# SAINTRIX Beta Development Guide: Complete Implementation with Cursor AI

## 1. Introduction

This comprehensive development guide provides step-by-step instructions for building SAINTRIX Beta from start to finish using Cursor AI as the primary development environment. The guide is specifically designed to enable Cursor AI to generate fully functional frontend and backend code, ensuring the application is production-ready and meets all requirements outlined in the Product Requirements Document.

SAINTRIX Beta represents a sophisticated credit repair software platform that requires careful attention to security, user experience, and regulatory compliance. This guide breaks down the development process into manageable phases, each with detailed implementation instructions, code examples, and testing procedures. By following this guide, Cursor AI will be able to create a complete, working application that handles sensitive financial data securely while providing an intuitive user experience for both clients and administrators.

The development approach emphasizes modern best practices including TypeScript for type safety, comprehensive testing, security-first design, and responsive user interfaces. Each section includes specific prompts and instructions that Cursor AI can use to generate the appropriate code, along with validation steps to ensure the implementation meets the specified requirements.

## 2. Development Environment Setup

### 2.1. Initial Project Structure

The first step in building SAINTRIX Beta involves setting up the development environment and creating the foundational project structure. This section provides detailed instructions for Cursor AI to establish the necessary tools, dependencies, and configurations.

**Supabase Project Initialization:**

Begin by creating a new Supabase project through the Supabase dashboard. The project should be configured with the following specifications:

- **Project Name:** SAINTRIX Beta
- **Database Password:** Generate a strong, unique password and store it securely
- **Region:** Select the region closest to your target users for optimal performance
- **Pricing Plan:** Start with the Pro plan to ensure adequate resources for development and testing

Once the Supabase project is created, navigate to the project settings and note the following critical values:
- Project URL (SUPABASE_URL)
- Anonymous public key (SUPABASE_ANON_KEY)
- Service role key (SUPABASE_SERVICE_ROLE_KEY) - Keep this secret and secure

**Next.js Application Setup:**

Create a new Next.js application with TypeScript support using the following command structure. Cursor AI should execute these commands in sequence:

```
npx create-next-app@latest saintrix-beta --typescript --tailwind --eslint --app --src-dir --import-alias "@/*"
cd saintrix-beta
```

**Essential Dependencies Installation:**

Install the required dependencies for the SAINTRIX Beta application. These packages provide the core functionality needed for authentication, database interaction, UI components, and form handling:

```
npm install @supabase/supabase-js @supabase/auth-helpers-nextjs @supabase/auth-helpers-react
npm install @hookform/resolvers react-hook-form zod
npm install @radix-ui/react-dialog @radix-ui/react-dropdown-menu @radix-ui/react-select
npm install @radix-ui/react-tabs @radix-ui/react-toast @radix-ui/react-tooltip
npm install lucide-react class-variance-authority clsx tailwind-merge
npm install recharts date-fns
npm install @types/node @types/react @types/react-dom
```

**Development Dependencies:**

```
npm install -D @types/jest jest jest-environment-jsdom
npm install -D @testing-library/react @testing-library/jest-dom
```

```
npm install -D prettier eslint-config-prettier
npm install -D @playwright/test
```

**Environment Configuration:**

Create the necessary environment files with the appropriate variables. Cursor AI should create these files with the following structure:

**.env.local:**

```
 NEXT_PUBLIC_SUPABASE_URL=your_supabase_project_url
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key
SUPABASE_SERVICE_ROLE_KEY=your_supabase_service_role_key

# Third-party API keys (to be configured later)
PLAID_CLIENT_ID=your_plaid_client_id
PLAID_SECRET=your_plaid_secret
PLAID_ENV=sandbox

# Email service configuration
RESEND_API_KEY=your_resend_api_key
```

**.env.example:**

```
 NEXT_PUBLIC_SUPABASE_URL=
NEXT_PUBLIC_SUPABASE_ANON_KEY=
SUPABASE_SERVICE_ROLE_KEY=

PLAID_CLIENT_ID=
PLAID_SECRET=
PLAID_ENV=sandbox

RESEND_API_KEY=
```

## 2.2. Project Structure Organization

Cursor AI should organize the project with a clear, scalable structure that separates concerns and makes the codebase maintainable. The following directory structure should be implemented:

```
saintrix-beta/
├─── src/
│    ├─── app/
│    │    ├─── (auth)/
│    │    │    ├─── login/
│    │    │    ├─── register/
```

```
│   │   │   └── reset-password/
│   │   ├── (client)/
│   │   │   ├── dashboard/
│   │   │   ├── credit-reports/
│   │   │   ├── disputes/
│   │   │   ├── documents/
│   │   │   └── profile/
│   │   ├── (admin)/
│   │   │   ├── admin/
│   │   │   │   ├── dashboard/
│   │   │   │   ├── clients/
│   │   │   │   ├── disputes/
│   │   │   │   ├── templates/
│   │   │   │   └── tasks/
│   │   ├── api/
│   │   │   ├── auth/
│   │   │   ├── credit/
│   │   │   ├── disputes/
│   │   │   ├── documents/
│   │   │   └── admin/
│   │   ├── globals.css
│   │   ├── layout.tsx
│   │   └── page.tsx
│   ├── components/
│   │   ├── ui/
│   │   ├── auth/
│   │   ├── client/
│   │   ├── admin/
│   │   └── shared/
│   ├── lib/
│   │   ├── supabase/
│   │   ├── utils/
│   │   ├── validations/
│   │   └── types/
│   ├── hooks/
│   └── styles/
├── supabase/
│   ├── functions/
│   ├── migrations/
│   └── seed.sql
├── tests/
│   ├── components/
│   ├── pages/
│   └── utils/
└── docs/
```

