MedNAIS SOP Marketplace - Comprehensive Analysis Report

Date: November 22, 2025
Analyzed by: DeepAgent
Purpose: Rebuild with Next.js, PostgreSQL, and TypeScript

Table of Contents

Executive Summary

MedNAIS SOP Marketplace is a comprehensive platform for creating, sharing, executing, and monetizing Standard Operating Procedures (SOPs). The application is built with **Next.js 14** (App Router), **TypeScript**, **PostgreSQL** (via Prisma ORM), and includes a **Python FastAPI** backend for Stripe integration.

Key Statistics:
- **Total Files:** 158+ files across 65 directories
- **UI Components:** 78 React components
- **API Routes:** 32 Next.js API endpoints
- **Database Models:** 15 Prisma models
- **Supported Languages:** 10 (English, Spanish, French, German, Chinese, Arabic, Portuguese, Polish, Russian, Ukrainian)

Core Business Model:
- Free SOP creation for personal and group use
- Marketplace for buying/selling SOPs

- 70/30 revenue split (Creator/Platform)
- Stripe-powered payment processing

Project Overview

Vision

A marketplace platform that enables users to create, share, execute, and monetize Standard Operating Procedures with built-in execution tracking, collaboration features, and payment processing.

Target Users

1. **SOP Creators:** Professionals who create and sell procedures
2. **SOP Buyers:** Individuals/teams who purchase proven procedures
3. **Teams:** Groups collaborating on internal SOPs
4. **Individual Users:** Personal SOP management

Business Value

- **Creator Monetization:** 70% revenue share on marketplace sales
- **Execution Analytics:** Track performance and completion rates
- **Collaboration:** Private groups for team SOP management
- **Multimedia Support:** Images, YouTube embeds, and timers

Project Structure

```
mednais-sop-marketplace-main/
├── app/                          # Next.js App Router (Pages & API Routes)
│   ├── api/                      # API endpoints (32 routes)
│   │   ├── assets/               # Asset management
│   │   ├── auth/                 # Authentication endpoints (6 routes)
│   │   ├── cart/                 # Shopping cart (2 routes)
│   │   ├── categories/           # Category management (2 routes)
│   │   ├── groups/               # Group management (3 routes)
│   │   ├── marketplace/          # Marketplace listings
│   │   ├── profile/              # User profile (2 routes)
│   │   ├── promo-codes/          # Promo code validation
│   │   ├── ratings/              # SOP ratings (2 routes)
│   │   ├── sessions/             # Execution sessions (2 routes)
│   │   ├── sop-executions/       # Execution tracking (2 routes)
│   │   ├── sops/                 # SOP CRUD (3 routes)
│   │   ├── stripe/               # Payment processing (2 routes)
│   │   ├── upload/               # File uploads
│   │   └── users/                # User management
│   ├── auth/                     # Auth pages (login, magic link)
│   ├── cart/                     # Shopping cart page
│   ├── dashboard/                # User dashboard
│   ├── groups/                   # Group management pages (5 pages)
│   ├── marketplace/              # Marketplace browse page
│   ├── profile/                  # User profile page
│   ├── purchase-success/         # Payment success page
│   ├── sessions/                 # Execution session pages (2 pages)
│   ├── sops/                     # SOP pages (6 pages)
│   ├── globals.css               # Global styles & CSS variables
│   ├── layout.tsx                # Root layout
│   └── page.tsx                  # Home page
│
├── backend/                      # Python FastAPI backend
│   ├── server.py                 # FastAPI server (proxies to Next.js)
│   ├── stripe_routes.py          # Stripe integration (Emergent)
│   └── requirements.txt          # Python dependencies
│
├── components/                   # React components (78 components)
│   ├── layout/                   # Layout components (5 components)
│   ├── sop-editor/               # SOP editor components (4 components)
│   ├── ui/                       # Reusable UI components (50+ components)
│   ├── add-to-cart-button.tsx
│   ├── copy-button.tsx
│   ├── custom-icon.tsx
│   ├── language-switcher.tsx
│   ├── mednais-logo.tsx
│   ├── navigation.tsx
│   ├── page-header.tsx
│   ├── providers.tsx
│   ├── purchase-sop-button.tsx
│   ├── rating-form.tsx
│   ├── rating-stars.tsx
│   ├── ratings-list.tsx
│   ├── sop-card.tsx
│   ├── sop-purchase-section.tsx
│   ├── sop-rating-badge.tsx
│   ├── sop-ratings-section.tsx
│   ├── theme-provider.tsx
│   └── theme-toggle.tsx
│
├── lib/                          # Utility libraries
│   ├── auth/                     # Authentication utilities (7 files)
│   │   ├── AuthContext.tsx       # React Context for auth state
│   │   ├── cookies.ts            # Cookie management
│   │   ├── email.ts              # Magic link email sending
│   │   ├── jwt.ts                # JWT generation/verification
│   │   ├── oauth.ts              # Google/Apple OAuth
│   │   ├── server.ts             # Server-side auth helpers
│   │   └── validation.ts         # Auth validation schemas
│   ├── cart/                     # Cart utilities
│   ├── i18n/                     # Internationalization (4 files)
│   │   ├── LanguageContext.tsx
│   │   ├── locales.ts            # Supported locales
│   │   ├── translations.ts       # Translation strings
│   │   └── translations-old.ts
│   ├── aws-s3-config.ts          # AWS S3 configuration
│   ├── db.ts                     # Prisma client
│   ├── firebase.ts               # Firebase config (client)
│   ├── firebase-admin.ts         # Firebase Admin config
│   ├── s3.ts                     # S3 upload utilities
│   ├── timer.ts                  # Timer utilities
│   ├── types.ts                  # TypeScript types
│   ├── utils.ts                  # General utilities
│   └── youtube.ts                # YouTube embed helpers
│
├── prisma/                       # Database schema & migrations
│   ├── migrations/               # 10 migration files
│   └── schema.prisma             # Database schema (15 models)
│
├── public/                       # Static assets
│   ├── uploads/avatars/          # User avatars
│   ├── favicon.svg
│   ├── mednais-logo.jpg
│   ├── og-image.png
│   └── robots.txt
│
├── scripts/                      # Database seeding scripts
│   ├── process_document.py       # Document processing
│   ├── seed.ts                   # TypeScript seed script
│   ├── seed_categories.py        # Python category seeding
│   ├── seed_categories.ts        # TypeScript category seeding
│   └── requirements.txt
│
├── types/                        # TypeScript type definitions
│   └── next-auth.d.ts
│
├── .emergent/                    # Emergent AI integration config
├── .qdrant/                      # Qdrant vector DB (if used)
├── next.config.js                # Next.js configuration
├── tailwind.config.ts            # Tailwind CSS configuration
├── tsconfig.json                 # TypeScript configuration
├── package.json                  # Node dependencies
├── yarn.lock                     # Yarn lock file
├── postcss.config.js             # PostCSS configuration
├── README.md                     # Project documentation
├── DEPLOYMENT_NOTES.md           # Deployment guide
├── TRANSLATION_GUIDE.md          # Translation guide
└── start_postgres.sh             # PostgreSQL startup script
```

Technology Stack

Frontend Stack

Core Framework

- **Next.js 14.2.28** - React framework with App Router
- **React 18.2.0** - UI library
- **TypeScript 5.2.2** - Type-safe JavaScript

Styling & UI

- **Tailwind CSS 3.3.3** - Utility-first CSS framework
- **Radix UI** - Accessible, unstyled component primitives (20+ components)
- Accordion, Alert Dialog, Avatar, Checkbox, Dialog, Dropdown Menu, etc.
- **tailwindcss-animate 1.0.7** - Animation utilities
- **Framer Motion 10.18.0** - Advanced animations
- **Lucide React 0.446.0** - Icon library (500+ icons)
- **class-variance-authority 0.7.0** - Component variants
- **clsx 2.1.1** - Conditional class names
- **tailwind-merge 2.5.2** - Merge Tailwind classes

State Management

- **Zustand 5.0.3** - Lightweight state management
- **Jotai 2.6.0** - Atomic state management
- **@tanstack/react-query 5.0.0** - Server state management
- **SWR 2.2.4** - Data fetching & caching

Forms & Validation

- **React Hook Form 7.53.0** - Form management
- **Formik 2.4.5** - Alternative form library
- **Zod 4.1.12** - Schema validation
- **Yup 1.3.0** - Alternative validation
- **@hookform/resolvers 3.9.0** - Form validation resolvers

Data Visualization

- **Recharts 2.15.3** - Chart library
- **Chart.js 4.4.9** - Alternative charting
- **React-Chartjs-2 5.3.0** - React wrapper for Chart.js
- **Plotly.js 2.35.3** - Advanced plotting
- **React-Plotly.js 2.6.0** - React wrapper for Plotly

Media & File Handling

- **React Dropzone 14.3.8** - File upload
- **React YouTube 10.1.0** - YouTube embeds
- **Embla Carousel React 8.3.0** - Carousel component
- **React Resizable Panels 2.1.3** - Resizable layouts

UI Utilities

- **React Hot Toast 2.4.1** - Toast notifications
- **Sonner 1.5.0** - Alternative toast library

- **React Intersection Observer 9.8.0** - Viewport detection
- **React Day Picker 8.10.1** - Date picker
- **React DatePicker 6.1.0** - Alternative date picker
- **React Select 5.8.0** - Select component
- **Input OTP 1.2.4** - OTP input component
- **CMDK 1.0.0** - Command menu
- **Vaul 0.9.9** - Drawer component

Utilities

- **date-fns 3.0.0** - Date manipulation
- **dayjs 1.11.13** - Alternative date library
- **lodash 4.17.21** - Utility functions
- **uuid 13.0.0** - UUID generation
- **gray-matter 4.0.3** - Front matter parsing
- **CSV 6.3.11** - CSV parsing
- **cookie 1.0.2** - Cookie handling

Backend Stack

API Layer

- **Next.js API Routes** - Primary API (32 endpoints)
- **Python FastAPI 0.115.0** - Stripe webhook handler
- **Uvicorn 0.34.0** - ASGI server for FastAPI
- **HTTPX 0.28.1+** - Async HTTP client

Database

- **PostgreSQL** - Primary database
- **Prisma 6.7.0** - ORM for TypeScript
- **@prisma/client 6.7.0** - Prisma client for Node.js
- **prisma-client-py 0.15.0** - Prisma client for Python

Authentication

- **jose 6.1.2** - JWT verification (JOSE standard)
- **jsonwebtoken 9.0.2** - JWT generation/verification
- **bcryptjs 3.0.3** - Password hashing
- **Firebase 12.6.0** - Firebase client SDK
- **Firebase Admin 13.6.0** - Firebase Admin SDK

Payment Processing

- **Stripe 19.2.0** - Backend Stripe SDK
- **@stripe/stripe-js 8.2.0** - Frontend Stripe SDK
- **emergentintegrations 0.1.0** - Stripe integration helper

