Legal Diary - Technical Documentation

Version: 1.0.0 **Last Updated:** December 7, 2025 **Author:** Development Team

---

Table of Contents

---

1. Executive Summary

1.1 Project Overview

Legal Diary is a comprehensive case management system designed specifically for law firms and legal practitioners. The application serves as a **Legal Referencer** - a daily task management tool that helps advocates track court hearings, manage cases, and maintain their legal practice efficiently.

1.2 Key Features

Feature	Description
Legal Referencer Dashboard	Daily view of scheduled hearings with previous/next dates
Case Management	Full CRUD operations for legal cases
Hearing Calendar	Interactive calendar for scheduling and viewing court dates
AI-Powered Analysis	GPT-4o integration for case analysis and insights
Document Management	Upload and manage case-related documents
Multi-Firm Support	Firm-based data isolation and user management
Responsive Design	Mobile-first design with vh/vw responsive units

1.3 Target Users

- Advocates and Lawyers
- Law Firm Administrators
- Legal Support Staff
- Paralegals

---

2. System Architecture

2.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                        CLIENT LAYER                              │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │                Next.js 16 (React 19) Frontend              │  │
│  │  ┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐      │  │
│  │  │Dashboard│ │  Cases   │ │Calendar  │ │Settings  │      │  │
│  │  └──────────┘ └──────────┘ └──────────┘ └──────────┘      │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
                              │
                              ▼
```

```
┌─────────────────────────────────────────────────────────────────┐
│                        API LAYER                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │              Next.js API Routes (App Router)              │  │
│  │  ┌─────────┐  ┌─────────┐  ┌─────────┐  ┌─────────┐      │  │
│  │  │  Auth   │  │  Cases  │  │Hearings │  │   AI    │      │  │
│  │  └─────────┘  └─────────┘  └─────────┘  └─────────┘      │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
                              │
              ┌───────────────┼───────────────┐
              ▼               ▼               ▼
┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐
│    PostgreSQL   │  │   File System   │  │    OpenAI API   │
│    (Prisma)     │  │   (Uploads)     │  │    (GPT-4o)     │
└─────────────────┘  └─────────────────┘  └─────────────────┘
```

2.2 Directory Structure

```
legal-diary/
├── src/
│   ├── app/                          # Next.js App Router
│   │   ├── api/                      # API Routes
│   │   │   ├── auth/                 # Authentication endpoints
│   │   │   │   ├── login/route.ts
│   │   │   │   ├── register/route.ts
│   │   │   │   └── logout/route.ts
│   │   │   ├── cases/                # Case management
│   │   │   │   ├── route.ts          # GET all, POST new
│   │   │   │   └── [id]/
│   │   │   │       ├── route.ts      # GET, PUT, DELETE
│   │   │   │       ├── upload/route.ts
│   │   │   │       └── ai/           # AI analysis endpoints
│   │   │   ├── hearings/             # Hearing management
│   │   │   │   ├── route.ts
│   │   │   │   └── [id]/route.ts
│   │   │   ├── dashboard/
│   │   │   │   └── today/route.ts    # Legal referencer
│   │   │   └── firms/route.ts
│   │   ├── dashboard/page.tsx        # Main dashboard
│   │   ├── cases/                    # Case pages
│   │   ├── calendar/page.tsx         # Hearing calendar
│   │   ├── login/page.tsx
│   │   ├── register/page.tsx
│   │   ├── globals.css              # Global styles
│   │   └── layout.tsx               # Root layout
│   ├── components/                  # Reusable components
│   │   ├── Layout/
│   │   │   └── DashboardLayout.tsx
│   │   ├── ProtectedRoute.tsx
│   │   ├── HearingCalendar/
│   │   └── Cases/
│   ├── context/
│   │   └── AuthContext.tsx          # Auth state management
│   ├── lib/                         # Utilities
│   │   ├── prisma.ts                # Database client
│   │   ├── auth.ts                  # Auth utilities
│   │   ├── middleware.ts            # Token verification
│   │   ├── rateLimit.ts             # Rate limiting
│   │   ├── openai.ts                # AI integration
│   │   └── fileProcessor.ts         # Document processing
│   └── generated/prisma/            # Prisma client
├── prisma/
│   ├── schema.prisma                # Database schema
│   └── migrations/                  # Database migrations
├── public/
│   └── uploads/                     # Uploaded documents
├── package.json
├── tsconfig.json
└── next.config.ts
```

3. Technology Stack

3.1 Frontend Technologies

Technology	Version	Purpose
Next.js	16.0.3	React framework with App Router
React	19.2.0	UI library
TypeScript	5.x	Type-safe JavaScript

Ant Design	5.28.x	UI component library
Day.js	1.x	Date manipulation
Turbopack	Built-in	Development bundler

3.2 Backend Technologies

Technology	Version	Purpose
Next.js API Routes	16.0.3	REST API endpoints
Prisma ORM	6.x	Database ORM
PostgreSQL	17.x	Relational database
bcryptjs	3.x	Password hashing
OpenAI SDK	4.x	AI integration

3.3 Development Tools

Tool	Purpose
ESLint	Code linting
TypeScript	Static type checking
Prisma Studio	Database GUI
Git	Version control

---

4. Database Design

4.1 Entity Relationship Diagram

