High-Level Design Report

1. Introduction

1.1 Purpose of the system

In today's fast-paced world, maintaining a healthy diet has become increasingly challenging due to busy schedules, high stress levels, and limited time for meal planning and preparation. Research indicates that approximately 69% of users abandon health and nutrition apps within the first 90 days, and around 70% stop using them within 100 days [1]. The primary reasons identified for these high drop-off rates include time-consuming manual data entry, limited personalization, poor user experience, and lack of sustained motivation [1], [2].

While there are many nutrition apps available today, most of them have significant limitations. They typically rely on users to manually search for and enter every single item they eat, which can be time-consuming. Furthermore, these apps often lack the features that provide emotional and social support, which are critical for long-term motivation [3]. They don't offer intelligent suggestions, adapt to the user's lifestyle, or create a sense of community and encouragement. This creates a clear market gap for a smarter, more engaging solution.

The primary purpose of this project is to develop NutriGame, an intelligent and user-friendly mobile application designed to address these challenges. Our motivation is to leverage the power of AI, social interaction, and gamification to make nutrition tracking simple, enjoyable, and sustainable [4], [5]. NutriGame is designed to be more than just a calorie counter; it aims to be a personal wellness companion that supports users on their journey to a healthier lifestyle.

The target of the project will be measured through clear and quantifiable indicators: achieving at least a 25% improvement in three-month user retention compared to benchmark nutrition apps, reducing the average meal logging time by 50% through AI-based automation, reaching a daily-to-weekly active user ratio above 40%, maintaining over 80% accuracy in AI-powered recognition for the top 100 most common foods in the target regions, and securing an average user satisfaction score of 4.2/5 or higher in pilot tests.

By combining technology, behavioral psychology, and community-driven motivation, NutriGame aims not only to improve individual health outcomes but also to contribute to broader public health by promoting long-term healthy eating behaviors.

1.2 Design goals

The design of NutriGame is based on practical goals that ensure the system works smoothly, is easy to build for our team, and keeps user data safe.

- **User Privacy and Security:** Given the sensitive nature of health data and personal photos, our primary security goal is to ensure users can strictly access only their own information. We achieve this by blocking unauthorized access. Additionally, we implement a privacy strategy where personal details are filtered from data before it is sent to any external AI service, protecting user identity.
- **Visual Appeal and Usability:** To ensure users enjoy using the app and do not feel overwhelmed by complex nutritional data, we prioritize a high-quality visual design. Our goal is to transform boring numbers into engaging, easy-to-read charts and a clean interface.
- **Performance and Responsiveness:** Our goal is to ensure the app is always fast and the screen never freezes. To achieve this, we design the system to handle heavy tasks, like AI food recognition, in the background (AI inference runs asynchronously so the UI thread is never blocked). This means the user does not have to wait on a loading screen and can keep navigating and using the app while the AI calculates the results.
- **Modularity and Code Structure:** Our primary goal is to write the code as modular as possible. Instead of keeping everything in large, complex files, we break the system down into small, independent pieces. This approach keeps the codebase clean, makes it easier to find and fix errors, and allows us to reuse the same code in different parts of the application.
- **Cross-Platform Portability:** The application is designed to be accessible to the widest possible audience by supporting both Android and iOS platforms. To achieve this without doubling the development time, we utilize React Native. This allows us to write a single codebase that runs smoothly on both operating systems, ensuring all users get the same features.

1.3 Definitions, acronyms, and abbreviations

Term	Definition
Semantic Versioning	Version numbering standard (MAJOR.MINOR.PATCH)
Static Analysis	Analyzing code without executing it
Regression Test	Testing to ensure that existing functionality is not broken by new changes
Data Scraping	Extracting data from websites
Label Encoding	Converting text labels into numerical values

Stop-word Removal	Removing insignificant words in natural language processing
Segmentation Mask	Labeled mask used to separate objects in an image
Bounding Box	Rectangular frame showing the position of an object in an image
Anonymization	Process of removing personal data from datasets
NutriGame	Mobile application for healthy living, meal tracking, and gamification
Meal Logging	Feature allowing users to record their meals in the system
AI Chatbot	AI-powered chatbot providing nutrition and diet advice
Gamification	System for increasing user motivation through points, badges, and goals
Streak	A series of consecutive days a user completes a certain task or goal. In NutriGame, it encourages consistency and habit-building.
BMI (Body Mass Index)	A number calculated from a person's weight and height. It is used to determine if someone is underweight, normal weight, overweight, or obese.
Data Processing	Processing nutrition, health, and activity data
Healthy Recipe Dataset	Dataset containing healthy food recipes
Psychology Dataset	Dataset containing psychological counseling conversations
Image Segmentation Dataset	Dataset with food images annotated with segmentation masks
Image Classification Dataset	Dataset categorizing food images into different classes
Traceability Matrix	Table mapping requirements to test scenarios
P0 (Critical)	Blocks core functionality must be fixed immediately before release.
P1 (High)	Major functionality issue, fix required in the next release cycle.
P2 (Medium)	Minor functionality issue, workaround available.
P3 (Low)	Cosmetic or non-critical issue.

Table 1.2. Definitions

Acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
CSV	Comma-Separated Values
GDPR	General Data Protection Regulation
KVKK	Turkish Personal Data Protection Law
OWASP	Open Web Application Security Project
PR	Pull Request
RCA	Root Cause Analysis
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
UAT	User Acceptance Testing
CI/CD	Continuous Integration / Continuous Deployment
UI	User Interface
UX	User Experience
DB	Database
ML	Machine Learning
NLP	Natural Language Processing

Table 1.3. Acronyms

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
CSV	Comma-Separated Values
GDPR	General Data Protection Regulation
KVKK	Turkish Personal Data Protection Law
OWASP	Open Web Application Security Project
PR	Pull Request
RCA	Root Cause Analysis
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
UAT	User Acceptance Testing

CI/CD	Continuous Integration / Continuous Deployment
UI	User Interface
UX	User Experience
DB	Database
ML	Machine Learning
NLP	Natural Language Processing

Table 1.4. Abbreviation

1.4 Overview

This High-Level Design (HLD) report is organized to guide the reader from the project's foundational goals to its specific technical implementation. The document structure is as follows:

- **Section 2 (Current Software Architecture):** Analyzes existing market solutions (such as MyFitnessPal and Lifesum) and identifies the architectural gaps that NutriGame aims to fill with its unified, AI-driven approach.Android
- **Section 3 (Proposed Software Architecture):** Defines the high-level system design, detailing the transition to a modular Client-Server architecture. This section covers Subsystem Decomposition, Hardware/Software Mapping, Persistent Data Management (using CockroachDB + Prisma), Access Control, Global Software Control strategies, and Boundary Conditions.
- **Section 4 (Subsystem Services):** Provides a granular breakdown of the five core subsystems—Mobile Client, Backend API, Data Access, External AI Services, and Notification System—specifying their responsibilities, technology stacks, and integration points.
- **Glossary & References:** Defines technical terminology and acronyms used throughout the document and lists the academic and industry sources supporting the design decisions.

