Recipe Discovery & Management Platform - Product Requirements Document

Executive Summary

A comprehensive recipe management platform that evolves from web-based recipe discovery to a full-featured family cooking assistant. The platform will be self-hosted on NAS infrastructure with eventual mobile native apps and browser extensions.

Vision Statement

Create a beautiful, intelligent recipe management ecosystem that transforms how families discover, organize, and cook recipes together, with complete data ownership through self-hosted infrastructure.

Phase 1: Recipe Discovery Web App

Objectives

- Build foundational recipe discovery and viewing capabilities
- Establish modern, responsive web interface
- Integrate external recipe APIs for content discovery

Core Features

Recipe Discovery Engine

- **Search & Browse**: Text search with real-time suggestions
- **Advanced Filters**:
  - Cuisine type (Italian, Mexican, Asian, etc.)
  - Dietary restrictions (Vegetarian, Vegan, Gluten-free, Keto, etc.)
  - Cook time (Under 15 min, 15-30 min, 30-60 min, 1+ hours)
  - Difficulty level (Beginner, Intermediate, Advanced)
  - Meal type (Breakfast, Lunch, Dinner, Snacks, Desserts)
  - Available ingredients (search by what you have)
  - Serving size range

Recipe Display

- **Rich Recipe Cards**: High-quality images, key metadata overlay
- **Detailed Recipe View**:
  - Hero image with fallback placeholder
  - Cook time, prep time, total time
  - Serving size with scaling capability
  - Ingredient list with quantities
  - Step-by-step instructions
  - Nutritional information (if available)
  - Source attribution and external link

User Interface

- **Responsive Design**: Mobile-first, tablet and desktop optimized
- **Dark/Light Theme**: System preference detection with manual toggle
- **Search History**: Recent searches and trending queries
- **Loading States**: Skeleton screens and progressive loading

Technical Specifications

Frontend

- **Framework**: React 18 with TypeScript
- **Styling**: Tailwind CSS with custom design system
- **State Management**: React Context + useReducer
- **Build Tool**: Vite for fast development and optimized builds
- **PWA Features**: Service worker for offline recipe viewing

API Integration

- **Primary API**: Spoonacular API
  - Endpoint: `/recipes/search` for discovery
  - Endpoint: `/recipes/{id}/information` for detailed recipes
  - Endpoint: `/recipes/random` for featured content
  - Rate Limiting: 150 requests/day (free tier)
- **Backup APIs**:
  - Edamam Recipe Search API
  - TheMealDB (free, limited content)

Data Models

```typescript
interface Recipe {
  id: string;
  title: string;
  image?: string;
  summary?: string;
  readyInMinutes: number;
  servings: number;
  cuisines: string[];
  diets: string[];
  ingredients: Ingredient[];
  instructions: Instruction[];
  nutrition?: NutritionInfo;
  sourceUrl?: string;
  sourceName?: string;
}

interface Ingredient {
  id: string;
  name: string;
  amount: number;
  unit: string;
  original: string; // Original text from API
}

interface Instruction {
  number: number;
  step: string;
  ingredients?: string[];
  equipment?: string[];
}
```

Success Metrics

- Recipe search functionality with <2s response time
- Responsive design across all device sizes
- 95%+ uptime during development phase
- Clean, accessible UI (WCAG 2.1 AA compliance)

Phase 2: Web Scraping & Database Integration

Objectives

- Add capability to extract recipes from any cooking website
- Implement local database for recipe storage
- Create web browsing interface within the app

Core Features

In-App Web Browser

- **Embedded Browser**: iframe-based browser with custom controls
- **URL Input**: Direct navigation to cooking websites
- **Recipe Detection**: Automatic detection of recipe content on pages
- **Extract Recipe Button**: One-click recipe extraction and save

Web Scraping Engine

- **Structured Data Parsing**:
  - JSON-LD structured data (Recipe schema)
  - Microdata markup parsing
  - OpenGraph meta tags
- **Fallback HTML Parsing**:
  - Common recipe site patterns (AllRecipes, Food Network, etc.)
  - AI-powered content extraction for unknown sites
  - Image URL extraction and validation

