Contents

Product Specification Document (PSD)

Calories Tracker Application

Document Information

- **Document Version**: 1.0
- **Date**: December 2024
- **Prepared By**: Software Development Team
- **Document Type**: Product Specification Document

---

1. Executive Summary

1.1 Product Overview

The Calories Tracker is a modern web-based application designed to help users monitor and manage their daily caloric intake through an intuitive interface. The application combines traditional manual entry with innovative AI-powered image analysis to provide users with a seamless calorie tracking experience.

1.2 Business Objectives

- **Primary Goal**: Enable users to easily track and monitor their daily caloric consumption
- **Secondary Goals**:
  - Provide visual insights through data analytics and charts
  - Offer AI-powered convenience through image-based calorie estimation
  - Support long-term health and wellness goals through trend analysis

1.3 Target Market

- Health-conscious individuals seeking calorie management
- Fitness enthusiasts monitoring dietary intake
- Users requiring medical dietary tracking
- General consumers interested in wellness and nutrition awareness

---

2. Product Architecture & Technology Stack

2.1 System Architecture

Architecture Pattern: Microservices with containerized deployment - **Frontend**: Single Page Application (SPA) - **Backend**: RESTful API service - **Mock Service**: AI simulation service for image analysis - **Database**: SQLite for development/testing environments - **Deployment**: Docker containerization with multi-service orchestration

2.2 Technology Stack

Frontend Technologies

- **Framework**: React 19.1.0 with TypeScript
- **Build Tool**: Vite 7.0.0
- **Routing**: React Router DOM 6.23.1
- **Authentication**: Google OAuth 2.0 (@react-oauth/google)
- **Data Visualization**: Chart.js 4.4.2 with React wrapper
- **UI Components**: Custom component library with React Icons
- **Styling**: CSS with theme system supporting light/dark modes
- **Performance**: React virtualization for large data sets

Backend Technologies

- **Framework**: NestJS 11.0.1 with TypeScript
- **Database ORM**: TypeORM 0.3.25
- **Authentication**: JWT tokens with Google OAuth verification
- **Validation**: Class-validator and class-transformer
- **Database**: SQLite 5.1.7 (configurable for production databases)
- **API Documentation**: RESTful endpoints with DTO validation

Infrastructure & DevOps

- **Containerization**: Docker with multi-stage builds
- **Orchestration**: Docker Compose for local development
- **Web Server**: Nginx for frontend serving
- **Development**: Hot reload enabled for rapid development

---

3. Core Features & Functionality

3.1 Authentication & User Management

3.1.1 Google OAuth Integration

- **Feature**: Secure authentication using Google Sign-In
- **Implementation**: JWT token-based session management
- **User Data**: Email, name, profile picture synchronization
- **Security**: Token verification with Google's authentication service
- **Session Management**: Persistent login state with sessionStorage

3.1.2 User Profile Management

- **User Entity Structure**:
  - Unique email identification
  - Profile information (name, picture)
  - Account status management
  - Creation and update timestamps
  - One-to-many relationship with calorie entries

3.2 Calorie Management System

3.2.1 Manual Calorie Entry

- **Core Functionality**: Add, edit, delete calorie entries
- **Data Fields**:
  - Description (text, required)
  - Calorie count (integer, minimum 1, required)
  - Automatic timestamp recording
- **Validation**: Client-side and server-side validation
- **User Experience**: Modal-based forms with real-time validation

3.2.2 AI-Powered Image Analysis

- **Feature**: Upload food images for automatic calorie estimation
- **Technology**: Mock AI service simulating real-world image analysis
- **Capabilities**:
  - Image upload and processing
  - Food item recognition with 80% success rate simulation
  - Automatic calorie estimation
  - Pre-populated form fields with AI results
- **User Feedback**: Loading states and error handling for failed recognition

3.2.3 Data Management

- **CRUD Operations**: Full create, read, update, delete functionality
- **Soft Deletion**: Entries marked as deleted rather than permanently removed
- **Data Relationships**: User-scoped data access with proper authorization
- **Pagination**: Support for large datasets with configurable limits

3.3 Analytics & Visualization

3.3.1 Daily Calorie Tracking

- **Aggregation**: Automatic daily calorie summation
- **Time Periods**: Configurable viewing periods (1 week, 2 weeks, 4 weeks)
- **Data Visualization**: Interactive bar charts with Chart.js
- **Today Highlighting**: Visual distinction for current day in charts

3.3.2 Historical Data Analysis

- **Trend Visualization**: Multi-week trend analysis

- **Data Completeness**: Automatic filling of missing days with zero values
- **Chart Features**:
  - Responsive design for all screen sizes
  - Color-coded bars (special highlighting for today)
  - Hover interactions for detailed information
  - Theme-aware styling (light/dark mode support)

3.3.3 Data Export & Management

- **Table View**: Comprehensive data table with sorting and actions
- **Entry Details**: Date, time, description, and calorie information
- **Bulk Operations**: Edit and delete capabilities from table interface
- **Test Data Generation**: Development feature for populating sample data

