FOOD NINJA

Technical Documentation

Architecture, Technologies, APIs, and AI Systems

Version 1.0
Document Date: March 13, 2026
Authored by: Albert Huang, Food Ninja Developer
Contact: alberthuang@food-ninja.com

App Name	Food Ninja
Platform	iOS (iPhone & iPad)
Developer	Albert Huang
Developer Location	Taiwan
Server Location	Nuremberg, Germany (Hetzner Online GmbH, EU)
Document Date	March 13, 2026
Document Version	1.0
Authored By	Albert Huang
Contact	alberthuang@food-ninja.com | support@food-ninja.com

1. Overview

Food Ninja is an iOS application designed to help individuals and households reduce food waste by tracking food inventory, expiration dates, and connecting users with donation opportunities. The app leverages on-device machine learning, voice recognition, AI-powered meal planning, and cloud-based subscription management to deliver an integrated, privacy-respecting experience.

This document provides a transparent, detailed account of all technologies, frameworks, APIs, third-party libraries, AI models, database systems, backend architecture, and platform integrations used to build and operate Food Ninja. It is intended for technically sophisticated readers who wish to understand exactly how the application is constructed and what systems it depends on.

2. Application Architecture

Food Ninja follows a client-server architecture with a clear separation of concerns:

- iOS Client (SwiftUI): all user-facing interaction, on-device data storage, on-device ML, voice transcription, camera capture, calendar integration, notifications, and widgets.
- Backend Server (Deno / TypeScript): authentication, subscription management, AI query orchestration, rate limiting, App Store transaction verification, and Google OAuth token management.
- Database (PostgreSQL): persistent server-side storage for user accounts, subscription state, App Store transaction metadata, credit usage, and Google refresh tokens.

All food records, expiration data, camera images, audio recordings, and on-device OCR results are stored exclusively on the user's device and are never transmitted to or stored by the server, except transiently when passed as context to AI queries (which are not logged or persisted).

3. iOS Client Technology Stack

3.1 Language & UI Framework

Technology	Version / Details	Purpose
Swift	Swift 6 (Strict Concurrency)	Primary programming language for all iOS application code
SwiftUI	iOS 26+	Declarative UI framework for all views, navigation, and animations

Technology	Version / Details	Purpose
Swift Concurrency	async/await, actors, Sendable	Structured concurrency for all asynchronous operations throughout the app

3.2 Data Persistence

Technology	Purpose
SwiftData	On-device persistent storage for food records (Record), app settings (AppSettings), donation center cache (PlaceModel), and AI chat history (ChatThread, ChatMessage). SwiftData is Apple's modern Swift-native persistence framework backed by Core Data.
App Group Container	Shared SwiftData model container enabling data sharing between the main app and home screen widgets.
UserDefaults	Lightweight key-value storage for user preferences, sync state identifiers (Google Calendar ID, Apple Calendar identifier, Reminder List identifier), and onboarding state.
Keychain Services	Secure encrypted storage for JWT authentication tokens using the system Keychain (kSecClassGenericPassword), scoped to the app's bundle identifier.

3.3 On-Device Machine Learning & Computer Vision

Food Ninja uses Apple's Vision framework exclusively for on-device image analysis. No images or camera frames are ever sent to a remote server for classification.

API / Model	Purpose
Apple Vision Framework (VNClassifyImageRequest / ClassifyImageRequest)	On-device food image classification. When a user photographs a food item, the Vision framework runs Apple's built-in image classification model on-device to produce classification identifiers and confidence scores. The top results are extracted as human-readable hints (e.g., 'apple', 'banana peel') to be passed as context to the backend AI.
Apple Vision Framework (VNRecognizeTextRequest / RecognizeTextRequest)	On-device Optical Character Recognition (OCR). Reads text from food packaging to detect expiration date labels in multiple languages, including English and Chinese (Traditional).