Recipe Metadata Schema

```typescript
interface RecipeMetadata {
  title: string;
  description?: string;
  image?: string;
  prepTime?: string; // ISO 8601 duration
  cookTime?: string;
  totalTime?: string;
  recipeYield?: string; // servings
  recipeCategory?: string[]; // meal type
  recipeCuisine?: string[];
  keywords?: string[];
  author?: {
    name: string;
    url?: string;
  };
  nutrition?: {
    calories?: string;
    protein?: string;
    carbs?: string;
    fat?: string;
    fiber?: string;
  };
  ingredients: string[];
  instructions: string[];
  sourceUrl: string;
  dateScraped: Date;
}
```

Database Implementation

- **Database**: SQLite with Prisma ORM
- **Tables**:
  - `recipes`: Core recipe data
  - `ingredients`: Normalized ingredient catalog
  - `recipe_ingredients`: Junction table with quantities
  - `instructions`: Step-by-step instructions
  - `tags`: Flexible tagging system
  - `scrape_sources`: Track scraped URLs to prevent duplicates

Local Recipe Management

- **Save Scraped Recipes**: Store extracted recipes locally
- **Edit Capabilities**: Modify scraped recipes (fix extraction errors)
- **Duplicate Detection**: Prevent saving same recipe multiple times
- **Search Local Recipes**: Full-text search across saved recipes
- **Export Options**: JSON, PDF recipe cards

Technical Specifications

Backend Architecture

- **Framework**: Node.js with Express
- **Database**: SQLite with Prisma
- **Web Scraping**:
  - Puppeteer for dynamic content
  - Cheerio for static HTML parsing
  - Custom recipe extraction algorithms

Scraping Strategy

1. **Structured Data First**: Parse JSON-LD, microdata
2. **Meta Tags**: Extract OpenGraph/Twitter cards
3. **Pattern Matching**: Site-specific selectors for major recipe sites
4. **AI Fallback**: Use text analysis for unknown sites
5. **Image Processing**: Download and store recipe images locally

API Endpoints

```
GET  /api/recipes          # List saved recipes
POST /api/recipes          # Save new recipe
GET  /api/recipes/:id      # Get specific recipe
PUT  /api/recipes/:id      # Update recipe
DELETE /api/recipes/:id    # Delete recipe
POST /api/scrape           # Scrape recipe from URL
GET  /api/scrape/preview   # Preview extraction without saving
```

Success Metrics

- Successfully extract recipes from 90%+ of major cooking websites
- Recipe extraction completes in <10 seconds
- Accurate ingredient and instruction parsing (95%+ accuracy)
- Local database supports 10,000+ recipes without performance degradation

---

Phase 3: Multi-User & Family Sharing

Objectives

- Add user authentication and profiles
- Enable recipe sharing between family members
- Implement collaborative features for family cooking

Core Features

User Management

- **User Registration/Login**: Email-based authentication
- **User Profiles**:
  - Display name and avatar
  - Dietary preferences and restrictions
  - Cooking skill level
  - Favorite cuisines
- **Family Groups**: Create and join family units
- **Permission Levels**: Admin, Member roles within families

Family Collaboration

- **Shared Recipe Collections**: Family recipe box accessible to all members
- **Personal Collections**: Private recipe saves per user
- **Recipe Sharing**: Share individual recipes between family members
- **Comments & Ratings**: Rate and comment on family recipes
- **Cooking History**: Track who cooked what and when

Social Features

- **Family Feed**: Recent recipe additions and cooking activities
- **Recipe Suggestions**: Recommend recipes based on family preferences
- **Cooking Assignments**: Assign recipes to family members for meal planning
- **Grocery List Collaboration**: Multiple family members can edit shopping lists

Technical Specifications

Authentication

- **Strategy**: JWT tokens with refresh token rotation
- **Session Management**: Redis for session storage
- **Password Security**: bcrypt hashing with salt rounds
- **Account Recovery**: Email-based password reset