---

4. User Interface & User Experience (UI/UX)

4.1 Design Philosophy

- **Minimalist Approach**: Clean, uncluttered interface focusing on core functionality
- **Accessibility**: High contrast, readable fonts, clear navigation
- **Responsiveness**: Mobile-first design with desktop optimization
- **Consistency**: Unified component library with standardized interactions

4.2 Theme System

- **Dual Theme Support**: Light and dark mode implementations
- **Theme Persistence**: User preference storage in localStorage
- **Dynamic Switching**: Real-time theme toggling without page refresh
- **Color Palette**:
  - **Light Mode**: Fresh green primary (#4CAF50), warm orange secondary (#FF9800)
  - **Dark Mode**: Softer green primary (#66BB6A), muted orange secondary (#FFB74D)

4.3 Component Architecture

- **Reusable Components**:
  - Button system with multiple variants (primary, secondary, danger)
  - Modal dialogs for forms and confirmations
  - Data tables with customizable columns and actions
  - Form inputs with validation styling
  - Layout components for consistent spacing

4.4 Navigation & User Flow

- **Landing Page**: Simple authentication with Google Sign-In
- **Dashboard**: Central hub with statistics and entry management
- **Protected Routes**: Authentication-required pages with automatic redirection
- **Header Navigation**: User profile display and logout functionality

4.5 Responsive Design

- **Mobile Optimization**: Touch-friendly interfaces and optimized layouts
- **Tablet Support**: Adaptive layouts for medium-screen devices
- **Desktop Enhancement**: Full-featured experience with expanded visualizations

---

5. API Specification & Data Models

5.1 Authentication Endpoints

```
POST /auth/token-signin
- Purpose: Google OAuth token verification and user session creation
- Input: { token: string }
- Output: { accessToken: string, user: UserProfile }
```

5.2 Calorie Management Endpoints

```
GET /calories
- Purpose: Retrieve user's calorie entries with filtering
- Parameters: startDate?, endDate?, skip?, limit?
- Output: CalorieEntry[]

GET /calories/by-day
- Purpose: Retrieve daily calorie aggregations
- Parameters: startDate?, endDate?, skip?, limit?
- Output: { date: string, totalCalories: number }[]

POST /calories
- Purpose: Create new calorie entry
- Input: { description: string, calories: number }
- Output: CalorieEntry

PUT /calories/:id
- Purpose: Update existing calorie entry
- Input: { description?: string, calories?: number }
- Output: CalorieEntry

DELETE /calories/:id
- Purpose: Soft delete calorie entry
- Output: { success: boolean }

POST /calories/test-data
- Purpose: Generate sample data for testing (development feature)
- Output: { success: boolean, message: string }
```

5.3 Data Models

User Entity

```
interface User {
  id: number;
  email: string;
  name?: string;
  phone?: string;
  picture?: string;
  accessToken?: string;
  status: UserStatusEnum;
  emailVerified?: boolean;
  calories?: Calorie[];
  createdAt: Date;
  updatedAt: Date;
}
```

Calorie Entity

```
interface Calorie {
  id: number;
  description: string;
  calories: number;
  userId?: number;
  user?: User;
  createdAt: Date;
  updatedAt: Date;
  deleted: boolean;
}
```

---

6. Security & Compliance

6.1 Authentication Security

- **Google OAuth 2.0**: Industry-standard authentication protocol
- **JWT Tokens**: Secure session management with expiration
- **Token Verification**: Server-side Google token validation
- **Session Isolation**: User-scoped data access controls

6.2 Data Protection

- **User Data Isolation**: Database-level user separation
- **Soft Deletion**: Data recovery capabilities for accidental deletions
- **Input Validation**: Comprehensive client and server-side validation
- **SQL Injection Prevention**: ORM-based query building

6.3 Privacy Considerations

- **Minimal Data Collection**: Only essential user information stored
- **Local Storage**: Theme preferences stored locally
- **Session Management**: Secure token storage in sessionStorage

- **Data Ownership**: Users maintain full control over their calorie data

---

7. Performance & Scalability

7.1 Frontend Performance

- **React Optimization**: Modern React 19 with efficient rendering
- **Code Splitting**: Vite-based bundling with optimized loading
- **Virtual Scrolling**: React virtualization for large data sets
- **Lazy Loading**: Component-based lazy loading for improved initial load

7.2 Backend Performance

- **Database Optimization**: Indexed queries and efficient relationships
- **Pagination**: Configurable data retrieval limits
- **Caching Strategy**: Session-based caching for frequently accessed data
- **Query Optimization**: TypeORM query builder for efficient database access

7.3 Scalability Considerations

- **Microservices Architecture**: Independently scalable services
- **Database Flexibility**: Configurable database backends (SQLite to PostgreSQL)
- **Containerization**: Docker-based deployment for easy scaling
- **Load Balancing**: Nginx configuration for traffic distribution