```
┌──────────────┐       ┌──────────────┐       ┌──────────────┐
│     User     │       │     Firm     │       │    Session   │
├──────────────┤       ├──────────────┤       ├──────────────┤
│ id (PK)      │◄──┬──►│ id (PK)      │       │ id (PK)      │
│ email        │   │   │ name         │       │ userId (FK)  │
│ name         │   │   │ address      │       │ token        │
│ password     │   │   │ phone        │       │ expiresAt    │
│ firmId (FK)  │───┘   │ email        │       │ createdAt    │
│ role         │       │ ownerId (FK) │───────┤              │
│ createdAt    │       │ createdAt    │       └──────────────┘
│ updatedAt    │       │ updatedAt    │
└──────────────┘       └──────────────┘
       │
       │ Creates
       ▼
┌──────────────┐       ┌──────────────┐       ┌──────────────┐
│     Case     │       │    Hearing   │       │   AISummary  │
├──────────────┤       ├──────────────┤       ├──────────────┤
│ id (PK)      │◄─────►│ id (PK)      │       │ id (PK)      │
│ caseNumber   │       │ caseId (FK)  │       │ caseId (FK)  │
│ clientName   │       │ hearingDate  │       │ summary      │
│ caseTitle    │       │ hearingTime  │       │ keyPoints    │
│ description  │       │ hearingType  │       │ insights     │
│ status       │       │ courtRoom    │       │ generatedAt  │
│ priority     │       │ notes        │       │ updatedAt    │
│ createdById  │       │ status       │       └──────────────┘
│ firmId (FK)  │       │ createdAt    │
│ opponents    │       │ updatedAt    │
│ courtName    │       └──────────────┘
│ judgeAssigned│              │
│ createdAt    │              │ Has
│ updatedAt    │              ▼
└──────────────┘       ┌──────────────┐
       │               │   Reminder   │
       │ Has           ├──────────────┤
       ▼               │ id (PK)      │
┌──────────────┐       │ hearingId    │
│FileDocument  │       │ reminderType │
├──────────────┤       │ reminderTime │
│ id (PK)      │       │ status       │
│ caseId (FK)  │       │ sentAt       │
│ fileName     │       │ createdAt    │
│ fileUrl      │       └──────────────┘
│ fileType     │
│ fileSize      │
│ uploadedAt    │
└───────────────┘
```

4.2 Model Definitions

User Model