Database Schema Updates

sql

```sql
-- Users table
CREATE TABLE users (
  id UUID PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  display_name VARCHAR(100),
  avatar_url VARCHAR(500),
  dietary_restrictions TEXT[], -- JSON array
  skill_level VARCHAR(20),
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Families table
CREATE TABLE families (
  id UUID PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  created_by UUID REFERENCES users(id),
  invite_code VARCHAR(20) UNIQUE,
  created_at TIMESTAMP DEFAULT NOW()
);

-- Family memberships
CREATE TABLE family_members (
  family_id UUID REFERENCES families(id),
  user_id UUID REFERENCES users(id),
  role VARCHAR(20) DEFAULT 'member', -- admin, member
  joined_at TIMESTAMP DEFAULT NOW(),
  PRIMARY KEY (family_id, user_id)
);

-- Recipe ownership and sharing
CREATE TABLE user_recipes (
  user_id UUID REFERENCES users(id),
  recipe_id UUID REFERENCES recipes(id),
  is_favorite BOOLEAN DEFAULT FALSE,
  personal_notes TEXT,
  rating INTEGER CHECK (rating >= 1 AND rating <= 5),
  last_cooked_at TIMESTAMP,
  times_cooked INTEGER DEFAULT 0,
  PRIMARY KEY (user_id, recipe_id)
);
```

API Security

- **Rate Limiting**: Per-user API rate limits
- **Input Validation**: Comprehensive request validation
- **CORS**: Proper CORS configuration for web app
- **Data Privacy**: User data isolation and family-based access control

Success Metrics

- Support 100+ concurrent family groups
- Sub-500ms response time for user operations
- 99.9% authentication uptime
- Zero unauthorized data access incidents

Phase 4: Dockerization & NAS Deployment

Objectives

- Containerize the entire application stack
- Enable easy deployment on NAS systems
- Ensure data persistence and backup capabilities

Core Features

Container Architecture

- **Multi-Service Setup**:
  - Frontend container (Nginx + React build)
  - Backend API container (Node.js)
  - Database container (PostgreSQL)
  - Redis container (session storage)
  - Reverse proxy (Traefik or Nginx Proxy Manager)

NAS Integration

- **OpenMediaVault Compatibility**: Tested deployment on OMV
- **Volume Mounting**: Persistent data storage on NAS drives
- **Backup Integration**: Automatic database backups to NAS storage
- **SSL/TLS**: Automatic HTTPS certificate management

Configuration Management

- **Environment Variables**: All config via environment files
- **Docker Compose**: Single-command deployment
- **Health Checks**: Container health monitoring
- **Auto-Restart**: Automatic recovery from failures

Technical Specifications

Docker Structure

yaml

```yaml
version: '3.8'
services:
  frontend:
    build: ./frontend
    ports:
      - "3000:80"
    depends_on:
      - backend

  backend:
    build: ./backend
    ports:
      - "3001:3001"
    environment:
      - DATABASE_URL=${DATABASE_URL}
      - JWT_SECRET=${JWT_SECRET}
      - REDIS_URL=${REDIS_URL}
    depends_on:
      - database
      - redis
    volumes:
      - recipe-images:/app/uploads

  database:
    image: postgres:15-alpine
    environment:
      - POSTGRES_DB=${DB_NAME}
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ./backups:/backups

  redis:
    image: redis:7-alpine
    volumes:
      - redis-data:/data

volumes:
  postgres-data:
    driver: local
    driver_opts:
      device: /srv/dev-disk-by-uuid-${NAS_UUID}/recipes/data
      o: bind
  recipe-images:
```

```
recipe-images:
  driver: local
  driver_opts:
    device: /srv/dev-disk-by-uuid-${NAS_UUID}/recipes/images
    o: bind
```

Deployment Process

1. **Pre-requisites Check**: Verify Docker and system requirements
2. **Configuration Setup**: Generate environment files
3. **SSL Certificate**: Obtain and configure HTTPS certificates
4. **Database Migration**: Run initial database setup
5. **Service Start**: Launch all containers
6. **Health Verification**: Confirm all services are running
7. **Backup Schedule**: Set up automated backups

Monitoring & Maintenance

- **Log Aggregation**: Centralized logging with rotation
- **Performance Monitoring**: Basic metrics collection
- **Update Mechanism**: Rolling updates without downtime
- **Backup Verification**: Automated backup integrity checks

