  meals: [
    {
      mealId: string (UUID),
      dayOfWeek: string (Monday-Sunday),
      mealType: string (Breakfast, Lunch, Dinner, Snack),
      recipeId: string (foreign key to recipe, nullable if no recipe),
      recipeName: string (denormalized for display),
      servings: integer (adjusted servings for this meal),
      participatingProfiles: [profileId strings],
      scalingFactor: number (calculated: servings / recipe base servings),
      nutritionalSummary: {
        totalCalories: integer,
        totalProtein: number,
        totalCarbs: number,
        totalFat: number,
        totalFiber: number,
        perPersonBreakdown: [
          {
            profileId: string,
            profileName: string,
            portionSize: number (decimal, e.g., 0.5 for half serving),
            calories: integer,
            protein: number,
            carbs: number,
            fat: number,
            fiber: number,
            meetsTargets: boolean
          }
        ]
      },
      isLeftover: boolean,
      leftoverFromMealId: string (nullable, reference to original meal),
      notes: string (optional, user notes about this meal slot),
      isLocked: boolean (user manually selected, don't auto-change)
    }
  ],

  weeklyNutritionalSummary: {
```

```
  byProfile: [
    {
      profileId: string,
      profileName: string,
      totalCalories: integer,
      totalProtein: number,
      totalCarbs: number,
      totalFat: number,
      totalFiber: number,
      targetCalories: integer,
      targetProtein: number,
      targetCarbs: number,
      targetFat: number,
      targetFiber: number,
      complianceScore: number (0-100, percentage of targets met)
    }
  ]
 }
}
```

4.5.2 Meal Planning Features

Meal Plan View:

- Weekly calendar grid layout:
  - Rows: Days of week (Monday-Sunday)
  - Columns: Meal types (Breakfast, Lunch, Dinner, Snacks)
  - Each cell displays: Recipe name, recipe image thumbnail, participating profile icons
- Multiple view modes:
  - Grid view (default)
  - List view (chronological meals)
  - Profile view (meals by person)
- Navigation: Previous/Next week, Jump to specific week, "This Week" quick button

Auto-Generate Meal Plan:

- Button: "Generate Meal Plan" on meal plan screen
- AI generation using Claude API with following inputs:
  - All family profiles with availability and dietary preferences
  - Recipe library with ratings, times used, last used date
  - Current inventory items

- Weekly staples
- User preferences for variety (configurable setting)
- Nutritional targets for each profile

AI Generation Algorithm (Claude Prompt Specification):

```
Input context:
- Profile data (availability, likes, dislikes, allergies, macro targets)
- Recipe database with metadata (ratings, frequency, last used)
- Current inventory with expiry dates
- Staples list
- Variety preference: high/medium/low

Output required:
- Complete weekly meal assignments matching profile availability
- Recipe selection criteria:
  * Prioritize recipes rated 8+ (but include variety)
  * Avoid same recipe twice in same week
  * Boost probability for recipes not used in last 2 weeks
  * Additional weighting for recipes manually selected in past (track selection count)
  * Consider manually selected recipes as "favorites" with 1.5x selection weight
  * Use ingredients from inventory approaching expiry
  * Match recipes to profile dietary preferences
  * Ensure variety of cuisines throughout week
- Portion scaling based on participating profiles per meal
- Account for meal prep recipes yielding multiple servings
- Assign leftovers appropriately to subsequent meals
- Calculate per-meal and per-person nutritional breakdowns
- Ensure daily and weekly macro targets are approached for each profile
- Flag any profiles whose targets cannot be met with available recipes

Constraints:
- No recipe may appear more than once per week
- All meals must have >= 1 participating profile
- Recipe must not contain ingredients in any participant's dislikes list
- Recipe must not contain allergens for severe allergies (moderate/mild show warning)
- Prefer recipes using expiring inventory items (within 5 days)
```

Manual Meal Selection:

- Click any meal slot to open recipe selector
- Search/filter recipes compatible with available profiles for that meal/day
- Preview recipe with nutritional info for selected profiles

- "Add to Plan" button
- "Clear Slot" option to remove assigned recipe
- Lock icon to prevent auto-generation from changing this slot

Meal Slot Actions:

- Swap meals (drag-and-drop between slots)
- Copy meal to another slot
- Mark as leftover (link to original meal)
- Add profiles to meal (checkboxes)
- Remove profiles from meal
- Adjust servings (override auto-scaling)
- Add meal notes
- Lock/unlock slot

Profile-Specific Meal Management:

- Select profile filter to highlight their meals
- "Edit [Profile] Week" mode showing only their meals
- Quick assign different recipe for specific profile on a given day
- Example use case: Kids have different dinner than adults on Tuesday
  - Solution: Create two separate "Dinner" entries for Tuesday, one for kids, one for adults

Leftover Management:

- When meal prep recipe is added, AI suggests leftover assignments
- Leftover meals show "🍱 Leftover from [Original Meal]" indicator
- Leftover slots don't require additional ingredients (smart shopping list)
- User can convert leftover slots to new meals

Variety Optimization:

- User setting: Variety level (High/Medium/Low)
  - High: No cuisine type repeated in same week, maximum ingredient diversity
  - Medium: Cuisine types max 2x per week, some ingredient overlap acceptable
  - Low: Prioritize highly rated recipes even if repeated cuisine/ingredients
- Display variety score for current meal plan (0-100)

- "Increase Variety" button to re-run AI generation with higher variety weights

Nutritional Overview:

- Per-Day view: Shows each profile's macro totals vs targets for selected day
- Per-Week summary: Shows each profile's weekly totals vs weekly targets (daily target × 7)
- Color coding:
  - Green: Within 10% of target
  - Yellow: 10-20% off target
  - Red: >20% off target or missing target entirely
- Drill-down to see which meals contribute most to specific macros

AI Nutritionist Feedback:

- Real-time analysis button on meal plan screen
- Claude API call with meal plan + profile targets as context
- Returns structured feedback:

```typescript
```

```
{
  overallScore: number (0-100),
  profileFeedback: [
    {
      profileId: string,
      profileName: string,
      calorieStatus: string ("on track", "under", "over"),
      macroBalance: string (analysis of protein/carb/fat ratios),
      recommendations: [string array of suggestions],
      warnings: [string array of concerns],
      positives: [string array of praise]
    }
  ],
  weeklyRecommendations: [string array],
  suggestedSwaps: [
    {
      originalMealId: string,
      originalRecipeName: string,
      suggestedRecipeId: string,
      suggestedRecipeName: string,
      reason: string
    }
  ]
}
```

- Display feedback in expandable sections per profile
- One-click apply suggested swaps
- Examples of feedback:
  - "John's protein target is 150g/day but averaging 95g. Consider adding protein-rich snacks or increasing portions."
  - "Sarah's fiber intake is excellent at 32g/day!"
  - "Family carbohydrate intake skewed high on weekdays. Suggest lower-carb lunches Mon-Wed."

Finalize Meal Plan:

- "Finalize Plan" button (prominent, primary action)
- Confirmation dialog showing summary:
  - Total unique recipes
  - Number of meals per profile
  - Number of profiles meeting nutritional targets

    - Inventory items to be used
    - Estimated new ingredients needed
- Upon finalization:
    - Status changes to "Finalized"
    - Generate shopping list (see 4.7)
    - Trigger inventory depletion for used items
    - Lock meal plan from auto-regeneration
    - Prompt to rate recipes after meals are consumed (notification system)

Meal Plan Versioning:

- Auto-save draft changes
- "Start Over" option to regenerate from scratch
- Undo/Redo for manual changes (last 10 actions)

Past Meal Plans:

- Archive finalized meal plans automatically when new week starts
- View past meal plans in read-only mode
- "Repeat Week" option to copy past meal plan to current week
- Analytics: Most used recipes over last month/quarter

4.5.3 Validation Rules

- Each meal slot must have at least 1 participating profile
- Participating profiles must be marked as available for that day/meal type
- Recipe must not conflict with severe allergies of participating profiles
- Cannot finalize plan with empty required meal slots for available profiles
- Leftover meals must reference valid original meal in same week
- Servings must be > 0 for all meals

4.6 Shopping List Management

4.6.1 Shopping List Data Model