```
model User {
  id            String     @id @default(cuid())
  email         String     @unique
  name          String?
  password      String                         // bcrypt hashed
  ownedFirm     Firm?      @relation("FirmOwner")
  firmMember    Firm?      @relation("FirmMembers", fields: [firmId])
  firmId        String?
  role          UserRole   @default(ADVOCATE) // ADVOCATE, ADMIN, SUPPORT_STAFF
  createdAt     DateTime   @default(now())
  updatedAt     DateTime   @updatedAt
  cases         Case[]     @relation("CaseAdvocate")
  sessions      Session[]

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

Case Model

```
model Case {
  id            String      @id @default(cuid())
  caseNumber    String      @unique
  clientName    String
  clientEmail   String?
  clientPhone   String?
  caseTitle     String
  description   String?
  status        CaseStatus  @default(ACTIVE)
  priority      Priority    @default(MEDIUM)
  createdById   String
  firmId        String
  opponents     String?     // Comma-separated
  courtName     String?
  judgeAssigned String?
  createdAt     DateTime    @default(now())
  updatedAt     DateTime    @updatedAt

  // Relations
  createdBy     User        @relation("CaseAdvocate", fields: [createdById])
  firm          Firm        @relation(fields: [firmId], onDelete: Cascade)
  fileDocuments FileDocument[]
  hearings      Hearing[]
  aiSummary     AISummary?

  @@index([firmId])
  @@index([createdById])
  @@index([status])
  @@index([caseNumber])
}
```

Hearing Model

```
model Hearing {
  id          String        @id @default(cuid())
  caseId      String
  hearingDate DateTime
  hearingTime String?       // HH:MM format
  hearingType HearingType   @default(ARGUMENTS)
  courtRoom   String?
  notes       String?
  status      HearingStatus @default(SCHEDULED)
  createdAt   DateTime      @default(now())
  updatedAt   DateTime      @updatedAt

  // Relations
  case        Case          @relation(fields: [caseId], onDelete: Cascade)
  reminders   Reminder[]

  @@index([caseId])
  @@index([hearingDate])
}
```

4.3 Enumerations

```
enum UserRole {
  ADVOCATE
  ADMIN
  SUPPORT_STAFF
}

enum CaseStatus {
  ACTIVE
  PENDING_JUDGMENT
  CONCLUDED
  APPEAL
  DISMISSED
}

enum Priority {
  LOW
  MEDIUM
  HIGH
  URGENT
}

enum HearingType {
  ARGUMENTS
  EVIDENCE_RECORDING
  FINAL_HEARING
  INTERIM_HEARING
  JUDGMENT_DELIVERY
  PRE_HEARING
  OTHER
}

enum HearingStatus {
  SCHEDULED
  POSTPONED
  COMPLETED
  CANCELLED
}

enum ReminderType {
  ONE_DAY_BEFORE
  THREE_DAYS_BEFORE
  ONE_WEEK_BEFORE
  CUSTOM
}

enum ReminderStatus {
  PENDING
  SENT
  FAILED
}
```

---

5. Authentication & Security

5.1 Authentication Flow

```
┌─────────┐      ┌─────────┐      ┌─────────┐      ┌─────────┐
│  User   │      │ Frontend│      │   API   │      │ Database│
└────┬────┘      └────┬────┘      └────┬────┘      └────┬────┘
     │                │                │                │
     │  Login Form    │                │                │
     │───────────────►│                │                │
     │                │                │                │
     │                │  POST /login   │                │
     │                │───────────────►│                │
     │                │                │                │
     │                │                │  Query User    │
     │                │                │───────────────►│
     │                │                │                │
     │                │                │◄───────────────│
     │                │                │   User Data    │
     │                │                │                │
     │                │                │ Verify Password│
     │                │                │ (bcrypt)       │
     │                │                │                │
     │                │                │ Create Session │
     │                │                │───────────────►│
     │                │                │                │
     │                │  JWT Token     │                │
     │                │◄───────────────│                │
     │                │                │                │
     │  Store Token   │                │                │
     │◄───────────────│                │                │
     |  (localStorage)|                |                |
     |                |                |                |
```