Cloud Storage

- **@aws-sdk/client-s3 3.0.0** - AWS S3 client
- **@aws-sdk/s3-request-presigner 3.0.0** - S3 pre-signed URLs

Email

- **nodemailer 7.0.10** - Email sending (magic links)

Security & Encryption

- **uncrypto 0.1.3** - Cryptographic utilities

Development Tools

Build & Bundling

- **Webpack 5.99.5** - Module bundler
- **PostCSS 8.4.30** - CSS processing
- **Autoprefixer 10.4.15** - CSS vendor prefixes
- **@next/swc-wasm-nodejs 13.5.1** - SWC compiler

Code Quality

- **ESLint 9.24.0** - JavaScript linter
- **eslint-config-next 15.3.0** - Next.js ESLint config
- **@typescript-eslint/eslint-plugin 7.0.0** - TypeScript ESLint
- **@typescript-eslint/parser 7.0.0** - TypeScript parser
- **eslint-plugin-prettier 5.1.3** - Prettier ESLint integration
- **eslint-plugin-react-hooks 4.6.0** - React Hooks linting

Type Definitions

- **@types/node 20.6.2**
- **@types/react 18.2.22**
- **@types/react-dom 18.2.7**
- **@types/bcryptjs 3.0.0**
- **@types/nodemailer 7.0.4**
- **@types/uuid 11.0.0**
- **@types/jsonwebtoken 9.0.10**
- **@types/plotly.js 2.35.5**
- **@types/react-plotly.js 2.6.3**

Development Servers

- **ts-node 10.9.2** - TypeScript execution
- **tsx 4.20.3** - TypeScript execution (faster alternative)
- **dotenv 16.5.0** - Environment variables

External Services

Required Services

1. **PostgreSQL** - Database
2. **Stripe** - Payment processing
3. **AWS S3** - File storage
4. **Firebase** - Optional (auth provider)
5. **Google OAuth** - Social authentication
6. **Apple OAuth** - Social authentication

Optional Services

1. **Qdrant** - Vector database (AI features)
2. **Emergent AI** - AI integrations

Browser Support

- **IE 11+**
- **> 0.5% market share**
- **Last 2 versions**
- **Not dead browsers**

Database Schema & Models

The application uses **PostgreSQL** as the primary database with **Prisma ORM** for type-safe database access.

Schema Overview

Total Models: 15
Total Migrations: 10
Database Provider: PostgreSQL

Entity Relationship Diagram (Text)

```
User
├── AuthProvider (1:N) - Multiple auth methods per user
├── RefreshToken (1:N) - Multiple sessions per user
├── SOP (1:N - creator) - User creates SOPs
├── Group (1:N - owner) - User owns groups
├── GroupMembership (1:N) - User joins groups
├── Purchase (1:N - buyer) - User purchases SOPs
├── Purchase (1:N - seller) - User sells SOPs
├── SOPExecution (1:N) - User executes SOPs
├── Rating (1:N) - User rates SOPs
└── CategorySuggestion (1:N) - User suggests categories

SOP
├── SOPStep (1:N) - SOP has steps
├── Purchase (1:N) - SOP can be purchased
├── SOPExecution (1:N) - SOP can be executed
├── Rating (1:N) - SOP can be rated
├── CartItem (1:N) - SOP can be in carts
├── User (N:1 - creator) - SOP belongs to creator
├── Group (N:1) - SOP belongs to group (optional)
└── Category (N:1) - SOP belongs to category (optional)

Group
├── GroupMembership (1:N) - Group has members
├── SOP (1:N) - Group has SOPs
└── User (N:1 - owner) - Group has owner

Category
├── SOP (1:N) - Category has SOPs
├── CategorySuggestion (1:N) - Suggestions for category
├── Category (1:N - subcategories) - Self-referencing hierarchy
└── Category (N:1 - parent) - Parent category
```

Detailed Model Definitions

1. User Model

Purpose: Core user entity for authentication and profile

```
model User {
  id            String    @id @default(uuid())
  email         String?   @unique
  name          String?
  avatar_url    String?
  bio           String?
  location      String?
  website       String?
  twitter       String?
  linkedin      String?
  github        String?
  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt

  // Relations
  authProviders       AuthProvider[]
  refreshTokens       RefreshToken[]
  sops                SOP[]                 @relation("UserSOPs")
  ownedGroups         Group[]               @relation("GroupOwner")
  memberOf            Group[]               @relation("GroupMembers")
  groupMemberships    GroupMembership[]
  purchasedSOPs       Purchase[]            @relation("SOPPurchaser")
  soldSOPs            Purchase[]            @relation("SOPSeller")
  executions          SOPExecution[]
  ratings             Rating[]
  categorySuggestions  CategorySuggestion[]
}
```

Key Features:
- UUID as primary key
- Optional email (social auth may not provide)
- Rich profile fields (bio, location, social links)
- Multiple authentication providers
- Tracks both purchases and sales

2. AuthProvider Model

Purpose: Multi-provider authentication support

```
model AuthProvider {
  id              String   @id @default(uuid())
  userId          String
  provider        String   // 'email' | 'google' | 'apple'
  providerUserId  String
  createdAt       DateTime @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerUserId])
}
```

Supported Providers:
- Email (magic link)

- Google OAuth
- Apple Sign In

Key Features:
- One user can have multiple auth providers
- Prevents duplicate provider accounts
- Cascade delete when user is deleted

3. MagicLink Model

Purpose: Passwordless email authentication

```
model MagicLink {
  id        String    @id @default(uuid())
  email     String
  token     String    @unique
  expiresAt DateTime
  usedAt    DateTime?
  createdAt DateTime  @default(now())

  @@index([token])
  @@index([email])
}
```

Key Features:
- Time-limited tokens
- One-time use tracking
- Indexed for fast lookup

4. RefreshToken Model

Purpose: Long-lived authentication sessions

```
model RefreshToken {
  id        String    @id @default(uuid())
  userId    String
  token     String    @unique
  userAgent String?
  ip        String?
  revokedAt DateTime?
  createdAt DateTime  @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId])
  @@index([token])
}
```

Key Features:
- 7-day lifespan
- Device tracking (user agent, IP)
- Revocation support
- Multiple sessions per user

5. SOP Model

Purpose: Core SOP entity

```
model SOP {
  id          String   @id @default(cuid())
  title       String
  description String
  type        SOPType  @default(PERSONAL)
  price       Float?   // Price in CENTS
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  // Relations
  creatorId String
  creator   User   @relation("UserSOPs", fields: [creatorId], references: [id], onDele
te: Cascade)

  groupId   String?
  group     Group?  @relation(fields: [groupId], references: [id], onDelete: SetNull)

  categoryId String?
  category   Category? @relation(fields: [categoryId], references: [id], onDelete: Set
Null)

  steps      SOPStep[]
  purchases  Purchase[]
  executions SOPExecution[]
  ratings    Rating[]
  cartItems  CartItem[]
}
```

SOP Types:

```
enum SOPType {
  PERSONAL      // Private, owned by user
  GROUP         // Shared within a group
  MARKETPLACE   // Public, for sale
}
```

Key Features:
- **CUID** primary key (shorter than UUID)
- **Price in cents** (e.g., 1000 = $10.00)
- **Cascade delete** when creator is deleted
- **SetNull** when group/category is deleted
- Required fields: title, description, type

6. SOPStep Model

Purpose: Individual steps within a SOP

```
model SOPStep {
  id          String  @id @default(cuid())
  sopId       String
  order       Int
  title       String
  description String
  imageId     String?  // S3 key or Firebase Storage ID
  youtubeUrl  String?
  timerSeconds Int?
  references  String?  // JSON array of reference URLs
  question    String?  // Optional yes/no question
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  sop           SOP            @relation(fields: [sopId], references: [id],
onDelete: Cascade)
  stepExecutions StepExecution[]
}
```

Key Features:
- **Order field** for step sequencing
- **Multimedia support:** Images, YouTube videos
- **Timer integration:** Optional time limit per step
- **References:** JSON array for citations/links
- **Question field:** Optional yes/no checkpoint
- Cascade delete with parent SOP

7. Group Model

Purpose: Team collaboration spaces

```
model Group {
  id          String   @id @default(cuid())
  name        String
  description String
  inviteCode  String   @unique
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  ownerId     String
  owner       User              @relation("GroupOwner", fields: [ownerId],
references: [id], onDelete: Cascade)
  members     User[]            @relation("GroupMembers")
  memberships GroupMembership[]
  sops        SOP[]
}
```

Key Features:
- **Unique invite code** for joining
- **Owner** has admin privileges
- **Members** many-to-many relationship
- **Memberships** tracks approval status

8. GroupMembership Model

Purpose: Track group membership requests and approvals

```
model GroupMembership {
  id        String                @id @default(cuid())
  status    GroupMembershipStatus @default(PENDING)
  createdAt DateTime              @default(now())
  updatedAt DateTime              @updatedAt

  groupId String
  group   Group  @relation(fields: [groupId], references: [id], onDelete: Cascade)

  userId String
  user   User   @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([groupId, userId])
}

enum GroupMembershipStatus {
  PENDING
  APPROVED
  REJECTED
}
```

Key Features:
- Prevents duplicate membership requests
- Approval workflow
- Cascade delete with group/user

9. Purchase Model

Purpose: Track marketplace transactions

```
model Purchase {
  id              String   @id @default(cuid())
  price           Float    // Total price in cents
  platformFee     Float    // 30% platform commission
  creatorRevenue  Float    // 70% creator revenue
  stripePaymentId String?
  createdAt       DateTime @default(now())

  buyerId  String
  buyer    User   @relation("SOPPurchaser", fields: [buyerId], references: [id], onDel
ete: Cascade)

  sellerId String
  seller   User   @relation("SOPSeller", fields: [sellerId], references: [id], onDelet
e: Cascade)

  sopId String
  sop   SOP    @relation(fields: [sopId], references: [id], onDelete: Cascade)
}
```

Revenue Split:
- **Creator:** 70%
- **Platform:** 30%

Key Features:
- Stores split amounts for accounting
- Links to Stripe payment ID
- Tracks buyer and seller

10. SOPExecution Model

Purpose: Track SOP execution sessions

```
model SOPExecution {
  id          String          @id @default(cuid())
  status      ExecutionStatus @default(IN_PROGRESS)
  startedAt   DateTime        @default(now())
  completedAt DateTime?
  totalTimeSeconds Int?

  userId String
  user   User   @relation(fields: [userId], references: [id], onDelete: Cascade)

  sopId String
  sop   SOP    @relation(fields: [sopId], references: [id], onDelete: Cascade)

  stepExecutions StepExecution[]
}

enum ExecutionStatus {
  IN_PROGRESS
  COMPLETED
  ABANDONED
}
```

Key Features:
- Tracks start and completion times
- Calculates total execution time
- Links to individual step executions