```
typescript
```

```
{
 shoppingListId: string (UUID),
 linkedMealPlanId: string (foreign key to meal plan),
 weekStartDate: date,
 status: string (Generated, In Progress, Completed),
 generatedDate: timestamp,
 completedDate: timestamp (nullable),

 items: [
  {
   itemId: string (UUID),
   itemName: string,
   quantity: number,
   unit: string,
   category: string,
   source: string (Recipe, Staple, Manual, Multiple),
   sourceDetails: [
    {
     sourceType: string (Recipe/Staple/Manual),
     sourceName: string (recipe name or staple name),
     quantityNeeded: number,
     unit: string
    }
   ],
   isConsolidated: boolean (true if multiple sources combined),
   inInventory: boolean (item exists in current inventory),
   inventoryQuantity: number (how much already owned),
   netQuantityNeeded: number (calculated: quantity - inventoryQuantity),
   isPurchased: boolean (checkbox for shopping),
   customNote: string (optional),
   priority: string (High, Medium, Low),
   estimatedPrice: number (optional, for future cost tracking)
  }
 ],

 categoryOrder: [string array of category names in custom order],

 manualItems: [
  {
   itemId: string (UUID),
   itemName: string,
   quantity: number,
   unit: string,
   category: string,
   notes: string (optional),
   isPurchased: boolean
```

```
    }
  ]
}
```

4.6.2 Shopping List Generation

Automatic Generation (on Meal Plan Finalization):

1. Extract all ingredients from all recipes in finalized meal plan
2. Adjust quantities based on servings/scaling factors
3. Add all active weekly staples
4. Consolidate duplicate ingredients:
   - Sum quantities of same ingredient with same unit
   - Convert compatible units if possible (e.g., 500ml + 1L = 1.5L)
   - Flag items consolidated from multiple sources
5. Check inventory:
   - Mark items already in inventory
   - Calculate net quantity needed (total needed - inventory amount)
   - If inventory sufficient, set netQuantityNeeded to 0 (item still appears but marked as covered)
6. Exclude ingredients from leftover meals (already accounted for in original meal)
7. Categorize all items by ingredient category
8. Sort within categories (configurable order)
9. Create shopping list record with status "Generated"

Consolidation Display:

- Show consolidated items with source breakdown
- Example: "Milk - 3 cups (2 cups from Pancake Recipe, 1 cup from Smoothie Recipe)"
- Bracket notation for visibility: "Milk (3 cups) [from 2 recipes]"

Inventory Integration:

- Items in inventory shown with ✓ icon
- Show "You have: [inventory amount], Need: [net amount]"
- If inventory covers need entirely: "✓ Already in inventory"
- User can override inventory check (e.g., inventory item expired)

4.6.3 Shopping List Features

View & Interact:

- List view organized by category (collapsible sections)
- Checkbox for each item (mark as purchased)
- Swipe actions: Delete, Edit quantity, Move to different category
- Search/filter: By category, by recipe source, show only unpurchased, show only purchased
- Sort options: Category order, alphabetical, by recipe

Custom Category Order:

- Settings screen: "Customize Shopping List Categories"
- Drag-and-drop to reorder categories to match local store layout
- Presets: "Standard Grocery Store", "Warehouse Club", "Farmers Market"
- Add custom categories
- Example order: Produce → Dairy → Meat → Frozen → Bakery → Pantry → Beverages → Other

Manual Add Items:

- "+" button to add item not from recipes
- Quick add form: item name, quantity, unit, category
- Suggested items from past shopping lists
- Voice input option for hands-free add
- Manual items show with " ✋ Manual" tag

Edit Shopping List:

- Adjust quantities inline
- Remove items (confirmation if from recipe)
- Split item into separate entries (e.g., buy 2 packs instead of bulk)
- Add notes to items (e.g., "organic preferred", "brand: XYZ")
- Set priority (High/Medium/Low) with visual indicator

Shopping Mode:

- "Start Shopping" button changes UI to shopping-friendly layout
- Large checkboxes and text

- Category sections collapse after all items checked
- Progress bar showing % complete
- "Mark All in Category" quick action
- Audio feedback on check (optional, settings)

Share Shopping List:

- Export as text, PDF, or email
- Generate shareable link (read-only or collaborative)
- If collaborative link, real-time sync of checked items across devices
- Print view optimized for paper list

Post-Shopping Actions:

- "Complete Shopping" button
- Option to add purchased items to inventory
- Bulk add with default expiry dates
- Review unfinished items (not checked)
- Option to move unpurchased items to "Next Week" list

Shopping List History:

- View past shopping lists linked to meal plans
- Copy items from past list to current list
- Analytics: Most frequently purchased items, average cost (if tracking enabled)

4.6.4 Validation Rules

- Item name required for all entries
- Quantity must be > 0
- Cannot mark shopping complete if linked meal plan not finalized
- Consolidated quantities must be mathematically accurate
- Category must be valid (from predefined list or custom additions)

4.7 AI Integration Specifications

4.7.1 AI Provider

- **Primary AI Service:** Anthropic Claude API
- **Recommended Model:** Claude 3.5 Sonnet (balance of speed and capability)
- **Fallback:** Claude 3 Opus for complex nutritional analysis if needed

4.7.2 AI Use Cases & Implementation

Use Case 1: Meal Plan Auto-Generation

- **Trigger:** User clicks "Generate Meal Plan" button
- **Input:** JSON payload with profiles, recipes, inventory, staples, preferences
- **Claude Prompt Template:**

You are a family meal planning assistant. Generate a complete weekly meal plan based on the following context:

FAMILY PROFILES:
{profiles with availability, preferences, allergies, macro targets}

AVAILABLE RECIPES:
{recipe library with ratings, times used, last used date, ingredients, nutritional info}

CURRENT INVENTORY:
{items with quantities and expiry dates}

WEEKLY STAPLES:
{staple items}

REQUIREMENTS:
1. Assign recipes to meal slots matching profile availability
2. Prioritize recipes rated 8+ but ensure variety
3. Avoid repeating same recipe within the week
4. Give extra weight to recipes not used in last 2 weeks
5. Manually selected recipes have 1.5x selection weight
6. Use ingredients expiring within 5 days when possible
7. Respect all dietary preferences and severe allergies
8. Scale servings based on participating profiles
9. Identify meal prep recipes and assign leftovers
10. Calculate per-meal nutritional breakdown per profile
11. Ensure daily macro targets are approached for each profile

OUTPUT FORMAT:
Return a JSON object with structure matching the mealPlan data model, including:
- Complete meal assignments for the week
- Scaling factors and participating profiles per meal
- Leftover meal linkages
- Per-person nutritional breakdown per meal
- Weekly nutritional summary per profile
- Reasoning notes for selections

- **Response Parsing:** Validate JSON structure, handle errors gracefully
- **Error Handling:** If AI fails, provide manual selection mode with helpful defaults
- **Caching:** Cache recipe database context to reduce API calls for iterative generation

Use Case 2: AI Nutritionist Feedback

- **Trigger:** User clicks "Get Nutritionist Feedback" on meal plan
- **Input:** Current meal plan with all nutritional data

- **Claude Prompt Template:**

```
You are a professional nutritionist analyzing a family's weekly meal plan. Provide constructive, specific feedback.

MEAL PLAN DATA:
{complete meal plan with per-person nutritional breakdowns}

PROFILE TARGETS:
{each profile's macro goals and activity level}

ANALYZE:
1. Calorie adequacy per profile (is each person meeting daily needs?)
2. Macro balance (protein/carb/fat ratios)
3. Meal distribution (is energy evenly distributed throughout day?)
4. Micronutrient considerations (fiber, general healthy eating patterns)
5. Specific concerns for age groups (e.g., children need adequate calcium)
6. Variety and sustainability of plan

PROVIDE:
1. Overall score (0-100) for meal plan quality
2. Per-profile feedback with:
   - What's going well
   - Areas for improvement
   - Specific recommendations
   - Warnings if targets significantly missed
3. Suggested recipe swaps to improve nutritional balance (provide specific recipe IDs from available library)

OUTPUT FORMAT: JSON matching nutritionist feedback data model

TONE: Supportive and educational, not judgmental. Praise good choices and offer practical suggestions.
```

- **Response Display:** Format feedback in user-friendly sections with expand/collapse
- **Actionable Suggestions:** Make recipe swap suggestions one-click actions

Use Case 3: Recipe Import from URL

- **Trigger:** User pastes recipe URL and clicks "Import"
- **Input:** Recipe URL
- **Process:**
  1. Fetch HTML content from URL (web scraping)
  2. Send HTML to Claude with parsing prompt
- **Claude Prompt Template:**

```
Extract recipe information from the following HTML content:

{HTML content}

EXTRACT:
- Recipe title
- Description/summary
- Servings count
- Prep time (minutes)
- Cook time (minutes)
- Ingredients list with quantities, units, and names
- Step-by-step instructions
- Nutritional information per serving (if available)
- Cuisine type (best guess if not stated)
- Meal category (breakfast/lunch/dinner)