---

8. Testing & Quality Assurance

8.1 Testing Strategy

- **Unit Testing**: Jest framework for backend services
- **Integration Testing**: API endpoint testing with Supertest
- **Frontend Testing**: React Testing Library for component testing
- **Type Safety**: TypeScript for compile-time error detection

8.2 Development Tools

- **Code Quality**: ESLint and Prettier for consistent code style
- **Type Checking**: Comprehensive TypeScript configuration
- **Hot Reload**: Development environment with instant feedback
- **Debugging**: Source map support for production debugging

8.3 Mock Services

- **AI Simulation**: Mock image analysis service for development/testing
- **Realistic Responses**: Configurable success/failure rates
- **Data Generation**: Automated test data creation capabilities

---

9. Deployment & DevOps

9.1 Containerization

- **Multi-Stage Builds**: Optimized Docker images for production
- **Service Orchestration**: Docker Compose for development environment
- **Environment Configuration**: Environment-specific variable management

9.2 Production Deployment

- **Nginx Configuration**: Production-ready web server setup
- **Database Migration**: TypeORM synchronization for schema updates
- **Health Checks**: Service monitoring and restart capabilities

9.3 Configuration Management

- **Environment Variables**: Secure configuration through environment files
- **API Endpoints**: Configurable service URLs for different environments
- **Feature Flags**: Conditional feature enabling through configuration

---

10. Future Roadmap & Enhancements

10.1 Planned Features

- **Real AI Integration**: Replace mock service with actual image recognition API
- **Nutritional Analysis**: Detailed macro and micronutrient tracking
- **Goal Setting**: Personalized calorie targets and progress tracking
- **Social Features**: Sharing achievements and collaborative tracking
- **Mobile Applications**: Native iOS and Android applications

10.2 Technical Improvements

- **Database Migration**: Production-ready database implementation
- **Caching Layer**: Redis implementation for improved performance
- **Real-time Updates**: WebSocket integration for live data updates
- **Advanced Analytics**: Machine learning for personalized insights

10.3 Integration Possibilities

- **Fitness Trackers**: Wearable device integration
- **Nutrition Databases**: Third-party nutrition API integration
- **Health Platforms**: Integration with health monitoring systems
- **Recipe Services**: Meal planning and recipe suggestion features

---

11. Success Metrics & KPIs

11.1 User Engagement Metrics

- **Daily Active Users**: Users logging calories daily

- **Feature Adoption**: Image analysis vs. manual entry usage rates
- **Session Duration**: Time spent in application per session
- **Data Retention**: Consistency of daily logging over time

11.2 Technical Performance Metrics

- **Response Time**: API endpoint response times under 200ms
- **Uptime**: 99.9% application availability target
- **Error Rates**: Less than 1% error rate for critical operations
- **Load Capacity**: Support for concurrent user sessions

11.3 Business Impact Metrics

- **User Retention**: Monthly active user retention rates
- **Feature Satisfaction**: User feedback on core features
- **Support Requests**: Minimized user support ticket volume
- **Accessibility Compliance**: WCAG 2.1 AA compliance achievement

---

12. Risk Assessment & Mitigation

12.1 Technical Risks

- **Third-party Dependencies**: Google OAuth service availability
- **Database Limitations**: SQLite scalability constraints
- **Image Analysis Accuracy**: Mock service limitations in production

12.2 Mitigation Strategies

- **Fallback Authentication**: Alternative authentication methods
- **Database Migration Path**: Clear upgrade path to production databases
- **Gradual AI Integration**: Phased rollout of real AI services

12.3 Security Considerations

- **Data Breach Prevention**: Regular security audits and updates
- **Compliance Monitoring**: Ongoing privacy regulation compliance
- **Access Control**: Robust user authentication and authorization

---

13. Conclusion

The Calories Tracker application represents a modern, user-centric approach to dietary monitoring, combining traditional calorie tracking with innovative AI-powered features. The application's architecture ensures scalability, security, and maintainability while providing an excellent user experience across all devices.

Key Strengths:

- **Modern Technology Stack**: Built with latest React and NestJS frameworks
- **Innovative Features**: AI-powered image analysis for calorie estimation
- **User-Centric Design**: Intuitive interface with accessibility considerations
- **Scalable Architecture**: Microservices design ready for growth
- **Comprehensive Analytics**: Visual insights for long-term health tracking

Implementation Readiness:

The application is development-ready with a clear path to production deployment. The mock services provide immediate development capabilities while allowing for seamless integration of production services as they become available.

This Product Specification Document serves as a comprehensive guide for development teams, stakeholders, and future enhancement planning, ensuring the Calories Tracker application meets both current requirements and future growth expectations.

Document Control - **Next Review Date**: Quarterly review cycle - **Approval Required**: Product Owner, Technical Lead, UI/UX Designer - **Distribution**: Development Team, QA Team, Product Management, Stakeholders