2. Current software architecture (if any)

The current landscape is dominated by separate specialized systems, representing the most popular apps in the nutrition and wellness field. Each implements distinct architectural components that address different aspects of nutrition tracking:

A. Database-Centric Architecture (MyFitnessPal)

- One of the most widely used nutrition tracking apps globally
- Comprehensive nutritional database with barcode scanning and manual entry
- Focuses on quantitative tracking but lacks AI automation, emotional support, and gamification

B. Rule-Based Personalization Architecture (Lifesum)

- Leading app for personalized diet and wellness planning
- Offers personalized diet plans and health assessments with structured guidance
- Missing AI-based photo logging and emotional well-being tracking capabilities

C. Visual-First Journaling Architecture (AteMate)

- Popular choice for mindful eating and visual food tracking
- Provides mindful eating support through visual food journaling
- Lacks nutritional analysis, calorie tracking, and personalized meal planning features

Each of these popular platforms successfully implements part of the solution:

- MyFitnessPal [6] provides the nutritional database foundation
- Lifesum [7] offers the personalization and planning layer
- AteMate [8] delivers the visual journaling approach

However, these features exist in isolation across different platforms. Users must switch between multiple apps to achieve comprehensive nutrition and wellness tracking. Additionally, none of these market-leading platforms include a social community component for sharing healthy lifestyle content and connecting with others.

NutriGame's Approach: Integrate the proven architectural components from these popular systems into a unified platform, enhanced with:

- AI-powered image recognition for automatic food logging
- Emotional well-being tracking integrated with nutrition data
- Gamification with daily strike logic
- Social media platform dedicated to healthy living and nutritious content sharing
- Holistic data architecture that combines nutrition + emotion + behavior + community in one cohesive system

This creates a consolidated architecture that merges the best features of existing market leaders while adding intelligent automation, social connectivity, and engagement mechanisms currently absent from the market.

3. Proposed software architecture

3.1 Overview

The NutriGame system utilizes a Client-Server Architecture powered by a unified TypeScript ecosystem, ensuring type safety and consistency across the stack [9].

- **Frontend:** The mobile client is developed using React Native offering a native experience on both iOS and Android [10].
- **Backend:** The server logic is built on Node.js using the Express framework, providing a robust REST API.

- **Data Layer:** Database interactions are managed via Prisma ORM, providing a structured abstraction layer over the database.
- **AI Layer:** The system integrates specialized AI modules for two primary functions; Food Recognition to analyze meal photos for nutritional content, and Psychological Support to provide empathetic, motivational coaching via a chatbot interface.

Communication occurs primarily via RESTful HTTP requests, utilizing JSON for data interchange. The system integrates external AI services (Gemini, Hugging Face) via API calls from the Node.js backend to handle intelligence tasks without burdening the application server.

3.2 Subsystem decomposition

We divided the NutriGame system into four main parts. Each part has a specific job, which keeps the code clean and easy to manage.

3.2.1. Mobile Client Subsystem (Frontend)

- **Role**: This is the mobile app that users install on their phone.
- **Technology:** It is built using React Native (Expo)
- **Main Features:**
    - **Visual Interface:** Displays the dashboard, meal logs, and profile settings.
    - **Social Hub:** A place where users can find friends, look at leaderboards, and send challenges to each other or themselves.
    - **Badge and Streak Case:** A visually engaging profile area that displays earned achievements and continues activeness using high-quality Lottie animations.
    - **Camera Module:** Takes food photos to be sent to the AI for analysis.

3.2.2. Backend API Subsystem

- **Role:** The main server that runs the logic behind the scenes.
- **Technology:** It runs on Node.js with Express.
- **Main Features:**
    - **Orchestrator:** Acts as a bridge that connects the mobile app, the database, and the AI tools together.
    - **Challenge Logic:** Manages the rules of the game. For example, if User A challenges User B, the server tracks User B's meals to verify if the challenge is completed.
    - **Reward System:** Automatically gives points and badges when a user reaches a goal.
    - **Matchmaking:** Handles friend requests and social lists.

3.2.3. Data Access Subsystem

- **Role:** The persistent database.
- **Technology:** CockroachDB Cloud (PostgreSQL-compatible), accessed via Prisma ORM [11].
- **Main Features:**
    - **Core Data**: Safely saves user passwords, profiles, and food history.
    - **Social Graph:** Keeps track of relationships (who follows whom) and the status of active challenges (Pending, Accepted, Completed).
    - **Notification History:** Stores a log of past alerts so users can read them later.

3.2.4. AI Intelligence Subsystem

- **Role:** The external smart layer using Google Gemini and Hugging Face.
- **Technology:** Integrates Google Gemini and Hugging Face APIs.
- **Main Responsibilities:**
    - **Food Recognition:** Analyzes user-uploaded photos to identify food items and estimate calories automatically with image processing.
    - **NutriCoach:** Uses NLP to understand user chats and provide psychological support or motivation.

3.2.5. Notification Subsystem

- **Role:** The system responsible for real-time alerts.
- **Main Responsibilities:**
    - **Instant Alerts:** Sends immediate push notifications to the user's phone for time-sensitive events (e.g., "Friend Request Accepted" or "Challenge Won").
    - **Smart Reminders:** Runs scheduled jobs to remind users to log their meals if they haven't done so by specific times of the day.