### 2.3. TypeScript Configuration

Cursor AI should configure TypeScript with strict settings to ensure type safety throughout the application. The `tsconfig.json` file should be updated with the following configuration:

```
{
 "compilerOptions": {
  "target": "es5",
  "lib": ["dom", "dom.iterable", "es6"],
  "allowJs": true,
  "skipLibCheck": true,
  "strict": true,
  "noEmit": true,
  "esModuleInterop": true,
  "module": "esnext",
  "moduleResolution": "bundler",
  "resolveJsonModule": true,
  "isolatedModules": true,
  "jsx": "preserve",
  "incremental": true,
  "plugins": [
   {
    "name": "next"
   }
  ],
  "baseUrl": ".",
  "paths": {
   "@/*": ["./src/*"],
   "@/components/*": ["./src/components/*"],
   "@/lib/*": ["./src/lib/*"],
   "@/hooks/*": ["./src/hooks/*"],
   "@/types/*": ["./src/lib/types/*"]
  }
 },
 "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx", ".next/types/**/*.ts"],
 "exclude": ["node_modules"]
}
```

## 3. Database Implementation

### 3.1. Supabase Database Schema Creation

The database schema forms the foundation of SAINTRIX Beta's data management capabilities. Cursor AI should implement the complete database schema as defined in

the technical architecture document. This section provides the exact SQL commands and implementation steps.

**Step 1: Enable Required Extensions**

First, enable the necessary PostgreSQL extensions in the Supabase SQL editor:

```
-- Enable required extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE EXTENSION IF NOT EXISTS "pg_stat_statements";

-- Enable Row Level Security globally
ALTER DATABASE postgres SET row_security = on;
```

**Step 2: Create Core Tables**

Implement the core tables in the following order to maintain referential integrity:

```
-- Profiles table (extends auth.users)
CREATE TABLE profiles (
    id UUID PRIMARY KEY REFERENCES auth.users(id) ON DELETE CASCADE,
    email TEXT UNIQUE NOT NULL,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    phone_number TEXT,
    date_of_birth DATE,
    ssn_encrypted TEXT,
    address_line1 TEXT,
    address_line2 TEXT,
    city TEXT,
    state TEXT,
    zip_code TEXT,
    user_type TEXT NOT NULL DEFAULT 'client' CHECK (user_type IN ('client',
'admin', 'super_admin')),
    client_status TEXT DEFAULT 'active' CHECK (client_status IN ('active', 'on_hold',
'completed', 'suspended')),
    onboarding_completed BOOLEAN DEFAULT FALSE,
    terms_accepted_at TIMESTAMP WITH TIME ZONE,
    privacy_policy_accepted_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    last_login_at TIMESTAMP WITH TIME ZONE
);

-- Credit reports table
CREATE TABLE credit_reports (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES profiles(id) ON DELETE CASCADE,
```

```
    bureau_name TEXT NOT NULL CHECK (bureau_name IN ('equifax', 'experian',
'transunion')),
    report_date DATE NOT NULL,
    credit_score INTEGER,
    score_model TEXT,
    raw_data JSONB,
    parsed_data JSONB,
    import_status TEXT DEFAULT 'pending' CHECK (import_status IN ('pending',
'processing', 'completed', 'failed')),
    import_error_message TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Credit accounts table
CREATE TABLE credit_accounts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    credit_report_id UUID NOT NULL REFERENCES credit_reports(id) ON DELETE
CASCADE,
    account_number_encrypted TEXT,
    account_name TEXT NOT NULL,
    account_type TEXT NOT NULL CHECK (account_type IN ('credit_card',
'mortgage', 'auto_loan', 'personal_loan', 'student_loan', 'other')),
    creditor_name TEXT NOT NULL,
    account_status TEXT CHECK (account_status IN ('open', 'closed', 'paid',
'charged_off', 'collection')),
    balance DECIMAL(12,2),
    credit_limit DECIMAL(12,2),
    payment_history JSONB,
    date_opened DATE,
    date_closed DATE,
    last_payment_date DATE,
    is_negative BOOLEAN DEFAULT FALSE,
    negative_reason TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Credit inquiries table
CREATE TABLE credit_inquiries (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    credit_report_id UUID NOT NULL REFERENCES credit_reports(id) ON DELETE
CASCADE,
    inquiry_type TEXT NOT NULL CHECK (inquiry_type IN ('hard', 'soft')),
    creditor_name TEXT NOT NULL,
    inquiry_date DATE NOT NULL,
    purpose TEXT,
    is_disputed BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Public records table
CREATE TABLE public_records (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    credit_report_id UUID NOT NULL REFERENCES credit_reports(id) ON DELETE
CASCADE,
    record_type TEXT NOT NULL CHECK (record_type IN ('bankruptcy', 'tax_lien',
'judgment', 'foreclosure')),
    filing_date DATE,
    amount DECIMAL(12,2),
    status TEXT,
    court_name TEXT,
    case_number TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Dispute items table
CREATE TABLE dispute_items (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES profiles(id) ON DELETE CASCADE,
    credit_account_id UUID REFERENCES credit_accounts(id) ON DELETE SET NULL,
    credit_inquiry_id UUID REFERENCES credit_inquiries(id) ON DELETE SET NULL,
    public_record_id UUID REFERENCES public_records(id) ON DELETE SET NULL,
    dispute_type TEXT NOT NULL CHECK (dispute_type IN ('account', 'inquiry',
'public_record', 'personal_info')),
    dispute_reason TEXT NOT NULL,
    client_notes TEXT,
    admin_notes TEXT,
    status TEXT DEFAULT 'pending_review' CHECK (status IN (
        'pending_review', 'approved', 'rejected', 'letter_generated',
        'letter_sent', 'response_received', 'resolved', 'unresolved'
    )),
    priority TEXT DEFAULT 'medium' CHECK (priority IN ('low', 'medium', 'high',
'urgent')),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    resolved_at TIMESTAMP WITH TIME ZONE,

    CONSTRAINT dispute_item_single_reference CHECK (
        (credit_account_id IS NOT NULL)::int +
        (credit_inquiry_id IS NOT NULL)::int +
        (public_record_id IS NOT NULL)::int = 1
    )
);
```

