NutriScanner

Developer Documentation

Technical Guide for Building and Extending
AI-Powered Nutritional Information Extraction

Technology Stack

Next.js 15 | React 19 | TypeScript | Tailwind CSS
Vercel AI SDK | OpenAI GPT-4o | Google Gemini | Zod
shadcn/ui | Framer Motion | Lucide Icons | Sonner Toasts

Version 1.0

October 26, 2025

Contents

1 Overview

1.1 Introduction

NutriScanner is a production-ready, AI-powered web application that extracts allergen information and nutritional values from food product PDF documents. Built with Next.js 15 and powered by advanced vision language models, it provides real-time streaming extraction with bilingual support for Hungarian and English.

i Information

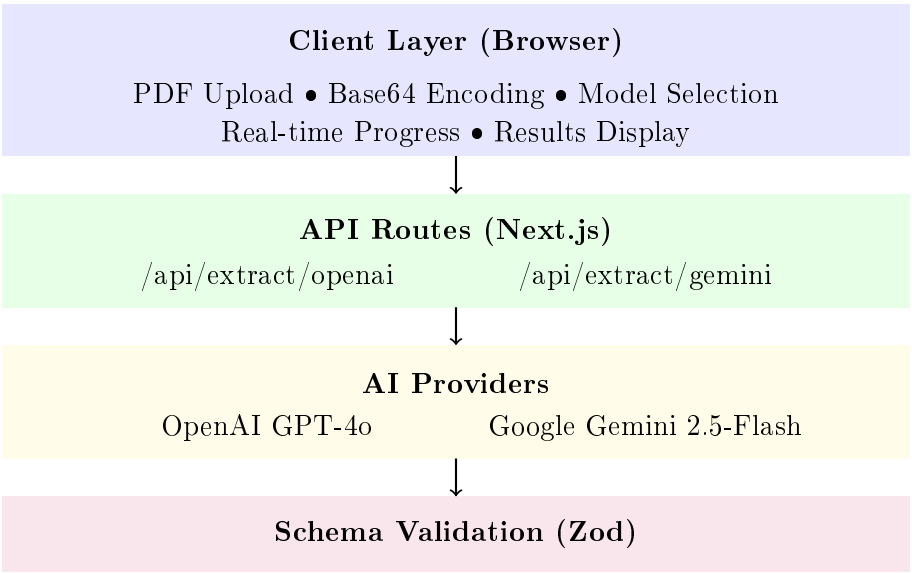
This documentation is intended for developers who want to understand, modify, extend, or deploy the NutriScanner application. For end-user documentation, refer to the User Manual.

1.2 Key Features

- **Dual AI Models**: OpenAI GPT-4o and Google Gemini 2.5-Flash
- **Real-time Streaming**: Live progress tracking (10-100%)
- **Bilingual Support**: Hungarian & English auto-detection
- **10 EU Allergens**: Smart detection with confidence scoring
- **Type-Safe**: Zod schema validation with TypeScript
- **Modern Stack**: Next.js 15, React 19, Tailwind CSS
- **Serverless**: API routes with 60s max duration
- **Vision AI**: Handles both PDFs and scanned images

1.3 Architecture Overview

NutriScanner follows a modern serverless architecture with clear separation of concerns:



2 Technology Stack

2.1 Frontend Technologies

Technology	Version	Purpose
Next.js	15.1.0	React framework with App Router
React	19.0.0	UI library with hooks
TypeScript	5.7.2	Type safety and developer experience
Tailwind CSS	3.4.16	Utility-first CSS framework
shadcn/ui	Latest	Component library (Radix UI)
Framer Motion	11.14.1	Animation library
Lucide React	0.468.0	Icon library
Sonner	1.7.1	Toast notifications

2.2 Backend & AI Technologies

Technology	Version	Purpose
Vercel AI SDK	5.0.0	AI model integration framework
@ai-sdk/openai	2.0.53	OpenAI GPT-4o integration
@ai-sdk/google	2.0.23	Google Gemini integration
@ai-sdk/react	2.0.79	React hooks for streaming
Zod	3.24.1	Runtime schema validation