Success Metrics

- One-command deployment on fresh NAS system
- <5 minute deployment time from start to running
- 99.9% uptime over 30-day period
- Successful automated backups daily

Phase 5: Android Native App

Objectives

- Create native Android app with full feature parity
- Implement offline-first architecture for kitchen use
- Optimize for touch interactions and mobile cooking workflows

Core Features

Native Mobile Experience

- **Material Design 3**: Modern Android UI patterns
- **Gesture Navigation**: Swipe, pinch, long-press interactions
- **Adaptive UI**: Dynamic color theming, large screen support
- **Accessibility**: TalkBack support, high contrast mode

Kitchen-Optimized Features

- **Always-On Display**: Keep screen on during cooking
- **Voice Commands**: "Next step", "Start timer", hands-free navigation
- **Large Text Mode**: Easy reading while cooking
- **Splash Protection**: Hide sensitive controls when phone is near water

Offline-First Capabilities

- **Recipe Caching**: Store frequently used recipes locally
- **Sync Queue**: Queue actions when offline, sync when connected
- **Offline Search**: Local recipe database search
- **Image Caching**: Store recipe images for offline viewing

Integration Features

- **Camera Recipe Scanning**: Take photo of printed recipe for OCR extraction
- **Barcode Scanner**: Scan ingredients for automated shopping lists
- **Share Integration**: Share recipes via Android's native sharing
- **Calendar Integration**: Add cooking time to calendar

Technical Specifications

Development Stack

- **Language**: Kotlin with Coroutines
- **Architecture**: MVVM with Repository pattern
- **Database**: Room (SQLite) with offline-first sync
- **Networking**: Retrofit with OkHttp, automatic retry logic
- **Image Loading**: Coil for efficient image caching
- **Navigation**: Navigation Component with type-safe arguments

Offline Sync Strategy

- **Conflict Resolution**: Last-write-wins with user override option
- **Delta Sync**: Only sync changed data to minimize bandwidth
- **Background Sync**: Periodic sync using WorkManager
- **Sync Status**: Clear indicators of sync state to users

API Integration

```kotlin
interface RecipeApiService {
    @GET("recipes")
    suspend fun getRecipes(
        @Query("search") search: String?,
        @Query("cuisine") cuisine: String?,
        @Query("diet") diet: String?
    ): Response<RecipesResponse>

    @POST("recipes")
    suspend fun saveRecipe(@Body recipe: Recipe): Response<Recipe>

    @GET("recipes/{id}")
    suspend fun getRecipe(@Path("id") id: String): Response<Recipe>
}
```

Performance Optimizations

- **LazyLoading**: Pagination for recipe lists
- **Image Compression**: Automatic image optimization before upload
- **Memory Management**: Proper lifecycle-aware components
- **Battery Optimization**: Efficient background processing

Success Metrics

- App launch time <2 seconds
- Smooth 60fps scrolling and animations
- Offline functionality works for 100% of cached recipes
- 4.5+ star rating on Google Play Store

Phase 6: Meal Planning & Shopping Lists

Objectives

- Add comprehensive meal planning calendar
- Generate smart shopping lists with store optimization
- Implement ingredient inventory tracking

Core Features

Meal Planning Calendar

- **Weekly/Monthly Views**: Visual meal planning interface
- **Drag & Drop Scheduling**: Assign recipes to specific meals/days
- **Recurring Meals**: Set up weekly meal patterns
- **Family Coordination**: Multiple family members can plan meals
- **Nutrition Tracking**: Weekly nutrition summaries
- **Leftover Management**: Track and plan leftover usage

Smart Shopping Lists

- **Auto-Generation**: Create lists from planned meals
- **Store Layout Optimization**: Organize by grocery store sections
- **Price Tracking**: Integrate with grocery price APIs where available
- **Quantity Optimization**: Consolidate ingredients across recipes
- **Substitution Suggestions**: Alternative ingredient recommendations
- **Shared Lists**: Real-time collaboration on shopping lists