11. StepExecution Model

Purpose: Track time and answers for each step

```
model StepExecution {
  id           String @id @default(cuid())
  timeSeconds  Int
  answer       Boolean?  // true = Yes, false = No, null = no question
  completedAt  DateTime @default(now())

  executionId String
  execution   SOPExecution @relation(fields: [executionId], references: [id],
onDelete: Cascade)

  stepId String
  step   SOPStep @relation(fields: [stepId], references: [id], onDelete: Cascade)
}
```

Key Features:
- Granular time tracking per step
- Optional yes/no answers
- Analytics-ready data structure

12. Category Model

Purpose: Hierarchical SOP organization

```
model Category {
  id          String   @id @default(cuid())
  name        String   @unique
  description String?
  parentId    String?  // For subcategories
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  parent       Category?  @relation("CategoryHierarchy", fields: [parentId], refer-
ences: [id], onDelete: Cascade)
  subcategories Category[] @relation("CategoryHierarchy")
  sops        SOP[]
  suggestions CategorySuggestion[]
}
```

Key Features:
- Self-referencing hierarchy
- Unlimited nesting depth
- Unique category names

13. CategorySuggestion Model

Purpose: User-submitted category requests

```
model CategorySuggestion {
  id          String   @id @default(cuid())
  name        String
  description String?
  status      String   @default("PENDING")  // PENDING, APPROVED, REJECTED
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  userId   String
  user     User   @relation(fields: [userId], references: [id], onDelete: Cascade)

  categoryId String?
  category   Category? @relation(fields: [categoryId], references: [id], onDelete: Set
Null)
}
```

Workflow:
1. User suggests new category
2. Admin reviews suggestion
3. If approved, links to created category

14. Rating Model

Purpose: User ratings for marketplace SOPs

```
model Rating {
  id        String   @id @default(cuid())
  rating    Int      // 1-5 stars
  comment   String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  userId String
  user   User   @relation(fields: [userId], references: [id], onDelete: Cascade)

  sopId String
  sop   SOP    @relation(fields: [sopId], references: [id], onDelete: Cascade)

  @@unique([userId, sopId])
}
```

Key Features:
- 5-star rating system
- Optional comment
- One rating per user per SOP
- Editable (updatedAt tracking)

15. PaymentTransaction Model

Purpose: Stripe payment tracking

```
model PaymentTransaction {
  id            String   @id @default(cuid())
  sessionId     String   @unique  // Stripe checkout session ID
  paymentId     String?           // Stripe payment intent ID
  amount        Float             // Amount in dollars
  currency      String   @default("usd")
  status        String   @default("PENDING")  // PENDING, COMPLETED, FAILED, EXPIRED
  paymentStatus String?           // Stripe payment status
  metadata      Json?             // SOP ID, user info, etc.
  userId        String?
  userEmail     String?
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt

  @@index([sessionId])
  @@index([userId])
}
```

Payment Flow:
1. Create checkout session → PENDING
2. User completes payment → COMPLETED
3. Payment fails → FAILED
4. Session expires → EXPIRED

16. Cart Model

Purpose: Shopping cart for bulk purchases

```
model Cart {
  id        String     @id @default(cuid())
  userId    String     @unique
  createdAt DateTime   @default(now())
  updatedAt DateTime   @updatedAt

  items     CartItem[]
}
```

Key Features:
- One cart per user
- Persistent storage

17. CartItem Model

Purpose: Items in shopping cart

```
model CartItem {
  id        String   @id @default(cuid())
  cartId    String
  sopId     String
  addedAt   DateTime @default(now())

  cart      Cart     @relation(fields: [cartId], references: [id], onDelete: Cascade)
  sop       SOP      @relation(fields: [sopId], references: [id], onDelete: Cascade)

  @@unique([cartId, sopId])
}
```

Key Features:
- Prevents duplicate items
- Cascade delete with cart
- Tracks when item was added

18. PromoCode Model

Purpose: Discount codes for marketing

```
model PromoCode {
  id              String   @id @default(cuid())
  code            String   @unique
  discountType    String   // "PERCENTAGE" or "FIXED"
  discountValue   Float    // 20 for 20% or 5.00 for $5 off
  minPurchase     Float?
  maxDiscount     Float?
  maxUses         Int?
  currentUses     Int      @default(0)
  isActive        Boolean  @default(true)
  startsAt        DateTime?
  expiresAt       DateTime?
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  @@index([code])
  @@index([isActive])
}
```

Discount Types:
- **PERCENTAGE:** e.g., 20% off
- **FIXED:** e.g., $5.00 off

Key Features:
- Usage limits
- Minimum purchase requirements
- Maximum discount caps
- Time-based activation
- Active/inactive toggle

Database Indexes

Optimized Queries:
- User email lookups
- Magic link token lookups
- Refresh token validation
- Payment session lookups
- Promo code validation
- User ID foreign key lookups

Migration History

1. **20251116112621_init** - Initial schema
2. **20251116112903_remove_password_field** - Passwordless auth
3. **20251116145242_rename_imageurl_to_imageid** - S3 migration
4. **20251120110431_add_category_hierarchy** - Nested categories
5. **20251120133050_add_ratings** - Rating system
6. **20251120192420_add_custom_auth_tables** - Custom auth flow
7. **20251120213958_add_user_profile_fields** - Rich profiles
8. **20251121083029_add_payment_transactions** - Stripe tracking
9. **20251121085505_add_shopping_cart** - Cart feature
10. **20251121112602_add_promo_codes** - Discount codes

Authentication System

Architecture Overview

The application implements a **custom authentication system** with **JWT tokens** and **refresh token rotation**, supporting multiple authentication providers.

Authentication Methods:
1. **Email Magic Link** (passwordless)
2. **Google OAuth 2.0**
3. **Apple Sign In**

Token Strategy:
- **Access Tokens:** 30-minute expiry (JWT)
- **Refresh Tokens:** 7-day expiry (stored in database, hashed)
- **Secure Cookies:** httpOnly, sameSite, secure flags

Authentication Flow

1. Email Magic Link Flow

```
1. User enters email → POST /api/auth/magic-link/request
2. Server generates unique token with 15-minute expiry
3. Server sends email with magic link (via nodemailer)
4. User clicks link → GET /api/auth/magic-link/verify?token=...
5. Server validates token (not expired, not used)
6. Server creates/finds user in database
7. Server generates access + refresh tokens
8. Server stores hashed refresh token in database
9. Server sets refresh_token cookie (httpOnly)
10. User is redirected to dashboard
```

Email Template:

```
// lib/auth/email.ts
const mailOptions = {
  from: process.env.EMAIL_FROM,
  to: email,
  subject: 'Your Magic Link to Sign In',
  html: `
    <p>Click the link below to sign in:</p>
    <a href="${magicLink}">Sign In</a>
    <p>This link expires in 15 minutes.</p>
  `
};
```

2. Google OAuth Flow

```
1. User clicks "Sign in with Google"
2. Frontend redirects to Google OAuth consent screen
3. User approves permissions
4. Google redirects back with ID token
5. Frontend sends ID token → POST /api/auth/google
6. Server verifies ID token with Google JWKS
7. Server extracts user info (email, name, picture)
8. Server creates/updates user in database
9. Server creates AuthProvider record (provider: 'google')
10. Server generates access + refresh tokens
11. Server sets refresh_token cookie
12. User is redirected to dashboard
```

Token Verification:

```
// lib/auth/oauth.ts
import { jwtVerify, createRemoteJWKSet } from 'jose';

export async function verifyGoogleToken(idToken: string) {
  const JWKS = createRemoteJWKSet(
    new URL('https://www.googleapis.com/oauth2/v3/certs')
  );

  const { payload } = await jwtVerify(idToken, JWKS, {
    issuer: ['https://accounts.google.com', 'accounts.google.com'],
    audience: process.env.GOOGLE_CLIENT_ID,
  });

  return payload;
}
```

3. Apple Sign In Flow

Similar to Google OAuth, but uses Apple’s JWKS endpoint:

```
https://appleid.apple.com/auth/keys
```

Key Differences:
- Email may be hidden (relay email)
- `is_private_email` flag in payload
- Requires Apple Developer account configuration

4. Token Refresh Flow

```
1. Access token expires (30 minutes)
2. Frontend receives 401 Unauthorized
3. Frontend calls POST /api/auth/refresh
4. Server reads refresh_token cookie
5. Server verifies JWT signature
6. Server checks database for stored token (not revoked)
7. Server generates NEW refresh token
8. Server revokes old refresh token in database
9. Server stores new hashed refresh token
10. Server returns new access token + sets new refresh_token cookie
```

Refresh Token Rotation: Prevents token replay attacks by invalidating old tokens.

JWT Implementation

Token Structure

Access Token Payload:

```
{
  userId: string;
  email: string;
  type: 'access';
  iat: number;  // Issued at
  exp: number;  // Expires at
}
```

Refresh Token Payload:

```
{
  userId: string;
  email: string;
  type: 'refresh';
  iat: number;
  exp: number;
}
```

Token Generation

```
// lib/auth/jwt.ts
import jwt from 'jsonwebtoken';

const JWT_SECRET = process.env.JWT_SECRET;
const ACCESS_TOKEN_EXPIRY = '30m';
const REFRESH_TOKEN_EXPIRY = '7d';

export function generateAccessToken(userId: string, email: string): string {
  return jwt.sign(
    { userId, email, type: 'access' },
    JWT_SECRET,
    { expiresIn: ACCESS_TOKEN_EXPIRY }
  );
}

export function generateRefreshToken(userId: string, email: string): string {
  return jwt.sign(
    { userId, email, type: 'refresh' },
    JWT_SECRET,
    { expiresIn: REFRESH_TOKEN_EXPIRY }
  );
}
```

Token Verification

```
// lib/auth/jwt.ts
export function verifyToken(token: string): JWTPayload | null {
  try {
    return jwt.verify(token, JWT_SECRET) as JWTPayload;
  } catch (error) {
    console.error('JWT verification failed:', error);
    return null;
  }
}
```

Token Storage

Refresh Tokens: Hashed with SHA-256 before database storage

```
// lib/auth/jwt.ts
export function hashToken(token: string): string {
  const crypto = require('crypto');
  return crypto.createHash('sha256').update(token).digest('hex');
}
```

Database Record:

```
await prisma.refreshToken.create({
  data: {
    userId: user.id,
    token: hashToken(refreshToken),
    userAgent: request.headers.get('user-agent'),
    ip: request.headers.get('x-forwarded-for'),
  }
});
```

Server-Side Authentication

getCurrentUser() Helper

Used in API routes and server components:

```
// lib/auth/server.ts
import { cookies } from 'next/headers';

export async function getCurrentUser() {
  const cookieStore = await cookies();
  const refreshToken = cookieStore.get('refresh_token')?.value;

  if (!refreshToken) return null;

  const payload = verifyToken(refreshToken);
  if (!payload || payload.type !== 'refresh') return null;

  const hashedToken = hashToken(refreshToken);
  const storedToken = await prisma.refreshToken.findUnique({
    where: { token: hashedToken },
    include: { user: true },
  });

  if (!storedToken || storedToken.revokedAt) return null;

  return storedToken.user;
}
```

requireAuth() Helper

Redirects to login if not authenticated:

```
// lib/auth/server.ts
export async function requireAuth() {
  const user = await getCurrentUser();
  if (!user) return null;
  return user;
}
```