**Step 3: Create Supporting Tables**

Continue with the supporting tables for document management, templates, and administrative functions:

```
 -- Letter templates table
CREATE TABLE letter_templates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
```

```
    name TEXT NOT NULL,
    description TEXT,
    template_content TEXT NOT NULL,
    dispute_type TEXT CHECK (dispute_type IN ('account', 'inquiry', 'public_record',
'personal_info')),
    is_active BOOLEAN DEFAULT TRUE,
    version INTEGER DEFAULT 1,
    created_by UUID REFERENCES profiles(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Dispute letters table
CREATE TABLE dispute_letters (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    dispute_item_id UUID NOT NULL REFERENCES dispute_items(id) ON DELETE
CASCADE,
    template_id UUID REFERENCES letter_templates(id),
    letter_content TEXT NOT NULL,
    letter_pdf_path TEXT,
    status TEXT DEFAULT 'draft' CHECK (status IN ('draft', 'generated', 'sent',
'response_received')),
    sent_date DATE,
    response_deadline DATE,
    bureau_response TEXT,
    response_received_date DATE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Documents table
CREATE TABLE documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES profiles(id) ON DELETE CASCADE,
    dispute_item_id UUID REFERENCES dispute_items(id) ON DELETE SET NULL,
    file_name TEXT NOT NULL,
    file_path TEXT NOT NULL,
    file_size BIGINT NOT NULL,
    file_type TEXT NOT NULL,
    document_type TEXT CHECK (document_type IN ('evidence', 'identity',
'bureau_response', 'internal')),
    description TEXT,
    uploaded_by UUID REFERENCES profiles(id),
    is_client_visible BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Admin tasks table
CREATE TABLE admin_tasks (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    assigned_to UUID REFERENCES profiles(id) ON DELETE SET NULL,
    dispute_item_id UUID REFERENCES dispute_items(id) ON DELETE CASCADE,
```

```
    task_type TEXT NOT NULL CHECK (task_type IN (
        'review_dispute', 'generate_letter', 'send_letter',
        'follow_up', 'process_response', 'client_contact'
    )),
    title TEXT NOT NULL,
    description TEXT,
    priority TEXT DEFAULT 'medium' CHECK (priority IN ('low', 'medium', 'high',
'urgent')),
    status TEXT DEFAULT 'pending' CHECK (status IN ('pending', 'in_progress',
'completed', 'cancelled')),
    due_date TIMESTAMP WITH TIME ZONE,
    completed_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Client messages table
CREATE TABLE client_messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    recipient_id UUID NOT NULL REFERENCES profiles(id) ON DELETE CASCADE,
    sender_id UUID NOT NULL REFERENCES profiles(id) ON DELETE CASCADE,
    subject TEXT NOT NULL,
    message_content TEXT NOT NULL,
    message_type TEXT DEFAULT 'general' CHECK (message_type IN ('general',
'dispute_update', 'welcome', 'reminder')),
    is_read BOOLEAN DEFAULT FALSE,
    read_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Audit logs table
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES profiles(id) ON DELETE SET NULL,
    action TEXT NOT NULL,
    table_name TEXT NOT NULL,
    record_id UUID,
    old_values JSONB,
    new_values JSONB,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

### 3.2. Database Indexes and Performance Optimization

Create strategic indexes to optimize query performance for common access patterns:

```
 -- Performance indexes for profiles
CREATE INDEX idx_profiles_user_type ON profiles(user_type);
```

```
CREATE INDEX idx_profiles_client_status ON profiles(client_status) WHERE
user_type = 'client';
CREATE INDEX idx_profiles_email ON profiles(email);

-- Performance indexes for credit reports
CREATE INDEX idx_credit_reports_user_id ON credit_reports(user_id);
CREATE INDEX idx_credit_reports_bureau_date ON credit_reports(bureau_name,
report_date);
CREATE INDEX idx_credit_reports_import_status ON credit_reports(import_status);

-- Performance indexes for credit accounts
CREATE INDEX idx_credit_accounts_report_id ON credit_accounts(credit_report_id);
CREATE INDEX idx_credit_accounts_negative ON credit_accounts(is_negative)
WHERE is_negative = TRUE;
CREATE INDEX idx_credit_accounts_creditor ON credit_accounts(creditor_name);

-- Performance indexes for dispute items
CREATE INDEX idx_dispute_items_user_id ON dispute_items(user_id);
CREATE INDEX idx_dispute_items_status ON dispute_items(status);
CREATE INDEX idx_dispute_items_created_at ON dispute_items(created_at);
CREATE INDEX idx_dispute_items_user_status ON dispute_items(user_id, status);

-- Performance indexes for admin tasks
CREATE INDEX idx_admin_tasks_assigned_to ON admin_tasks(assigned_to);
CREATE INDEX idx_admin_tasks_status ON admin_tasks(status);
CREATE INDEX idx_admin_tasks_due_date ON admin_tasks(due_date);
CREATE INDEX idx_admin_tasks_assigned_status_due ON
admin_tasks(assigned_to, status, due_date);

-- JSONB indexes for credit report data
CREATE INDEX idx_credit_reports_raw_data_score ON credit_reports USING GIN
((raw_data->'score'));
CREATE INDEX idx_credit_accounts_payment_history ON credit_accounts USING
GIN (payment_history);

-- Partial indexes for active records
CREATE INDEX idx_active_dispute_items ON dispute_items(created_at)
WHERE status NOT IN ('resolved', 'unresolved');

CREATE INDEX idx_pending_admin_tasks ON admin_tasks(due_date)
WHERE status = 'pending';
```

### 3.3. Row Level Security (RLS) Policies

Implement comprehensive Row Level Security policies to ensure data access is properly controlled:

```
 -- Enable RLS on all tables
ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;
```