Pantry Management

- **Inventory Tracking**: Track what ingredients you have at home
- **Expiration Alerts**: Notifications for expiring ingredients
- **Recipe Suggestions**: Recommend recipes based on available ingredients
- **Shopping List Refinement**: Remove items already in pantry
- **Barcode Scanning**: Quick pantry additions via mobile app

Budget Management

- **Meal Cost Estimation**: Calculate approximate meal costs
- **Budget Tracking**: Set and monitor monthly food budgets
- **Cost per Serving**: Track cost efficiency of different recipes
- **Shopping History**: Track spending patterns over time

Technical Specifications

Database Schema Extensions

sql

```sql
-- Meal plans
CREATE TABLE meal_plans (
  id UUID PRIMARY KEY,
  family_id UUID REFERENCES families(id),
  date DATE NOT NULL,
  meal_type VARCHAR(20) NOT NULL, -- breakfast, lunch, dinner, snack
  recipe_id UUID REFERENCES recipes(id),
  servings INTEGER DEFAULT 1,
  notes TEXT,
  created_by UUID REFERENCES users(id),
  created_at TIMESTAMP DEFAULT NOW()
);

-- Shopping lists
CREATE TABLE shopping_lists (
  id UUID PRIMARY KEY,
  family_id UUID REFERENCES families(id),
  name VARCHAR(100) NOT NULL,
  created_by UUID REFERENCES users(id),
  is_completed BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP DEFAULT NOW()
);

-- Shopping list items
CREATE TABLE shopping_list_items (
  id UUID PRIMARY KEY,
  shopping_list_id UUID REFERENCES shopping_lists(id),
  ingredient_name VARCHAR(255) NOT NULL,
  quantity DECIMAL(10,2),
  unit VARCHAR(50),
  category VARCHAR(50), -- produce, dairy, meat, etc.
  is_purchased BOOLEAN DEFAULT FALSE,
  estimated_price DECIMAL(10,2),
  notes TEXT
);

-- Pantry inventory
CREATE TABLE pantry_items (
  id UUID PRIMARY KEY,
  family_id UUID REFERENCES families(id),
  ingredient_name VARCHAR(255) NOT NULL,
  quantity DECIMAL(10,2),
  unit VARCHAR(50),
  expiration_date DATE,
  location VARCHAR(100), -- fridge, freezer, pantry
```

```
  location VARCHAR(100), -- fridge, freezer, pantry
  added_by UUID REFERENCES users(id),
  added_at TIMESTAMP DEFAULT NOW()
);
```

Advanced Features

- **Recipe Scaling**: Automatic ingredient quantity adjustments
- **Meal Prep Batching**: Group similar prep tasks across recipes
- **Seasonal Recommendations**: Suggest recipes based on seasonal ingredients
- **Dietary Goal Tracking**: Monitor nutrition goals across planned meals

Success Metrics

- 80% of users create meal plans within first week
- Shopping list generation accuracy >95%
- Pantry tracking reduces food waste by 20%
- Average meal planning time reduced by 60%

Phase 7: Browser Extension

Objectives

- Create seamless recipe saving from any website
- Provide one-click export to recipe management system
- Integrate with existing browser workflows

Core Features

Browser Integration

- **Universal Recipe Detection**: Automatically detect recipes on any webpage
- **One-Click Save**: Save button overlay when recipe detected
- **Quick Preview**: Preview extracted recipe before saving
- **Folder Organization**: Save to specific recipe collections
- **Duplicate Detection**: Warn if recipe already saved

Enhanced Extraction

- **Improved Accuracy**: Better extraction algorithms than web version
- **Manual Editing**: Edit extracted data before saving
- **Image Selection**: Choose best recipe image from page
- **Tag Suggestions**: AI-powered tag recommendations
- **Rating Integration**: Import existing ratings from recipe sites

Workflow Integration

- **Right-Click Menu**: Save recipe option in context menu
- **Keyboard Shortcuts**: Power user keyboard shortcuts
- **Bookmark Sync**: Integrate with browser bookmark system
- **History Tracking**: Track visited recipe sites
- **Bulk Import**: Import existing browser bookmarks