Client-Side Authentication

AuthContext Provider

React Context for client-side auth state:

```tsx
// lib/auth/AuthContext.tsx
'use client';

import { createContext, useContext, useState, useEffect } from 'react';

interface AuthContextType {
  user: User | null;
  isLoading: boolean;
  login: (email: string) => Promise<void>;
  logout: () => Promise<void>;
  refreshUser: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function AuthProvider({ children }) {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  // Load user on mount
  useEffect(() => {
    loadUser();
  }, []);

  async function loadUser() {
    // Calls /api/auth/me or similar
  }

  return (
    <AuthContext.Provider value={{ user, isLoading, login, logout, refreshUser }}>
      {children}
    </AuthContext.Provider>
  );
}

export const useAuth = () => useContext(AuthContext);
```

Usage in Components

```tsx
'use client';

import { useAuth } from '@/lib/auth/AuthContext';

export default function Dashboard() {
  const { user, isLoading } = useAuth();

  if (isLoading) return <LoadingSpinner />;
  if (!user) return <Redirect to="/auth" />;

  return <div>Welcome, {user.name}!</div>;
}
```

Cookie Configuration

```
// lib/auth/cookies.ts
export function setAuthCookie(response: NextResponse, token: string) {
  response.cookies.set('refresh_token', token, {
    httpOnly: true,      // Prevents XSS
    secure: true,        // HTTPS only
    sameSite: 'strict',  // Prevents CSRF
    maxAge: 7 * 24 * 60 * 60,  // 7 days
    path: '/',
  });
}
```

Logout Flow

```
// app/api/auth/logout/route.ts
export async function POST(req: NextRequest) {
  const user = await getCurrentUser();
  if (!user) return NextResponse.json({ error: 'Not authenticated' }, { status: 401 })
;

  const cookieStore = await cookies();
  const refreshToken = cookieStore.get('refresh_token')?.value;

  if (refreshToken) {
    const hashedToken = hashToken(refreshToken);

    // Revoke refresh token
    await prisma.refreshToken.update({
      where: { token: hashedToken },
      data: { revokedAt: new Date() }
    });
  }

  // Clear cookie
  const response = NextResponse.json({ success: true });
  response.cookies.delete('refresh_token');

  return response;
}
```

Security Considerations

✅ Implemented Security Measures

1. **HttpOnly Cookies** - Prevents XSS token theft
2. **Token Hashing** - Refresh tokens hashed in database
3. **Token Rotation** - Old tokens invalidated on refresh
4. **JWT Expiration** - Short-lived access tokens (30 min)
5. **HTTPS Only** - Secure cookie flag
6. **SameSite Strict** - CSRF protection
7. **Device Tracking** - User agent and IP logging
8. **Token Revocation** - Manual logout support
9. **OAuth Token Verification** - JWKS validation for Google/Apple

⚠️ Potential Vulnerabilities

1. **JWT Secret Key** - Hardcoded fallback (should be env-only)
2. **No Rate Limiting** - Magic link requests not rate-limited
3. **No CAPTCHA** - Susceptible to email spam
4. **No IP Whitelisting** - No geo-blocking for suspicious IPs
5. **No 2FA** - No two-factor authentication option
6. **No Session Limit** - Users can have unlimited sessions
7. **No Email Verification** - OAuth emails trusted immediately

Environment Variables

```
# JWT Secret (256-bit recommended)
JWT_SECRET="your-256-bit-secret-key-change-in-production"

# OAuth Configuration
GOOGLE_CLIENT_ID="your-google-client-id"
GOOGLE_CLIENT_SECRET="your-google-client-secret"
APPLE_CLIENT_ID="your-apple-client-id"
APPLE_CLIENT_SECRET="your-apple-client-secret"  # P8 key

# Email Configuration (for magic links)
EMAIL_HOST="smtp.gmail.com"
EMAIL_PORT=587
EMAIL_USER="your-email@gmail.com"
EMAIL_PASSWORD="your-app-specific-password"
EMAIL_FROM="noreply@yourdomain.com"

# Application URL
NEXTAUTH_URL="http://localhost:3000"
```

API Routes Summary

Endpoint	Method	Purpose
`/api/auth/magic-link/request`	POST	Send magic link email
`/api/auth/magic-link/verify`	GET	Verify magic link token
`/api/auth/google`	POST	Google OAuth login
`/api/auth/apple`	POST	Apple Sign In login
`/api/auth/refresh`	POST	Refresh access token
`/api/auth/logout`	POST	Revoke refresh token
`/api/auth/test-login`	POST	Test/demo login (dev only)

Features & Functionality

Core Features

1. SOP Creation & Management

Personal SOPs
- Create private SOPs for personal use
- Unlimited steps with rich formatting
- Edit and delete owned SOPs
- Duplicate SOPs for quick creation

Group SOPs
- Share SOPs within private groups
- Group-specific SOP visibility
- Collaborative editing (owner only)

Marketplace SOPs
- Publish SOPs for sale
- Set pricing (minimum $1.00)
- Public visibility
- Revenue sharing (70% creator, 30% platform)

SOP Step Features:
- **Title & Description** - Clear step instructions
- **Image Upload** - Visual guides (AWS S3)
- **YouTube Embeds** - Video tutorials
- **Timer Integration** - Step time limits
- **References** - Citations/source links (JSON array)
- **Questions** - Yes/No checkpoints
- **Step Reordering** - Drag-and-drop step management

SOP Editor Components:
- `StepEditor.tsx` - Individual step editing
- `StepsList.tsx` - Step management with reordering
- `TopBar.tsx` - SOP metadata editor (title, description, category)

2. Marketplace

Browse & Search
- **Search:** Filter by title/description keywords
- **Category Filter:** Browse by category/subcategory
- **Price Filter:** Min/max price range
- **Sorting Options:**
- Newest first
- Oldest first
- Price: Low to High
- Price: High to Low
- Most Purchased
- Highest Rated

SOP Discovery
- Category hierarchy navigation
- User-suggested categories

- Related SOPs recommendations
- Creator profiles with all SOPs

Purchase Flow
1. Browse marketplace
2. View SOP details (step count, preview)
3. Add to cart OR buy now
4. Stripe checkout
5. Immediate access after payment

3. Shopping Cart

Cart Features
- Add multiple SOPs to cart
- Remove items from cart
- View cart total with breakdown
- Apply promo codes
- Bulk checkout with Stripe

Promo Code System
- **Discount Types:**
- Percentage off (e.g., 20%)
- Fixed amount off (e.g., $5.00)
- **Restrictions:**
- Minimum purchase requirements
- Maximum discount caps
- Usage limits (total uses)
- Time-based activation
- Active/inactive toggle

Cart Persistence
- Stored in database (not localStorage)
- Survives logout/login
- One cart per user

4. Payment Processing (Stripe)

Checkout Flow
- Stripe Checkout integration
- Secure payment processing
- Multiple payment methods
- Payment status tracking

Payment Transaction Tracking
- Session ID tracking
- Payment intent ID linking
- Payment status monitoring
- Metadata storage (SOP, user, etc.)

Revenue Calculation
- **Creator Revenue:** 70% of price
- **Platform Fee:** 30% of price
- Automatic split on purchase creation

Purchase Records
- Buyer and seller tracking
- Price snapshot at purchase time
- Stripe payment ID reference
- Purchase timestamp

Post-Purchase
- Immediate SOP access
- Purchase history tracking
- Access to execute purchased SOPs

5. SOP Execution & Tracking

Execution Features
- Step-by-step guided execution
- Progress tracking (N of M steps)
- Timer per step (if configured)
- Total execution time tracking
- Yes/No question answering
- Pause and resume execution

Execution UI
- Progress bar
- Current step highlighting
- Previous/Next navigation
- Step timer with start/pause
- Image/video display
- Reference links

Execution Data Captured
- Start time
- Completion time (if completed)
- Total execution time
- Per-step time spent
- Answers to questions
- Execution status (IN_PROGRESS, COMPLETED, ABANDONED)

Execution Sessions
- Resume interrupted sessions
- Multiple executions of same SOP
- Execution history per SOP
- User-specific execution data

6. Analytics & Insights

SOP Analytics
- Total executions count
- Average execution time
- Completion rate (% completed vs. abandoned)
- Step-level time analysis
- Question answer patterns

User Analytics
- SOPs created count

- SOPs purchased count
- SOPs executed count
- Revenue earned (sellers)
- Execution history

Marketplace Analytics
- Most purchased SOPs
- Highest-rated SOPs
- Popular categories
- Top creators

7. Rating & Review System

Rating Features
- 5-star rating system
- Optional text comments
- One rating per user per SOP
- Editable ratings
- Rating timestamps

Rating Display
- Average rating calculation
- Total rating count
- Rating distribution (5★, 4★, etc.)
- Recent reviews list
- Creator average rating

Rating Components
- `rating-form.tsx` - Submit/edit rating
- `rating-stars.tsx` - Star display
- `ratings-list.tsx` - Review list
- `sop-rating-badge.tsx` - Compact rating display
- `sop-ratings-section.tsx` - Full ratings section

8. Group Collaboration

Group Features
- Create private groups
- Unique invite codes
- Group member management
- Membership approval workflow
- Group-specific SOPs

Group Membership Flow
1. User enters invite code
2. Membership request created (PENDING)
3. Group owner reviews request
4. Owner approves/rejects
5. Approved members can access group SOPs

Group Roles
- **Owner:** Full control, can delete group
- **Members:** View and execute group SOPs
- No editing permissions for members (currently)

Group Pages
- Group list page
- Group detail page (SOPs, members)
- Group settings page
- Group creation page
- Join group page (invite code)

9. User Profile Management

Profile Fields
- Name
- Email (from auth)
- Avatar image
- Bio
- Location
- Website URL
- Social links (Twitter, LinkedIn, GitHub)

Profile Features
- Avatar upload (AWS S3)
- Profile editing
- Public profile view
- Creator profile with SOPs
- Creator ratings/reviews

Profile API
- GET `/api/profile` - Get current user profile
- PUT `/api/profile` - Update profile
- POST `/api/profile/avatar` - Upload avatar

10. Category Management

Category Features
- Hierarchical categories (parent/child)
- Category descriptions
- Unlimited nesting depth
- Category-based SOP filtering

User-Suggested Categories
- Users can suggest new categories
- Approval workflow (PENDING → APPROVED/REJECTED)
- Linked to created category if approved

Category API
- GET `/api/categories` - List all categories
- POST `/api/categories/suggestions` - Suggest category
- GET `/api/categories/suggestions` - List suggestions (admin)