API / Model	Purpose
NSDataDetector	System text analysis to extract and normalize date expressions from OCR-recognized text.
CoreImage (CIContext)	Image preprocessing pipeline for CGImage conversion prior to Vision framework analysis.
ImageIO	Low-level image format handling and orientation management.

3.4 Speech Recognition & Voice Input

API / Framework	Purpose
Apple Speech Framework (SFSpeechAnalyzer)	Real-time on-device speech-to-text transcription for the Voice Log feature. Users speak food item names and details; the transcript is converted to text on-device before being sent to the backend AI for structured record generation.
SFSpeechTranscriber	Streaming transcription component providing real-time partial transcript segments with speaker turn detection.
AVFoundation (AVAudioEngine)	Low-level audio capture pipeline, microphone input session management, and audio routing.
Speech Framework Authorization	SFSpeechRecognizer.requestAuthorization — explicit user permission required before any speech processing begins.

3.5 Authentication

Framework / API	Purpose
AuthenticationServices (Sign in with Apple)	ASAuthorizationAppleIDProvider, ASAuthorizationController. Apple's native Single Sign-On framework. Requests .fullName and .email scopes. Issues an Apple identity token (JWT) sent to the backend for verification and account creation.
GoogleSignIn SDK (Google Sign-In for iOS)	GIDSignIn. Google's official iOS SDK for OAuth 2.0 authentication. Requests profile, email, and openid scopes, plus offline_access for obtaining a server-side refresh token to enable Google Calendar and Google Tasks sync. Configured via GoogleService-Info.plist.
Keychain Services	Stores the app's custom JWT session token securely in the iOS Keychain after successful authentication.

3.6 In-App Purchases & Subscription Management

Framework / API	Purpose
StoreKit 2	Apple's modern Swift-native in-app purchase framework. Used for fetching subscription products (Product.products(for:)), initiating purchase flows (product.purchase()), and observing real-time transaction updates (Transaction.updates async stream).
Transaction.currentEntitlements	StoreKit 2 API to verify current subscription entitlements locally on-device before syncing with the backend.
AppStore.sync()	Forces a StoreKit transaction sync with the App Store, used for account recovery flows.

3.7 Calendar & Reminders Integration

Framework / API	Purpose
EventKit (Apple Calendar)	EKEventStore, EKEvent, EKCalendar. Writes food expiration records as all-day calendar events to a dedicated 'Food Ninja' calendar in Apple Calendar. Requires explicit user permission (requestFullAccessToEvents). All processing is on-device; no server involvement.
EventKit (Apple Reminders)	EKReminder, EKCalendar (for Reminders). Creates expiration reminders in a dedicated reminder list. Requires explicit user permission (requestFullAccessToReminders). All processing is on-device.
Google Calendar REST API v3	Server-side HTTP API (https://www.googleapis.com/calendar/v3). Used to create, update, and delete all-day calendar events in a dedicated 'Food Ninja' Google Calendar on the user's Google account. Authenticated via Google OAuth 2.0 access tokens obtained using the stored refresh token. Managed by the backend server on the user's behalf.
Google Tasks REST API v1	Server-side HTTP API (https://tasks.googleapis.com/tasks/v1). Creates and manages tasks with due dates in a dedicated task list. Authenticated identically to Google Calendar. Also managed by the backend server.

3.8 Maps & Location

Framework / API	Purpose
MapKit	MKMapView, Map SwiftUI view. Displays an interactive map for donation center discovery and browsing, including pin annotations for nearby centers.
Core Location	CLLocationManager. Requests When In Use location authorization to determine the user's current position for distance-based donation center sorting and map centering. Location data is processed on-device and never transmitted to our servers.

3.9 Notifications

Framework / API	Purpose
UserNotifications	UNUserNotificationCenter, UNMutableNotificationContent, UNCalendarNotificationTrigger. Schedules local push notifications for upcoming food expirations at specific intervals: monthly, 7 days, 3 days, 5 days, 1 day, day-of, and post-expiry. All notifications are local (no server push involvement).
UNUserNotificationCenterDelegate	Custom NotificationDelegate handles foreground notification presentation and notification-tap deep linking into specific records.

