PLOS Module-by-Module Function Details

1. Dashboard Module

Core Components

- **WelcomeBanner**: Displays personalized greeting
- **QuoteCard**: Shows daily motivational quote from OpenAI
- **StatsGrid**: Visualizes key metrics from all modules
- **ActionCards**: Quick access to frequently used features

Key Functions

- `useDashboardStore`: Zustand store for dashboard state
  - `stats`: Object containing metrics (steps, mood, tasksCompleted, etc.)
  - `updateStat(key, value)`: Updates a specific metric

API Endpoints

- `/api/quote`: Fetches daily motivational quote from OpenAI

Data Flow

1. User opens dashboard
2. App loads statistics from all modules via Zustand store
3. App fetches daily quote from OpenAI
4. Dashboard renders with personalized data

2. Journal Module

Core Components

- **JournalEditor**: Rich text editor with markdown support
- **JournalList**: Displays list of journal entries
- **SentimentDisplay**: Shows emotional analysis of entries

Key Functions

- `JournalEditor`: React component for writing/editing entries
  - `mode`: Toggles between 'edit' and 'preview' modes

- `onSubmit`: Saves entry and triggers sentiment analysis
- `analyzeSentiment`: Sends content to OpenAI for analysis

API Endpoints

- `/api/journal`: CRUD operations for journal entries
- `/api/journal/sentiment`: Analyzes entry sentiment via OpenAI

Database Schema

- **journal_entries**:
  - `id`: UUID primary key
  - `user_id`: Reference to auth.users
  - `title`: Entry title
  - `content`: Entry content (markdown)
  - `sentiment`: Emotional tone (from OpenAI)
  - `created_at`: Timestamp
  - `updated_at`: Timestamp

Data Flow

1. User writes journal entry
2. Content is saved to Supabase
3. OpenAI API analyzes sentiment
4. Sentiment is stored with entry

3. Goals & Planner Module

Core Components

- **GoalCreator**: Form for creating SMART goals
- **TaskList**: Drag-and-drop task management
- **ProgressTracker**: Visual indicator of goal completion
- **VisionBoard**: Visual representation of goals

Key Functions

- Goal creation and tracking

- Task management with priority sorting
- Progress visualization
- Deadline notifications

API Endpoints

- `/api/goals`: CRUD operations for goals
- `/api/goals/tasks`: Manage tasks within goals

Database Schema

- **goals**:
  - `id`: UUID primary key
  - `user_id`: Reference to auth.users
  - `title`: Goal title
  - `description`: Detailed description
  - `deadline`: Target completion date
  - `progress`: Completion percentage
  - `status`: Active/Completed/Abandoned

Data Flow

1. User creates goals and tasks
2. Progress is updated as tasks are completed
3. Dashboard aggregates goal progress metrics

4. Mental Health Module

Core Components

- **MoodTracker**: Interface for recording daily mood
- **MoodCalendar**: Calendar heatmap of mood history
- **AICompanion**: Supportive chatbot interface
- **MeditationGuide**: Audio-guided meditation sessions

Key Functions

- Mood recording and analysis

- Pattern recognition in emotional states
- Supportive AI conversations based on mood history
- Guided breathing and meditation exercises

API Endpoints

- `/api/mental/mood`: Record and retrieve mood entries
- `/api/mental/insights`: Generate insights from mood data
- `/api/mental/chat`: Interface with OpenAI for supportive conversations

Database Schema

- **mood_entries**:
  - `id`: UUID primary key
  - `user_id`: Reference to auth.users
  - `mood_score`: Numerical mood rating
  - `notes`: User comments on mood
  - `factors`: Array of contributing factors
  - `created_at`: Timestamp

Data Flow

1. User logs daily mood
2. App stores mood data in Supabase
3. AI analyzes patterns and provides insights
4. Dashboard shows mood trends over time

5. Physical Health Module

Core Components

- **MetricsTracker**: Form for logging health metrics
- **ActivityLog**: Record of physical activities
- **HealthCharts**: Visualizations of health trends
- **RecommendationPanel**: AI-generated health advice

Key Functions

- Health metric logging and tracking
- Activity recording with duration and intensity
- Trend visualization with Recharts
- AI-powered recommendations based on health data

API Endpoints

- `/api/health/metrics`: Record and retrieve health metrics
- `/api/health/activities`: Manage physical activities
- `/api/health/recommendations`: Get AI-generated health advice

Database Schema

- **health_metrics**:
  - `id`: UUID primary key
  - `user_id`: Reference to auth.users
  - `weight`: User weight
  - `heart_rate`: Heart rate measurement
  - `sleep_hours`: Hours slept
  - `steps`: Daily step count
  - `created_at`: Timestamp