2.3 Development Tools

- **ESLint**: Code linting and formatting
- **PostCSS**: CSS processing
- **npm**: Package management
- **Git**: Version control

3 Project Structure

3.1 Directory Layout

> Project Structure

nutriscan/ |– app/ | |– (preview)/ | | |– page.tsx Main application UI | | |– layout.tsx Root layout with metadata | | |– globals.css Global styles | | +– docs/ | | +– page.tsx Developer documentation | +– api/ | +– extract/ | |– openai/ | | +– route.ts OpenAI GPT-4o endpoint | +– gemini/ | +– route.ts Google Gemini endpoint |– components/ | +– ui/ | |– results-table.tsx Results display component | |– button.tsx Button component | |– card.tsx Card container | |– progress.tsx Progress bar | +– ... Other shadcn/ui components |– lib/ | |– schemas.ts Zod validation schemas | |– system-prompt.ts AI extraction instructions | +– utils.ts Utility functions |– public/ | |– nutriscanner.svg App logo | |– openai.svg OpenAI logo | +– google.svg Google logo +– Configuration files

3.2 Key Files Explained

`app/(preview)/page.tsx` Main application page with PDF upload UI, model selection, and results display

`app/api/extract/*/route.ts` API routes for OpenAI and Gemini extraction

`lib/schemas.ts` Zod schemas defining data structure and validation

`lib/system-prompt.ts` Comprehensive 190-line AI instruction set

`components/ui/results-table.tsx` Custom component for displaying extraction results

4 Developer Setup

4.1 Prerequisites

Before starting development, ensure you have:

- **Node.js 18+** — Required for Next.js 15
- **npm 10+** — Or yarn/pnpm equivalent
- **Git** — For version control
- **Code Editor** — VS Code recommended
- **OpenAI API Key** — From https://platform.openai.com/api-keys
- **Google Gemini API Key** — From https://aistudio.google.com/app/apikey

4.2 Installation Steps

> Clone and Install

1. Clone the repository git clone https://github.com/purelyricky/nutriscan.git cd nutriscan
2. Install dependencies npm install
3. Create environment file cp .env.example .env.local Edit .env.local with your API keys

4.3 Environment Configuration

Create a `.env.local` file in the root directory:

> Environment Variables

OpenAI Configuration (Required) $\text{OPENAI}_{A}PI_{K}EY = sk - proj - your_{o}penai_{k}ey_{h}ere$
Google Gemini Configuration (Required) $\text{GOOGLE}_{G}ENERATIVE_{A}I_{A}PI_{K}EY = AIzaSyD_{y}our_{g}oogle_{k}ey_{h}ere$

! Important

Never commit `.env.local` to version control. It's already in `.gitignore`.

4.4 Development Commands

> Common Commands

Start development server (with hot reload) npm run dev
Build for production npm run build
Start production server npm start
Run linting npm run lint

The development server runs at `http://localhost:3000`

5 Data Schemas

All schemas are defined in `lib/schemas.ts` using Zod for runtime validation and TypeScript type inference.

5.1 Allergen Schema

lib/schemas.ts — Allergen Schema

```
export const allergenSchema = z.object( gluten: z.boolean().describe("Contains gluten (Glutén)"), egg: z.boolean().describe("Contains egg (Tojás)"), crustaceans: z.boolean().describe("Contains crustaceans (Rák)"), fish: z.boolean().describe("Contains fish (Hal)"), peanut: z.boolean().describe("Contains peanut (Földimogyoró)"), soy: z.boolean().describe("Contains soy (Szója)"), milk: z.boolean().describe("Contains milk (Tej)"), treeNuts: z.boolean().describe("Contains tree nuts (Diófélék)"), celery: z.boolean().describe("Contains celery (Zeller)"), mustard: z.boolean().describe("Contains mustard (Mustár)"), );
export type Allergen = z.infer<typeof allergenSchema>;
```

! Best Practice

All 10 allergen fields are **required** and must be boolean values. Descriptions include Hungarian translations to help the AI model understand bilingual documents.