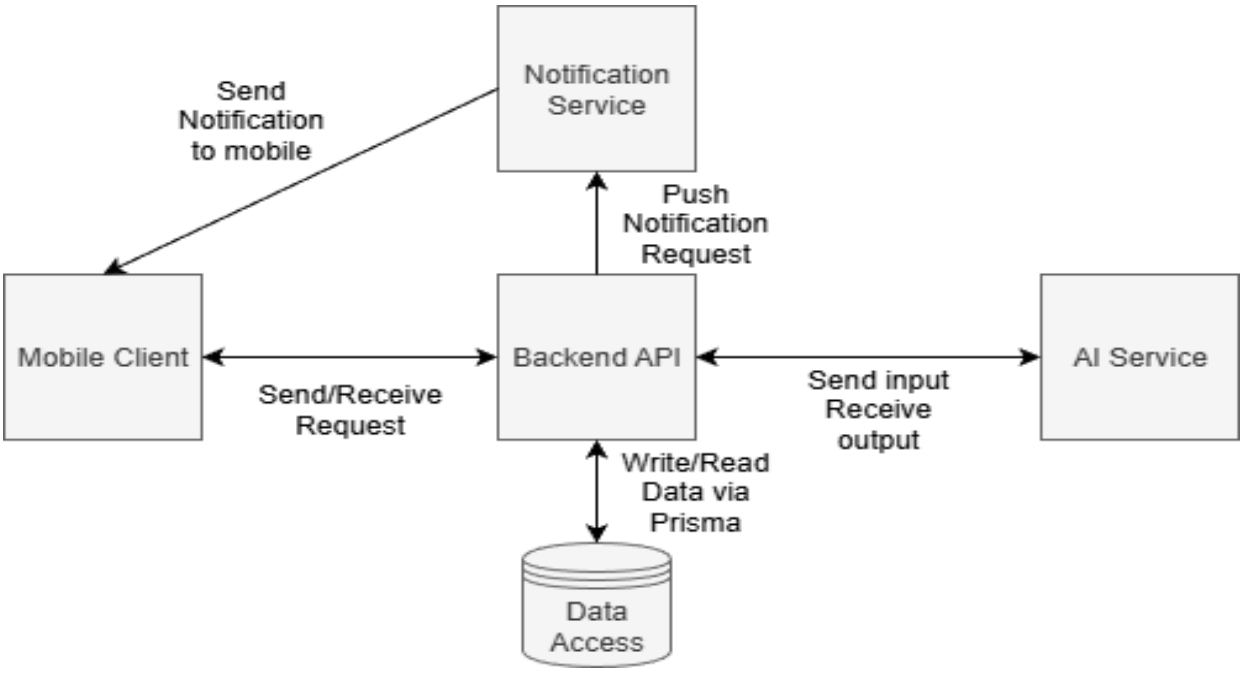


Figure 3.1 Subsystem Diagram

3.3 Hardware/software mapping

The NutriGame system is designed as a distributed application where software components are deployed across five distinct hardware environments: the user's mobile device, the application backend server, the database server, external AI clusters, and notification infrastructure.

3.3.1. Client Node

- **Hardware:** Personal Mobile Device.
  - **Requirements*:*** iOS or Android capable device with Camera and Internet connectivity.
- **Software:**
  - **OS:** Android 11.0+ or iOS 14+.
  - **Artifact:** NutriGame Mobile Application (Native Binary - APK/IPA).
  - **Function:** Executes the Mobile Client Subsystem. It handles the UI rendering, captures images via the camera, and manages local caching for offline performance.

3.3.2. Application Backend Server Node

- **Hardware:** Virtualized Cloud Container
- **Software:**
  - **OS:** Linux (Ubuntu/Alpine).
  - **Artifact:** Compiled Backend Service (TypeScript Build).
  - **Function:** Executes the Backend API Subsystem. It runs the business logic, manages authentication, calculates gamification rewards,and daily calorie targets and orchestrates communication between the database and external services.

3.3.3. Data Server Node

- **Hardware:** Managed Cloud Database Infrastructure.
- **Software:**
    - **Artifact:** PostgreSQL (Relational Database Engine).
    - **Function:** Hosts the Data Access Subsystem. It is responsible for the persistent storage of user profiles, food logs, and the social graph.

3.3.4. External AI Computation Node

- **Hardware:** High-Performance GPU Clusters.
- **Software:**
    - **Artifact:** Google Gemini API & Hugging Face Inference API.
    - **Function:** Executes the AI Intelligence Subsystem. These powerful remote servers handle complex image recognition and Natural Language Processing tasks for psychological support chatbot.

3.3.5. Notification Delivery Node

- **Hardware**: Push Notification Service Infrastructure(Expo)
- **Software**:
    - **Artifact**: Push Delivery Service.
    - **Function**: Executes the Notification Subsystem. It ensures reliable delivery of alerts to user devices even when the application is not running.

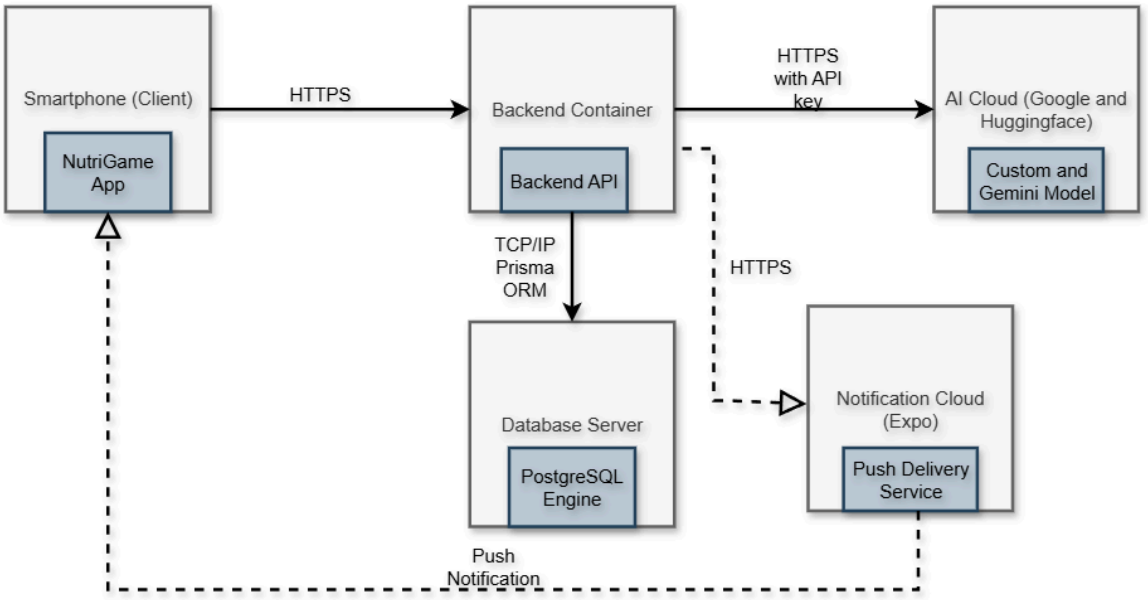


Figure 3.2 Deployment Diagram

3.4 Persistent Data Management

The NutriGame system requires a robust and scalable storage solution to handle complex relationships between users, nutritional logs, and social gamification events. The system uses CockroachDB Cloud as its primary data store. Its PostgreSQL compatibility and distributed architecture ensure data integrity, scalability, and high availability.