OUTPUT FORMAT: JSON matching recipe data model

RULES:
- Parse ingredients into structured format: quantity (number), unit (string), name (string)
- Standardize units to metric (g, kg, ml, L) when possible
- If nutritional info not found, estimate based on ingredients (mark as auto-calculated)
- Number instruction steps sequentially
- If images are referenced, extract primary recipe image URL
```

- **Error Handling:** If parsing fails, return partial data and prompt user to complete manually
- **User Review:** Always show parsed data for confirmation before saving

Use Case 4: Recipe Photo Recognition

- **Trigger:** User uploads photo of cooked dish
- **Input:** Image file
- **Claude Vision Prompt:**

Analyze this food image and provide:
1. Identify the dish/recipe name (be specific, e.g., "Chicken Tikka Masala" not just "curry")
2. List likely main ingredients
3. Suggest cuisine type
4. Estimate difficulty level
5. Suggest meal category (breakfast/lunch/dinner)
6. If recognizable packaged items, extract brand and product name

OUTPUT FORMAT: JSON with fields: dishName, ingredients (array), cuisine, difficulty, mealCategory, confidence (0-100)

Be specific but acknowledge uncertainty if image is ambiguous.

- **Confidence Handling:** If confidence < 70%, show "uncertain" warning and suggest manual entry
- **Follow-up:** User confirms dish name, then optionally trigger recipe search or manual entry

Use Case 5: Inventory Photo Recognition

- **Trigger:** User uploads photo of groceries or fridge contents
- **Input:** Image file
- **Claude Vision Prompt:**

```
Identify all food items in this image and provide structured inventory data.

IDENTIFY:
- Item names (be specific: "cherry tomatoes" not just "tomatoes")
- Estimated quantities (count for items, approximate weight/volume for others)
- Appropriate units
- Likely storage location (fridge, freezer, pantry)
- Typical expiry timeframes from today's date

HANDLE:
- Multiple items in one image
- Packaged goods (read labels for exact quantities if visible)
- Loose produce (estimate count or weight)
- Partial visibility (estimate conservatively)

OUTPUT FORMAT: JSON array of items matching inventory data model, including:
{
  itemName: string,
  quantity: number,
  unit: string,
  category: string (Produce, Dairy, Meat, Pantry, etc.),
  location: string (Fridge/Freezer/Pantry),
  estimatedExpiryDays: number (days from now),
  confidence: number (0-100 per item)
}

If unable to identify an item, return object with itemName: "Unknown item" and confidence: 0.
```

- **Response Handling:** Display list of identified items with checkboxes, allow user to edit before adding to inventory
- **Batch Processing:** Support multiple photos in sequence, aggregate all items

Use Case 6: Macro Calculation & Estimation

- **Trigger:** User enters recipe without nutritional info, clicks "Estimate Macros"
- **Input:** Recipe ingredients with quantities
- **Claude Prompt Template:**

Estimate nutritional information for this recipe based on ingredients.

RECIPE INGREDIENTS:
{ingredients list with quantities and units}

SERVINGS: {number}

CALCULATE per serving:
- Calories (kcal)
- Protein (g)
- Carbohydrates (g)
- Fat (g)
- Fiber (g)
- Sugar (g)
- Sodium (mg)

Use standard USDA nutritional data for calculations. Account for:
- Cooking methods (oils, reductions, etc. if mentioned in recipe)
- Edible portions (e.g., chicken bone weight vs. edible meat)

OUTPUT FORMAT: JSON with macros object matching recipe nutritional data model

Include field: autoCalculated: true

- **Display:** Show calculated macros with "🤖 AI Estimated" badge
- **User Override:** Allow manual editing with indicator that it's no longer auto-calculated

4.7.3 AI Rate Limiting & Error Handling

- Implement request queuing for AI calls
- Show loading states during API calls (spinners with progress messages)
- Timeout after 30 seconds with user-friendly error message
- Cache common AI responses (e.g., frequently imported recipes)
- Provide manual fallback for all AI features
- Rate limiting: Max 100 AI requests per user per day (configurable)
- Error messages: Specific and actionable (e.g., "Unable to parse recipe. Please check URL is accessible." vs. "Error")

4.7.4 AI Response Validation

- All AI JSON responses must be validated against expected schema
- Catch and handle malformed responses gracefully

- Log AI failures for monitoring and improvement
- Implement confidence thresholds for auto-acceptance (>90% auto-accept, 70-90% user review, <70% manual mode)

4.8 Additional Features & Enhancements

4.8.1 Recipe Suggestions & Discovery

Smart Recipe Suggestions:

- "Recipes You Might Like" section based on:
  - Highly rated recipes similar to past favorites
  - Recipes matching family profiles
  - Trending recipes (if social features added later)
  - Seasonal recipes based on current month
- "Use Up Your Inventory" recipe search:
  - Show recipes that use ingredients expiring soon
  - Filter recipes by ingredients on hand
  - "What can I make with..." search

Recipe Collections:

- User-created collections (e.g., "Summer BBQ", "Kid Favorites", "Quick Weeknight Meals")
- Pre-made collections by app (e.g., "Heart Healthy", "High Protein", "Budget Friendly")
- Share collections with other users (future social feature)

4.8.2 Meal Planning Templates

- Save current week as template (e.g., "Typical School Week", "Holiday Week")
- Apply template to new week with one click
- Edit template to adjust for season/preferences
- Pre-made templates for different family types:
  - "Family with Young Kids"
  - "Athletic Family"
  - "Vegetarian Week"
  - "Busy Professionals"

4.8.3 Notifications & Reminders

Push Notifications (optional, user can enable/disable):

- "Time to plan next week's meals" (Sunday evening)
- "Don't forget to shop!" (before week starts, if shopping list not completed)
- "Items expiring soon in inventory" (3 days before)
- "Rate your meals from this week" (end of week prompt)
- "Recipe prep reminder" (e.g., "Start marinating chicken for tomorrow's dinner")

In-App Reminders:

- Banner on home screen for pending tasks
- Badge counts for unread notifications

4.8.4 Settings & Preferences

App Settings:

- Account management (email, password, profile image)
- Notification preferences (toggle types, frequency)
- Unit preferences (metric/imperial - though metric is default)
- Theme: Light mode, Dark mode, Auto
- Language (English default, future multi-language support)

Meal Planning Preferences:

- Default variety level (High/Medium/Low)
- Auto-generate on week start (boolean)
- Include snacks in meal plan (boolean)
- Leftover preference (maximize/minimize/balanced)
- Recipe complexity preference (easy/mixed/advanced)

Shopping List Preferences:

- Default category order (customizable)
- Auto-add staples (boolean)
- Show inventory items on list (boolean)
- Consolidate duplicate items (boolean)

AI Preferences:

- AI assistance level (Full/Partial/Manual)
- Nutritionist feedback auto-run (boolean)
- Auto-estimate macros for new recipes (boolean)

Privacy Settings:

- Data sharing opt-in/opt-out
- Delete all data option
- Export data (JSON format)

4.8.5 Onboarding & Tutorials

First-Time User Experience:

- Welcome screen with app overview
- Step-by-step setup wizard:
  1. Create first family profile
  2. Add 2-3 favorite recipes
  3. Set up weekly staples
  4. Generate first meal plan
  5. View shopping list
- Interactive tooltips on key features
- "Getting Started" checklist in app

In-App Help:

- Help icon (?) on each major screen with contextual tips
- FAQ section
- Video tutorials (optional future enhancement)
- Contact support form

4.8.6 Data Export & Backup

- Export recipes as PDF cookbook
- Export meal plan as PDF weekly menu
- Export shopping list as text/PDF/CSV

- Export all user data as JSON (for backup or migration)
- Import recipes from JSON/CSV file

4.8.7 Analytics & Insights (Future Enhancement)

- Dashboard showing:
  - Recipes used this month
  - Most popular recipes
  - Nutritional trends over time
  - Food waste metrics (expired inventory items)
  - Money saved estimates (if cost tracking added)
  - Achievement badges (e.g., "10 weeks planned!", "5-star nutritionist score")

4.8.8 Social Features (Future Enhancement)

- Share recipes with friends/family
- Public recipe library with user submissions
- Follow other users for recipe inspiration
- Recipe comments and ratings from community
- Weekly meal plan sharing (e.g., "Meal Prep Sunday" social posts)

---

5. User Flows & Wireframe Requirements

5.1 Core User Flows

Flow 1: New User Onboarding

1. User creates account (email/password)
2. Welcome screen with app value proposition
3. Prompt: "Let's set up your family profiles"
4. Add first profile form (name, age, basic preferences)
5. Optional: Add more profiles or skip
6. Prompt: "Add your first recipe" with options: Manual, URL, or Skip
7. Optional: Set up weekly staples or skip
8. Prompt: "Ready to plan your first week?"

9. Generate meal plan (AI auto-generates)
10. Review meal plan screen with tips overlay
11. "Finalize Plan" → generates shopping list
12. Success message: "Your first meal plan is ready!"

Flow 2: Weekly Meal Planning (Primary Use Case)

1. User opens app on Sunday evening
2. Home screen shows: Current week meal plan (if exists) or "Plan This Week" prompt
3. User navigates to meal planner
4. Clicks "Generate Meal Plan" button
5. AI generates meal plan (loading indicator with progress)
6. Meal plan displays in weekly grid view
7. User reviews meals, clicks on any slot to modify
8. Optional: Clicks "Get Nutritionist Feedback" → AI provides analysis
9. Optional: Makes manual adjustments based on feedback
10. Clicks "Finalize Plan"
11. Confirmation dialog with summary
12. System generates shopping list
13. Success message: "Meal plan finalized! View shopping list?"
14. Redirect to shopping list

Flow 3: Shopping

1. User navigates to Shopping List
2. Views categorized list of items
3. Optional: Adds manual items
4. Optional: Adjusts quantities or removes items
5. Clicks "Start Shopping"
6. Shopping mode activates (large checkboxes, simplified UI)
7. User checks off items as purchased
8. Progress bar updates
9. Clicks "Complete Shopping"

10. Prompt: "Add purchased items to inventory?"
11. User confirms or skips
12. If confirmed: Bulk add with default expiry dates
13. Success message: "Shopping complete! Enjoy your meals."

Flow 4: Adding Recipe from URL

1. User navigates to Recipes section
2. Clicks "Add Recipe" → selects "Import from URL"
3. Pastes recipe URL
4. Clicks "Import"
5. AI parses recipe (loading indicator)
6. Parsed recipe displays in review form (all fields editable)
7. User reviews and adjusts as needed
8. Optional: Uploads better image
9. Clicks "Save Recipe"
10. Recipe added to library
11. Success message: "Recipe saved! Add to this week's plan?"

Flow 5: Inventory Management with Photo

1. User navigates to Inventory
2. Clicks "Add Items" → selects "Photo"
3. Camera opens or photo selector
4. User takes photo or selects from gallery
5. AI analyzes photo (loading with "Identifying items...")
6. List of identified items displays with checkboxes and editable fields
7. User reviews, adjusts quantities, expiry dates
8. Checks items to add
9. Clicks "Add to Inventory"
10. Items added with confirmation
11. Inventory list updates

5.2 Screen Layout Requirements

Navigation Structure:

- Bottom tab navigation (mobile) or side navigation (web):
  - Home / Dashboard
  - Meal Planner
  - Recipes
  - Shopping List
  - Inventory
  - Settings

Home/Dashboard Screen:

- Current week meal plan preview (mini calendar)
- Quick actions: "Plan This Week", "View Shopping List", "Add Inventory"
- Upcoming expiry alerts (if any)
- Recent activity feed
- Nutritional summary for current week (collapsible)

Meal Planner Screen:

- Week selector (Previous/This/Next week navigation)
- Meal plan grid (days × meal types)
- Action buttons: "Generate Plan", "Get Feedback", "Finalize Plan"
- Profile filter toggle
- Nutritional summary panel (collapsible, per profile)

Recipe Library Screen:

- Search bar at top
- Filter/sort controls
- Grid or list view toggle
- Recipe cards with: image, name, rating, prep time, tags
- Floating "+" button to add recipe
- Categories/tabs: All, Favorites, Recent, By Cuisine

Recipe Detail Screen:

- Hero image
- Recipe name and rating (editable inline)
- Metadata: servings, prep/cook time, difficulty, cuisine
- Ingredients list (with checkboxes for shopping)
- Instructions (numbered steps)
- Nutritional info panel
- Actions: Edit, Delete, Add to Plan, Share, Favorite toggle
- Portion calculator slider

Shopping List Screen:

- Category sections (collapsible)
- Items with checkboxes
- Search bar
- Filter controls: All/Purchased/Unpurchased
- Actions: "Start Shopping", "Add Item", "Edit Categories"
- Progress indicator (% complete)
- Share/Export button

Inventory Screen:

- Filter tabs: All, Fridge, Freezer, Pantry, Expiring Soon
- Search bar
- Sort controls
- Items list with: name, quantity, location, expiry date
- Color coding for expiry status (green/yellow/red)
- Actions: "Add Item", "Add Photo", "Clean Expired"
- Quick actions per item: Edit, Move, Use, Delete

Profile Management Screen:

- List of family profiles (card-based)
- Each card shows: name, age, dietary summary
- Actions: Add Profile, Edit, Delete

- Profile detail form with tabs: Basic Info, Dietary Preferences, Meal Availability, Nutritional Targets

Settings Screen:

- Sections: Account, Meal Planning, Shopping, Notifications, AI Preferences, Help
- Each section expands to show relevant settings
- Save button at bottom (or auto-save on change)

5.3 Responsive Design Requirements

- Mobile-first design approach
- Breakpoints:
  - Mobile: < 768px (stacked layouts, bottom navigation)
  - Tablet: 768px - 1024px (side navigation, multi-column where appropriate)
  - Desktop: > 1024px (full side navigation, multi-panel layouts)
- Touch-friendly targets (min 44×44px for buttons/checkboxes)
- Swipe gestures on mobile (swipe to delete, swipe between weeks)
- Keyboard shortcuts on desktop (e.g., "N" for new recipe, "/" for search)

---

6. Data Models & Database Schema

6.1 Entity Relationship Overview