```sql
ALTER TABLE credit_reports ENABLE ROW LEVEL SECURITY;
ALTER TABLE credit_accounts ENABLE ROW LEVEL SECURITY;
ALTER TABLE credit_inquiries ENABLE ROW LEVEL SECURITY;
ALTER TABLE public_records ENABLE ROW LEVEL SECURITY;
ALTER TABLE dispute_items ENABLE ROW LEVEL SECURITY;
ALTER TABLE dispute_letters ENABLE ROW LEVEL SECURITY;
ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
ALTER TABLE admin_tasks ENABLE ROW LEVEL SECURITY;
ALTER TABLE letter_templates ENABLE ROW LEVEL SECURITY;
ALTER TABLE client_messages ENABLE ROW LEVEL SECURITY;
ALTER TABLE audit_logs ENABLE ROW LEVEL SECURITY;

-- Profiles policies
CREATE POLICY "Users can view own profile" ON profiles
  FOR SELECT USING (auth.uid() = id);

CREATE POLICY "Users can update own profile" ON profiles
  FOR UPDATE USING (auth.uid() = id);

CREATE POLICY "Admins can view all profiles" ON profiles
  FOR SELECT USING (
    EXISTS (
      SELECT 1 FROM profiles
      WHERE id = auth.uid()
      AND user_type IN ('admin', 'super_admin')
    )
  );

-- Credit reports policies
CREATE POLICY "Users can view own credit reports" ON credit_reports
  FOR SELECT USING (
    user_id = auth.uid() OR
    EXISTS (
      SELECT 1 FROM profiles
      WHERE id = auth.uid()
      AND user_type IN ('admin', 'super_admin')
    )
  );

CREATE POLICY "System can insert credit reports" ON credit_reports
  FOR INSERT WITH CHECK (
    user_id = auth.uid() OR
    EXISTS (
      SELECT 1 FROM profiles
      WHERE id = auth.uid()
      AND user_type IN ('admin', 'super_admin')
    )
  );

-- Dispute items comprehensive policy
CREATE POLICY "dispute_items_access_policy" ON dispute_items
  FOR ALL USING (
```

```
      user_id = auth.uid()
      OR
      EXISTS (
        SELECT 1 FROM profiles
        WHERE id = auth.uid()
        AND user_type IN ('admin', 'super_admin')
      )
    );

-- Documents policies
CREATE POLICY "Users can view own documents" ON documents
    FOR SELECT USING (
      (user_id = auth.uid() AND is_client_visible = TRUE) OR
      EXISTS (
        SELECT 1 FROM profiles
        WHERE id = auth.uid()
        AND user_type IN ('admin', 'super_admin')
      )
    );

CREATE POLICY "Users can upload own documents" ON documents
    FOR INSERT WITH CHECK (user_id = auth.uid());

-- Admin tasks policies
CREATE POLICY "Admins can view assigned tasks" ON admin_tasks
    FOR SELECT USING (
      assigned_to = auth.uid() OR
      EXISTS (
        SELECT 1 FROM profiles
        WHERE id = auth.uid()
        AND user_type = 'super_admin'
      )
    );

-- Letter templates policies (admin only)
CREATE POLICY "Admins can manage letter templates" ON letter_templates
    FOR ALL USING (
      EXISTS (
        SELECT 1 FROM profiles
        WHERE id = auth.uid()
        AND user_type IN ('admin', 'super_admin')
      )
    );

-- Client messages policies
CREATE POLICY "Users can view own messages" ON client_messages
    FOR SELECT USING (recipient_id = auth.uid());

CREATE POLICY "Admins can send messages" ON client_messages
    FOR INSERT WITH CHECK (
      EXISTS (
        SELECT 1 FROM profiles
```

```
        WHERE id = auth.uid()
        AND user_type IN ('admin', 'super_admin')
    )
  );

-- Audit logs policies (super admin only)
CREATE POLICY "Super admins can view audit logs" ON audit_logs
  FOR SELECT USING (
    EXISTS (
      SELECT 1 FROM profiles
      WHERE id = auth.uid()
      AND user_type = 'super_admin'
    )
  );
```

### 3.4. Database Functions and Triggers

Implement automated functions and triggers for maintaining data integrity and automating workflows:

```
-- Function to update the updated_at timestamp
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = NOW();
  RETURN NEW;
END;
$$ language 'plpgsql';

-- Apply timestamp triggers to relevant tables
CREATE TRIGGER update_profiles_updated_at
  BEFORE UPDATE ON profiles
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_dispute_items_updated_at
  BEFORE UPDATE ON dispute_items
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_admin_tasks_updated_at
  BEFORE UPDATE ON admin_tasks
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_letter_templates_updated_at
  BEFORE UPDATE ON letter_templates
  FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

-- Function to log changes to audit_logs table
CREATE OR REPLACE FUNCTION log_audit_changes()
RETURNS TRIGGER AS $$
```

```
BEGIN
  INSERT INTO audit_logs (
    user_id, action, table_name, record_id, old_values, new_values
  ) VALUES (
    auth.uid(),
    TG_OP,
    TG_TABLE_NAME,
    COALESCE(NEW.id, OLD.id),
    CASE WHEN TG_OP = 'DELETE' THEN to_jsonb(OLD) ELSE NULL END,
    CASE WHEN TG_OP IN ('INSERT', 'UPDATE') THEN to_jsonb(NEW) ELSE NULL
END
  );

  RETURN COALESCE(NEW, OLD);
END;
$$ language 'plpgsql';

-- Apply audit logging to sensitive tables
CREATE TRIGGER audit_dispute_items
  AFTER INSERT OR UPDATE OR DELETE ON dispute_items
  FOR EACH ROW EXECUTE FUNCTION log_audit_changes();

CREATE TRIGGER audit_dispute_letters
  AFTER INSERT OR UPDATE OR DELETE ON dispute_letters
  FOR EACH ROW EXECUTE FUNCTION log_audit_changes();