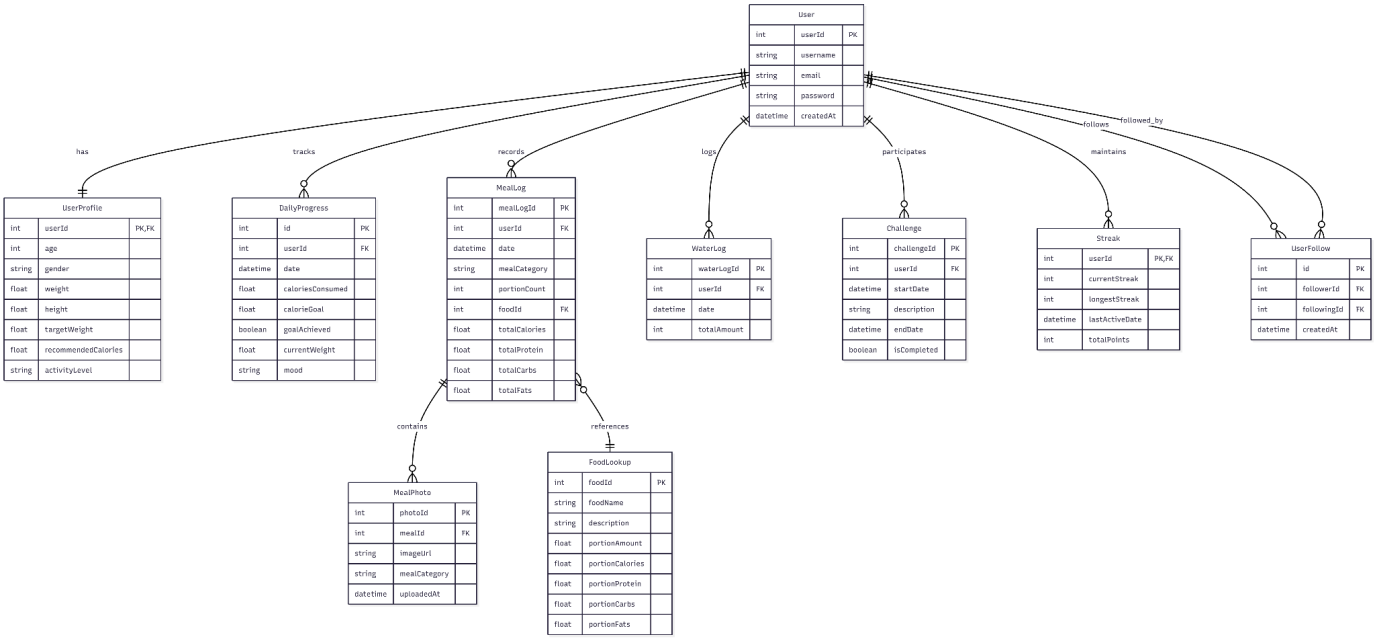
3.4.1. Database Management

- **Technology:** PostgreSQL
- **Rationale:** We chose a relational database strategy because our data is highly structured and interconnected [12].

3.4.2. Data Schema

The application employs a relational database structure that supports nutrition tracking, user management, and social features. Figure 3.3 The Entity-Relationship Diagram illustrates all entities, attributes, and relationships.

- **User Entity:** The central table storing authentication credentials (userId, username, email, password) and account creation timestamp. It maintains one-to-one relationship with UserProfile and one-to-many relationships with all other user-specific entities.
- **UserProfile Entity:** Stores detailed personal metrics linked to User via userId. Contains age, gender, weight, height, targetWeight, recommendedCalories, and activityLevel to enable personalized nutrition recommendations.
- **DailyProgress Entity:** Tracks daily metrics including date, caloriesConsumed, calorieGoal, goalAchieved status, currentWeight, and mood. Enables historical progress tracking through one-to-many relationship with User.
- **FoodLookup Entity:** A global, read-only reference table containing standardized food items with foodId, foodName, description, and complete macronutrient data (portionAmount, portionCalories, portionProtein, portionCarbs, portionFats). Ensures consistency in nutritional calculations.
- **MealLog Entity:** The central ledger for dietary intake, recording mealCategory, date, portionCount, and macronutrient totals. References FoodLookup via foodId while allowing portion customization, maintaining one-to-many relationships with both User and MealPhoto.
- **MealPhoto Entity:** Stores visual meal records with photoId, mealId, imageUrl, mealCategory, and uploadedAt timestamp. Supports AI-powered food recognition and visual meal history.
- **WaterLog Entity:** Dedicated hydration tracking table recording userId, date, and totalAmount (in milliliters). Enables daily hydration pattern analysis.
- **Challenge Entity:** Tracks user participation in fitness challenges with challengeId, userId, startDate, endDate, description, and isCompleted status. Supports multiple simultaneous challenge participation.
- **Streak Entity:** Maintains engagement metrics with one-to-one relationship to User. Tracks currentStreak, longestStreak, lastActiveDate, and totalPoints to encourage consistent usage.
- **UserFollow Entity:** Implements social networking through a self-referencing table with followerId and followingId (both referencing User), creating many-to-many relationships for bidirectional social connections.



3.3 Entity-Relationship Diagram

3.4.3. Access Control and Security

- **Authentication:** Users must log in to receive a secure token (JWT). Without this token, the API rejects all requests.
- **Row Level Security (RLS):** We enforce security rules directly in the database. A user can strictly only read or edit their own data. Access to other users' private logs is blocked at the engine level [14].

3.5 Access Control and Security

We treat health logs as private information, enforcing strict access controls at the application, device, and database levels

3.5.1. Authentication Strategy (Identity Verification)

- **Mechanism:** We utilize Database Authentication Tools to handle identity management via secure JSON Web Tokens (JWT) [13].
- **Token Policy:** Every API request must carry a valid token. If a token is missing or expired, the backend immediately rejects the request with a *401 Unauthorized* error.

3.5.2. Authorization Policy (Row Level Security)

Authorization determines what a verified user is allowed to do. We implement a strict "User-Isolation Policy" enforced directly in the database engine via Row Level Security (RLS). Users are strictly limited to querying and modifying only

their own data rows (user_id = auth.uid()). Access to another user's private logs is physically blocked by the database.

3.5.3. AI Privacy

Since the system interacts with external AI services (Google Gemini), we enforce strict data sanitization to prevent data leakage:

- **Chatbot Filtering:** If a user accidentally shares sensitive details in the chat (e.g., phone numbers, addresses, or IDs), the backend attempts to detect and mask this information before forwarding the prompt to the AI model.
- **Image Metadata:** Before sending food photos for analysis, the system strips all EXIF metadata to ensure user anonymity.

3.5.4. Device Permissions (Hardware Security)

The application adheres to strict OS-level privacy standards regarding hardware access. The user is explicitly asked to grant permission ("Allow Camera Access") only once, upon their first attempt to use the camera feature. If granted, the app retains this access for seamless future use. If denied, the app handles the restriction gracefully by falling back to manual text entry.

3.6 Global Software Control

The NutriGame system follows an Event-Driven Architecture [17]. The control flow is not a continuous linear process; instead, the software components remain in an idle state until triggered by specific external events, such as user interactions (UI taps), API requests, or database updates.

3.6.1. Control Strategy

Since the system relies heavily on network operations (fetching data, uploading images, waiting for AI), we utilize an Asynchronous Control Flow to ensure a smooth user experience. The mobile app does not freeze while waiting for the server. Instead, it displays loading indicators (spinners) while the logic executes in the background. Also, the backend handles multiple user requests simultaneously using Node.js's non-blocking event loop. It does not wait for one user's AI analysis to finish before serving another user.