5.2 Token Management

Token Generation (`src/lib/auth.ts`):

```
import { randomBytes } from 'crypto';

export function generateSessionToken(): string {
  return randomBytes(32).toString('hex'); // 64-character hex string
}
```

Token Verification (`src/lib/middleware.ts`):

```
export async function verifyToken(token: string) {
  const session = await prisma.session.findUnique({
    where: { token },
    include: { user: { include: { firmMember: true } } },
  });

  if (!session) return null;

  // Check expiration
  if (session.expiresAt < new Date()) {
    await prisma.session.delete({ where: { id: session.id } });
    return null;
  }

  return session.user;
}
```

5.3 Password Security

- **Hashing Algorithm:** bcrypt with 10 salt rounds
- **Storage:** Only hashed passwords stored in database
- **Verification:** Constant-time comparison to prevent timing attacks

```
import bcrypt from 'bcryptjs';

export async function hashPassword(password: string): Promise<string> {
  return bcrypt.hash(password, 10);
}

export async function verifyPassword(password: string, hash: string): Promise<bc
  return bcrypt.compare(password, hash);
}
```

5.4 Rate Limiting

Configuration:

- **Max Attempts:** 5 per email
- **Window:** 15 minutes
- **Lockout:** 1 hour after exceeding limit

```
const MAX_ATTEMPTS = 5;
const WINDOW_MS = 15 * 60 * 1000;     // 15 minutes
const LOCKOUT_MS = 60 * 60 * 1000;    // 1 hour

export function isRateLimited(email: string): boolean {
  const record = rateLimitStore.get(email);
  if (!record) return false;
  if (Date.now() > record.resetTime) {
    rateLimitStore.delete(email);
    return false;
  }
  return record.attempts >= MAX_ATTEMPTS;
}
```

5.5 Data Isolation

All data queries are scoped by `firmId` to ensure firm-level data isolation:

```
// Example: Fetching cases
const cases = await prisma.case.findMany({
  where: { firmId: user.firmId },
});
```

6. API Reference

6.1 Authentication Endpoints

POST /api/auth/register

Description: Register a new user with firm association.

Request Body:

```json
{
  "email": "user@example.com",
  "password": "securePassword123",
  "name": "John Doe",
  "firmName": "Law Offices of John Doe",
  "firmAddress": "123 Legal Street"
}
```

Response (201):

```json
{
  "message": "User registered successfully",
  "user": {
    "id": "clxyz123",
    "email": "user@example.com",
    "name": "John Doe",
    "firmId": "clxyz456"
  },
  "token": "64-character-hex-token"
}
```

POST /api/auth/login

Description: Authenticate user and receive session token.

Request Body:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response (200):

```
{
  "token": "64-character-hex-token",
  "user": {
    "id": "clxyz123",
    "email": "user@example.com",
    "name": "John Doe",
    "firmId": "clxyz456"
  }
}
```

Error Responses:

- `401` : Invalid credentials
- `429` : Too many attempts (rate limited)

POST /api/auth/logout

Description: Invalidate current session.

Headers:

```
Authorization: Bearer <token>
```

Response (200):

```
{
  "message": "Logged out successfully"
}
```

6.2 Case Endpoints

GET /api/cases

Description: Retrieve all cases for the user's firm.

Headers:

```
Authorization: Bearer <token>
```

Response (200):

```
[
  {
    "id": "case123",
    "caseNumber": "CS/2024/001",
    "caseTitle": "Smith v. Jones",
    "clientName": "John Smith",
    "status": "ACTIVE",
    "priority": "HIGH",
    "hearings": [...],
    "fileDocuments": [...],
    "aiSummary": {...}
  }
]
```

POST /api/cases

Description: Create a new case.

Request Body:

```
{
  "caseNumber": "CS/2024/002",
  "caseTitle": "Contract Dispute",
  "clientName": "Jane Doe",
  "clientEmail": "jane@example.com",
  "clientPhone": "+1-555-0123",
  "description": "Breach of contract case...",
  "courtName": "High Court of Delhi",
  "judgeAssigned": "Hon. Justice Smith",
  "opponents": "ABC Corp, XYZ Ltd",
  "priority": "MEDIUM"
}
```

Response (201):

```
{
  "id": "case456",
  "caseNumber": "CS/2024/002",
  "status": "ACTIVE",
  ...
}
```

GET /api/cases/[id]

Description: Retrieve a single case with all related data.