3.10 Home Screen Widgets

Framework / API	Purpose
WidgetKit	Widget extension providing home screen and lock screen widgets that display expiring food items. Uses shared App Group SwiftData container to read live food record data without launching the main app.
AppIntentTimelineProvider	Widget timeline generation using App Intents for configuration.
SwiftData (Widget Extension)	Direct read access to the shared model container for fetching food records sorted by expiration date.

3.11 Siri & App Shortcuts

Framework / API	Purpose
AppIntents	AddFoodRecordIntent and ListExpiringFoodIntent. Exposes key app actions to Siri, Spotlight, and the Shortcuts app. Users can add food records by voice via Siri without opening the app.
AppShortcuts	FoodNinjaShortcuts AppShortcutsProvider. Registers pre-built Siri phrase shortcuts (e.g., 'Add food to Food Ninja') discoverable without manual user setup.

3.12 Feature Discovery

Framework / API	Purpose
TipKit	Apple's TipKit framework for contextual in-app feature tips. Tips are displayed conditionally based on user behavior (e.g., first login, first record created) to guide users to key features without overwhelming them.

4. Backend Server Technology Stack

The Food Ninja backend is a standalone TypeScript REST API server. It handles authentication, subscription verification, AI orchestration, rate limiting, and Google OAuth token management.

4.1 Runtime & Language

Technology	Version	Purpose
Deno	Latest stable	Server-side JavaScript/TypeScript runtime. Deno is used as the backend execution environment. It provides built-in TypeScript support, a permission-based security model and native async I/O.
TypeScript	Deno-native	All backend logic is written in TypeScript with strict typing for request/response payloads, database queries, subscription states, and AI model configuration.

4.2 HTTP Framework & Middleware

Library	Version	Purpose
Oak (deno.land/x/oak)	N/A for security reasons	HTTP server framework for Deno. Provides routing (Router), middleware pipeline (Application), request/response context management, and structured error handling.
oakCors (deno.land/x/cors)	N/A for security reasons	CORS (Cross-Origin Resource Sharing) middleware for Oak. Configures allowed origins for browser-based access.

4.3 Database

Technology	Version / Details	Purpose
PostgreSQL	Managed by the developer on the server off Hetzner (EU, Germany)	Relational database for all server-side persistent storage. Used with the pgcrypto extension for UUID primary key

Technology	Version / Details	Purpose
		generation (gen_random_uuid()).
postgres.js (deno.land/x/postgresjs)	N/A for security reasons	PostgreSQL client for Deno. Used for all database queries via tagged template literals. Supports connection pooling and parameterized queries to prevent SQL injection.

4.3.1 Database Schema Summary

The PostgreSQL database contains the following tables:

Table	Purpose
Users Table	Stores user accounts: UUID primary key, email, first/last name, OAuth provider (apple or google), and provider-specific user ID. Indexed on email and provider+provider_user_id.
Subscriptions Data Table	Stores per-user subscription state, billing period, and consumed credits count. Linked to users via foreign key with CASCADE deletion.
Credentials Table	Stores encrypted Google OAuth refresh tokens (as BYTEA) for users who sign in with Google and enable calendar/task sync. Permanently deleted on account deletion after OAuth revocation.
Transactions Data Table	Stores Apple App Store JWS-signed transaction and renewal payloads for subscription lifecycle management and billing dispute resolution.
Grounding Table	Tracks daily aggregate usage of Gemini grounding (real-time web search) requests to enforce server-level daily quotas across all users.
Migrations Data Table	Migration tracking table: records which SQL migration files have been applied, enabling safe incremental schema upgrades.