3.6.2. Dual-Input Execution Flow

The system handles two primary user interaction flows that merge into a single data processing pipeline:

- **AI-Assistaned Logging Flow:** The user taps "Capture Meal". The Controller activates the camera. Upon confirmation, the Backend receives the image and delegates the analysis task to the trained model in Huggingface. The Backend logic "awaits" the AI response without blocking other users. Once the AI responds, the flow resumes, sending the prediction (Food Name + Calories) back to the client for user verification.

- **Manual Logging Flow:** The user searches from the Global Food Library or inputs water amount. Since the data comes from a trusted internal source (the library), the flow bypasses the AI verification step and commits directly to the database.
- **Post-Persistence Streak Logic:** Upon successfully inserting a record into either the meal_log or water_log tables, the Backend triggers a sequential update on the streak table, which is relationally linked to the user table:
    - **Date Validation:** The system retrieves and compares the lastActiveDate from the streak record with the current date.
    - **Streak Update Operation:**
        - **Increment:** If lastActiveDate == Yesterday, the system executes an SQL update to increment the current_streak column by +1.
        - **Reset:** If lastActiveDate < Yesterday (implying a missed day), the system resets the current_streak column to 1.
        - **Response:** The newly updated streak value is returned immediately in the API response payload to refresh the UI.

3.6.3. Social Challenge and Notification Service Flow

This section details how the system handles multiplayer challenges created specifically by users for themselves or their friends.and notifies friends when a task is completed.

- **Phase0:** Before challenges begin, the system establishes the social graph. Users send requests to add other users as friends. Additionally, users can upload and share personal recipes on their profiles, which can serve as inspiration for custom challenges.
- **Phase1:** A user (Challenger) selects a goal type (e.g., "No Sugar") and assigns it to a friend or a group via the app. The invited user receives the request and must explicitly Accept or Reject it.The Backend creates a new record in the Challenge table, linking the participants with a status of "Active".
- **Phase2:** A participant logs a relevant item (e.g., logs a "Sugar-Free Meal"). The Gamification Engine detects this log, identifies the specific challenge linked to this user, and increments their progress bar.
- **Phase3:** The system compares the new progress against the target. if (User_Progress >= Target) -> Mark as Completed.
- **Phase4:** Once completed, the system identifies the creator (Challenger) and other group members. It sends a notification to everyone: *"@Dwight has completed the challenge you assigned!*

3.6.4. AI Chatbot Flow

Unlike standard chatbots, the NutriGame Assistant utilizes a Context Injection strategy combined with a strict Persona Initialization to act as a supportive psychological coach.

1. **Trigger:** The user asks a question (e.g., *"I feel guilty for eating pizza."*).

2. **Context Retrieval:** The Backend queries the database for the user's daily stats to understand the physiological context (e.g., *"Calories: 1800/2000, Water: 1.2L"*).
3. **Prompt Construction:** The Backend constructs a hidden "System Instruction" that defines the AI's behavior before contacting the external service: "You are NutriGame Coach, an empathetic psychological assistant specializing in nutrition. Do not judge the user. Focus on motivation and mental well-being while giving dietary advice."
4. **Safety & Inference:** The input passes through a text filter to ensure no sensitive personal data (like phone numbers) is sent to the AI provider. The combined prompt (Instruction + Context + Query) is sent to the Gemini API.
5. **Response:** The AI generates a psychologically supportive answer (e.g., *"It's okay to enjoy food! You are still within your calorie limit. One meal doesn't define your progress."*) which is streamed back to the UI.

3.6.5. Exception Handling Control

The control flow includes strategies to handle edge cases

- **AI Service Failure:** If the AI API times out, the Backend Control Flow redirects the user to the Manual Search screen instead of crashing.

3.7 Boundary Conditions

Robust systems must gracefully handle edge cases and operational limits. The NutriGame architecture includes specific strategies to manage scenarios where the system operates near or beyond its normal boundaries.

3.7.1. AI Boundaries (AI Failures & Non-Food Inputs)

- **Condition A (Service Outage):** The external AI Service (Gemini) is down or times out (>10 seconds).
  - Handling: The Backend implements a Circuit Breaker Pattern [18]. If the AI fails, the system automatically falls back to "Manual Search Mode," prompting the user to select the food from the Global Library rather than showing a crash error.
- **Condition B (Invalid Image):** The user uploads a photo of a non-food object (e.g., a shoe or a pet).
  - Handling: The AI model includes a classification check. If the "Food Confidence Score" is below a specific threshold (e.g., <60%), the system rejects the input with a user-friendly message: "We couldn't detect any food. Please try again or log manually."

3.7.2. Logical & Data Validity Limits

- **Condition:** A user attempts to input unrealistic values (e.g., drinking 50 liters of water or logging -500 calories).
- **Handling:** The Zod Validation Layer in the Backend enforces strict logical boundaries:

- Calories: Must be between 0 and 5000 per entry.
- Water: Must be between 0ml and 1000ml per entry.
- Date: Future dates are blocked (users cannot log meals for tomorrow).

3.7.3. Input Size Boundaries (Storage Protection)

- **Condition:** A user tries to upload an extremely large image file (e.g., 20MB raw photo) or an unsupported format (e.g., PDF/Video).
- **Handling:** The backend enforces a strict file size limit of 5MB and accepts only JPEG/PNG formats. Files exceeding these limits are rejected immediately at the server entry point to save bandwidth and processing power.

Subsystem Services

The system is decomposed into the following logical subsystems based on identified dependencies and functional cohesion:

1. Mobile Client Subsystem (Frontend)
2. Backend API Subsystem
3. Data Access Subsystem
4. AI Intelligence Subsystem
5. Notification Subsystem

4.1. Mobile Client Subsystem (Frontend)

The mobile client subsystem encompasses all user-facing components running on iOS and Android devices. This subsystem is responsible for rendering the user interface, handling user interactions, managing local state, and communicating with the backend API.