PUT /api/cases/[id]

Description: Update case details.

Allowed Fields:

- caseTitle, description, clientName, clientEmail, clientPhone
- status, priority, courtName, judgeAssigned, opponents

DELETE /api/cases/[id]

Description: Delete a case and all related data (cascading).

6.3 Hearing Endpoints

GET /api/hearings

Description: Retrieve all hearings for the firm.

Response:

```json
[
  {
    "id": "hearing123",
    "caseId": "case456",
    "hearingDate": "2024-12-15T10:00:00Z",
    "hearingTime": "10:00",
    "hearingType": "ARGUMENTS",
    "status": "SCHEDULED",
    "case": {
      "caseNumber": "CS/2024/001",
      "caseTitle": "Smith v. Jones",
      "clientName": "John Smith"
    }
  }
]
```

POST /api/hearings

Description: Schedule a new hearing.

Request Body:

```json
{
  "caseId": "case456",
  "hearingDate": "2024-12-20T00:00:00Z",
  "hearingTime": "10:30",
  "hearingType": "FINAL_HEARING",
  "courtRoom": "Court Room 5",
  "notes": "Prepare final arguments"
}
```

PUT /api/hearings/[id]

Description: Update hearing details.

DELETE /api/hearings/[id]

Description: Remove a hearing.

6.4 Dashboard Endpoints

GET /api/dashboard/today

Description: Retrieve today's hearings with previous/next dates.

Response:

```json
{
  "date": "2024-12-07",
  "totalCount": 3,
  "hearings": [
    {
      "id": "hearing123",
      "caseId": "case456",
      "caseNumber": "CS/2024/001",
      "partyName": "John Smith",
      "caseTitle": "Smith v. Jones",
      "stage": "ACTIVE",
      "courtName": "High Court",
      "hearingType": "ARGUMENTS",
      "previousDate": "2024-11-20T00:00:00Z",
      "currentDate": "2024-12-07T00:00:00Z",
      "nextDate": "2024-12-20T00:00:00Z"
    }
  ]
}
```

6.5 AI Analysis Endpoints

POST /api/cases/[id]/ai/reanalyze

Description: Re-analyze case with AI including all documents.

Response:

```json
{
  "summary": "This case involves...",
  "keyPoints": [
    "Point 1",
    "Point 2"
  ],
  "insights": "Recommendations for this case..."
}
```

POST /api/cases/[id]/ai/analyze-documents

Description: Analyze specific documents.

Request Body:

```json
{
  "documentIds": ["doc1", "doc2"]
}
```

POST /api/cases/[id]/ai/custom-analysis

Description: Ask custom questions about the case.

Request Body:

```json
{
  "prompt": "What are the strongest arguments in this case?",
  "documentIds": ["doc1"]
}
```

---

7. Frontend Architecture

7.1 Component Hierarchy

```
App (layout.tsx)
├── AuthProvider (context)
│   ├── Public Routes
│   │   ├── LoginPage
│   │   └── RegisterPage
│   │
│   └── Protected Routes (ProtectedRoute wrapper)
│       └── DashboardLayout
│           ├── Sidebar (Desktop)
│           ├── Drawer (Mobile)
│           ├── Header
│           └── Content
│               ├── DashboardPage (Legal Referencer)
│               ├── CasesPage
│               │   ├── CasesList
│               │   ├── CreateCasePage
│               │   └── CaseDetailPage
│               │       └── AIAnalysisTab
│               └── CalendarPage
│                   └── HearingCalendar
```