4.4 Authentication & Token Management

Library / Standard	Version	Purpose
djwt (deno.land/x/djwt)	N/A for security reasons	JWT (JSON Web Token) creation and verification for the app's custom session tokens. Used to issue signed JWTs to authenticated users after Apple or Google identity verification,

Library / Standard	Version	Purpose
		and to verify those tokens on all subsequent API requests.
jose (npm:jose)	N/A for security reasons	Full-featured JWT/JWK library for Apple token verification. Used to verify Apple's ES256-signed identity tokens against Apple's public JWKS endpoint (https://appleid.apple.com/auth/keys).
@peculiar/x509 (npm:@peculiar/x509)	latest	X.509 certificate parsing for App Store receipt verification. Used to parse and validate Apple's root CA certificates when verifying JWS-signed App Store transaction payloads during server-to-server notification processing.
google-auth-library (npm:google-auth-library)	N/A for security reasons	Google's official OAuth 2.0 client library. Used to verify Google ID tokens (OAuth2Client.verifyIdToken), exchange authorization codes for refresh/access token pairs, and refresh access tokens using stored refresh tokens for Google Calendar and Tasks API calls.
Apple Sign In (SIWA)	Apple Identity Service	Verifies Apple identity tokens using Apple's public JWKS (JSON Web Key Set) fetched from https://appleid.apple.com/auth/keys. Token audience (aud) is validated against the app's bundle ID.
Google Sign In OAuth 2.0	Google Identity Platform	Verifies Google ID tokens and manages OAuth 2.0 authorization code exchange. Google refresh tokens are encrypted (AES-GCM via Web Crypto API) before storage in the database.

4.5 AI & Generative AI

Food Ninja uses Google's Gemini family of generative AI models exclusively for all AI-powered features. The AI is accessed through the backend server; the iOS client never calls AI APIs directly.

Library	Version	Purpose
@google/generative-ai (npm:gemini)	latest (npm)	Google's official Generative AI JavaScript SDK. Used to interact with the Gemini API for all AI-powered backend features,

Library	Version	Purpose
		including chat responses, voice log parsing, photo capture item generation, and conversation title generation.

4.5.1 AI Models Used

The following Google Gemini models are deployed by the backend, selectable per request based on task type and subscription plan:

Model Name	Model ID	Role in Food Ninja
Gemini 2.5 Flash Lite	gemini-2.5-flash-lite	Lightweight, fast model. Used for lower-cost AI tasks including basic chat responses, voice log record generation, photo capture record generation, and conversation title generation. Default for the Freemium plan. Smarter answers with the same model for the Helper plan.
Gemini 2.5 Flash	gemini-2.5-flash	Mid-tier model with stronger reasoning. Used for recipe generation, meal planning and grounding-enabled searches. Available to Ninja subscribers.
Gemini 3 Flash Preview (with Extended Thinking)	gemini-3-flash-preview w/thinking	Highest-tier model with extended thinking (chain-of-thought reasoning). Used for donation eligibility checks to ensure safety. Available to Ninja subscribers. When 'thinking' is enabled, the model performs multi-step reasoning before generating a response.

4.5.2 AI Features & Task Types

AI requests are categorized by task type, which determines model selection and system prompt behavior:

Task Type	Description	Models Used
recipe	Generates structured recipes from the user's food inventory. Output includes recipe title, servings, time estimate,	Flash Lite or Flash

Task Type	Description	Models Used
	ingredient list with amounts, step-by-step instructions, and tags.	
meal_plan	Generates a multi-day structured meal plan based on pantry contents. Output includes daily meal schedules, prep notes, and a shopping list.	Flash Lite or Flash
donation_safety	Assesses whether specific food items are safe and appropriate to donate to a given donation center based on the center's accepted items, hours, and policies.	Flash Lite or Flash Thinking
general	General-purpose food-related chat. Handles pantry questions, cooking advice, nutrition queries, and anti-food-waste guidance.	Flash Lite or Flash
voice_log	Parses a voice transcript to extract structured food record data: item names, emoji categories, storage locations, expiration dates, and notes.	Flash Lite
capture	Combines on-device Vision classification hints, OCR text, and optional voice transcript to generate one or more structured food records.	Flash Lite
title	Generates a short, descriptive conversation title from the first user message.	Flash Lite

4.5.3 Grounding (Real-Time Web Search)

For certain AI tasks (primarily donation safety assessments), the backend may enable Gemini's 'grounding' capability. When grounding is enabled, Gemini accesses real-time information from the web to supplement its response with up-to-date details (e.g., current donation center policies, food safety guidelines). Grounding is subject to daily server-wide quotas and is not enabled for every request. The app indicates to the user when grounding was used and whether it was limited due to quota constraints.