11. File Upload & Storage

AWS S3 Integration
- Secure file uploads
- Pre-signed URLs for downloads
- Organized bucket structure
- Image optimization

Upload Features
- Avatar uploads (JPEG, PNG, WebP)
- SOP step images
- File size limits
- MIME type validation

S3 Configuration

```
// lib/s3.ts
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

const s3Client = new S3Client({
  region: process.env.AWS_REGION,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  },
});
```

12. Internationalization (i18n)

Supported Languages: 10 languages
- English (en)
- Spanish (es)
- French (fr)
- German (de)
- Chinese (zh)
- Arabic (ar)
- Portuguese (pt)
- Polish (pl)
- Russian (ru)
- Ukrainian (uk)

i18n Implementation
- React Context for language state
- Translation JSON files
- Language switcher component
- RTL support for Arabic
- next-intl integration (v4.5.5)

Translation Coverage
- UI labels and buttons
- Form validations
- Error messages
- Email templates
- Marketplace content

13. Theme System

Theme Features
- Light mode
- Dark mode
- System preference detection

- Smooth transitions
- Persistent theme preference

Theme Colors (Samplify Brand)
- **Primary:** Samplify Red (#E63946)
- **Dark Background:** Samplify Navy (hsl(215, 28%, 17%))
- **Accent:** Red for CTAs and highlights

Theme Components
- `theme-provider.tsx` - next-themes integration
- `theme-toggle.tsx` - Light/dark switcher

14. AI-Powered SOP Generation

Document Processing
- Upload PDF documents
- AI extracts steps from document
- Auto-generate SOP structure
- Edit AI-generated steps

API Endpoint
- POST `/api/sops/generate-from-file`
- Accepts PDF files
- Returns structured SOP data

Integration
- `process_document.py` script
- Emergent AI integration
- Qdrant vector database (optional)

Additional Features

Dashboard

- Welcome message
- Quick stats (SOPs created, purchased, executed)
- Recent executions
- Recent purchases
- Create SOP CTA

Navigation

- Responsive navigation bar
- User avatar dropdown
- Theme toggle
- Language switcher
- Cart icon with count
- Mobile menu

Empty States

- Custom empty state component
- Context-aware messaging
- CTA buttons
- Illustrations

Loading States

- Loading spinners
- Skeleton screens
- Optimistic UI updates

Error Handling

- Toast notifications (sonner, react-hot-toast)
- Form validation errors
- API error messages
- 404/403 error pages

Accessibility

- Radix UI components (WCAG compliant)
- Keyboard navigation
- Screen reader support
- Focus management
- ARIA labels

API Endpoints

Complete API Route List (32 Routes)

Authentication (7 routes)

Endpoint	Method	Purpose	Auth Required
`/api/auth/magic-link/request`	POST	Send magic link email	No
`/api/auth/magic-link/verify`	GET	Verify magic link token	No
`/api/auth/google`	POST	Google OAuth login	No
`/api/auth/apple`	POST	Apple Sign In login	No
`/api/auth/refresh`	POST	Refresh access token	Yes
`/api/auth/logout`	POST	Revoke refresh token	Yes
`/api/auth/test-login`	POST	Test login (dev only)	No

SOPs (3 routes)

Endpoint	Method	Purpose	Auth Required
`/api/sops`	GET	List user's SOPs (personal/group)	Yes
`/api/sops`	POST	Create new SOP	Yes
`/api/sops/[id]`	GET	Get SOP details	Yes*
`/api/sops/[id]`	PUT	Update SOP	Yes
`/api/sops/[id]`	DELETE	Delete SOP	Yes
`/api/sops/generate-from-file`	POST	AI-generate SOP from PDF	Yes

*Access control: Creator, group member, or purchased user only

Marketplace (1 route)

Endpoint	Method	Purpose	Auth Required
`/api/marketplace`	GET	Browse marketplace SOPs	No

Query Parameters:
- `search` - Keyword search
- `categoryId` - Filter by category
- `minPrice` - Minimum price (cents)
- `maxPrice` - Maximum price (cents)
- `sortBy` - Sort order (newest, price-asc, price-desc, purchased, rated)

Groups (3 routes)

Endpoint	Method	Purpose	Auth Required
`/api/groups`	GET	List user's groups	Yes
`/api/groups`	POST	Create new group	Yes
`/api/groups/join`	POST	Join group by invite code	Yes
`/api/groups/user-groups`	GET	Get groups user owns/members	Yes

Categories (2 routes)

Endpoint	Method	Purpose	Auth Required
`/api/categories`	GET	List all categories	No
`/api/categories/suggestions`	POST	Suggest new category	Yes
`/api/categories/suggestions`	GET	List suggestions (admin)	Yes

Ratings (2 routes)

Endpoint	Method	Purpose	Auth Required
`/api/ratings`	POST	Submit/update rating	Yes
`/api/ratings/creator/[creatorId]`	GET	Get creator's ratings	No

Rating POST Body:

```
{
  "sopId": "clxy123...",
  "rating": 5,
  "comment": "Excellent SOP!"
}
```

Shopping Cart (2 routes)

Endpoint	Method	Purpose	Auth Required
`/api/cart`	GET	Get user's cart	Yes
`/api/cart`	POST	Add item to cart	Yes
`/api/cart`	DELETE	Remove item from cart	Yes
`/api/cart/checkout`	POST	Checkout cart (Stripe)	Yes

Payment (Stripe) (2 routes)

Endpoint	Method	Purpose	Auth Required
`/api/stripe/create-checkout-session`	POST	Create Stripe checkout	Yes
`/api/stripe/checkout-status/[session_id]`	GET	Get payment status	Yes

Handled by Python FastAPI backend:
- `/api/stripe/webhook` - Stripe webhook handler

Promo Codes (1 route)

Endpoint	Method	Purpose	Auth Required
`/api/promo-codes/validate`	POST	Validate promo code	Yes

Request Body:

```json
{
  "code": "SAVE20",
  "totalAmount": 2500
}
```

Response:

```json
{
  "valid": true,
  "discount": 500,
  "finalAmount": 2000
}
```

SOP Executions (2 routes)

Endpoint	Method	Purpose	Auth Required
`/api/sop-executions`	POST	Start execution session	Yes
`/api/sop-executions/[id]/complete`	POST	Complete execution	Yes

Start Execution Body:

```
{
  "sopId": "clxy123..."
}
```

Complete Execution Body:

```
{
  "status": "COMPLETED",
  "stepExecutions": [
    {
      "stepId": "clxy456...",
      "timeSeconds": 45,
      "answer": true
    }
  ]
}
```

Sessions (2 routes)

Endpoint	Method	Purpose	Auth Required
`/api/sessions`	GET	List user’s execution sessions	Yes
`/api/sessions/[id]`	GET	Get session details	Yes

Profile (2 routes)

Endpoint	Method	Purpose	Auth Required
`/api/profile`	GET	Get current user profile	Yes
`/api/profile`	PUT	Update user profile	Yes
`/api/profile/avatar`	POST	Upload avatar image	Yes

Users (1 route)

Endpoint	Method	Purpose	Auth Required
`/api/users/[userId]`	GET	Get user profile by ID	No

File Upload (1 route)

Endpoint	Method	Purpose	Auth Required
`/api/upload`	POST	Upload file to S3	Yes

Assets (1 route)

Endpoint	Method	Purpose	Auth Required
`/api/assets/[id]`	GET	Get asset pre-signed URL	Yes

API Response Formats

Success Response

```
{
  "success": true,
  "data": { ... }
}
```

Error Response

```
{
  "error": "Error message",
  "details": { ... }
}
```

Pagination (Not Currently Implemented)

```
{
  "data": [ ... ],
  "pagination": {
    "page": 1,
    "pageSize": 20,
    "total": 150,
    "totalPages": 8
  }
}
```

API Authentication

Authentication Methods:
1. **Cookie-based:** refresh_token cookie (primary)
2. **Header-based:** Authorization: Bearer (optional)

Protected Routes:
All routes except auth, marketplace browse, and public profiles require authentication.

Authorization Checks:
- **SOP Access:** Creator, group member, or purchaser
- **SOP Edit/Delete:** Creator only
- **Group Edit/Delete:** Owner only
- **Profile Edit:** Own profile only

UI/UX Design System

Design Philosophy

Brand Identity: Samplify/MedNAIS
Design System: Custom design with Radix UI primitives
Styling Approach: Utility-first (Tailwind CSS)

Color Palette

Brand Colors

Primary Colors:

```
--samplify-red: #E63946;           /* Primary brand color */
--samplify-red-dark: #D62839;      /* Hover/active state */
--samplify-navy: hsl(215, 28%, 17%); /* Dark background */
--samplify-navy-light: hsl(215, 28%, 22%); /* Lighter navy */
```

Light Mode Colors

```
:root {
  --background: 0 0% 100%;            /* White */
  --foreground: 222.2 84% 4.9%;       /* Near black */

  --card: 0 0% 100%;                  /* White */
  --card-foreground: 222.2 84% 4.9%;

  --primary: 354 85% 57%;             /* Samplify Red */
  --primary-foreground: 0 0% 100%;    /* White text on red */

  --secondary: 210 40% 96.1%;         /* Light gray */
  --secondary-foreground: 222.2 47.4% 11.2%;

  --muted: 210 40% 96.1%;             /* Light gray */
  --muted-foreground: 215.4 16.3% 46.9%;

  --accent: 354 85% 57%;              /* Samplify Red */
  --accent-foreground: 0 0% 100%;

  --destructive: 0 84.2% 60.2%;       /* Red for errors */
  --destructive-foreground: 210 40% 98%;

  --border: 214.3 31.8% 91.4%;        /* Light gray border */
  --input: 214.3 31.8% 91.4%;         /* Input border */
  --ring: 354 85% 57%;                /* Focus ring (red) */
}
```

Dark Mode Colors

```
.dark {
  --background: 215 28% 17%;          /* Samplify Navy */
  --foreground: 0 0% 98%;             /* Near white */

  --card: 215 25% 20%;                /* Darker card */
  --card-foreground: 0 0% 98%;

  --primary: 354 85% 57%;             /* Samplify Red (same) */
  --primary-foreground: 0 0% 100%;

  --secondary: 215 25% 25%;           /* Darker gray */
  --secondary-foreground: 0 0% 98%;

  --muted: 215 25% 25%;
  --muted-foreground: 215 20% 65%;

  --accent: 354 85% 57%;              /* Samplify Red (same) */
  --accent-foreground: 0 0% 100%;

  --destructive: 0 62.8% 30.6%;       /* Darker red */
  --destructive-foreground: 0 0% 98%;

  --border: 215 25% 25%;              /* Dark border */
  --input: 215 25% 25%;
  --ring: 354 85% 57%;                /* Focus ring (red) */
}
```