5.2 Nutritional Value Schema

lib/schemas.ts — Nutritional Value Schema

```
export const nutritionalValueSchema = z.object( energy: z.string().optional() .describe("Energy value (Energia) - include unit (kJ/kcal)"), fat: z.string().optional() .describe("Fat content (Zsír) - include unit (g)"), carbohydrate: z.string().optional() .describe("Carbohydrate content (Szénhidrát) - include unit (g)"), sugar: z.string().optional() .describe("Sugar content (Cukor) - include unit (g)"), protein: z.string().optional() .describe("Protein content (Fehérje) - include unit (g)"), sodium: z.string().optional() .describe("Sodium content (Nátrium) - include unit (mg/g)"), );
export type NutritionalValue = z.infer<typeof nutritionalValueSchema>;
```

i Information

All nutritional fields are **optional** and include units as strings. Only fields found in the document are included in the response.

5.3 Extraction Result Schema

lib/schemas.ts — Main Extraction Schema

```
export const extractionResultSchema = z.object( allergens: allergenSchema .describe("Allergen information extracted from the document"), nutritionalValues: nutritionalValueSchema .describe("Nutritional values extracted from the document"), detectedLanguage: z.enum(["hungarian", "english", "both", "unknown"]) .describe("The language de-
```

```
tected in the document"), productName: z.string().optional() .describe("Product name extracted from content or filename"), confidence: z.enum(["high", "medium", "low"]) .describe("REQUIRED: Confidence level based on document quality"), );
export type ExtractionResult = z.infer<typeof extractionResultSchema>;
```

6 API Routes

NutriScanner implements two parallel API routes, one for each AI model provider.

6.1 OpenAI Route

Endpoint: `POST /api/extract/openai`
Model: GPT-4o (OpenAI's vision-language model)
Max Duration: 60 seconds

app/api/extract/openai/route.ts

```
import openai from "@ai-sdk/openai"; import streamObject from "ai"; import extractionResultSchema from "@/lib/schemas"; import EXTRACTION_SYSTEM_PROMPT from "@/lib/system-prompt";
export const maxDuration = 60;
export async function POST(req: Request) try const files = await req.json();
if (!files || files.length === 0) throw new Error("No files provided");
// Extract base64 data and convert to Buffer let firstFile = files[0].data; if (!firstFile.startsWith('data:')) firstFile = `data:application/pdf;base64,${firstFile}`;
const base64Data = firstFile.split(',')[1]; const fileBuffer = Buffer.from(base64Data, 'base64');
// Stream extraction with OpenAI const result = streamObject( model: openai("gpt-4o"), messages: [ role: "system", content: EXTRACTION_SYSTEM_PROMPT, , role : "user", content : [type : "text", text : `Extractallergenandnutr
extractionResultSchema, );
return result.toTextStreamResponse(); catch (error) return new Response(JSON.stringify( error: String(error) ), status: 500, headers: "Content-Type": "application/json" , );
```

6.2 Gemini Route

Endpoint: `POST /api/extract/gemini`
Model: Gemini 2.5-Flash (Google's vision model)
Max Duration: 60 seconds

app/api/extract/gemini/route.ts

```
import google from "@ai-sdk/google"; import streamObject from "ai"; import extractionResultSchema from "@/lib/schemas"; import EXTRACTION_SYSTEM_PROMPT from "@/lib/system-prompt";
export const maxDuration = 60;
export async function POST(req: Request) try const files = await req.json();
if (!files || files.length === 0) throw new Error("No files provided");
const base64Data = files[0].data.split(',')[1]; const fileBuffer = Buffer.from(base64Data, 'base64');
// Stream extraction with Gemini const result = streamObject( model: google("gemini-2.5-flash"), // Only difference! messages: [ role: "system", content: EXTRACTION_SYSTEM_PROMPT, , role : "user", content : [type : "text", text : `Extractallergenandnutr
extractionResultSchema, );
return result.toTextStreamResponse(); catch (error) return new Response(JSON.stringify( error: String(error) ), status: 500, headers: "Content-Type": "application/json" , );
```

! Best Practice

Both routes follow identical patterns. The only difference is the model provider: `openai("gpt-4o")` vs `google("gemini-2.5-flash")`. This consistency makes the codebase easy to maintain and extend.