```
User (1) ──< (M) FamilyProfile
User (1) ──< (M) Recipe
User (1) ──< (M) MealPlan
User (1) ──< (M) ShoppingList
User (1) ──< (M) InventoryItem
User (1) ──< (M) WeeklyStaple

MealPlan (1) ──< (M) Meal
Meal (M) ──< (1) Recipe
Meal (M) ──< (M) FamilyProfile (many-to-many through participation)

Recipe (1) ──< (M) RecipeIngredient
Recipe (1) ──< (M) RecipeInstruction
Recipe (M) ──< (M) FamilyProfile (many-to-many for compatibility)

ShoppingList (1) ──< (M) ShoppingListItem
```

6.2 Database Tables (PostgreSQL)

Table: users

```sql
CREATE TABLE users (
  user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  last_login TIMESTAMP,
  preferences JSONB DEFAULT '{}'::jsonb
);
```

Table: family_profiles

```sql
CREATE TABLE family_profiles (
  profile_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(user_id) ON DELETE CASCADE,
  profile_name VARCHAR(50) NOT NULL,
  age INTEGER CHECK (age >= 0 AND age <= 120),
  avatar_url VARCHAR(500),
  food_likes TEXT[],
  food_dislikes TEXT[],
  allergies JSONB DEFAULT '[]'::jsonb, -- [{name: string, severity: string}]
  meal_availability JSONB DEFAULT '{}'::jsonb, -- {day: {meal: boolean}}
  activity_level VARCHAR(50),
  daily_calorie_target INTEGER,
  daily_protein_target DECIMAL(6,2),
  daily_carbs_target DECIMAL(6,2),
  daily_fat_target DECIMAL(6,2),
  daily_fiber_target DECIMAL(6,2),
  macro_tracking_enabled BOOLEAN DEFAULT false,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(user_id, profile_name)
);
CREATE INDEX idx_family_profiles_user_id ON family_profiles(user_id);
```

Table: recipes