4.6 App Store Server API

Food Ninja uses Apple's App Store Server API and Server-to-Server Notifications V2 to manage subscription lifecycles authoritatively.

API / Standard	Purpose
App Store Server Notifications V2 (HTTPS POST endpoint)	Apple sends signed server-to-server webhook notifications for subscription events: SUBSCRIBED, DID_RENEW, DID_FAIL_TO_RENEW, EXPIRED, REFUND, CANCEL, GRACE_PERIOD_EXPIRED, etc. The backend verifies the JWS payload and updates subscription state accordingly.
JWS (JSON Web Signature) Transaction Decoding	All App Store transaction and renewal payloads are delivered as JWS-encoded strings. The backend decodes and verifies these using Apple's certificate chain (validated against Apple's root CA certificates) using @peculiar/x509 and jose.
App Store Server API — Get All Subscription Statuses	GET https://api.storekit.itunes.apple.com/inApps/v1/subscriptions/{originalTransactionId}. Used to query the current authoritative subscription state from Apple during sync operations.
App Store Connect API (JWT Bearer Auth)	The backend generates a signed ES256 JWT using the App Store Connect private key (P8 format) and key ID to authenticate requests to the App Store Server API.

4.7 Rate Limiting & Security

Mechanism	Details
IP-based Rate Limiting	Enforced. Details N/A for security reasons.
User-based Rate Limiting	Enforced. Details N/A for security reasons.
Burst Allowance	Enforced. Details N/A for security reasons.
JWT Authentication	Enforced. Details N/A for security reasons.
HTTP Security Headers	Enforced. Details N/A for security reasons.
Input Validation	Maximum character limits are enforced server-side on all AI request fields: message content, conversation history, and record attachment data to prevent prompt injection and resource exhaustion.
Request Tracing	Traced with headers for debugging purposes. Details N/A for security reasons.

5. Third-Party Services & External APIs

Service	Provider	Purpose in Food Ninja
Gemini Generative AI API	Google LLC	All server-side AI inference: recipe generation, meal planning, voice log parsing, food capture analysis, donation safety assessments, and chat. Accessed via the @google/generative-ai SDK with an API key.
App Store Server API	Apple Inc.	Authoritative subscription lifecycle management. Receives server-to-server webhook notifications for all subscription events and allows querying subscription status by original transaction ID.
Apple Identity Services (SIWA)	Apple Inc.	Identity token issuance and verification for Sign in with Apple. JWKS endpoint: https://appleid.apple.com/auth/keys.
Google Identity Platform (OAuth 2.0)	Google LLC	Identity token verification and OAuth 2.0 authorization code exchange for Sign in with Google. Manages refresh token lifecycle for Google Calendar/Tasks sync.
Google Calendar API v3	Google LLC	Creates, updates, and deletes calendar events on behalf of users who enable Google Calendar sync. Requires user-granted OAuth 2.0 offline access.
Google Tasks API v1	Google LLC	Creates and manages task list entries on behalf of users who enable Google Tasks sync. Requires user-granted OAuth 2.0 offline access.
Apple Calendar (EventKit)	Apple Inc.	On-device calendar sync via EventKit framework. No network calls to Apple servers; processed entirely locally.
Apple Reminders (EventKit)	Apple Inc.	On-device reminders sync via EventKit framework. Entirely local; no server involvement.
Apple Maps (MapKit)	Apple Inc.	Interactive map display for donation center discovery. MapKit is an Apple-provided SDK using Apple's mapping tile servers.