6.3 Request Format

Request Body

"files": ["name": "product-label.pdf", "type": "application/pdf", "data": "data:application/pdf;base64,JVBERi0xLjQK..."]

6.4 Response Format

Streaming JSON response with progressive extraction:

Response (Streaming)

"allergens": "gluten": true, "egg": false, // ... other allergens , "nutritionalValues": "energy": "1500kJ/350kcal", "fat": "12g", // ... other nutrients , "detectedLanguage": "hungarian", "productName": "Sample Product", "confidence": "high"

7 Frontend Implementation

7.1 Main Page Component

The main application page (`app/(preview)/page.tsx`) handles the entire user flow.

Main Page Setup

"use client";
import useState from "react"; import experimental$_{u}$seObjectasuseObject$from$"@$ai-sdk/react$";$importextractionResultSchema, ExtractionResult$from$"@/lib/schemas";$importtoast$from$"$s$
export default function NutriScanner() const [files, setFiles] = useState<File[]>([]); const [selectedModel, setSelectedModel] = useState<"openai" | "gemini">("openai"); const [extractionResult, setExtractionResult] = useState<ExtractionResult | null>(null);
// Create hooks for each model const openaiHook = useObject(api: "/api/extract/openai", schema: extractionResultSchema, onFinish: (object, error) => if (error) toast.error("Extraction failed"); return; setExtractionResult(object); toast.success("Extraction completed!"); ,);
const geminiHook = useObject(api: "/api/extract/gemini", schema: extractionResultSchema, onFinish: (object, error) => if (error) toast.error("Extraction failed"); return; setExtractionResult(object); toast.success("Extraction completed!"); ,);
// Select active hook based on model const currentHook = selectedModel === "openai" ? openaiHook : geminiHook; const submit, object: partialResult, isLoading = currentHook;
// ... render UI

7.2 File Upload Handler

File Upload and Encoding

const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => const selectedFiles = Array.from(e.target.files || []); const validFiles = selectedFiles.filter((file) => file.type === "application/pdf" file.size <= 5 * 1024 * 1024);
if (validFiles.length !== selectedFiles.length) toast.error("Only PDF files under 5MB are allowed.");
setFiles(validFiles); ;
const encodeFileAsBase64 = (file: File): Promise<string> => return new Promise((resolve, reject) => const reader = new FileReader(); reader.readAsDataURL(file); reader.onload = () => resolve(reader.result as string); reader.onerror = (error) => reject(error);); ;
const handleSubmitWithFiles = async (e: React.FormEvent) => e.preventDefault(); const encodedFiles = await Promise.all(files.map(async (file) => (name: file.name, type: file.type, data: await encodeFileAsBase64(file),))); submit(files: encodedFiles); ;

7.3 Real-time Progress Calculation

Progress Tracking

const calculateProgress = () => if (!isLoading) return 0; if (!partialResult) return 10; // Initial processing
let completed = 0; const total = 3; // allergens, nutritionalValues, detectedLanguage
if (partialResult.allergens Object.keys(partialResult.allergens).length > 0) completed += 1; if (partialResult.nutritionalValues Object.keys(partialResult.nutritionalValues).length >

```
0)  completed += 1;  if (partialResult.detectedLanguage)  completed += 1;
// Map to percentage (10return 10 + Math.round((completed / total) * 90); ;
const progress = calculateProgress();
```

! Best Practice

The progress calculation provides real-time feedback to users by tracking which fields have been populated in the streaming response. This creates a much better UX than a simple loading spinner.

8 System Prompt Engineering

The `EXTRACTION_SYSTEM_PROMPT` in `lib/system-prompt.ts` is a comprehensive 190-line instruction set that guides the AI model.