Technology Stack:

- Framework: Expo (React Native)
- Language: TypeScript
- UI Components: React functional components with hooks
- Styling: React Native StyleSheet and standard libraries
- Navigation: @react-navigation/native and @react-navigation/native-stack
- State Management: React Context API or Redux (if applicable)

a) Visual Interface

- Functionality: Displays the dashboard, meal logs, and profile settings
- Components: Home screen, meal history, user profile, settings

b) Social Hub

- Functionality: A place where users can find friends, view leaderboards, and send challenges
- Features: Friend discovery, global/friend leaderboards, challenge creation

c) Badge and Streak Case

- Library: lottie-react-native
- Functionality: A visually engaging profile area that displays earned achievements and active streaks
- Animations: High-quality Lottie animations for gamification feedback

d) Push Receiver

- Functionality: A background service that listens for messages (e.g., "Challenge Received") and displays them even if the app is closed
- Integration: Works with Expo Push Notifications

e) Camera Module

- Library: expo-camera
- Functionality: Captures food photographs to be sent to the AI for nutritional analysis
- Permissions: Manages camera access permissions

f) Chat Interface

- Library: react-native-gifted-chat
- Functionality: Provides conversational UI for AI assistant (NutriCoach)
- Features: Message bubbles, typing indicators, timestamp display

g) Secure Storage

- Library: expo-secure-store
- Functionality: Securely stores JWT tokens and user preferences
- Security: Platform-specific encryption (Keychain on iOS, Keystore on Android)

h) Navigation Service

- Libraries: @react-navigation/native, @react-navigation/native-stack
- Functionality: Manages screen transitions and navigation flow
- Navigation Patterns: Stack navigation, tab navigation

i) Onboarding & Registration Flow

- Functionality: Guides new users through account creation and initial setup
- Components: Login, signup, age/gender input, weight/height input, goal selection, avatar picker
- Purpose: Collects user data required for personalized calorie calculations

j) Meal Logging Interface

- Functionality: Allows users to add meals through search or camera scan

- Components: Meal category buttons (Breakfast/Lunch/Dinner), search bar, scan button, calorie progress indicator
- Integration: Works with Camera Module and Backend API

k) Date Navigation

- Functionality: Allows users to view and navigate between different days
- Components: Date picker with previous/next day navigation
- Purpose: Review or log meals for past days

4.2. Backend API Subsystem

The backend API subsystem serves as the core application logic layer, orchestrating communication between the mobile client, database, and external services. It enforces business rules, validates data, manages authentication, and processes requests.

Technology Stack:

- Runtime: Node.js 18+ LTS
- Language: TypeScript
- Web Framework: Express.js
- Validation: Zod
- Authentication: JSON Web Tokens (JWT)

Architecture Layers:

a) Web Server Layer

- Framework: Express.js
- Responsibilities: HTTP request routing, middleware execution (CORS, logging, error handling), request parsing

b) Validation Layer

- Library: Zod
- Responsibilities: Runtime validation of request bodies, type-safe schema definitions
- Use Cases: User registration, meal logging, challenge creation

c) Business Logic Layer

- Structure: Controller-Service pattern
- Orchestrator: Acts as a bridge connecting the mobile app, database, and AI tools
- Key Controllers:
    - AuthController: Handles user registration, login, token refresh
    - MealController: Processes meal creation, retrieval, update, deletion

    - ChatController: Manages chatbot message exchanges
    - GoalController: Calculates and updates user nutrition goals
    - GamificationController: Awards points, tracks streaks, unlocks achievements
    - SocialController: Manages posts, follows, and challenges

d) Authentication & Authorization Service

- Mechanism: JWT-based stateless authentication
- Token Management: Issuance, verification, refresh
- Security: Tokens stored securely in expo-secure-store on client

e) Challenge Logic

- Functionality: Manages challenge rules and verification
- Example: If User A challenges User B, the server tracks User B's meals to verify completion

f) Reward System

- Functionality: Automatically awards points and badges when users reach goals
- Integration: Works with GamificationController

g) Matchmaking

- Functionality: Handles friend requests and social connections
- Features: Follow/unfollow, friend suggestions

4.3. Data Access Subsystem

The data access subsystem abstracts database interactions, providing a clean interface between business logic and persistent storage. It manages schema definitions, query construction, and data validation at the database level.

Technology Stack:

- ORM: Prisma (@prisma/client)
- Database: CockroachDB
- Migration Tool: Prisma Migrate

Key Services:

a) ORM Service (Prisma)

- Responsibilities: Maps tables to TypeScript types, constructs type-safe queries, manages connections
- Features: Connection pooling, transaction support, code generation
- Example: await prisma.meal.create({ data: {...} })

b) Database Service

- Platform: Managed PostgreSQL with additional features

- Features: Real-time subscriptions, row-level security (RLS), automated backups
- Access: Via Prisma using connection string

c) Data Models

Primary Domains:

- User Domain: Stores user identity information, current level, total points, daily calorie goals
- Global Food Library: A read-only, pre-populated catalog containing thousands of standard food items with verified macronutrient values and portion sizes
- Nutritional Logs: Records food meals (referencing Global Library or AI predictions) and water hydration with timestamps
- Gamification & Social Graph: Tracks challenges, earned badges, and follower/following relationships

Key Collections:

- User: Account information, profile data, goals, gamification metrics
- Meal: Meal logs with nutritional data and timestamps
- Post: User-generated content (recipes, progress updates)
- Achievement: Badge definitions and unlock criteria
- Challenge: Challenge definitions with status (Pending, Accepted, Completed)
- Follow: Follower/following relationships
- ChatMessage: Chatbot conversation history
- NotificationHistory: Log of past alerts for users to review later

4.4. AI Intelligence Subsystem

The AI Intelligence Subsystem serves as the external smart layer using Google Gemini and Hugging Face APIs.

Technology Stack:

- Providers: Google Gemini API, Hugging Face
- Integration Pattern: Backend acts as gateway
- GPU Infrastructure: High-performance GPU clusters for AI computation

Main Responsibilities:

a) Food Recognition

- Functionality: Analyzes user-uploaded photos using object detection to identify food items and estimate calories automatically
- Model: Custom-trained computer vision model
- Use Cases: Photo-based meal logging
- Error Handling: Circuit Breaker Pattern - falls back to manual search if AI fails

b) NutriCoach (Psychological Support Chatbot)

- Functionality: Uses NLP to understand user messages and provide psychological support, motivation, and wellness advice
- Model: Gemini API
- Features: Context-aware responses, empathetic coaching
- Privacy: Personal information filtered before sending to AI

c) Error Handling & Fallbacks

- Circuit Breaker: If AI service times out (>10 seconds), system falls back to manual mode
- Invalid Input Detection: If confidence score < threshold, rejects input with user-friendly message
- Privacy Protection: Strips EXIF metadata from images before AI processing

4.5. Notification Subsystem

The Notification Subsystem is responsible for real-time alerts and scheduled reminders.

Technology Stack:

- Platform: Expo Push Notification Service
- Integration: Firebase Cloud Messaging (FCM) as alternative (Expo uses it)
- Delivery: Push notifications to iOS and Android devices

Main Responsibilities:

a) Instant Alerts

- Functionality: Sends immediate push notifications for time-sensitive events
- Examples:
  - "Friend Request Accepted"
  - "Challenge Won"
  - "@Dwight has completed the challenge you assigned!"
  - "You've unlocked a new badge!"
- Triggers: Goal achievements, challenge updates, social interactions

b) Smart Reminders

- Functionality: Runs scheduled jobs to remind users to log meals
- Logic: Checks if user hasn't logged meals by specific times
- Examples:
  - "Don't forget to log your lunch!"
  - "You're on a 5-day streak - keep it going!"