Service	Provider	Purpose in Food Ninja
Hetzner Online GmbH	Hetzner (EU/Germany)	Cloud hosting provider for the backend server with the PostgreSQL database. All user data at rest is physically located in Nuremberg, Germany, subject to GDPR.

6. Data Flow & Privacy Architecture

6.1 On-Device Only Data

The following data is processed and stored exclusively on the user's device and is never transmitted to or stored by Food Ninja's servers:

- Food records: names, categories, storage locations, expiration dates, notes, emoji identifiers
- Camera images and photos captured in-app
- Audio recordings and raw voice input
- On-device OCR output (expiration date text extracted from packaging)
- On-device image classification results (Vision framework output)
- Donation center browsing and interaction history
- App settings and user preferences
- Notification schedules

6.2 Data Transmitted to the Backend Server

The following data is transmitted to Food Ninja's backend server for processing:

- Authentication: Apple or Google identity tokens for account creation and login
- AI Queries: user message text, conversation history, food record data selected as context (serialized as structured JSON, not images), and donation center metadata — transmitted for real-time AI response generation and NOT stored after the response is returned
- Voice Log: speech transcript text (not audio) and on-device Vision classification output — for AI-powered structured record generation; not stored after response
- Photo Capture: on-device Vision classification hints, OCR text, optional supplemental transcript — for AI-powered item identification; not stored after response
- App Store Transactions: Apple JWS-signed transaction and renewal payloads — stored for subscription entitlement management
- Google OAuth Authorization Code: exchanged server-side for a refresh token; the refresh token is encrypted (AES-GCM) and stored solely for Google Calendar/Tasks sync and revocation on account deletion

6.3 Server-Side Persistent Data

The only data persistently stored on Food Ninja's servers is:

- User account: email address, first and last name, OAuth provider, provider user ID
- Subscription state: plan, status, billing period, AI credit usage counters
- App Store transaction metadata: transaction IDs, product IDs, purchase/expiry/revocation dates, JWS payloads (for billing dispute resolution)
- Google OAuth refresh token: encrypted, used only for sync operations and revocation
- JWT session tokens: invalidated on logout and account deletion

7. Infrastructure & Deployment

Component	Details
Cloud Provider	Hetzner Online GmbH — a European cloud infrastructure provider headquartered in Germany. Selected for GDPR compliance (all data processed within EU jurisdiction) and strong data processor agreements.
Server Location	Nuremberg, Germany (European Union). All user data is physically stored and processed in the EU regardless of user location.
Backend Runtime	Deno runtime on a Linux-based virtual machine hosted on Hetzner infrastructure.
Database	PostgreSQL, managed on server based on Hetzner infrastructure in Germany.
Transport Security	All client-server communication uses TLS (HTTPS). HTTP is not permitted in production. HSTS (HTTP Strict Transport Security) is enforced with a one-year max-age and subdomain inclusion.
Database Migrations	Automated sequential SQL migration system: migration files are applied in alphabetical order on server startup, tracked in the schema_migrations table to prevent re-application.
Environment Configuration	All secrets (API keys, JWT secrets, private keys) are loaded exclusively from environment variables at runtime. No secrets are embedded in source code or configuration files.

8. Subscription System

Food Ninja offers three subscription tiers managed jointly by Apple's App Store and the Food Ninja backend. Billing is handled exclusively by Apple; Food Ninja servers only receive transaction metadata to verify entitlement.

Plan Display Name	Product ID	Features
Freemium	(free, no purchase required)	Core pantry tracking, voice logging, photo capture, local notifications, widgets, calendar sync, and access to AI features with a basic monthly credit allocation using the Gemini Flash Lite model. Limited usage.
Helper	FOOD_NINJA_SUBPLAN_PANTRY	Higher monthly AI credit allocation, access to the Gemini Flash Lite model for improved recipe and meal planning quality, and Gemini grounding (real-time web search) for donation safety checks.
Ninja	FOOD_NINJA_SUBPLAN_CHEF	Highest monthly AI credit allocation, access to the Gemini Flash & Thinking model (extended reasoning) for complex meal plans and recipes, and all grounding capabilities.