```sql
```

```sql
CREATE TABLE recipes (
  recipe_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(user_id) ON DELETE CASCADE,
  recipe_name VARCHAR(100) NOT NULL,
  description TEXT,
  image_url VARCHAR(500),
  servings INTEGER NOT NULL DEFAULT 4 CHECK (servings > 0),
  prep_time_minutes INTEGER,
  cook_time_minutes INTEGER,
  total_time_minutes INTEGER GENERATED ALWAYS AS (prep_time_minutes + cook_time_minutes) STORED,
  cuisine_type VARCHAR(50),
  meal_category TEXT[], -- ['Breakfast', 'Lunch', 'Dinner', 'Snack']
  difficulty_level VARCHAR(20),
  recipe_source VARCHAR(50),
  source_url VARCHAR(1000),
  times_used INTEGER DEFAULT 0,
  times_manually_selected INTEGER DEFAULT 0,
  last_used_date TIMESTAMP,
  family_rating DECIMAL(3,1) CHECK (family_rating >= 1 AND family_rating <= 10),
  rating_date TIMESTAMP,
  notes TEXT,
  tags TEXT[],
  yields_multiple_meals BOOLEAN DEFAULT false,
  meals_yielded INTEGER,
  leftover_instructions TEXT,
  freezable BOOLEAN DEFAULT false,
  reheating_instructions TEXT,
  calories_per_serving INTEGER,
  protein_per_serving DECIMAL(6,2),
  carbs_per_serving DECIMAL(6,2),
  fat_per_serving DECIMAL(6,2),
  fiber_per_serving DECIMAL(6,2),
  sugar_per_serving DECIMAL(6,2),
  sodium_per_serving INTEGER,
  nutrition_auto_calculated BOOLEAN DEFAULT false,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  is_archived BOOLEAN DEFAULT false
);
CREATE INDEX idx_recipes_user_id ON recipes(user_id);
CREATE INDEX idx_recipes_rating ON recipes(family_rating);
CREATE INDEX idx_recipes_times_used ON recipes(times_used);
CREATE INDEX idx_recipes_meal_category ON recipes USING GIN(meal_category);
```

Table: recipe_ingredients

```sql
CREATE TABLE recipe_ingredients (
  ingredient_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  recipe_id UUID REFERENCES recipes(recipe_id) ON DELETE CASCADE,
  ingredient_name VARCHAR(100) NOT NULL,
  quantity DECIMAL(10,3) NOT NULL CHECK (quantity > 0),
  unit VARCHAR(20) NOT NULL,
  category VARCHAR(50),
  notes TEXT,
  sort_order INTEGER DEFAULT 0
);
CREATE INDEX idx_recipe_ingredients_recipe_id ON recipe_ingredients(recipe_id);
```

Table: recipe_instructions

```sql
CREATE TABLE recipe_instructions (
  instruction_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  recipe_id UUID REFERENCES recipes(recipe_id) ON DELETE CASCADE,
  step_number INTEGER NOT NULL,
  instruction TEXT NOT NULL,
  timer_minutes INTEGER,
  sort_order INTEGER DEFAULT 0
);
CREATE INDEX idx_recipe_instructions_recipe_id ON recipe_instructions(recipe_id);
```

Table: recipe_profile_compatibility

```sql
CREATE TABLE recipe_profile_compatibility (
  recipe_id UUID REFERENCES recipes(recipe_id) ON DELETE CASCADE,
  profile_id UUID REFERENCES family_profiles(profile_id) ON DELETE CASCADE,
  is_compatible BOOLEAN DEFAULT true,
  incompatible_ingredients TEXT[],
  PRIMARY KEY (recipe_id, profile_id)
);
```

Table: meal_plans

```sql
```

```
CREATE TABLE meal_plans (
  meal_plan_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(user_id) ON DELETE CASCADE,
  week_start_date DATE NOT NULL,
  week_end_date DATE NOT NULL,
  status VARCHAR(20) DEFAULT 'Draft' CHECK (status IN ('Draft', 'Finalized', 'Archived')),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  finalized_at TIMESTAMP,
  weekly_nutritional_summary JSONB DEFAULT '{}'::jsonb
);
CREATE INDEX idx_meal_plans_user_id ON meal_plans(user_id);
CREATE INDEX idx_meal_plans_week ON meal_plans(week_start_date, week_end_date);
```

Table: meals

sql

```
CREATE TABLE meals (
  meal_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  meal_plan_id UUID REFERENCES meal_plans(meal_plan_id) ON DELETE CASCADE,
  day_of_week VARCHAR(10) NOT NULL CHECK (day_of_week IN ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday
  meal_type VARCHAR(20) NOT NULL CHECK (meal_type IN ('Breakfast', 'Lunch', 'Dinner', 'Snack')),
  recipe_id UUID REFERENCES recipes(recipe_id) ON DELETE SET NULL,
  recipe_name VARCHAR(100),
  servings INTEGER CHECK (servings > 0),
  scaling_factor DECIMAL(5,2),
  is_leftover BOOLEAN DEFAULT false,
  leftover_from_meal_id UUID REFERENCES meals(meal_id) ON DELETE SET NULL,
  notes TEXT,
  is_locked BOOLEAN DEFAULT false,
  nutritional_summary JSONB DEFAULT '{}'::jsonb,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_meals_meal_plan_id ON meals(meal_plan_id);
CREATE INDEX idx_meals_recipe_id ON meals(recipe_id);
```

Table: meal_participants

sql

```
CREATE TABLE meal_participants (
  meal_id UUID REFERENCES meals(meal_id) ON DELETE CASCADE,
  profile_id UUID REFERENCES family_profiles(profile_id) ON DELETE CASCADE,
  portion_size DECIMAL(4,2) DEFAULT 1.0,
  PRIMARY KEY (meal_id, profile_id)
);
```

Table: weekly_staples

sql

```
CREATE TABLE weekly_staples (
  staple_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(user_id) ON DELETE CASCADE,
  item_name VARCHAR(100) NOT NULL,
  quantity DECIMAL(10,3) NOT NULL CHECK (quantity > 0),
  unit VARCHAR(20) NOT NULL,
  category VARCHAR(50),
  auto_add_to_list BOOLEAN DEFAULT true,
  notes TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_weekly_staples_user_id ON weekly_staples(user_id);
```

Table: inventory_items

sql

```sql
CREATE TABLE inventory_items (
  inventory_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(user_id) ON DELETE CASCADE,
  item_name VARCHAR(100) NOT NULL,
  quantity DECIMAL(10,3) NOT NULL CHECK (quantity > 0),
  unit VARCHAR(20) NOT NULL,
  category VARCHAR(50),
  location VARCHAR(50) CHECK (location IN ('Fridge', 'Freezer', 'Pantry', 'Custom')),
  expiry_date DATE,
  auto_populated_expiry BOOLEAN DEFAULT false,
  added_by VARCHAR(50) DEFAULT 'Manual',
  notes TEXT,
  is_used_in_planned_meal BOOLEAN DEFAULT false,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_inventory_user_id ON inventory_items(user_id);
CREATE INDEX idx_inventory_expiry ON inventory_items(expiry_date);
```

Table: shopping_lists

sql

```sql
CREATE TABLE shopping_lists (
  shopping_list_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(user_id) ON DELETE CASCADE,
  meal_plan_id UUID REFERENCES meal_plans(meal_plan_id) ON DELETE SET NULL,
  week_start_date DATE NOT NULL,
  status VARCHAR(20) DEFAULT 'Generated' CHECK (status IN ('Generated', 'In Progress', 'Completed')),
  category_order TEXT[],
  generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  completed_at TIMESTAMP
);
CREATE INDEX idx_shopping_lists_user_id ON shopping_lists(user_id);
CREATE INDEX idx_shopping_lists_meal_plan ON shopping_lists(meal_plan_id);
```

Table: shopping_list_items

sql

```
CREATE TABLE shopping_list_items (
  item_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  shopping_list_id UUID REFERENCES shopping_lists(shopping_list_id) ON DELETE CASCADE,
  item_name VARCHAR(100) NOT NULL,
  quantity DECIMAL(10,3) NOT NULL CHECK (quantity > 0),
  unit VARCHAR(20) NOT NULL,
  category VARCHAR(50),
  source VARCHAR(50), -- 'Recipe', 'Staple', 'Manual', 'Multiple'
  source_details JSONB DEFAULT '[]'::jsonb,
  is_consolidated BOOLEAN DEFAULT false,
  in_inventory BOOLEAN DEFAULT false,
  inventory_quantity DECIMAL(10,3),
  net_quantity_needed DECIMAL(10,3),
  is_purchased BOOLEAN DEFAULT false,
  custom_note TEXT,
  priority VARCHAR(20) DEFAULT 'Medium' CHECK (priority IN ('High', 'Medium', 'Low')),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_shopping_list_items_list_id ON shopping_list_items(shopping_list_id);
```

Table: app_settings

```
sql

CREATE TABLE app_settings (
  user_id UUID PRIMARY KEY REFERENCES users(user_id) ON DELETE CASCADE,
  theme VARCHAR(20) DEFAULT 'Auto',
  notifications_enabled BOOLEAN DEFAULT true,
  notification_preferences JSONB DEFAULT '{}'::jsonb,
  meal_planning_preferences JSONB DEFAULT '{}'::jsonb,
  shopping_list_preferences JSONB DEFAULT '{}'::jsonb,
  ai_preferences JSONB DEFAULT '{}'::jsonb,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6.3 Data Integrity & Constraints