Technical Specifications

Extension Architecture

- **Manifest V3**: Modern Chrome extension format
- **Content Scripts**: Recipe detection and extraction
- **Background Service**: API communication and data sync
- **Popup Interface**: Quick recipe preview and save options
- **Options Page**: Extension settings and account management

Cross-Browser Support

- **Chrome/Chromium**: Primary target platform
- **Firefox**: WebExtensions API compatibility
- **Edge**: Chromium-based compatibility
- **Safari**: Safari Web Extensions (Phase 7.1)

Extension Manifest

```json
{
  "manifest_version": 3,
  "name": "Family Recipe Saver",
  "version": "1.0.0",
  "permissions": [
    "activeTab",
    "storage",
    "contextMenus"
  ],
  "host_permissions": [
    "*://*/*"
  ],
  "content_scripts": [{
    "matches": ["*://*/*"],
    "js": ["content.js"],
    "css": ["content.css"]
  }],
  "background": {
    "service_worker": "background.js"
  },
  "action": {
    "default_popup": "popup.html",
    "default_title": "Save Recipe"
  }
}
```

Privacy & Security

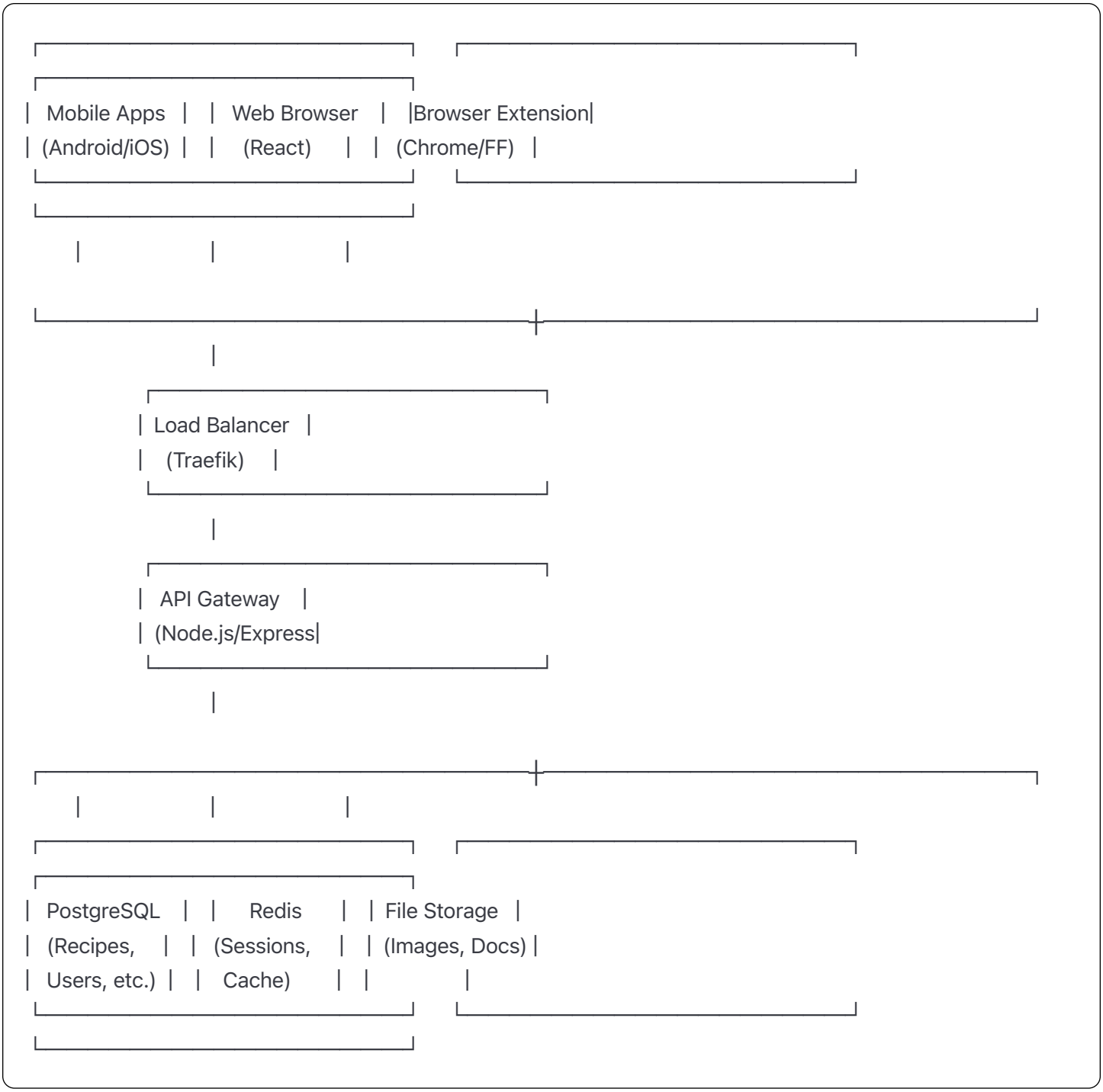
- **Minimal Permissions**: Request only necessary permissions
- **Local Processing**: Process recipe data locally when possible
- **Secure Communication**: HTTPS-only API communication
- **Data Encryption**: Encrypt stored authentication tokens