Chart Colors

```
--chart-1: hsl(var(--chart-1));
--chart-2: hsl(var(--chart-2));
--chart-3: hsl(var(--chart-3));
--chart-4: hsl(var(--chart-4));
--chart-5: hsl(var(--chart-5));
```

Typography

Font Family: Inter (Google Font)

```
import { Inter } from "next/font/google";
const inter = Inter({ subsets: ["latin"] });
```

Text Scales:
- **Headings:** `text-4xl`, `text-3xl`, `text-2xl`, `text-xl`
- **Body:** `text-base`, `text-sm`, `text-xs`
- **Weights:** `font-normal`, `font-medium`, `font-semibold`, `font-bold`

Spacing & Layout

Border Radius:

```
--radius: 0.75rem;  /* 12px */
border-radius: var(--radius);
border-radius: calc(var(--radius) - 2px);  /* Nested elements */
border-radius: calc(var(--radius) - 4px);  /* Double-nested */
```

Spacing Scale: Tailwind default (4px base)
- `p-1` → 4px
- `p-2` → 8px
- `p-4` → 16px
- `p-6` → 24px
- `p-8` → 32px

Component Patterns

Buttons

Primary Button:

```
<Button className="bg-primary text-primary-foreground hover:bg-primary/90">
  Primary Action
</Button>
```

Secondary Button:

```
<Button variant="outline">
  Secondary Action
</Button>
```

Samplify Branded Button:

```
<Button className="btn-samplify">
  Get Started
</Button>
```

Button Variants:
- `default` - Primary color
- `destructive` - Red for delete actions
- `outline` - Bordered, transparent background
- `secondary` - Muted background
- `ghost` - No background
- `link` - Text only

Button Sizes:
- `default` - Regular size
- `sm` - Small
- `lg` - Large
- `icon` - Square, icon only

Cards

Standard Card:

```
<Card>
  <CardHeader>
    <CardTitle>Card Title</CardTitle>
    <CardDescription>Card description</CardDescription>
  </CardHeader>
  <CardContent>
    Card content here
  </CardContent>
  <CardFooter>
    <Button>Action</Button>
  </CardFooter>
</Card>
```

SOP Card:

```
<SOPCard
  sop={{
    id,
    title,
    description,
    price,
    creator: { name, avatar },
    _count: { executions, purchases },
  }}
/>
```

Forms

Form with React Hook Form:

```
<Form {...form}>
  <form onSubmit={form.handleSubmit(onSubmit)}>
    <FormField
      control={form.control}
      name="title"
      render={({ field }) => (
        <FormItem>
          <FormLabel>Title</FormLabel>
          <FormControl>
            <Input placeholder="Enter title" {...field} />
          </FormControl>
          <FormDescription>
            Brief description of the field
          </FormDescription>
          <FormMessage />
        </FormItem>
      )}
    />
    <Button type="submit">Submit</Button>
  </form>
</Form>
```

Dialogs & Modals

```
<Dialog open={isOpen} onOpenChange={setIsOpen}>
  <DialogTrigger asChild>
    <Button>Open Dialog</Button>
  </DialogTrigger>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Dialog Title</DialogTitle>
      <DialogDescription>
        Dialog description
      </DialogDescription>
    </DialogHeader>
    <div>Dialog content</div>
    <DialogFooter>
      <Button onClick={() => setIsOpen(false)}>Close</Button>
    </DialogFooter>
  </DialogContent>
</Dialog>
```

Alerts & Toasts

Toast Notifications (Sonner):

```
import { toast } from 'sonner';

// Success toast
toast.success('Operation completed!');

// Error toast
toast.error('Something went wrong');

// Custom toast
toast('Custom message', {
  description: 'Additional details',
  action: {
    label: 'Undo',
    onClick: () => console.log('Undo'),
  },
});
```

Alert Component:

```
<Alert variant="destructive">
  <AlertCircle className="h-4 w-4" />
  <AlertTitle>Error</AlertTitle>
  <AlertDescription>
    Your operation failed. Please try again.
  </AlertDescription>
</Alert>
```

Navigation

Header Navigation:

```
<Navigation>
  <div className="flex items-center gap-4">
    <MednaisLogo />
    <nav className="hidden md:flex gap-4">
      <Link href="/dashboard">Dashboard</Link>
      <Link href="/marketplace">Marketplace</Link>
      <Link href="/groups">Groups</Link>
    </nav>
  </div>
  <div className="flex items-center gap-4">
    <LanguageSwitcher />
    <ThemeToggle />
    <CartButton />
    <UserMenu />
  </div>
</Navigation>
```

Data Display

Table:

```
<Table>
  <TableHeader>
    <TableRow>
      <TableHead>Name</TableHead>
      <TableHead>Status</TableHead>
      <TableHead>Actions</TableHead>
    </TableRow>
  </TableHeader>
  <TableBody>
    {data.map((item) => (
      <TableRow key={item.id}>
        <TableCell>{item.name}</TableCell>
        <TableCell>
          <Badge variant={item.status === 'active' ? 'default' : 'secondary'}>
            {item.status}
          </Badge>
        </TableCell>
        <TableCell>
          <Button size="sm">Edit</Button>
        </TableCell>
      </TableRow>
    ))}
  </TableBody>
</Table>
```

Badges:

```
<Badge>Default</Badge>
<Badge variant="secondary">Secondary</Badge>
<Badge variant="destructive">Error</Badge>
<Badge variant="outline">Outline</Badge>
```

Responsive Design

Breakpoints (Tailwind default):

- `sm` : 640px
- `md` : 768px

- `lg` : 1024px
- `xl` : 1280px
- `2xl` : 1536px

Responsive Pattern:

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
  {/* Mobile: 1 column, Tablet: 2 columns, Desktop: 3 columns */}
</div>
```

Animations

Accordion Animation:

```
@keyframes accordion-down {
  from { height: 0; }
  to { height: var(--radix-accordion-content-height); }
}

@keyframes accordion-up {
  from { height: var(--radix-accordion-content-height); }
  to { height: 0; }
}
```

Framer Motion:
Used for page transitions, component entrance animations, and micro-interactions.

Tailwind Animate:
Provides utility classes for common animations:
- `animate-spin` - Loading spinners
- `animate-pulse` - Skeleton loaders
- `animate-bounce` - Attention-grabbing

Accessibility

Focus States:
All interactive elements have visible focus states using `ring` color:

```
focus:ring-2 focus:ring-ring focus:ring-offset-2
```

Screen Reader Only:

```
<span className="sr-only">Hidden from visual users</span>
```

ARIA Labels:
Radix UI components include proper ARIA attributes by default.

Keyboard Navigation:
All interactive components support keyboard navigation (Tab, Enter, Escape, Arrow keys).

Icon System

Lucide React:

```
import { Home, Search, User, Settings, ChevronRight } from 'lucide-react';

<Button>
  <Home className="h-4 w-4 mr-2" />
  Home
</Button>
```

Custom MedNAIS Logo:

```
<MednaisLogo className="h-8 w-auto" />
```

Loading States

Loading Spinner:

```
<LoadingSpinner className="h-8 w-8" />
```

Skeleton Loader:

```
<Skeleton className="h-4 w-[250px]" />
<Skeleton className="h-4 w-[200px]" />
```

Empty States

```
<EmptyState
  icon={<Search className="h-12 w-12" />}
  title="No results found"
  description="Try adjusting your search criteria"
  action={
    <Button onClick={resetFilters}>Clear Filters</Button>
  }
/>
```

Key Components

Component Library (78 Components)

Layout Components (5)

1. **footer.tsx** - Site footer
2. **header.tsx** - Site header
3. **page-layout.tsx** - Client-side page wrapper
4. **server-header.tsx** - Server-side header
5. **server-page-layout.tsx** - Server-side page wrapper

SOP Editor Components (4)

1. **StepEditor.tsx** - Individual step editing
   - Title and description fields
   - Image upload
   - YouTube URL input

   - Timer configuration
   - References input
   - Question toggle

2. **StepsList.tsx** - Step management
   - Drag-and-drop reordering (react-sortable-hoc)
   - Step preview
   - Delete step
   - Add step

3. **TopBar.tsx** - SOP metadata editor
   - Title input
   - Description textarea
   - Category selector
   - Type selector (PERSONAL, GROUP, MARKETPLACE)
   - Price input (for MARKETPLACE)
   - Save/Cancel buttons

4. **types.ts** - TypeScript types for editor

Feature-Specific Components (14)

1. **add-to-cart-button.tsx** - Add SOP to cart
2. **copy-button.tsx** - Copy text to clipboard
3. **custom-icon.tsx** - Custom icon wrapper
4. **language-switcher.tsx** - Language selection dropdown
5. **mednais-logo.tsx** - SVG logo component
6. **navigation.tsx** - Main navigation bar
7. **page-header.tsx** - Page title and breadcrumbs
8. **providers.tsx** - App-level providers (Auth, Theme, Query)
9. **purchase-sop-button.tsx** - Purchase button with Stripe
10. **rating-form.tsx** - Submit/edit rating form
11. **rating-stars.tsx** - Star rating display
12. **ratings-list.tsx** - List of user reviews
13. **sop-card.tsx** - SOP preview card
14. **sop-purchase-section.tsx** - Purchase section on SOP detail

UI Components (55+)

Radix UI-based Components:

1. **accordion.tsx** - Collapsible sections
2. **alert-dialog.tsx** - Confirmation dialogs
3. **alert.tsx** - Alert messages
4. **aspect-ratio.tsx** - Aspect ratio container
5. **avatar.tsx** - User avatar
6. **badge.tsx** - Status badges
7. **breadcrumb.tsx** - Breadcrumb navigation
8. **button.tsx** - Button with variants
9. **calendar.tsx** - Date picker calendar
10. **card.tsx** - Card container

11. **carousel.tsx** - Image carousel
12. **checkbox.tsx** - Checkbox input
13. **collapsible.tsx** - Collapsible content
14. **command.tsx** - Command palette
15. **context-menu.tsx** - Right-click menu
16. **date-range-picker.tsx** - Date range selector
17. **dialog.tsx** - Modal dialog
18. **drawer.tsx** - Side drawer
19. **dropdown-menu.tsx** - Dropdown menu
20. **empty-state.tsx** - Empty state placeholder
21. **form.tsx** - Form wrapper (React Hook Form)
22. **hover-card.tsx** - Hover popover
23. **image-upload.tsx** - Image upload with dropzone
24. **input-otp.tsx** - OTP input
25. **input.tsx** - Text input
26. **label.tsx** - Form label
27. **loading-spinner.tsx** - Loading spinner
28. **menubar.tsx** - Menu bar
29. **navigation-menu.tsx** - Navigation menu
30. **pagination.tsx** - Pagination controls
31. **popover.tsx** - Popover tooltip
32. **progress.tsx** - Progress bar
33. **radio-group.tsx** - Radio button group
34. **resizable.tsx** - Resizable panels
35. **scroll-area.tsx** - Scrollable area
36. **select.tsx** - Select dropdown
37. **separator.tsx** - Divider line
38. **sheet.tsx** - Side sheet
39. **skeleton.tsx** - Skeleton loader
40. **slider.tsx** - Range slider
41. **sonner.tsx** - Toast notifications
42. **switch.tsx** - Toggle switch
43. **table.tsx** - Data table
44. **tabs.tsx** - Tab navigation
45. **task-card.tsx** - Task card (custom)
46. **textarea.tsx** - Multi-line text input
47. **timer.tsx** - Timer component (custom)
48. **toast.tsx** - Toast notification
49. **toaster.tsx** - Toast container
50. **toggle-group.tsx** - Toggle button group
51. **toggle.tsx** - Toggle button
52. **tooltip.tsx** - Tooltip

Custom Components:

1. **theme-provider.tsx** - Theme context provider

2. **theme-toggle.tsx** - Light/dark mode toggle
3. **sop-rating-badge.tsx** - Compact rating display
4. **sop-ratings-section.tsx** - Full ratings section

Component Usage Examples

SOP Card