7.2 State Management

AuthContext (`src/context/AuthContext.tsx`):

```typescript
interface AuthContextType {
  user: User | null;
  token: string | null;
  isLoading: boolean;
  login: (email: string, password: string) => Promise<void>;
  logout: () => void;
  register: (data: RegisterData) => Promise<void>;
}
```

Local Storage Keys:

- `token` : JWT session token
- `user` : Serialized user object

7.3 Protected Route Pattern

```
export default function ProtectedRoute({ children }: { children: React.ReactNode
  const { user, isLoading } = useAuth();
  const router = useRouter();

  useEffect(() => {
    if (!isLoading && !user) {
      router.push('/login');
    }
  }, [user, isLoading, router]);

  if (isLoading) return <LoadingSpinner />;
  if (!user) return null;

  return <>{children}</>;
}
```

7.4 Responsive Design System

CSS Variables (`globals.css`):

```
:root {
  --primary-color: #000000;      /* Pure Black */
  --primary-light: #333333;      /* Dark Gray */
  --secondary-color: #666666;    /* Medium Gray */
  --accent-color: #1a1a1a;       /* Almost Black */
  --text-primary: #1a1a1a;
  --text-secondary: #757575;
  --bg-primary: #ffffff;
  --bg-secondary: #fafafa;
  --border-color: #f0f0f0;

  --shadow-sm: 0 1px 3px rgba(0, 0, 0, 0.08);
  --shadow-md: 0 2px 8px rgba(0, 0, 0, 0.12);
  --shadow-lg: 0 4px 16px rgba(0, 0, 0, 0.15);
}
```

Responsive Units:

- Use `vh` , `vw` , `rem` instead of `px`
- Typography uses `clamp()` for fluid scaling

- Example: `font-size: clamp(0.9rem, 2vw, 1.1rem)`

---

8. AI Integration

8.1 OpenAI Configuration

```
// src/lib/openai.ts
import OpenAI from 'openai';

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

8.2 AI Functions

analyzeCaseWithAI

Generates case summary, key points, and insights.

Input:

```
interface CaseAnalysisRequest {
  caseTitle: string;
  caseDescription: string;
  documents?: Array<{ fileName: string; content: string }>;
}
```

Output:

```
interface CaseAnalysisResult {
  summary: string;
  keyPoints: string[];
  insights: string;
}
```

analyzeDocumentsWithAI

Analyzes uploaded documents for legal insights.

Output:

```
interface DocumentAnalysisResult {
  summary: string;
  keyFindings: string[];
  risks: string[];
  recommendations: string[];
}
```

performCustomAnalysis

Answers custom questions about a case.

Output:

```
interface CustomAnalysisResult {
  analysis: string;
  timestamp: string;
}
```

8.3 Model Configuration

- **Model:** `gpt-4o`
- **Max Tokens:** 1500-2048 (varies by function)
- **Response Format:** JSON parsing from text response

---

9. File Management

9.1 Upload Configuration

- **Max File Size:** 10MB (configurable via env)
- **Storage Location:** `public/uploads/`

- **Supported Types:** PDF, DOCX, DOC, TXT, XLSX, XLS

9.2 File Processor

Location: `src/lib/fileProcessor.ts`

Functions:

```
extractTextFromPDF(filePath: string): Promise<ExtractionResult>
extractTextFromDocx(filePath: string): Promise<ExtractionResult>
extractTextFromText(filePath: string): Promise<ExtractionResult>
extractTextFromExcel(filePath: string): Promise<ExtractionResult>
safeExtractFileContent(filePath: string, fileType: string): Promise<ExtractionRe
extractMultipleFiles(filePaths: Array<...>): Promise<DocumentContent[]>
```

9.3 File Upload Flow