8.1 Prompt Structure

1. **Task Definition** — Defines the AI's role as a food product data extraction specialist
2. **Critical Requirements** — Lists mandatory requirements (all allergen fields, confidence level)
3. **Allergen Detection Rules** — Explains when to mark allergens as true vs false
4. **Nutritional Extraction** — Where to find nutrition facts and how to extract them
5. **Product Name Extraction** — How to identify product names
6. **Confidence Assessment** — Criteria for high/medium/low confidence
7. **Language Detection** — How to detect Hungarian/English/Both/Unknown
8. **Edge Cases** — Handling blank PDFs, non-food documents, poor quality scans
9. **Example Outputs** — 4 detailed examples with expected JSON

8.2 Key Prompt Sections

Allergen Detection Rules (Excerpt)

/** * ALLERGEN DETECTION RULES * * Mark as TRUE only if: * - Document explicitly states "contains" / "tartalmaz" * - "van jelen" (is present) * - Listed in allergen section without qualifiers * - Appears in ingredients with emphasis (bold, underlined) * * Mark as FALSE if: * - Not mentioned * - "may contain traces" / "nyomokban tartalmazhat" * - "produced in facility that processes" * - Any conditional/uncertain phrasing */

8.3 Bilingual Terminology

The prompt includes both Hungarian and English terms for all allergens and nutrients:

English	Hungarian	Variations
Gluten	Glutén	búzaliszt, rozsliszt
Egg	Tojás	tojásfehérje, tojássárgája
Milk	Tej	tejpor, tejfehérje, whey, savó
Energy	Energia	kJ, kcal
Nutrition Facts	Tápértékadatok	Átlagos tápérték

9 AI Integration

9.1 Vercel AI SDK

NutriScanner uses the Vercel AI SDK for seamless integration with multiple AI providers.

Streaming with streamObject

```
import streamObject from "ai"; import openai from "@ai-sdk/openai"; import google from "@ai-sdk/google";
const result = streamObject( model: openai("gpt-4o"), // or google("gemini-2.5-flash") messages: [ role: "system", content: EXTRACTION_SYSTEM_PROMPT, role : "user", content : [type : "text", text : "Extractinformation...", t
extractionResultSchema, onFinish : (object, error) => if(error)console.error("Extractionerror : ", error); return;//Validateandusetheextracteddataconstvalida
return result.toTextStreamResponse();
```

9.2 Client-Side Streaming

React Hook for Streaming

```
import experimental_useObjectasuseObjectfrom"@ai − sdk/react";
const submit, object: partialResult, isLoading = useObject( api: "/api/extract/openai", schema: extractionResultSchema, initialValue: undefined, onError: (error) => console.error("Extraction error:", error); toast.error("Failed to extract information"); , onFinish: ( object, error ) => if (error) toast.error("Failed to process the PDF"); return; if (object) setExtractionResult(object); toast.success("Extraction completed successfully!"); , );
```

9.3 Model Selection Strategy

Scenario	Recommended Model	Reason
Critical data	OpenAI GPT-4o	Highest accuracy
Batch processing	Google Gemini	Faster, cost-effective
Poor quality scans	OpenAI GPT-4o	Better OCR
Clear digital PDFs	Google Gemini	Equally accurate
Complex layouts	OpenAI GPT-4o	Better understanding

10 Deployment

10.1 Vercel Deployment (Recommended)

1. Push your code to GitHub
2. Import project at https://vercel.com/new
3. Configure environment variables:
   - `OPENAI_API_KEY`
   - `GOOGLE_GENERATIVE_AI_API_KEY`
4. Deploy

> Vercel CLI Deployment

Install Vercel CLI npm i -g vercel
Login to Vercel vercel login
Deploy vercel
Deploy to production vercel –prod

10.2 Environment Variables

! Important

Production Environment Variables
Ensure these are set in your deployment platform:

- `OPENAI_API_KEY` — Your production OpenAI API key
- `GOOGLE_GENERATIVE_AI_API_KEY` — Your production Google Gemini API key

Never commit API keys to version control. Use platform-specific secret management.