AI credits are consumed per prompt. Each subscription plan has a defined monthly credit limit and per-prompt cost that resets at the start of each billing period. Unused credits do not roll over.

9. Open-Source & Third-Party Libraries Summary

The following table summarizes all external open-source and third-party libraries used in Food Ninja:

9.1 iOS Client Libraries

Library	Source	License	Purpose
GoogleSignIn-iOS	google/GoogleSignIn-iOS (GitHub / CocoaPods / SPM)	Apache 2.0	Sign in with Google OAuth 2.0 on iOS
SwiftUI	Apple (system)	Proprietary (Apple)	Declarative UI framework
SwiftData	Apple (system)	Proprietary (Apple)	On-device persistent storage
Vision	Apple (system)	Proprietary (Apple)	On-device image classification & OCR
Speech	Apple (system)	Proprietary (Apple)	On-device speech recognition
AVFoundation	Apple (system)	Proprietary (Apple)	Audio capture
AuthenticationServices	Apple (system)	Proprietary (Apple)	Sign in with Apple
StoreKit	Apple (system)	Proprietary (Apple)	In-app purchases
EventKit	Apple (system)	Proprietary (Apple)	Calendar & Reminders sync
MapKit	Apple (system)	Proprietary (Apple)	Maps display
UserNotifications	Apple (system)	Proprietary (Apple)	Local push notifications
WidgetKit	Apple (system)	Proprietary (Apple)	Home screen widgets
AppIntents	Apple (system)	Proprietary (Apple)	Siri & Shortcuts integration
TipKit	Apple (system)	Proprietary (Apple)	Contextual feature tips
CoreLocation	Apple (system)	Proprietary (Apple)	Device location for map centering
CoreImage	Apple (system)	Proprietary (Apple)	Image preprocessing
ImageIO	Apple (system)	Proprietary (Apple)	Image format handling
Combine	Apple (system)	Proprietary (Apple)	Reactive state management

9.2 Backend Libraries (Deno / npm)

Library	Registry	Version	License	Purpose
oak	deno.land/x/oak	N/A for security reasons	MIT	HTTP server framework & router

Library	Registry	Version	License	Purpose
oakCors	deno.land/x/cors	N/A for security reasons	MIT	CORS middleware for Oak
postgres.js	deno.land/x/postgresjs	N/A for security reasons	MIT	PostgreSQL client for Deno
djwt	deno.land/x/djwt	N/A for security reasons	MIT	JWT creation & verification
@google/generative-ai	npm	latest	Apache 2.0	Google Gemini AI SDK
google-auth-library	npm	N/A for security reasons	Apache 2.0	Google OAuth 2.0 client
jose	npm	N/A for security reasons	MIT	JWT/JWK/JWS processing for Apple token verification
@peculiar/x509	npm	latest	MIT	X.509 certificate parsing for App Store receipt verification

10. Security Practices

Area	Practice
Secrets Management	All API keys, JWT signing secrets, private keys, and database credentials are injected exclusively via environment variables at runtime. No secrets are committed to source code or configuration files.
Google Refresh Token Storage	Google OAuth refresh tokens are encrypted with (N/A) using the (N/A) API before being written to the database. The encryption key is a server environment variable. The encrypted BYTEA is meaningless without the key.
JWT Session Tokens	App-issued JWTs use (N/A) signing via djwt. All tokens are verified on every authenticated API request. Tokens are immediately invalidated (deleted from the Keychain client-side) on logout and account deletion.
Apple Token Verification	Apple identity tokens (SIWA) are verified against Apple's live JWKS endpoint (https://appleid.apple.com/auth/keys) using (N/A) asymmetric cryptography via jose. The audience claim (aud) is validated against the app's registered bundle ID.
App Store JWS Verification	App Store transaction payloads are verified against Apple's X.509 certificate chain using @peculiar/x509. Root CA certificates are pinned server-side. Online OCSP/CRL checks are optionally performed.
SQL Injection Prevention	All database queries use postgres.js parameterized tagged template literals. Dynamic SQL is never constructed via string concatenation.
Input Size Limits	Strictly enforced. Details N/A for security reasons.
Rate Limiting	Dual-layer rate limiting (per IP and per authenticated user) with sliding window counters protects against brute force, credential stuffing, and denial-of-service attacks.
HTTPS / TLS Only	All client-server communication requires TLS.
Security Response Headers	Enforced. N/A for security reasons.
Keychain Storage (iOS)	The JWT session token is stored in the iOS Keychain scoped to the app's bundle identifier. It is not stored in UserDefaults or NSCache.