```
<SOPCard
  sop={{
    id: "clxy123",
    title: "How to Make Sourdough Bread",
    description: "Step-by-step guide to making artisan sourdough",
    price: 499,  // $4.99 in cents
    creator: {
      id: "user123",
      name: "John Doe",
      avatar_url: "https://i.pinimg.com/474x/b0/82/c0/b082c01b099e8a0e8b-
d6a50c2b06e135.jpg",
    },
    category: {
      id: "cat1",
      name: "Cooking",
    },
    steps: [ /* array of steps */ ],
    _count: {
      executions: 42,
      purchases: 15,
    },
  }}
/>
```

Rating Form

```
<RatingForm
  sopId="clxy123"
  existingRating={{
    rating: 4,
    comment: "Great SOP!",
  }}
  onSuccess={() => {
    toast.success('Rating submitted');
  }}
/>
```

Add to Cart Button

```
<AddToCartButton
  sopId="clxy123"
  sopTitle="How to Make Sourdough Bread"
  sopPrice={499}
  creatorId="user123"
/>
```

Configuration & Environment

Environment Variables

Required Variables:

```
# Database
DATABASE_URL="postgresql://user:password@localhost:5432/sop_marketplace"

# JWT Authentication
JWT_SECRET="your-256-bit-secret-key-change-in-production"

# Application
NEXTAUTH_URL="http://localhost:3000"
NODE_ENV="development"

# OAuth Providers
GOOGLE_CLIENT_ID="your-google-client-id.apps.googleusercontent.com"
GOOGLE_CLIENT_SECRET="your-google-client-secret"
APPLE_CLIENT_ID="com.yourcompany.yourapp"
APPLE_CLIENT_SECRET="your-apple-p8-key-content"

# Email (Magic Link)
EMAIL_HOST="smtp.gmail.com"
EMAIL_PORT="587"
EMAIL_USER="your-email@gmail.com"
EMAIL_PASSWORD="your-app-specific-password"
EMAIL_FROM="noreply@yourdomain.com"

# Stripe
STRIPE_PUBLISHABLE_KEY="pk_test_..."
STRIPE_SECRET_KEY="sk_test_..."
STRIPE_API_KEY="sk_test_..."
STRIPE_WEBHOOK_SECRET="whsec_..."

# AWS S3
AWS_ACCESS_KEY_ID="your-aws-access-key"
AWS_SECRET_ACCESS_KEY="your-aws-secret-key"
AWS_REGION="us-east-1"
AWS_S3_BUCKET_NAME="your-bucket-name"

# Optional: Firebase (if using)
NEXT_PUBLIC_FIREBASE_API_KEY="your-firebase-api-key"
NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN="your-app.firebaseapp.com"
NEXT_PUBLIC_FIREBASE_PROJECT_ID="your-project-id"
NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET="your-app.appspot.com"
NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID="123456789"
NEXT_PUBLIC_FIREBASE_APP_ID="1:123456789:web:abc123"
FIREBASE_SERVICE_ACCOUNT_KEY='{"type":"service_account",...}'
```

Next.js Configuration

```js
// next.config.js
module.exports = {
  reactStrictMode: true,
  images: {
    domains: [
      'your-s3-bucket.s3.amazonaws.com',
      'firebasestorage.googleapis.com',
    ],
  },
  experimental: {
    serverActions: true,
  },
};
```

Tailwind Configuration

```ts
// tailwind.config.ts
import type { Config } from 'tailwindcss';

const config: Config = {
  darkMode: ['class'],
  content: [
    './pages/**/*.{js,ts,jsx,tsx,mdx}',
    './components/**/*.{js,ts,jsx,tsx,mdx}',
    './app/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {
      colors: {
        samplify: {
          navy: 'hsl(215, 28%, 17%)',
          'navy-light': 'hsl(215, 28%, 22%)',
          red: '#E63946',
          'red-dark': '#D62839',
        },
      },
    },
  },
  plugins: [require('tailwindcss-animate')],
};
```

TypeScript Configuration

```json
{
  "compilerOptions": {
    "target": "es5",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "incremental": true,
    "plugins": [{ "name": "next" }],
    "paths": {
      "@/*": ["./*"]
    }
  },
  "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx", ".next/types/**/*.ts"],
  "exclude": ["node_modules"]
}
```

Prisma Configuration

```
// prisma/schema.prisma
generator client {
    provider = "prisma-client-js"
    binaryTargets = ["native"]
}

generator db {
    provider = "prisma-client-py"
    interface = "asyncio"
}

datasource db {
    provider = "postgresql"
    url      = env("DATABASE_URL")
}
```

Package Manager

Yarn 1.22.22 (Classic)

```json
"packageManager":
"yarn@1.22.22+sha512.a6b2f7906b721bba3d67d4aff083df04dad64c399707841b7acf00f6b133b7ac2
4255f2652fa22ae3534329dc6180534e98d17432037ff6fd140556e2bb3137e"
```

Internationalization

Supported Languages

The application supports **10 languages** with full UI translations:

1. **English (en)** - Default
2. **Spanish (es)** - Español
3. **French (fr)** - Français
4. **German (de)** - Deutsch
5. **Chinese (zh)** - 中文
6. **Arabic (ar)** - العربية (RTL)
7. **Portuguese (pt)** - Português
8. **Polish (pl)** - Polski
9. **Russian (ru)** - Русский
10. **Ukrainian (uk)** - Українська

Implementation

Library: next-intl (v4.5.5)

Language Context:

```tsx
// lib/i18n/LanguageContext.tsx
'use client';

import { createContext, useContext, useState, useEffect } from 'react';
import type { Locale } from './locales';

interface LanguageContextType {
  locale: Locale;
  setLocale: (locale: Locale) => void;
  t: (key: string) => string;
}

const LanguageContext = createContext<LanguageContextType | undefined>(undefined);

export function LanguageProvider({ children }) {
  const [locale, setLocale] = useState<Locale>('en');

  // Load locale from localStorage
  useEffect(() => {
    const saved = localStorage.getItem('locale');
    if (saved) setLocale(saved as Locale);
  }, []);

  // Save locale to localStorage
  useEffect(() => {
    localStorage.setItem('locale', locale);
  }, [locale]);

  const t = (key: string) => {
    // Translation lookup logic
  };

  return (
    <LanguageContext.Provider value={{ locale, setLocale, t }}>
      {children}
    </LanguageContext.Provider>
  );
}

export const useLanguage = () => useContext(LanguageContext);
```

Translation Structure

Translation Keys (Example):

```
// lib/i18n/translations.ts
export const translations = {
  en: {
    common: {
      save: 'Save',
      cancel: 'Cancel',
      delete: 'Delete',
      edit: 'Edit',
    },
    navigation: {
      dashboard: 'Dashboard',
      marketplace: 'Marketplace',
      groups: 'Groups',
      profile: 'Profile',
    },
    sop: {
      create: 'Create SOP',
      title: 'Title',
      description: 'Description',
      steps: 'Steps',
    },
  },
  es: {
    common: {
      save: 'Guardar',
      cancel: 'Cancelar',
      delete: 'Eliminar',
      edit: 'Editar',
    },
    // ... more translations
  },
  // ... other languages
};
```

Language Switcher

```
// components/language-switcher.tsx
import { useLanguage } from '@/lib/i18n/LanguageContext';
import { locales, localeNames } from '@/lib/i18n/locales';

export function LanguageSwitcher() {
  const { locale, setLocale } = useLanguage();

  return (
    <Select value={locale} onValueChange={setLocale}>
      <SelectTrigger className="w-[150px]">
        <SelectValue />
      </SelectTrigger>
      <SelectContent>
        {locales.map((loc) => (
          <SelectItem key={loc} value={loc}>
            {localeNames[loc]}
          </SelectItem>
        ))}
      </SelectContent>
    </Select>
  );
}
```

RTL Support

Arabic Language: Right-to-left layout