4.6. Subsystem Communication

Subsystems communicate through well-defined interfaces and protocols:

- **Mobile Client ↔ Backend API**
  - Protocol: RESTful API over HTTPS
  - Data Format: JSON
  - Authentication: JWT tokens in Authorization header
  - Example: POST /api/meals with JSON payload

- **Backend API ↔ Database (CockroachDB via Prisma)**
  - Protocol: PostgreSQL wire protocol over TLS
  - ORM: Prisma client for type-safe queries
  - Connection: Connection pooling for performance
  - Example: await prisma.meal.create({ data: {...} })

- **Backend API ↔ AI Services (Gemini/Hugging Face)**
  - Protocol: HTTPS
  - Data Format: JSON (text), Binary (images)
  - Authentication: API keys
  - Example: POST to Gemini API with user message or food image

- **Backend API ↔ Notification Service (Expo)**
  - Protocol: HTTPS
  - Data Format: JSON
  - Authentication: Expo access token
  - Example: Send push notification with title, body, and data payload

4.7. Subsystem Deployment

Mobile Client Subsystem

- Platform: iOS (14+) and Android (11.0+)
- Build Tool: Expo Application Services (EAS Build)

Backend API Subsystem

- Hosting: Cloud platform
- Containerization: Docker
- Runtime: Node.js 18+ LTS with TypeScript compilation
- Build Process: npm install && npm run build
- Deployment: Git push triggers automatic CI/CD pipeline
- Environment Variables: Database URL, API keys, JWT secret
- Region: Europe (GDPR compliance) or geographically optimized

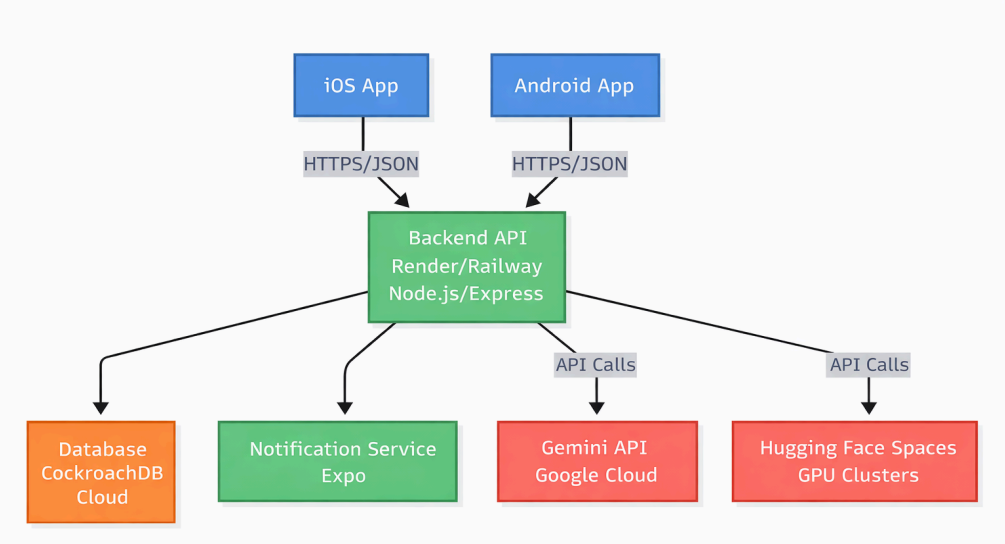
Database Subsystem

- Hosting: CockroachDB(cloud-hosted PostgreSQL)
- Tier: Free tier (development), Pro tier (production)

- Backup: Automated daily snapshots
- Migration: Prisma Migrate manages schema changes

External Services

- AI Services: Gemini API (Google Cloud), Hugging Face Spaces (GPU-enabled)
- Notification Service: Expo Push Notification Service
- Region: Distributed globally for low latency



External Services of the System Diagram

CI/CD Pipeline:

- Version Control: GitHub/GitLab
- Development Branch: Auto-deploy to staging environment
- Main Branch: Auto-deploy to production environment
- Testing: Automated tests run before deployment
- Monitoring: Application logs via platform dashboards

4. Glossary

This glossary provides definitions for technical terms, acronyms, and domain-specific concepts referenced throughout this document.

AI (Artificial Intelligence) : Machine learning systems that analyze data to make intelligent predictions. In NutriGame, AI identifies food items from photos and generates personalized chatbot responses.

AI Chatbot : An AI-powered conversational interface providing nutrition advice, diet guidance, and motivational support. Implemented using Gemini API or Hugging Face language models.

Anonymization : The process of removing personally identifiable information from datasets to protect user privacy. Applied to images and chat data before sending to external AI services.
API (Application Programming Interface) : A set of protocols enabling software applications to communicate and exchange data. NutriGame uses APIs to connect the mobile app, backend server, and external services.
Asynchronous Execution : A programming approach where operations run in the background without blocking the main application. Ensures the UI remains responsive while waiting for AI processing or network requests.
Badge : A digital achievement awarded to users for reaching specific milestones (e.g., 7-day streak, completing challenges).

BMI (Body Mass Index) : A numerical value derived from weight and height (kg/m²) used to categorize body composition.
Bounding Box : A rectangular frame indicating the position of a detected object within an image. Used in food recognition to isolate food items.
Calorie : A unit of energy. In nutritional contexts, typically refers to kilocalories (kcal). NutriGame tracks daily calorie intake to help users meet health goals.
Challenge : A community goal-oriented event where users commit to achieving a specific objective (e.g., "7-Day Water Challenge"). Created by users and shared with followers through invitations.
Circuit Breaker Pattern : A design pattern that prevents system failures by detecting errors and falling back to alternative functionality. Used when external AI services are unavailable.
Client-Server Architecture : A distributed computing model where client applications request services from centralized servers. NutriGame uses this architecture with mobile clients and cloud backend.
CSV (Comma-Separated Values) : A simple file format for storing tabular data. Used for importing or exporting datasets in NutriGame development.
Data Processing : The collection, transformation, and analysis of nutrition, health, and activity data to provide insights and recommendations.
Data Scraping : The automated extraction of data from websites or online sources. May be used to populate the Global Food Library with nutritional information.
Event-Driven Architecture : A software design pattern where system components respond to events (user actions, API calls, database changes) rather than continuous polling.
EXIF Metadata : Data embedded in image files containing information such as camera settings, timestamp, and GPS location. Stripped from food photos before AI processing to protect privacy.
Express.js : A minimal and flexible Node.js web application framework. Used to build NutriGame's backend REST API with routing and middleware support.
Gamification : The integration of game mechanics (points, badges, leaderboards, challenges) into non-game applications to increase user engagement and motivation.