```
Client (FormData) → API Route → Save to /public/uploads/
                             → Create FileDocument record
                             → Return file URL
```

---

10. Deployment Guide

10.1 Vercel Deployment

1. **Connect Repository:**

   ```
   vercel link
   ```

2. **Configure Environment Variables:**

   - `DATABASE_URL` (Production PostgreSQL)
   - `OPENAI_API_KEY`
   - `NEXTAUTH_SECRET`

    - `NEXTAUTH_URL` (Your Vercel domain)

3. **Build Command:**

   ```
   npx prisma generate && next build
   ```

4. **Deploy:**

   ```
   vercel --prod
   ```

10.2 Database Setup (Production)

1. **Create PostgreSQL Instance:**
    - Recommended: Supabase, Railway, or Neon

2. **Run Migrations:**

   ```
   npx prisma migrate deploy
   ```

3. **Generate Client:**

   ```
   npx prisma generate
   ```

---

11. Environment Configuration

11.1 Required Variables

```
# Database
DATABASE_URL="postgresql://user:password@host:5432/database"
```

```
# OpenAI
OPENAI_API_KEY="sk-proj-..."

# NextAuth (for future implementation)
NEXTAUTH_SECRET="your-secret-key"
NEXTAUTH_URL="http://localhost:3000"

# File Upload
NEXT_PUBLIC_MAX_FILE_SIZE=10485760
UPLOAD_DIR=public/uploads
```

11.2 Development vs Production

Variable	Development	Production
DATABASE_URL	localhost	Cloud PostgreSQL
NEXTAUTH_URL	http://localhost:3000	https://your-domain.com
Debug logs	Enabled	Disabled

---

12. Code Standards & Patterns

12.1 API Route Pattern

```
export async function GET(request: NextRequest) {
  try {
    // 1. Extract and verify token
    const token = request.headers.get('authorization')?.replace('Bearer ', '');
    if (!token) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
    }

    const user = await verifyToken(token);
    if (!user || !user.firmId) {
      return NextResponse.json({ error: 'Invalid token' }, { status: 401 });
    }
    // 2. Query with firm scope
    const data = await prisma.model.findMany({
      where: { firmId: user.firmId },
    });

    // 3. Return response
    return NextResponse.json(data);
  } catch (error) {
    console.error('Error:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}
```

12.2 Component Pattern

```
'use client';

import { useState, useEffect } from 'react';
import { useAuth } from '@/context/AuthContext';

interface Props {
  // Define props
}

export default function ComponentName({ prop1 }: Props) {
  const { token, user } = useAuth();
  const [data, setData] = useState<DataType | null>(null);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    if (token) {
      fetchData();
    }
  }, [token]);

  const fetchData = async () => {
    setLoading(true);
    try {
      const response = await fetch('/api/endpoint', {
        headers: { Authorization: `Bearer ${token}` },
      });
      if (response.ok) {
        setData(await response.json());
      }
    } catch (error) {
      console.error('Error:', error);
    } finally {
      setLoading(false);
    }
  };

  return (
    // JSX
  );
}
```

12.3 Error Handling

- API routes return consistent error format: `{ error: string }`
- HTTP status codes follow REST conventions
- Client displays user-friendly messages via Ant Design `message` component

12.4 Type Safety

- All API request/response types defined
- Prisma generates TypeScript types from schema
- Strict TypeScript configuration enabled

Appendix A: Quick Reference

Common Commands

```
# Development
npm run dev              # Start dev server (Turbopack)
npm run build            # Build for production
npm run start            # Start production server
npm run lint             # Run ESLint

# Database
npx prisma migrate dev   # Create/apply migrations
npx prisma generate      # Generate Prisma client
npx prisma studio        # Open database GUI
npx prisma db push       # Push schema without migration

# Git
git status               # Check changes
git add .                # Stage changes
git commit -m "message"  # Commit
git push                 # Push to remote
```

API Testing (cURL)

```
# Login
curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"user@example.com","password":"password"}'

# Get Cases
curl http://localhost:3000/api/cases \
  -H "Authorization: Bearer YOUR_TOKEN"

# Create Hearing
curl -X POST http://localhost:3000/api/hearings \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"caseId":"case123","hearingDate":"2024-12-20"}'
```

End of Technical Documentation

Document Version: 1.0.0 Generated: December 7, 2025