Area	Practice
No AI Data Retention	AI query inputs (prompts, food record context, donation center data) are processed transiently and are not logged, stored, or retained by Food Ninja's servers after a response is returned.

11. Localization & Internationalization

Food Ninja supports multiple languages for the user interface and date/text parsing. Localization is implemented using Apple's standard Localizable.strings system and SwiftUI's built-in localization support.

- Supported UI languages: English (en), Traditional Chinese (zh-Hant)
- Date format normalization: the app handles international date formats on food packaging including yyyy-MM-dd, MM/dd/yyyy, dd/MM/yyyy, and multiple delimiter styles
- OCR expiration keyword detection: English (exp, expires, use by, best before, sell by), Traditional Chinese (有效期限, 保存期限, 到期)
- Locale-aware AI responses: the user's current device locale identifier is transmitted with all AI requests so responses are generated in the user's preferred language

12. AI Transparency Statement

Food Ninja is committed to transparency about how AI is used within the application. The following principles govern all AI features:

- AI Provider: all AI features use Google's Gemini generative AI models exclusively. No other AI providers are used.
- No Training on User Data: Food Ninja does not use any user's food records, queries, voice transcripts, or chat history to train or fine-tune AI models. User data is not shared with Google for training purposes beyond what is governed by Google's standard API terms.
- No Persistent AI Memory: the AI has no memory between separate chat sessions. Each session begins fresh. Within a session, conversation history is transmitted with each request to maintain context.
- AI Model Transparency: the app displays which AI model tier was used and whether grounding (real-time web search) was used for each response, via the model info included in the API response payload.
- Thinking (Extended Reasoning): when the Gemini Flash Thinking model is used, extended chain-of-thought reasoning is performed server-side before the final response is generated. The internal reasoning trace is not exposed to the user.

- Grounding (Web Search): when grounding is active, Gemini accesses real-time information from the public web. The app indicates when grounding was used and if it was limited by quota.
- Human Review: AI-generated recipes, meal plans, and donation safety assessments are provided as suggestions only. Users are advised to apply their own judgment, particularly for donation safety decisions.

13. Contact Information

For technical questions, security disclosures, or documentation inquiries regarding Food Ninja, please use the following contact details:

Developer	Albert Huang
Developer Location	Taiwan
General Support	support@food-ninja.com
Privacy Inquiries	privacy@food-ninja.com
Legal Matters	legal@food-ninja.com
Developer & Business	alberthuang@food-ninja.com
Security Disclosures	legal@food-ninja.com
Document Date	March 13, 2026
Document Author	Albert Huang
Document Version	1.0

For privacy rights requests, account deletion, or data access inquiries, please refer to the Food Ninja Privacy Policy. For legal matters including GDPR data subject requests, use the legal email address above.

14. Document Revision History

Version	Date	Author	Summary of Changes
1.0	March 13, 2026	Albert Huang	Initial publication. Full documentation of app architecture, all technologies, APIs, AI models, database schema, security practices, and data flow.

This document will be updated when significant architectural changes, new integrations, or major AI model changes are introduced. The documented date on the cover page reflects the date of the most recent revision. We are not responsible for outdated or inaccurate information in this documentation, as it is not maintained at all times.