GDPR (General Data Protection Regulation) : European Union regulation governing data protection and privacy. NutriGame complies with GDPR principles including user consent and data minimization [15].
Global Food Library : A pre-populated database containing thousands of standard food items with verified nutritional values (calories, protein, fats, carbs) and portion sizes. Enables quick food selection without manual entry.
Hugging Face : A platform for hosting and deploying machine learning models. NutriGame will use Hugging Face for food recognition.
Image Classification Dataset : A labeled collection of food images categorized by food type. Used to train machine learning models for automated food recognition.
JSON (JavaScript Object Notation) : A lightweight, human-readable data format for transmitting structured data. Used for API communication between mobile app and backend.
JWT (JSON Web Token) : A compact, URL-safe token format for securely transmitting authentication credentials. Used to maintain user sessions in NutriGame.

KVKK (Kişisel Verilerin Korunması Kanunu) : Turkish Personal Data Protection Law. NutriGame complies with KVKK requirements for user consent, data security, and privacy protection [16].
Label Encoding : The process of converting text labels into numerical values for machine learning model training. Example: "apple" → 1, "banana" → 2.
Lottie Animation : A library for rendering animations in JSON format. Used in NutriGame to display visually engaging gamification feedback (badge unlocks, streak celebrations).
Meal Logging : The feature allows users to record their meals in the system through manual entry, text input, or photo capture with AI recognition.
ML (Machine Learning) : A subset of AI focused on training algorithms to learn patterns from data. Used in NutriGame for food recognition and personalized recommendations.
Natural Language Processing (NLP) : A branch of AI focused on enabling computers to understand, interpret, and generate human language. Used in chatbot conversations and text-based meal logging.
NutriCoach : The AI-powered psychological support component of NutriGame that provides motivational messages and wellness advice through conversational interactions.
NutriGame : An intelligent mobile application for nutrition tracking, meal planning, and healthy living with AI-powered features, gamification, and social interaction
Object Detection : A computer vision technique that identifies and locates objects within an image by drawing bounding boxes around them. Used in NutriGame to detect multiple food items in a single meal photo.
OWASP (Open Web Application Security Project) : A nonprofit organization providing resources and best practices for web application security. NutriGame follows OWASP guidelines for secure development.
P0 (Critical Priority) : A bug severity level indicating issues that block core functionality and must be fixed immediately before release.
P1 (High Priority) : A major functionality issue requiring a fix in the next release cycle.

P2 (Medium Priority) : A minor functionality issue with an available workaround.
P3 (Low Priority) : A cosmetic or non-critical issue that does not significantly impact functionality.
PostgreSQL : An open-source relational database management system. Used via CockroachDB for storing NutriGame's structured data.
Prisma ORM : An object-relational mapping tool that provides a type-safe interface between Node.js application code and the PostgreSQL database.
Push Notification : A server-initiated message delivered to a user's mobile device that appears even when the application is not actively running.
React Native : A framework for building cross-platform mobile applications using JavaScript/React. NutriGame uses Expo, a toolchain built on React Native.
Regression Test : Testing performed to ensure that existing functionality continues to work correctly after new code changes are introduced.
RESTful API : An architectural style for networked applications using stateless HTTP requests. NutriGame's backend follows REST principles with standardized methods (GET, POST, PUT, DELETE).

Row Level Security (RLS) : A database security feature that restricts data access at the row level based on user identity. Ensures users can only access their own data.
Semantic Versioning : A version numbering standard using the format MAJOR.MINOR.PATCH to indicate the type and scope of changes in software releases.
Static Analysis : The process of analyzing source code without executing it to detect potential errors, security vulnerabilities, or code quality issues.
Stop-word Removal : The process of filtering out common, insignificant words (e.g., "the", "is", "and") in natural language processing to improve analysis accuracy.
Streak : The number of consecutive days a user has successfully met their daily goals. Designed to encourage habit formation and sustained engagement.
CockroachDB Cloud: A distributed, PostgreSQL-compatible SQL database service that provides strong consistency, automatic replication, fault tolerance, and high availability for application data.
Traceability Matrix : A table that maps software requirements to their corresponding test scenarios, ensuring complete test coverage.
TypeScript : A programming language that extends JavaScript with static type checking. Used in NutriGame backend to improve code quality and catch errors early.
UAT (User Acceptance Testing) : The final phase of testing where actual users validate that the system meets their requirements and expectations before deployment.
UI (User Interface) : The visual components and layouts that users interact with in an application. NutriGame emphasizes clean, intuitive UI design.
UX (User Experience) : The overall experience a user has when interacting with an application, including usability, accessibility, and satisfaction.
Zod : A TypeScript-first schema validation library. Used in NutriGame backend to validate incoming API requests and ensure data integrity.

5. References

[1] K. R. Schneider, J. J. Liu, and A. A. Voida, “Retention and Engagement in Mobile Health Apps: A Longitudinal Study of User Behavior,” Journal of Medical Internet Research, vol. 26, no. 1, p. e56897, Jan. 2024. [Online]. Available: https://www.jmir.org/2024/1/e56897

[2] C. R. Maher, K. R. Ferguson, and J. Olds, “When and Why Adults Abandon Lifestyle, Behavior, and Mental Health Apps: Qualitative Analysis of User Experiences,” JMIR mHealth and uHealth, vol. 10, no. 11, p. e38938, Nov. 2022. [Online]. Available: https://researchnow.flinders.edu.au/en/publications/when-and-why-adults-abandon-lifestyle-behavior-and-mental-health

[3] S. Michie et al., “The Behaviour Change Wheel: A Guide to Designing Interventions,” Silverback Publishing, 2014.

[4] J. Hamari, J. Koivisto, and H. Sarsa, “Does Gamification Work? — A Literature Review,” HICSS, 2014.

[5] E. L. Deci and R. M. Ryan, “Self-Determination Theory,” Psychological Inquiry, 2000.

[6] MyFitnessPal Official Website. https://www.myfitnesspal.com

[7] Lifesum Official Website. https://lifesum.com

[8] AteMate Official Website. https://www.atemate.io

[9] R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” PhD Dissertation, 2000.

[10] Meta, “React Native Documentation.” https://reactnative.dev

[11] Cockroach Labs, “CockroachDB Architecture Overview.” https://www.cockroachlabs.com/docs/

[12] A. Silberschatz et al., “Database System Concepts,” McGraw-Hill, 7th Edition.

[13] RFC 7519 – JSON Web Token (JWT). https://www.rfc-editor.org/rfc/rfc7519

[14] PostgreSQL Documentation – Row Level Security. https://www.postgresql.org/docs/current/ddl-rowsecurity.html

[15] GDPR – General Data Protection Regulation. https://gdpr.eu

[16] KVKK – Kişisel Verilerin Korunması Kanunu. https://www.kvkk.gov.tr

[17] M. Fowler, “Event-Driven Architecture.” https://martinfowler.com

[18] M. Nygard, “Release It!: Design and Deploy Production-Ready Software,” Pragmatic Bookshelf.