10.3 Alternative Platforms

NutriScanner can be deployed on any Next.js-compatible platform:

- **Netlify** — Configure build: `npm run build`
- **AWS Amplify** — Connect GitHub and set environment variables
- **DigitalOcean App Platform** — Deploy from GitHub with Node.js 18+
- **Self-hosted** — Run `npm run build && npm start`

10.4 Build Configuration

next.config.mjs

/** @type import('next').NextConfig */ const nextConfig = typescript: ignoreBuildErrors: true, // For development flexibility , eslint: ignoreDuringBuilds: true, // For development flexibility , ;
export default nextConfig;

! Important

In production, consider enabling strict TypeScript and ESLint checks by setting these to `false`.

11 Extending the Application

11.1 Adding a New AI Model

To add support for another AI provider:

1. **Create new API route**

```
> Create Route
mkdir -p app/api/extract/newmodel touch app/api/extract/newmodel/route.ts
```

2. **Implement the route** (follow OpenAI/Gemini pattern)

3. **Add provider to frontend**

```
# Update Frontend
const [selectedModel, setSelectedModel] = useState<"openai" | "gemini" | "new-
model">("openai");
const newmodelHook = useObject( api: "/api/extract/newmodel", schema: extraction-
ResultSchema, // ... same pattern );
const currentHook = selectedModel === "openai" ? openaiHook : selectedModel ===
"gemini" ? geminiHook : newmodelHook;
```

4. **Add UI button** for model selection

5. **Test thoroughly** with various PDF types

11.2 Adding New Allergens

To add additional allergens beyond the EU-regulated 10:

1. Update `lib/schemas.ts`

```
# Add Allergen Field
export const allergenSchema = z.object( // ... existing allergens sesame:
z.boolean().describe("Contains sesame (Szezám)"), lupin: z.boolean().describe("Contains
lupin (Csillagfürt)"), );
```

2. Update `lib/system-prompt.ts` with detection rules

3. Update `components/ui/results-table.tsx` to display new allergens

4. Update bilingual labels

11.3 Adding New Nutritional Fields

Add Nutritional Field

```
export const nutritionalValueSchema = z.object( // ... existing fields fiber: z.string().optional() .describe("Fiber content (Rost) - include unit (g)"), saturatedFat: z.string().optional() .describe("Saturated fat (Telített zsír) - include unit (g)"), );
```

11.4 Customizing the UI

All UI components use Tailwind CSS. To customize:

1. Edit `tailwind.config.ts` for color scheme
2. Modify component styles in `components/ui/`
3. Update `app/(preview)/globals.css` for global styles

12 Troubleshooting

12.1 Common Development Issues

12.1.1 TypeScript Errors

Build fails with type errors

Solutions:

- Ensure all imports match exact file paths (case-sensitive)
- Run `npm install` to ensure all types are installed
- Check `tsconfig.json` for correct configuration
- Verify `@ai-sdk/react` uses `experimental_useObject`

12.1.2 Streaming Not Working

No real-time progress updates

Checklist:

1. Verify API route returns `result.toTextStreamResponse()`
2. Check schema is passed to `streamObject()`
3. Ensure using `experimental_useObject` hook
4. Confirm `maxDuration` is set in API route

12.1.3 Schema Validation Failing

AI returns invalid data

Debug steps:

1. Log the raw response: `console.log(object)`
2. Use `schema.safeParse(object)` to see specific errors
3. Review system prompt for clarity
4. Ensure all required fields are marked in prompt
5. Check confidence field is always included

12.2 API Issues

Issue	Solution
401 Unauthorized	Check API keys in .env.local
429 Rate Limit	Implement retry logic or upgrade plan
504 Timeout	Increase maxDuration or optimize PDF size
Invalid JSON	Check AI response format in logs

12.3 Performance Optimization

- **PDF Size**: Compress large PDFs before upload
- **Caching**: Consider caching results for identical PDFs
- **Model Selection**: Use Gemini for faster processing when accuracy allows
- **Error Handling**: Implement exponential backoff for retries