- All foreign key relationships enforce CASCADE deletes where appropriate
- CHECK constraints on enums and value ranges
- UNIQUE constraints on profile names within user
- NOT NULL constraints on required fields
- Indexes on foreign keys and frequently queried fields
- JSONB fields for flexible nested data (preferences, nutritional summaries)

6.4 Database Migrations

- Use migration tool: Prisma, TypeORM, or Sequelize with migration support
- Version control all migrations
- Naming convention: `YYYYMMDDHHMMSS_description.sql`
- Include rollback scripts for each migration
- Test migrations on staging before production

---

7. API Specifications

7.1 API Architecture

- RESTful API design
- Base URL: `https://api.familymealplanner.com/v1` (example)
- Authentication: JWT tokens in Authorization header
- Request/Response format: JSON
- Rate limiting: 1000 requests per hour per user
- Error responses: Standard HTTP status codes with error details

7.2 Authentication Endpoints

POST /auth/register

- Request: `{email, password, confirmPassword}`
- Response: `{userId, email, token}`
- Creates new user account

POST /auth/login

- Request: `{email, password}`
- Response: `{userId, email, token, expiresIn}`
- Authenticates user and returns JWT

POST /auth/logout

- Request: JWT in header
- Response: `{success: true}`
- Invalidates current token

POST /auth/refresh-token

- Request: `{refreshToken}`
- Response: `{token, expiresIn}`
- Issues new JWT

POST /auth/forgot-password

- Request: `{email}`
- Response: `{success: true, message}`
- Sends password reset email

POST /auth/reset-password

- Request: `{token, newPassword}`
- Response: `{success: true}`
- Resets password with token from email

7.3 Profile Endpoints

GET /profiles

- Response: Array of family profiles
- Query params: `?include=availability,macros`

POST /profiles

- Request: Profile object (see data model)
- Response: Created profile with ID

GET /profiles/:profileId

- Response: Single profile with full details

PUT /profiles/:profileId

- Request: Partial profile object (fields to update)
- Response: Updated profile

DELETE /profiles/:profileId

- Response: `{success: true}`
- Soft deletes profile (archives)

7.4 Recipe Endpoints

GET /recipes

- Response: Paginated array of recipes
- Query params: `?page=1&limit=20&search=keyword&cuisine=Italian&rating=>8&sort=rating_desc`

POST /recipes

- Request: Recipe object with ingredients and instructions
- Response: Created recipe with ID

GET /recipes/:recipeId

- Response: Full recipe details including ingredients, instructions, nutrition

PUT /recipes/:recipeId

- Request: Partial recipe object
- Response: Updated recipe

DELETE /recipes/:recipeId

- Response: `{success: true}`
- Archives recipe

POST /recipes/import-url

- Request: `{url: string}`
- Response: Parsed recipe object (for user review)
- Uses Claude API to parse

POST /recipes/import-photo

- Request: `{imageFile: multipart/form-data}`
- Response: `{dishName, ingredients[], cuisine, confidence}`
- Uses Claude Vision API

POST /recipes/:recipeId/rate

- Request: `{rating: number (1-10), notes: string}`
- Response: Updated recipe with new rating

GET /recipes/suggestions

- Response: Array of suggested recipes based on user preferences
- Query params: `?type=expiring_inventory|favorites|similar_to=recipeId`

7.5 Meal Plan Endpoints

GET /meal-plans

- Response: Array of meal plans (current, past)
- Query params: `?status=Finalized&week=2024-W42`

POST /meal-plans

- Request: `{weekStartDate: date}`
- Response: New meal plan with ID (empty/draft status)

GET /meal-plans/:mealPlanId

- Response: Full meal plan with all meals and nutritional summaries

PUT /meal-plans/:mealPlanId

- Request: Partial meal plan updates
- Response: Updated meal plan

POST /meal-plans/:mealPlanId/generate

- Request: Optional `{varietyLevel: string, lockedMealIds: string[]}`
- Response: Auto-generated meal plan
- Uses Claude API

POST /meal-plans/:mealPlanId/finalize

- Response: Finalized meal plan + generated shopping list ID

POST /meal-plans/:mealPlanId/nutritionist-feedback

- Response: AI nutritionist analysis (see data model)
- Uses Claude API

POST /meal-plans/:mealPlanId/meals

- Request: Meal object
- Response: Created meal with ID

PUT /meal-plans/:mealPlanId/meals/:mealId

- Request: Partial meal updates
- Response: Updated meal

DELETE /meal-plans/:mealPlanId/meals/:mealId

- Response: `{success: true}`

POST /meal-plans/:mealPlanId/duplicate-week

- Request: `{targetWeekStartDate: date}`
- Response: New meal plan (copy of specified week)

7.6 Shopping List Endpoints

GET /shopping-lists

- Response: Array of shopping lists
- Query params: `?status=Generated&weekStartDate=2024-11-18`

GET /shopping-lists/:shoppingListId

- Response: Full shopping list with all items

POST /shopping-lists/:shoppingListId/items

- Request: `{itemName, quantity, unit, category, source: 'Manual'}`
- Response: Created shopping list item

PUT /shopping-lists/:shoppingListId/items/:itemId

- Request: Partial item updates (e.g., `{isPurchased: true}`)
- Response: Updated item

DELETE /shopping-lists/:shoppingListId/items/:itemId

- Response: `{success: true}`

POST /shopping-lists/:shoppingListId/complete

- Request: Optional `{addToInventory: boolean}`
- Response: `{success: true, addedToInventory: number}`
- Marks shopping complete, optionally bulk-adds to inventory

PUT /shopping-lists/:shoppingListId/category-order

- Request: `{categoryOrder: string[]}`

- Response: Updated shopping list

GET /shopping-lists/:shoppingListId/export

- Query params: `?format=pdf|text|csv`
- Response: File download or text content

7.7 Inventory Endpoints

GET /inventory

- Response: Array of inventory items
- Query params: `?location=Fridge&expiring=true&category=Produce`

POST /inventory

- Request: Inventory item object
- Response: Created inventory item with ID

POST /inventory/bulk-add

- Request: `{items: InventoryItem[]}`
- Response: `{created: number, errors: []}`
- For bulk adding after shopping

POST /inventory/photo-recognition

- Request: `{imageFile: multipart/form-data}`
- Response: `{items: InventoryItem[], confidence: number}`
- Uses Claude Vision API

PUT /inventory/:inventoryId

- Request: Partial inventory item updates
- Response: Updated inventory item

DELETE /inventory/:inventoryId

- Response: `{success: true}`

POST /inventory/use-item

- Request: `{inventoryId, quantityUsed}`
- Response: Updated inventory item (decremented quantity or deleted if 0)

DELETE /inventory/clear-expired

- Response: `{deletedCount: number}`
- Bulk deletes expired items

7.8 Staples Endpoints

GET /staples

- Response: Array of weekly staples

POST /staples

- Request: Staple object
- Response: Created staple with ID

PUT /staples/:stapleId

- Request: Partial staple updates
- Response: Updated staple

DELETE /staples/:stapleId

- Response: `{success: true}`

POST /staples/bulk-add

- Request: `{template: string}` (e.g., "Basic Pantry")
- Response: `{created: number}`
- Adds preset list of common staples

7.9 Settings Endpoints

GET /settings

- Response: User settings object

PUT /settings

- Request: Partial settings updates
- Response: Updated settings

7.10 AI Service Endpoints (Internal)

These endpoints wrap Claude API calls:

POST /ai/generate-meal-plan

- Request: Context data (profiles, recipes, inventory, preferences)
- Response: Generated meal plan structure
- Called by `/meal-plans/:id/generate`

POST /ai/nutritionist-feedback

- Request: Meal plan data
- Response: Nutritionist analysis
- Called by `/meal-plans/:id/nutritionist-feedback`

POST /ai/parse-recipe-url

- Request: `{url, html}`
- Response: Parsed recipe data
- Called by `/recipes/import-url`

POST /ai/identify-food-photo

- Request: `{imageBase64}`
- Response: Identified items array
- Called by `/inventory/photo-recognition`

POST /ai/estimate-macros

- Request: `{ingredients, servings}`
- Response: Nutritional breakdown
- Called when saving recipe without nutrition info

7.11 Error Response Format

```json
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Recipe name is required",
    "details": {
      "field": "recipe_name",
      "constraint": "NOT NULL"
    }
  }
}
```