CREATE TRIGGER audit_profiles
  AFTER UPDATE ON profiles
  FOR EACH ROW EXECUTE FUNCTION log_audit_changes();

-- Function to create admin tasks based on dispute status changes
CREATE OR REPLACE FUNCTION create_admin_task_on_dispute_status_change()
RETURNS TRIGGER AS $$
BEGIN
  -- Create task when dispute is approved
  IF NEW.status = 'approved' AND OLD.status = 'pending_review' THEN
    INSERT INTO admin_tasks (
      dispute_item_id, task_type, title, description, priority, due_date
    ) VALUES (
      NEW.id,
      'generate_letter',
      'Generate dispute letter for ' || (SELECT first_name || ' ' || last_name
FROM profiles WHERE id = NEW.user_id),
      'Generate and review dispute letter for approved dispute item',
      NEW.priority,
      NOW() + INTERVAL '2 days'
    );
  END IF;

  -- Create follow-up task when letter is sent
  IF NEW.status = 'letter_sent' AND OLD.status != 'letter_sent' THEN
    INSERT INTO admin_tasks (
```

```
            dispute_item_id, task_type, title, description, priority, due_date
        ) VALUES (
            NEW.id,
            'follow_up',
            'Follow up on dispute response',
            'Check for bureau response and update dispute status',
            'medium',
            NOW() + INTERVAL '30 days'
        );
    END IF;

    RETURN NEW;
END;
$$ language 'plpgsql';

CREATE TRIGGER create_admin_tasks_on_dispute_change
    AFTER UPDATE ON dispute_items
    FOR EACH ROW EXECUTE FUNCTION
create_admin_task_on_dispute_status_change();

-- Function to automatically create profile when user signs up
CREATE OR REPLACE FUNCTION handle_new_user()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO profiles (id, email, first_name, last_name)
    VALUES (
        NEW.id,
        NEW.email,
        COALESCE(NEW.raw_user_meta_data->>'first_name', ''),
        COALESCE(NEW.raw_user_meta_data->>'last_name', '')
    );
    RETURN NEW;
END;
$$ language 'plpgsql' SECURITY DEFINER;

CREATE TRIGGER on_auth_user_created
    AFTER INSERT ON auth.users
    FOR EACH ROW EXECUTE FUNCTION handle_new_user();
```

# 4. Authentication System Implementation

## 4.1. Supabase Authentication Configuration

The authentication system forms the security foundation of SAINTRIX Beta. Cursor AI should implement a comprehensive authentication system that includes user registration, login, password management, and multi-factor authentication capabilities.

**Supabase Auth Configuration:**

First, configure the Supabase authentication settings through the Supabase dashboard:

1. Navigate to Authentication > Settings in your Supabase project
2. Configure the following settings:
3. **Site URL:** Set to your development URL (http://localhost:3000) and later update for production
4. **Redirect URLs:** Add your application URLs for authentication redirects
5. **Email Templates:** Customize the email templates for confirmation, password reset, and magic links
6. **Security:** Enable email confirmations and configure password requirements

**Authentication Utilities Setup:**

Create the core authentication utilities that will be used throughout the application:

```
 // src/lib/supabase/client.ts
import { createClientComponentClient } from '@supabase/auth-helpers-nextjs'
import { Database } from '@/lib/types/database'

export const createClient = () => createClientComponentClient<Database>()

// src/lib/supabase/server.ts
import { createServerComponentClient } from '@supabase/auth-helpers-nextjs'
import { cookies } from 'next/headers'
import { Database } from '@/lib/types/database'

export const createServerClient = () => {
  const cookieStore = cookies()
  return createServerComponentClient<Database>({ cookies: () => cookieStore })
}

// src/lib/supabase/middleware.ts
import { createMiddlewareClient } from '@supabase/auth-helpers-nextjs'
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'
import { Database } from '@/lib/types/database'

export async function middleware(req: NextRequest) {
  const res = NextResponse.next()
  const supabase = createMiddlewareClient<Database>({ req, res })

  const {
    data: { session },
  } = await supabase.auth.getSession()

  // Protect admin routes
  if (req.nextUrl.pathname.startsWith('/admin')) {
    if (!session) {
      return NextResponse.redirect(new URL('/login', req.url))
```

```
    }

    // Check if user is admin
    const { data: profile } = await supabase
      .from('profiles')
      .select('user_type')
      .eq('id', session.user.id)
      .single()

    if (!profile || !['admin', 'super_admin'].includes(profile.user_type)) {
      return NextResponse.redirect(new URL('/dashboard', req.url))
    }
  }

  // Protect client routes
  if (req.nextUrl.pathname.startsWith('/dashboard') ||
      req.nextUrl.pathname.startsWith('/credit-reports') ||
      req.nextUrl.pathname.startsWith('/disputes')) {
    if (!session) {
      return NextResponse.redirect(new URL('/login', req.url))
    }
  }

  // Redirect authenticated users away from auth pages
  if ((req.nextUrl.pathname === '/login' ||
       req.nextUrl.pathname === '/register') && session) {
    const { data: profile } = await supabase
      .from('profiles')
      .select('user_type')
      .eq('id', session.user.id)
      .single()

    if (profile?.user_type === 'admin' || profile?.user_type === 'super_admin') {
      return NextResponse.redirect(new URL('/admin/dashboard', req.url))
    } else {
      return NextResponse.redirect(new URL('/dashboard', req.url))
    }
  }

  return res
}

export const config = {
  matcher: [
    '/dashboard/:path*',
    '/credit-reports/:path*',
    '/disputes/:path*',
    '/admin/:path*',
    '/login',
    '/register'
  ]
}
```

### 4.2. Authentication Components

Create reusable authentication components that provide a consistent user experience across the application:

```
// src/components/auth/AuthForm.tsx
'use client'

import { useState } from 'react'
import { useRouter } from 'next/navigation'
import { createClient } from '@/lib/supabase/client'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Label } from '@/components/ui/label'
import { Alert, AlertDescription } from '@/components/ui/alert'
import { Loader2, Eye, EyeOff } from 'lucide-react'
import { useForm } from 'react-hook-form'
import { zodResolver } from '@hookform/resolvers/zod'
import { z } from 'zod'

const loginSchema = z.object({
  email: z.string().email('Please enter a valid email address'),
  password: z.string().min(8, 'Password must be at least 8 characters'),
})

const registerSchema = loginSchema.extend({
  firstName: z.string().min(1, 'First name is required'),
  lastName: z.string().min(1, 'Last name is required'),
  confirmPassword: z.string(),
  termsAccepted: z.boolean().refine(val => val === true, {
    message: 'You must accept the terms and conditions'
  })
}).refine((data) => data.password === data.confirmPassword, {
  message: "Passwords don't match",
  path: ["confirmPassword"],
})

type LoginFormData = z.infer<typeof loginSchema>
type RegisterFormData = z.infer<typeof registerSchema>

interface AuthFormProps {
  mode: 'login' | 'register'
}

export function AuthForm({ mode }: AuthFormProps) {
  const [isLoading, setIsLoading] = useState(false)
  const [showPassword, setShowPassword] = useState(false)
  const [error, setError] = useState<string | null>(null)
  const [success, setSuccess] = useState<string | null>(null)
  const router = useRouter()
```

```
const supabase = createClient()

const schema = mode === 'login' ? loginSchema : registerSchema
const {
  register,
  handleSubmit,
  formState: { errors },
} = useForm<LoginFormData | RegisterFormData>({
  resolver: zodResolver(schema),
})

const onSubmit = async (data: LoginFormData | RegisterFormData) => {
  setIsLoading(true)
  setError(null)
  setSuccess(null)

  try {
    if (mode === 'login') {
      const { data: authData, error } = await supabase.auth.signInWithPassword({
        email: data.email,
        password: data.password,
      })

      if (error) throw error

      // Check user type and redirect accordingly
      const { data: profile } = await supabase
        .from('profiles')
        .select('user_type')
        .eq('id', authData.user.id)
        .single()

      if (profile?.user_type === 'admin' || profile?.user_type === 'super_admin') {
        router.push('/admin/dashboard')
      } else {
        router.push('/dashboard')
      }
    } else {
      const registerData = data as RegisterFormData
      const { error } = await supabase.auth.signUp({
        email: registerData.email,
        password: registerData.password,
        options: {
          data: {
            first_name: registerData.firstName,
            last_name: registerData.lastName,
          },
        },
      })

      if (error) throw error
```

```
        setSuccess('Please check your email to confirm your account.')
      }
    } catch (error: any) {
      setError(error.message || 'An unexpected error occurred')
    } finally {
      setIsLoading(false)
    }
  }

  return (
    <div className="w-full max-w-md mx-auto">
      <div className="bg-white shadow-lg rounded-lg p-8">
        <div className="text-center mb-8">
          <h1 className="text-2xl font-bold text-gray-900">
            {mode === 'login' ? 'Sign In' : 'Create Account'}
          </h1>
          <p className="text-gray-600 mt-2">
            {mode === 'login'
              ? 'Welcome back to SAINTRIX Beta'
              : 'Start your credit repair journey'
            }
          </p>
        </div>

        {error && (
          <Alert className="mb-6 border-red-200 bg-red-50">
            <AlertDescription className="text-red-800">{error}</AlertDescription>
          </Alert>
        )}

        {success && (
          <Alert className="mb-6 border-green-200 bg-green-50">
            <AlertDescription className="text-green-800">{success}</AlertDescription>
          </Alert>
        )}

        <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
          {mode === 'register' && (
            <div className="grid grid-cols-2 gap-4">
              <div>
                <Label htmlFor="firstName">First Name</Label>
                <Input
                  id="firstName"
                  type="text"
                  {...register('firstName')}
                  className={errors.firstName ? 'border-red-500' : ''}
                />
                {errors.firstName && (
                  <p className="text-red-500 text-sm mt-1">{errors.firstName.message}
</p>
                )}
              </div>
```

```
          <div>
            <Label htmlFor="lastName">Last Name</Label>
            <Input
              id="lastName"
              type="text"
              {...register('lastName')}
              className={errors.lastName ? 'border-red-500' : ''}
            />
            {errors.lastName && (
              <p className="text-red-500 text-sm mt-1">{errors.lastName.message}</
p>
            )}
          </div>
        </div>
      )}

      <div>
        <Label htmlFor="email">Email Address</Label>
        <Input
          id="email"
          type="email"
          {...register('email')}
          className={errors.email ? 'border-red-500' : ''}
        />
        {errors.email && (
          <p className="text-red-500 text-sm mt-1">{errors.email.message}</p>
        )}
      </div>

      <div>
        <Label htmlFor="password">Password</Label>
        <div className="relative">
          <Input
            id="password"
            type={showPassword ? 'text' : 'password'}
            {...register('password')}
            className={errors.password ? 'border-red-500' : ''}
          />
          <button
            type="button"
            onClick={() => setShowPassword(!showPassword)}
            className="absolute right-3 top-1/2 transform -translate-y-1/2"
          >
            {showPassword ? <EyeOff size={20} /> : <Eye size={20} />}
          </button>
        </div>
        {errors.password && (
          <p className="text-red-500 text-sm mt-1">{errors.password.message}</
p>
        )}
      </div>
```

```
        {mode === 'register' && (
          <>
            <div>
              <Label htmlFor="confirmPassword">Confirm Password</Label>
              <Input
                id="confirmPassword"
                type="password"
                {...register('confirmPassword')}
                className={errors.confirmPassword ? 'border-red-500' : ''}
              />
              {errors.confirmPassword && (
                <p className="text-red-500 text-sm
mt-1">{errors.confirmPassword.message}</p>
              )}
            </div>