Data Flow

1. User logs health metrics and activities
2. App stores data in Supabase
3. Charts visualize trends over time
4. AI generates personalized recommendations

6. Nutrition Module

Core Components

- **FoodLogger**: Interface for logging meals
- **NutrientDisplay**: Visualization of macronutrient intake
- **MealPlanner**: AI-assisted meal planning
- **WaterTracker**: Hydration tracking

Key Functions

- Food and meal logging with nutritional information
- Macronutrient calculation and visualization
- AI-generated meal suggestions based on nutritional gaps
- Recipe generation with dietary preferences

API Endpoints

- `/api/nutrition/logs`: Record and retrieve nutrition entries
- `/api/nutrition/suggestions`: Get AI-generated meal suggestions
- `/api/nutrition/recipes`: Generate recipes based on preferences

Database Schema

- **nutrition_logs**:
  - `id`: UUID primary key
  - `user_id`: Reference to auth.users
  - `meal_type`: Breakfast/Lunch/Dinner/Snack
  - `food_items`: Array of food items
  - `calories`: Total calories
  - `macros`: Object with protein/carbs/fat
  - `created_at`: Timestamp

Data Flow

1. User logs meals and food items
2. App calculates nutritional information
3. Charts display macronutrient balance
4. AI suggests improvements to diet

7. Family & Social Module

Core Components

- **EventTracker**: Calendar for social events
- **RelationshipManager**: Track interactions with important people
- **TimeAnalysis**: Visualization of social time distribution

- **ActivitySuggestions**: AI-generated social activity ideas

Key Functions

- Event scheduling and tracking
- Relationship management
- Time allocation analysis
- AI-powered activity recommendations

API Endpoints

- `/api/social/events`: Manage social events
- `/api/social/relationships`: Track relationships and interactions
- `/api/social/suggestions`: Get AI-generated activity suggestions

Database Schema

- **social_events**:
  - `id`: UUID primary key
  - `user_id`: Reference to auth.users
  - `title`: Event title
  - `description`: Event details
  - `people`: Array of people involved
  - `date`: Event date and time
  - `location`: Event location
  - `created_at`: Timestamp

Data Flow

1. User schedules and logs social events
2. App tracks time spent with different people
3. Dashboard shows social activity metrics
4. AI suggests activities based on patterns

AI Service Layer

Core Functions

- `generateCompletion(prompt, context)`: General-purpose text generation
  - Used for: quotes, suggestions, insights
  - Returns: AI-generated text response
- `analyzeSentiment(text)`: Emotional analysis of text
  - Used for: journal entries, messages
  - Returns: Sentiment category (positive/negative/neutral)

Implementation Pattern

typescript

```ts
// lib/services/ai.ts
import OpenAI from 'openai';

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
});

export async function generateCompletion(prompt: string, context: string = '') {
  try {
    const response = await openai.chat.completions.create({
      model: "gpt-3.5-turbo",
      messages: [
        {
          role: "system",
          content: `You are an AI assistant for a personal life management app. ${context}`
        },
        {
          role: "user",
          content: prompt
        }
      ],
      max_tokens: 200,
    });
    return {
      success: true,
      data: response.choices[0].message.content,
    };
  } catch (error) {
    console.error('OpenAI error:', error);
    return {
      success: false,
      error: "Failed to generate AI response",
    };
  }
}

export async function analyzeSentiment(text: string) {
  try {
    const response = await openai.chat.completions.create({
      model: "gpt-3.5-turbo",
      messages: [
        {
          role: "system",
```

```
          content: "Analyze the sentiment of the following text and respond with exactly one wo
        },
        {
          role: "user",
          content: text
        }
      ],
      max_tokens: 10,
    });
    return {
      success: true,
      sentiment: response.choices[0].message.content.trim().toLowerCase(),
    };
  } catch (error) {
    console.error('Sentiment analysis error:', error);
    return {
      success: false,
      sentiment: 'neutral', // Default fallback
    };
  }
}
```

Development Timeline

Phase 1: Core Infrastructure (2-3 weeks)

- Set up Next.js with Tailwind and Supabase
- Authentication flow and user profiles
- Basic dashboard with mock data

Phase 2: Module Implementation (8-10 weeks)

- Dashboard (Days 4-8)
- Journal (Days 9-13)
- Goals & Planner (Days 14-19)
- Mental Health (Days 20-25)
- Physical Health (Days 26-31)
- Nutrition (Days 32-37)
- Family & Social (Days 38-43)

Phase 3: AI Enhancement (4-6 weeks)

- Add OpenAI integration for key features
- Implement recommendation systems
- Add sentiment analysis and insights

Phase 4: Polish & Performance (2-3 weeks)

- Optimize loading performance
- Add animations and transitions
- Implement PWA features