Common Error Codes:

- `AUTHENTICATION_FAILED` (401)
- `FORBIDDEN` (403)
- `NOT_FOUND` (404)
- `VALIDATION_ERROR` (400)
- `RATE_LIMIT_EXCEEDED` (429)
- `INTERNAL_SERVER_ERROR` (500)
- `AI_SERVICE_ERROR` (502)

---

8. Non-Functional Requirements

8.1 Performance Requirements

- Page load time: < 2 seconds for main screens
- API response time: < 500ms for 95% of requests
- AI generation response: < 10 seconds for meal plan generation
- Photo recognition: < 5 seconds per image
- Database queries: < 100ms for simple queries, < 1s for complex aggregations
- Support 10,000 concurrent users without degradation

8.2 Security Requirements

- HTTPS only for all connections
- JWT tokens expire after 24 hours
- Passwords hashed with bcrypt (cost factor 12)
- SQL injection prevention via parameterized queries
- XSS protection via input sanitization and CSP headers
- CSRF tokens for state-changing operations
- Rate limiting on authentication endpoints (5 attempts per minute)
- File upload restrictions: max 5MB, allowed types only
- Regular security audits and dependency updates
- PII encryption at rest (for sensitive profile data)

8.3 Scalability Requirements

- Horizontal scaling for API servers (stateless design)
- Database connection pooling
- Caching layer (Redis) for frequently accessed data:
  - Recipe lists
  - User settings
  - Ingredient categories
- CDN for static assets and images
- Async processing for heavy AI operations (queue-based)
- Database read replicas for scaling reads

8.4 Availability & Reliability

- 99.5% uptime target
- Automated health checks every 60 seconds
- Graceful degradation if AI service unavailable (manual mode)
- Database backups: daily full, hourly incremental
- Disaster recovery plan with RTO < 4 hours, RPO < 1 hour
- Error logging and monitoring (Sentry, DataDog, or similar)
- Alerting for critical errors and downtime

8.5 Usability Requirements

- Mobile-responsive design on all screen sizes
- Accessibility: WCAG 2.1 Level AA compliance
  - Keyboard navigation support
  - Screen reader compatible
  - Color contrast ratios meet standards
  - Alt text for all images
- Internationalization support (i18n framework)
  - English as primary language
  - UI strings externalized for future translations
- Maximum 3 clicks to reach any feature from home screen

- Intuitive icon usage with tooltips
- Consistent design language across all screens

8.6 Compatibility Requirements

- **Mobile:** iOS 14+, Android 10+
- **Web Browsers:** Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- **Screen Sizes:** 320px (small mobile) to 2560px (desktop)
- **Network:** Works on 3G connections (graceful degradation)
- **Offline:** Basic functionality available offline (view cached data)

8.7 Maintainability Requirements

- Code coverage: minimum 70% unit test coverage
- Documentation: inline comments, API documentation (Swagger/OpenAPI)
- Git workflow: feature branches, pull requests, code reviews
- CI/CD pipeline: automated testing and deployment
- Linting and code formatting: ESLint, Prettier
- Semantic versioning for releases
- Changelog maintained for all releases

8.8 Data Privacy & Compliance

- GDPR compliance (for EU users):
  - Data portability (export feature)
  - Right to deletion (account deletion)
  - Privacy policy and terms of service
  - Cookie consent banner
- CCPA compliance (for California users)
- No third-party data sharing without explicit consent
- User data anonymization in analytics
- Audit logs for data access and modifications

---

9. Testing Requirements

9.1 Unit Testing

- All business logic functions
- Database models and queries
- API endpoints (request/response validation)
- AI prompt parsing and response handling
- Utility functions (date calculations, unit conversions, etc.)
- Target: 70%+ code coverage

9.2 Integration Testing

- End-to-end user flows (onboarding, meal planning, shopping)
- API endpoint integration with database
- AI service integration (mocked and live)
- Authentication and authorization flows
- Image upload and processing
- Data export/import functions

9.3 UI/UX Testing

- Cross-browser testing (Chrome, Firefox, Safari, Edge)
- Responsive design testing (mobile, tablet, desktop)
- Accessibility testing (keyboard nav, screen readers)
- Usability testing with real users (5-10 participants)
- A/B testing for key features (meal plan generation UI, shopping list layout)

9.4 Performance Testing

- Load testing: simulate 1000 concurrent users
- Stress testing: identify breaking point
- API response time benchmarking
- Database query optimization validation
- Image upload and processing performance

9.5 Security Testing

- Penetration testing (automated and manual)
- Authentication/authorization testing
- SQL injection testing
- XSS vulnerability testing
- CSRF protection testing
- File upload security testing

9.6 AI Testing

- Meal plan generation quality (manual review of 50+ generated plans)
- Recipe parsing accuracy from URLs (test with 100+ URLs)
- Photo recognition accuracy (test with diverse images)
- Nutritionist feedback relevance (qualitative assessment)
- Macro calculation accuracy (compare to known values)

9.7 User Acceptance Testing (UAT)

- Beta testing with 20-50 real families
- Feedback collection via in-app surveys
- Bug reporting mechanism
- Feature request tracking
- Iterative improvements based on feedback

---

10. Deployment & DevOps

10.1 Environment Setup

- **Development:** Local machines with Docker containers
- **Staging:** Cloud environment mirroring production
- **Production:** Scalable cloud infrastructure (AWS, GCP, or Azure)

10.2 CI/CD Pipeline

- **Version Control:** Git (GitHub, GitLab, or Bitbucket)
- **CI Tool:** GitHub Actions, GitLab CI, or CircleCI

- **Pipeline Stages:**
  1. Lint and format check
  2. Unit tests
  3. Integration tests
  4. Build Docker images
  5. Push to container registry
  6. Deploy to staging (automatic on main branch)
  7. Run smoke tests on staging
  8. Manual approval for production deploy
  9. Deploy to production
  10. Post-deployment health checks

10.3 Infrastructure as Code

- Terraform or CloudFormation for infrastructure provisioning
- Docker Compose for local development
- Kubernetes or ECS for container orchestration (production)

10.4 Monitoring & Logging

- **Application Monitoring:** Datadog, New Relic, or Application Insights
- **Error Tracking:** Sentry
- **Logging:** Centralized logging (ELK stack, CloudWatch, or similar)
- **Metrics:** Custom dashboards for key metrics (sign-ups, meal plans created, AI API usage)
- **Alerts:** Email/Slack notifications for critical issues

10.5 Backup & Disaster Recovery

- Database backups: automated daily full backup, retained for 30 days
- Incremental backups every 6 hours
- Backup storage in geographically separate region
- Disaster recovery runbook with step-by-step restoration process
- Quarterly DR drills

10.6 Scaling Strategy

- Auto-scaling for API servers based on CPU/memory usage

- Database vertical scaling (upgrade instance size) for initial growth
- Database horizontal scaling (read replicas) for read-heavy workload
- CDN for static content delivery
- Caching layer (Redis) for frequently accessed data
- Queue system (RabbitMQ, SQS) for async AI processing

---

11. Future Enhancements & Roadmap

Phase 1: MVP (Months 1-3)

- Core functionality: profiles, recipes, meal planning, shopping lists, inventory
- Basic AI integration: meal plan generation, nutritionist feedback
- Web and mobile apps (basic UI)
- Manual recipe entry and simple URL import

Phase 2: Enhanced AI & Usability (Months 4-6)

- Advanced AI: photo recognition for inventory and recipes
- Recipe suggestions and discovery
- Meal planning templates
- Improved UI/UX based on beta feedback
- Onboarding improvements

Phase 3: Advanced Features (Months 7-9)

- Cost tracking and budget management
- Grocery delivery integration (Instacart, etc.)
- Meal prep focus with batch cooking tools
- Social features: share recipes, community recipe library
- Advanced analytics and insights dashboard

Phase 4: Expansion (Months 10-12)

- Multi-language support
- Regional customization (local ingredient availability)
- Dietary supplements and meal timing optimization

- Integration with fitness trackers (Apple Health, Google Fit)
- AI meal coach (conversational interface)

Phase 5: Ecosystem (Year 2+)

- Smart kitchen appliance integration
- Voice assistant integration (Alexa, Google Assistant)
- Meal kit service partnerships
- Nutritionist marketplace (connect with professionals)
- Family wellness features (sleep tracking, mood logging)

---

12. Success Criteria & KPIs

Launch Success Metrics (First 3 Months)

- 1,000 active users (weekly active)
- 70% user retention after 4 weeks
- 80% meal plan completion rate (finalized vs. started)
- Average recipe rating > 7/10
- AI meal plan acceptance rate > 75%
- < 5% critical bug rate
- Average session duration > 10 minutes