            <div className="flex items-center space-x-2">
              <input
                id="termsAccepted"
                type="checkbox"
                {...register('termsAccepted')}
                className="rounded border-gray-300"
              />
              <Label htmlFor="termsAccepted" className="text-sm">
                I agree to the{' '}
                <a href="/terms" className="text-blue-600 hover:underline">
                  Terms of Service
                </a>{' '}
                and{' '}
                <a href="/privacy" className="text-blue-600 hover:underline">
                  Privacy Policy
                </a>
              </Label>
            </div>
            {errors.termsAccepted && (
              <p className="text-red-500 text-sm">{errors.termsAccepted.message}</
p>
            )}
          </>
        )}

        <Button
          type="submit"
          className="w-full"
          disabled={isLoading}
        >
          {isLoading && <Loader2 className="mr-2 h-4 w-4 animate-spin" />}
          {mode === 'login' ? 'Sign In' : 'Create Account'}
        </Button>
      </form>

      <div className="mt-6 text-center">
```

```
        <p className="text-gray-600">
          {mode === 'login' ? "Don't have an account?" : 'Already have an account?'}{' '}
          <a
            href={mode === 'login' ? '/register' : '/login'}
            className="text-blue-600 hover:underline font-medium"
          >
            {mode === 'login' ? 'Sign up' : 'Sign in'}
          </a>
        </p>
      </div>

      {mode === 'login' && (
        <div className="mt-4 text-center">
          <a
            href="/reset-password"
            className="text-sm text-blue-600 hover:underline"
          >
            Forgot your password?
          </a>
        </div>
      )}
    </div>
  </div>
 )
}
```

### 4.3. Authentication Pages

Create the authentication pages that utilize the AuthForm component:

```
// src/app/(auth)/login/page.tsx
import { AuthForm } from '@/components/auth/AuthForm'

export default function LoginPage() {
 return (
  <div className="min-h-screen bg-gray-50 flex items-center justify-center py-12 px-4 sm:px-6 lg:px-8">
    <AuthForm mode="login" />
  </div>
 )
}

// src/app/(auth)/register/page.tsx
import { AuthForm } from '@/components/auth/AuthForm'

export default function RegisterPage() {
 return (
  <div className="min-h-screen bg-gray-50 flex items-center justify-center py-12 px-4 sm:px-6 lg:px-8">
    <AuthForm mode="register" />
```

```
    </div>
  )
}

// src/app/(auth)/reset-password/page.tsx
'use client'

import { useState } from 'react'
import { createClient } from '@/lib/supabase/client'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Label } from '@/components/ui/label'
import { Alert, AlertDescription } from '@/components/ui/alert'
import { Loader2 } from 'lucide-react'

export default function ResetPasswordPage() {
  const [email, setEmail] = useState('')
  const [isLoading, setIsLoading] = useState(false)
  const [message, setMessage] = useState<string | null>(null)
  const [error, setError] = useState<string | null>(null)
  const supabase = createClient()

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault()
    setIsLoading(true)
    setError(null)
    setMessage(null)

    try {
      const { error } = await supabase.auth.resetPasswordForEmail(email, {
        redirectTo: `${window.location.origin}/reset-password/confirm`,
      })

      if (error) throw error

      setMessage('Check your email for the password reset link.')
    } catch (error: any) {
      setError(error.message || 'An unexpected error occurred')
    } finally {
      setIsLoading(false)
    }
  }

  return (
    <div className="min-h-screen bg-gray-50 flex items-center justify-center py-12
px-4 sm:px-6 lg:px-8">
      <div className="w-full max-w-md mx-auto">
        <div className="bg-white shadow-lg rounded-lg p-8">
          <div className="text-center mb-8">
            <h1 className="text-2xl font-bold text-gray-900">Reset Password</h1>
            <p className="text-gray-600 mt-2">
              Enter your email address and we'll send you a link to reset your password.
```

```
          </p>
        </div>

        {error && (
          <Alert className="mb-6 border-red-200 bg-red-50">
            <AlertDescription className="text-red-800">{error}</AlertDescription>
          </Alert>
        )}

        {message && (
          <Alert className="mb-6 border-green-200 bg-green-50">
            <AlertDescription className="text-green-800">{message}</
AlertDescription>
          </Alert>
        )}

        <form onSubmit={handleSubmit} className="space-y-6">
          <div>
            <Label htmlFor="email">Email Address</Label>
            <Input
              id="email"
              type="email"
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              required
            />
          </div>

          <Button type="submit" className="w-full" disabled={isLoading}>
            {isLoading && <Loader2 className="mr-2 h-4 w-4 animate-spin" />}
            Send Reset Link
          </Button>
        </form>

        <div className="mt-6 text-center">
          <a href="/login" className="text-blue-600 hover:underline">
            Back to Sign In
          </a>
        </div>
      </div>
    </div>
  </div>
 )
}
```

### 4.4. Authentication Context and Hooks

Create a context provider and custom hooks for managing authentication state throughout the application:

```tsx
// src/lib/contexts/AuthContext.tsx
'use client'

import { createContext, useContext, useEffect, useState } from 'react'
import { User, Session } from '@supabase/supabase-js'
import { createClient } from '@/lib/supabase/client'
import { Database } from '@/lib/types/database'

type Profile = Database['public']['Tables']['profiles']['Row']

interface AuthContextType {
  user: User | null
  profile: Profile | null
  session: Session | null
  loading: boolean
  signOut: () => Promise<void>
  refreshProfile: () => Promise<void>
}