13 Testing

13.1 Manual Testing Checklist

1. **Upload Validation**
   - Try uploading non-PDF files (should fail)
   - Try files over 5MB (should fail)
   - Upload valid PDFs (should succeed)
2. **Model Selection**
   - Switch between OpenAI and Gemini
   - Verify correct endpoint is called
   - Compare results between models
3. **Extraction Quality**
   - Test with Hungarian labels
   - Test with English labels
   - Test with bilingual labels
   - Test with poor quality scans
   - Test with blank PDFs
4. **Results Display**
   - Verify all allergens display correctly
   - Check nutritional values with units
   - Confirm confidence level is shown
   - Test "Scan Another" functionality

13.2 Sample Test PDFs

Create test cases with:

- Complete nutrition label (all fields)
- Allergens only (no nutrition data)
- Nutrition only (no allergen list)
- Bilingual label
- Poor quality scan
- Handwritten label

14 Best Practices

14.1 Code Quality

! Best Practice

Follow these guidelines:

- Use TypeScript strict mode
- Validate all inputs with Zod
- Handle errors gracefully with user-friendly messages
- Log errors for debugging but sanitize sensitive data
- Use semantic HTML and ARIA labels for accessibility
- Keep components small and focused

14.2 Security

Security Checklist

1. Never commit `.env.local` or API keys
2. Validate file types and sizes on both client and server
3. Sanitize all user inputs
4. Use HTTPS in production
5. Implement rate limiting for API routes
6. Keep dependencies updated

14.3 Performance

- Use React's `useMemo` and `useCallback` for expensive operations
- Lazy load components when possible
- Optimize images (logos should be SVG or WebP)
- Use Next.js Image component for automatic optimization
- Implement proper loading states

14.4 User Experience

- Always provide feedback (loading, success, error)
- Use progress indicators for long operations
- Make error messages actionable
- Support keyboard navigation

- Test on mobile devices
- Ensure good color contrast

15 API Reference

15.1 POST /api/extract/openai

Description: Extract nutritional data using OpenAI GPT-4o

Request Body:

```
# Request

"files": [ "name": string, // Original filename "type": "application/pdf", // MIME type
"data": string // Base64-encoded PDF with data URL prefix ]
```

Response: Streaming text/event-stream with JSON chunks

Status Codes:

- 200 — Success (streaming)
- 500 — Server error

15.2 POST /api/extract/gemini

Identical to `/api/extract/openai` but uses Google Gemini 2.5-Flash model.

16 Appendix

16.1 Dependencies Reference

Package	Version	Purpose
next	15.1.0	React framework
react	19.0.0	UI library
typescript	5.7.2	Type safety
ai	5.0.0	Vercel AI SDK
@ai-sdk/openai	2.0.53	OpenAI integration
@ai-sdk/google	2.0.23	Gemini integration
@ai-sdk/react	2.0.79	React hooks
zod	3.24.1	Schema validation
tailwindcss	3.4.16	CSS framework
framer-motion	11.14.1	Animations
lucide-react	0.468.0	Icons
sonner	1.7.1	Toasts

16.2 Useful Resources

- **Next.js Docs**: https://nextjs.org/docs
- **Vercel AI SDK**: https://sdk.vercel.ai/docs
- **OpenAI API**: https://platform.openai.com/docs
- **Google Gemini**: https://ai.google.dev/gemini-api/docs
- **Zod Documentation**: https://zod.dev
- **Tailwind CSS**: https://tailwindcss.com/docs
- **shadcn/ui**: https://ui.shadcn.com

16.3 Contributing

To contribute to NutriScanner:

1. Fork the repository
2. Create a feature branch: `git checkout -b feature/amazing-feature`
3. Commit changes: `git commit -m 'Add amazing feature'`
4. Push to branch: `git push origin feature/amazing-feature`
5. Open a Pull Request

16.4 License

MIT License — See LICENSE file for details

NutriScanner Developer Documentation

Next.js 15 • React 19 • TypeScript • Vercel AI SDK

OpenAI GPT-4o • Google Gemini 2.5-Flash

Version 1.0 — October 26, 2025

https://github.com/purelyricky/nutriscan