User Engagement Metrics

- Number of meal plans created per user per month
- Number of recipes added per user
- Shopping list completion rate
- Inventory items tracked per user
- AI feature usage rate (% of users using AI features)

Technical Performance Metrics

- API uptime > 99.5%
- Average API response time < 500ms
- AI service success rate > 95%

- Photo recognition accuracy > 80%
- Database query performance (P95 < 1s)

Business Metrics (If Applicable)

- User acquisition cost (CAC)
- Customer lifetime value (CLV)
- Monthly recurring revenue (MRR) if subscription model
- Conversion rate (free to paid, if applicable)
- Net Promoter Score (NPS) > 40

---

13. Risks & Mitigation Strategies

Technical Risks

1. **AI Service Reliability:**
   - Risk: Claude API downtime or rate limiting
   - Mitigation: Graceful degradation to manual mode, caching, queue-based retry logic
2. **Database Performance:**
   - Risk: Slow queries as data grows
   - Mitigation: Proper indexing, query optimization, read replicas, caching
3. **Photo Recognition Accuracy:**
   - Risk: Low accuracy leading to poor user experience
   - Mitigation: User review step, confidence thresholds, manual fallback

User Experience Risks

1. **Complexity Overload:**
   - Risk: Too many features confusing new users
   - Mitigation: Progressive disclosure, excellent onboarding, contextual help
2. **AI Trust:**
   - Risk: Users don't trust AI-generated meal plans
   - Mitigation: Transparency, allow full control, show reasoning, gradual adoption

Business Risks

1. **Market Fit:**

- Risk: Product doesn't solve real user pain points
- Mitigation: Beta testing, user feedback loops, iterative development

2. **Competition:**
   - Risk: Established competitors or new entrants
   - Mitigation: Focus on AI differentiation, excellent UX, community building

Data Privacy Risks

1. **Data Breach:**
   - Risk: User data compromised
   - Mitigation: Security best practices, encryption, regular audits, insurance
2. **Compliance Violations:**
   - Risk: GDPR/CCPA violations
   - Mitigation: Legal review, privacy-by-design, clear policies, user controls

---

14. Glossary

Family Profile: Individual person within a household account, with unique dietary preferences and nutritional goals.

Meal Plan: Weekly schedule of meals (breakfast, lunch, dinner, snacks) assigned to family profiles.

Recipe Library: Collection of recipes stored by the user, including ingredients, instructions, and ratings.

Shopping List: Consolidated list of ingredients needed for the week's meal plan, plus weekly staples and manual additions.

Inventory: Current stock of food items in the household (fridge, freezer, pantry).

Weekly Staples: Regular grocery items purchased every week (e.g., milk, bread, eggs).

Macros (Macronutrients): Protein, carbohydrates, and fat (plus fiber) tracked for nutritional goals.

Leftover: Meal using food from a previous meal prep recipe.

Meal Prep: Recipes that yield multiple servings for future meals.

AI Nutritionist: Claude-powered feature providing feedback on meal plan nutritional quality.

Portion Scaling: Adjusting recipe quantities based on number of servings needed.

Recipe Rating: Family's 1-10 score for a recipe based on enjoyment.

Recipe Source: Origin of recipe (Manual, URL Import, Photo Recognition).

Cuisine Type: Category of recipe (Italian, Mexican, Asian, etc.).

Meal Category: When recipe is typically eaten (Breakfast, Lunch, Dinner, Snack).

Finalize Plan: Action to lock meal plan and generate shopping list.

Shopping Mode: UI optimized for checking off items while shopping.

Category Order: Custom organization of shopping list by store layout.

Expiry Date: Date by which inventory item should be consumed.

Net Quantity Needed: Amount to purchase after subtracting inventory on hand.

Consolidated Item: Shopping list item combining quantities from multiple recipes.

---

15. Appendices

Appendix A: Sample Data

Sample Family Profile:

json

```
{
  "profile_name": "Sarah",
  "age": 35,
  "food_likes": ["chicken", "broccoli", "pasta", "salmon"],
  "food_dislikes": ["mushrooms", "olives"],
  "allergies": [
    {"name": "peanuts", "severity": "severe"}
  ],
  "meal_availability": {
    "Monday": {"Breakfast": true, "Lunch": false, "Dinner": true, "Snack": false},
    "Tuesday": {"Breakfast": true, "Lunch": false, "Dinner": true, "Snack": false}
    // ... rest of week
  },
  "activity_level": "Moderately Active",
  "daily_calorie_target": 1800,
  "daily_protein_target": 120,
  "daily_carbs_target": 180,
  "daily_fat_target": 60,
  "daily_fiber_target": 25,
  "macro_tracking_enabled": true
}
```

Sample Recipe:

```json
{
  "recipe_name": "Grilled Chicken Caesar Salad",
  "description": "Healthy and satisfying salad with grilled chicken",
  "servings": 4,
  "prep_time_minutes": 15,
  "cook_time_minutes": 20,
  "cuisine_type": "American",
  "meal_category": ["Lunch", "Dinner"],
  "difficulty_level": "Easy",
  "family_rating": 8.5,
  "ingredients": [
    {"name": "chicken breast", "quantity": 500, "unit": "g", "category": "Meat"},
    {"name": "romaine lettuce", "quantity": 2, "unit": "heads", "category": "Produce"},
    {"name": "caesar dressing", "quantity": 150, "unit": "ml", "category": "Pantry"},
    {"name": "parmesan cheese", "quantity": 50, "unit": "g", "category": "Dairy"}
  ],
  "instructions": [
    {"step_number": 1, "instruction": "Season chicken with salt and pepper"},
    {"step_number": 2, "instruction": "Grill chicken for 8-10 minutes per side", "timer_minutes": 20},
    {"step_number": 3, "instruction": "Chop romaine lettuce and place in large bowl"},
    {"step_number": 4, "instruction": "Slice grilled chicken and add to salad"},
    {"step_number": 5, "instruction": "Drizzle with caesar dressing and top with parmesan"}
  ],
  "calories_per_serving": 380,
  "protein_per_serving": 42,
  "carbs_per_serving": 12,
  "fat_per_serving": 18,
  "fiber_per_serving": 3
}
```

Appendix B: AI Prompt Examples

Meal Plan Generation Prompt (Condensed):

Generate a weekly meal plan for this family:

Profiles:
- Sarah (35, moderately active, 1800 cal/day, dislikes mushrooms)
- John (38, very active, 2200 cal/day, allergic to shellfish)
- Emma (8, lightly active, 1400 cal/day, picky eater)

Available Monday-Sunday:
- Sarah: Breakfast, Dinner
- John: Breakfast, Lunch, Dinner
- Emma: Breakfast, Dinner

Recipes (50 total, showing top rated):
1. Grilled Chicken Caesar (rating 8.5, used 3 times, last used 2 weeks ago)
2. Spaghetti Bolognese (rating 9, used 5 times, last used 1 week ago)
3. Salmon Teriyaki (rating 7.5, used 1 time, last used 4 weeks ago)
...

Inventory expiring soon:
- Chicken breast (expires in 2 days)
- Spinach (expires in 4 days)

Requirements:
- No recipe repeated in same week
- Prioritize recipes rated 8+
- Use expiring inventory
- Ensure Emma gets kid-friendly meals
- Balance macros for each person
- Assign leftovers where appropriate

Output: JSON meal plan with nutritional breakdowns per person per meal.

Appendix C: Unit Conversion Reference

Common conversions for metric system:

- 1 cup = 240 ml
- 1 tablespoon = 15 ml
- 1 teaspoon = 5 ml
- 1 lb = 453.6 g
- 1 oz = 28.35 g
- 1 quart = 0.95 L
- 1 gallon = 3.78 L

Implement conversion utility functions for shopping list consolidation.

Appendix D: Category Defaults

Ingredient Categories:

- Produce
- Dairy
- Meat
- Seafood
- Frozen
- Bakery
- Pantry
- Beverages
- Condiments
- Snacks
- Other

Default Shopping List Category Order:

1. Produce
2. Meat
3. Seafood
4. Dairy
5. Frozen
6. Bakery
7. Pantry
8. Beverages
9. Condiments
10. Snacks
11. Other

---

16. Sign-Off & Approvals

This PRD is ready for implementation by the development team. Any questions or ambiguities should be raised with the product owner for clarification before starting development.

Document Status: Ready for Development
Version: 1.0
Last Updated: December 5, 2025

END OF DOCUMENT