```
<html lang={locale} dir={locale === 'ar' ? 'rtl' : 'ltr'}>
  {children}
</html>
```

Areas for Optimization

Performance

1. **Database Query Optimization**
   - Missing pagination on SOP lists
   - No database connection pooling configuration
   - N+1 query issues (e.g., loading ratings separately)
   - Missing indexes on frequently queried fields

2. **Image Optimization**
   - No image compression before S3 upload
   - No responsive image sizes (srcset)
   - No lazy loading implementation
   - No CDN integration for S3 assets

3. **Bundle Size**
   - Large bundle size (multiple chart libraries)
   - Duplicate functionality (Formik + React Hook Form, Yup + Zod)
   - Plotly.js is heavy (~3MB)
   - Not using Next.js dynamic imports for heavy components

4. **Caching**
   - No Redis or in-memory caching
   - No SWR/React Query on all data fetches
   - No static page generation for marketplace
   - No API route caching headers

5. **API Performance**
   - No rate limiting
   - No request compression
   - No API response caching
   - Multiple round trips for related data

Security

1. **Authentication**
   - JWT secret has fallback value (insecure)
   - No rate limiting on magic link requests
   - No CAPTCHA on auth forms
   - No session limit per user
   - No 2FA option

2. **Authorization**
   - Inconsistent authorization checks
   - No role-based access control (RBAC)
   - Group member permissions too permissive

3. **Input Validation**
   - File upload validation inconsistent
   - No MIME type verification on all uploads
   - Missing max file size on some routes

4. **CSRF Protection**
   - Relying on SameSite cookies only
   - No CSRF tokens for sensitive actions

Code Quality

1. **Type Safety**
   - Inconsistent use of TypeScript types
   - Some `any` types in codebase
   - Missing type definitions for API responses

2. **Error Handling**
   - Inconsistent error responses
   - Generic error messages to users
   - No error tracking/monitoring (Sentry, etc.)

3. **Code Duplication**
   - Similar fetch logic repeated across components
   - Duplicate form validation schemas
   - Multiple implementations of similar features

4. **Testing**
   - No unit tests
   - No integration tests
   - No E2E tests
   - No test coverage tracking

User Experience

1. **Loading States**
   - Inconsistent loading indicators
   - No optimistic UI updates
   - Long loading times for image-heavy pages

2. **Error Feedback**
   - Generic error messages
   - No retry mechanisms
   - No offline support

3. **Accessibility**
   - Missing alt text on some images
   - Inconsistent focus management
   - No skip navigation links
   - No keyboard shortcuts

4. **Mobile Experience**
   - Some forms difficult to use on mobile
   - Image upload on mobile clunky
   - Marketplace filters hidden on mobile

Scalability

1. **Database**
   - No read replicas
   - No database sharding strategy
   - No archiving strategy for old data

2. **File Storage**
   - All files in single S3 bucket
   - No file lifecycle policies
   - No backup strategy

3. **Backend Architecture**
   - Next.js API routes not ideal for high traffic
   - No microservices architecture
   - Python backend only for Stripe (underutilized)

Monitoring & Observability

1. **Logging**
   - Console.log statements (not production-ready)
   - No structured logging
   - No log aggregation

2. **Metrics**
   - No application performance monitoring
   - No business metrics tracking
   - No user analytics

3. **Alerting**
   - No error alerting
   - No performance degradation alerts
   - No payment failure notifications

Recommendations

Immediate Priorities (Rebuild Phase)

1. Database Optimization

- **Implement pagination** on all list endpoints (page size: 20-50)
- **Add database indexes:**
  `sql`
  ```
   CREATE INDEX idx_sops_creator ON sops(creatorId);
   CREATE INDEX idx_sops_category ON sops(categoryId);
   CREATE INDEX idx_sops_type_price ON sops(type, price);
   CREATE INDEX idx_purchases_buyer ON purchases(buyerId);
   CREATE INDEX idx_ratings_sop ON ratings(sopId);
  ```

- **Configure connection pooling** in Prisma:
  ```prisma
  datasource db {
    provider = "postgresql"
    url      = env("DATABASE_URL")
    connectionLimit = 10
  }
  ```

2. Security Hardening

- **Remove JWT secret fallback** - Fail fast if not set
- **Add rate limiting** with `next-rate-limit` or Redis
- **Implement CAPTCHA** on auth forms (hCaptcha, reCAPTCHA)
- **Add CSRF tokens** for state-changing requests
- **Implement 2FA** with TOTP (optional for users)

3. Code Quality

- **Consolidate libraries:**
- Remove Formik, use React Hook Form everywhere
- Remove Yup, use Zod everywhere
- Remove one toast library (keep Sonner)
- Replace Plotly with Recharts (lighter)
- **Add TypeScript strict mode:**
  ```json
  {
    "compilerOptions": {
      "strict": true,
      "noImplicitAny": true,
      "strictNullChecks": true
    }
  }
  ```
- **Create API client wrapper** to reduce duplication

4. Performance Optimization

- **Implement image optimization:**
- Next.js Image component everywhere
- Sharp for image compression
- Multiple sizes (thumbnail, medium, large)
- WebP format with fallback
- **Add React Query** for all data fetching:
  ```typescript
  const { data, isLoading, error } = useQuery({
    queryKey: ['sops', filters],
    queryFn: () => fetchSOPs(filters),
    staleTime: 5 * 60 * 1000, // 5 minutes
  });
  ```
- **Implement code splitting:**
  ```typescript
  const SOPEditor = dynamic(() => import('@/components/sop-editor'), {
    loading: () => <LoadingSpinner />,
  ```

```
    ssr: false,
  });
```

5. Error Handling & Monitoring

- **Add Sentry** for error tracking
- **Implement structured logging** with Winston or Pino
- **Create error boundary components**
- **Add retry logic** for failed requests

Medium-Term Improvements

6. User Experience

- **Implement optimistic UI updates:**

```
typescript
  const mutation = useMutation({
    mutationFn: addToCart,
    onMutate: async (newItem) => {
      // Optimistically update UI
      queryClient.setQueryData(['cart'], (old) => [...old, newItem]);
    },
    onError: (err, newItem, context) => {
      // Rollback on error
      queryClient.setQueryData(['cart'], context.previousCart);
    },
  });
```

- **Add skeleton screens** for all loading states
- **Implement infinite scroll** for marketplace
- **Add keyboard shortcuts** for power users

7. Testing Strategy

- **Unit tests** with Jest + Testing Library
- Test all utility functions
- Test React hooks
- Test form validations
- **Integration tests** with Playwright
- Test API routes
- Test authentication flows
- Test purchase flows
- **E2E tests** with Cypress or Playwright
- Critical user journeys
- Purchase flow end-to-end
- SOP creation and execution

8. Analytics & Monitoring

- **Add Google Analytics or Plausible**
- **Implement custom event tracking:**
- SOP purchases
- SOP executions completed
- Cart conversions

- **Add Stripe analytics dashboard**
- **Track key business metrics:**
- Daily Active Users (DAU)
- Monthly Recurring Revenue (MRR)
- Conversion rates
- Average Order Value (AOV)

Long-Term Enhancements

9. Architecture Evolution

- **Migrate to microservices:**
- Payment service (Python/Node)
- File processing service
- Email service
- Analytics service
- **Implement event-driven architecture:**
- Use message queue (RabbitMQ, Redis Streams)
- Event bus for inter-service communication
- **Add CDN** for static assets (Cloudflare, Fastly)

10. Feature Enhancements

- **Advanced Search:**
- Elasticsearch integration
- Faceted search
- Autocomplete
- **AI Recommendations:**
- Personalized SOP suggestions
- Similar SOPs algorithm
- Trending SOPs
- **Social Features:**
- User following
- Activity feed
- Comments on SOPs
- SOP collections/playlists
- **Collaboration:**
- Real-time collaborative editing
- Version control for SOPs
- Change history tracking
- Team workspaces
- **Mobile App:**
- React Native app
- Offline SOP execution
- Push notifications

11. Internationalization Expansion

- **Add more languages:** Japanese, Korean, Italian, Hindi
- **Implement professional translation service**

- **Add currency conversion** for pricing
- **Localize date/time formats**

12. Business Features

- **Subscription plans:**
- Free tier (limited SOPs)
- Pro tier (unlimited SOPs)
- Team tier (group features)
- **Affiliate program:**
- Referral links
- Commission tracking
- **Enterprise features:**
- SSO integration
- Custom branding
- Dedicated support
- SLA guarantees

Technical Debt Priorities

1. **High Priority:**
   - Remove duplicate dependencies
   - Consolidate validation libraries
   - Implement proper error handling
   - Add rate limiting

2. **Medium Priority:**
   - Add comprehensive tests
   - Implement monitoring
   - Optimize database queries
   - Add CDN for assets

3. **Low Priority:**
   - Refactor component structure
   - Improve TypeScript coverage
   - Document API endpoints
   - Create component storybook

Migration to New Stack

Recommended Stack for Rebuild:

Frontend:
- Next.js 15 (latest)
- TypeScript 5.x (strict mode)
- React Query (TanStack Query v5)
- Zod (validation)
- React Hook Form (forms)
- Recharts (charts)
- Radix UI (components)
- Tailwind CSS 4.x (when released)

Backend:
- Next.js API Routes (for CRUD)
- tRPC or GraphQL (type-safe API)
- Prisma ORM
- PostgreSQL 16
- Redis (caching, rate limiting)

Infrastructure:
- Vercel (hosting)
- Neon or Supabase (PostgreSQL)
- Upstash (Redis)
- AWS S3 or Cloudflare R2 (storage)
- Stripe (payments)
- Resend or SendGrid (emails)

Monitoring:
- Sentry (errors)
- Vercel Analytics (performance)
- PostHog or Mixpanel (product analytics)
- LogSnag (notifications)

Migration Strategy:
1. Set up new Next.js project with recommended dependencies
2. Migrate database schema (Prisma migrations)
3. Migrate API routes one by one
4. Migrate UI components with improvements
5. Implement new features (tests, monitoring, etc.)
6. Beta testing with subset of users
7. Gradual rollout with feature flags

Appendices

A. File Structure Quick Reference

```
├── app/                    # Next.js pages and API
├── backend/                # Python FastAPI (Stripe)
├── components/             # React components (78 files)
├── lib/                    # Utilities and helpers
├── prisma/                 # Database schema
├── public/                 # Static assets
├── scripts/                # Seeding and utilities
└── types/                  # TypeScript definitions
```

B. Database Models Quick Reference

1. User
2. AuthProvider
3. MagicLink
4. RefreshToken
5. SOP

6. SOPStep
7. Group
8. GroupMembership
9. Purchase
10. SOPExecution
11. StepExecution
12. Category
13. CategorySuggestion
14. Rating
15. PaymentTransaction
16. Cart
17. CartItem
18. PromoCode

C. API Endpoints Quick Reference

Total: 32 routes
- Authentication: 7
- SOPs: 3
- Marketplace: 1
- Groups: 3
- Categories: 2
- Ratings: 2
- Cart: 2
- Payments: 2
- Promo Codes: 1
- Executions: 2
- Sessions: 2
- Profile: 2
- Users: 1
- Upload: 1
- Assets: 1

D. Key Dependencies Quick Reference

Frontend:
- Next.js 14.2.28
- React 18.2.0
- TypeScript 5.2.2
- Tailwind CSS 3.3.3
- Radix UI (20+ components)
- Prisma 6.7.0

Backend:
- FastAPI 0.115.0
- Prisma Python 0.15.0

External Services:
- Stripe 19.2.0
- AWS SDK 3.0.0
- Firebase 12.6.0

Conclusion

The MedNAIS SOP Marketplace is a well-architected application with a solid foundation. The current implementation includes:

✅ **Strengths:**
- Comprehensive feature set (marketplace, execution tracking, groups, ratings)
- Modern tech stack (Next.js 14, TypeScript, Prisma, PostgreSQL)
- Secure authentication system (JWT, OAuth, magic links)
- Rich UI component library (78 components)
- Internationalization (10 languages)
- Payment processing (Stripe integration)
- Clean database schema (15 models)

⚠️ **Areas for Improvement:**
- Performance optimization needed (caching, pagination, image optimization)
- Security hardening required (rate limiting, CAPTCHA, CSRF)
- Testing infrastructure missing (unit, integration, E2E tests)
- Monitoring and observability gaps (logging, metrics, alerting)
- Code quality improvements (reduce duplication, TypeScript strict mode)

Rebuild Recommendation:
This analysis provides a comprehensive foundation for rebuilding the application with Next.js, PostgreSQL, and TypeScript. The recommended approach is to:
1. Keep the core architecture and database schema
2. Improve performance and security
3. Add testing and monitoring
4. Enhance user experience
5. Reduce technical debt

The rebuild should prioritize **immediate security and performance optimizations** while maintaining **feature parity** with the current implementation.

End of Analysis Report