Success Metrics

- Recipe detection accuracy >90% on major cooking sites
- <100ms recipe detection response time
- 10,000+ Chrome Web Store installations
- 4.5+ star average rating

Technical Architecture Overview

System Architecture



Data Flow

1. **Recipe Discovery**: External APIs → Backend → Frontend
2. **Recipe Scraping**: Web Scraper → Content Parser → Database
3. **User Management**: Authentication → Session Management → Authorization
4. **Family Sharing**: User Actions → Permission Checks → Data Sync
5. **Offline Sync**: Mobile App ↔ Local DB ↔ Server API

Security Considerations

- **Data Encryption**: All data encrypted at rest and in transit
- **API Security**: Rate limiting, input validation, authentication required
- **User Privacy**: GDPR compliant, data export capabilities
- **Self-Hosted Benefits**: Complete data ownership, no third-party data sharing

---

Development Timeline

Phase 1 (4-6 weeks)

- Week 1-2: Project setup, API integration, basic UI
- Week 3-4: Recipe search and display functionality
- Week 5-6: Responsive design, polish, testing

Phase 2 (6-8 weeks)

- Week 1-2: Web scraping engine development
- Week 3-4: Database design and implementation
- Week 5-6: In-app browser and recipe extraction
- Week 7-8: Testing, bug fixes, performance optimization

Phase 3 (4-6 weeks)

- Week 1-2: Authentication system
- Week 3-4: Family sharing features
- Week 5-6: Social features, testing

Phase 4 (3-4 weeks)

- Week 1-2: Dockerization and container setup
- Week 3-4: NAS deployment, backup systems

Phase 5 (8-10 weeks)

- Week 1-2: Android project setup, basic UI
- Week 3-4: Core recipe features
- Week 5-6: Offline sync implementation
- Week 7-8: Kitchen-specific features
- Week 9-10: Testing, optimization, Play Store release

Phase 6 (6-8 weeks)

- Week 1-2: Meal planning calendar
- Week 3-4: Shopping list generation
- Week 5-6: Pantry management
- Week 7-8: Integration and testing

Phase 7 (4-6 weeks)

- Week 1-2: Browser extension development
- Week 3-4: Cross-browser compatibility
- Week 5-6: Store submission and release

Total Estimated Timeline: 35-48 weeks (8.5-12 months)

Success Metrics & KPIs

User Engagement

- Daily Active Users (DAU)
- Recipe saves per user per month
- Average session duration
- Recipe view-to-save conversion rate

Technical Performance

- API response time <500ms (95th percentile)
- Mobile app crash rate <0.1%
- Web scraping success rate >90%
- System uptime >99.9%

Family Collaboration

- Families with >1 active user
- Shared recipes per family
- Meal plans created per family per month
- Shopping list collaboration rate

Platform Growth

- New user registration rate
- Platform retention (1-day, 7-day, 30-day)
- Feature adoption rates
- User-generated content volume

This PRD provides a comprehensive roadmap for building a world-class, self-hosted recipe management platform that evolves from a simple web app to a complete family cooking ecosystem.