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

export function AuthProvider({ children }: { children: React.ReactNode }) {
  const [user, setUser] = useState<User | null>(null)
  const [profile, setProfile] = useState<Profile | null>(null)
  const [session, setSession] = useState<Session | null>(null)
  const [loading, setLoading] = useState(true)
  const supabase = createClient()

  const fetchProfile = async (userId: string) => {
    try {
      const { data, error } = await supabase
        .from('profiles')
        .select('*')
        .eq('id', userId)
        .single()

      if (error) throw error
      setProfile(data)
    } catch (error) {
      console.error('Error fetching profile:', error)
      setProfile(null)
    }
  }

  const refreshProfile = async () => {
    if (user) {
      await fetchProfile(user.id)
    }
  }

  const signOut = async () => {
```

```
  await supabase.auth.signOut()
  setUser(null)
  setProfile(null)
  setSession(null)
 }

 useEffect(() => {
  const getSession = async () => {
   const { data: { session } } = await supabase.auth.getSession()
   setSession(session)
   setUser(session?.user ?? null)

   if (session?.user) {
    await fetchProfile(session.user.id)
   }

   setLoading(false)
  }

  getSession()

  const { data: { subscription } } = supabase.auth.onAuthStateChange(
   async (event, session) => {
    setSession(session)
    setUser(session?.user ?? null)

    if (session?.user) {
     await fetchProfile(session.user.id)
    } else {
     setProfile(null)
    }

    setLoading(false)
   }
  )

  return () => subscription.unsubscribe()
 }, [])

 return (
  <AuthContext.Provider
   value={{
    user,
    profile,
    session,
    loading,
    signOut,
    refreshProfile,
   }}
  >
   {children}
  </AuthContext.Provider>
```

```
  )
}

export const useAuth = () => {
  const context = useContext(AuthContext)
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider')
  }
  return context
}

// src/hooks/useRequireAuth.ts
import { useEffect } from 'react'
import { useRouter } from 'next/navigation'
import { useAuth } from '@/lib/contexts/AuthContext'

export function useRequireAuth(requiredUserType?: 'admin' | 'super_admin') {
  const { user, profile, loading } = useAuth()
  const router = useRouter()

  useEffect(() => {
    if (!loading) {
      if (!user) {
        router.push('/login')
        return
      }

      if (requiredUserType && profile) {
        if (requiredUserType === 'super_admin' && profile.user_type !==
'super_admin') {
          router.push('/dashboard')
          return
        }

        if (requiredUserType === 'admin' &&
            !['admin', 'super_admin'].includes(profile.user_type)) {
          router.push('/dashboard')
          return
        }
      }
    }
  }, [user, profile, loading, requiredUserType, router])

  return { user, profile, loading }
}
```

# 5. Frontend Development

## 5.1. UI Component Library Setup

SAINTRIX Beta requires a comprehensive UI component library that provides consistent styling and behavior across the application. Cursor AI should implement a robust component system using Radix UI primitives and Tailwind CSS for styling.

**Base UI Components:**

Create the foundational UI components that will be used throughout the application:

```
// src/components/ui/button.tsx
import * as React from "react"
import { Slot } from "@radix-ui/react-slot"
import { cva, type VariantProps } from "class-variance-authority"
import { cn } from "@/lib/utils"

const buttonVariants = cva(
  "inline-flex items-center justify-center whitespace-nowrap rounded-md text-sm font-medium ring-offset-background transition-colors focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50",
  {
    variants: {
      variant: {
        default: "bg-primary text-primary-foreground hover:bg-primary/90",
        destructive: "bg-destructive text-destructive-foreground hover:bg-destructive/90",
        outline: "border border-input bg-background hover:bg-accent hover:text-accent-foreground",
        secondary: "bg-secondary text-secondary-foreground hover:bg-secondary/80",
        ghost: "hover:bg-accent hover:text-accent-foreground",
        link: "text-primary underline-offset-4 hover:underline",
      },
      size: {
        default: "h-10 px-4 py-2",
        sm: "h-9 rounded-md px-3",
        lg: "h-11 rounded-md px-8",
        icon: "h-10 w-10",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)
```

```tsx
export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement>,
    VariantProps<typeof buttonVariants> {
  asChild?: boolean
}

const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
  ({ className, variant, size, asChild = false, ...props }, ref) => {
    const Comp = asChild ? Slot : "button"
    return (
      <Comp
        className={cn(buttonVariants({ variant, size, className }))}
        ref={ref}
        {...props}
      />
    )
  }
)
Button.displayName = "Button"

export { Button, buttonVariants }

// src/components/ui/input.tsx
import * as React from "react"
import { cn } from "@/lib/utils"

export interface InputProps
  extends React.InputHTMLAttributes<HTMLInputElement> {}

const Input = React.forwardRef<HTMLInputElement, InputProps>(
  ({ className, type, ...props }, ref) => {
    return (
      <input
        type={type}
        className={cn(
          "flex h-10 w-full rounded-md border border-input bg-background px-3 py-2
text-sm ring-offset-background file:border-0 file:bg-transparent file:text-sm
file:font-medium placeholder:text-muted-foreground focus-visible:outline-none
focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2
disabled:cursor-not-allowed disabled:opacity-50",
          className
        )}
        ref={ref}
        {...props}
      />
    )
  }
)
Input.displayName = "Input"

export { Input }
```

```tsx
// src/components/ui/card.tsx
import * as React from "react"
import { cn } from "@/lib/utils"

const Card = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn(
      "rounded-lg border bg-card text-card-foreground shadow-sm",
      className
    )}
    {...props}
  />
))
Card.displayName = "Card"

const CardHeader = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("flex flex-col space-y-1.5 p-6", className)}
    {...props}
  />
))
CardHeader.displayName = "CardHeader"

const CardTitle = React.forwardRef<
  HTMLParagraphElement,
  React.HTMLAttributes<HTMLHeadingElement>
>(({ className, ...props }, ref) => (
  <h3
    ref={ref}
    className={cn(
      "text-2xl font-semibold leading-none tracking-tight",
      className
    )}
    {...props}
  />
))
CardTitle.displayName = "CardTitle"

const CardDescription = React.forwardRef<
  HTMLParagraphElement,
  React.HTMLAttributes<HTMLParagraphElement>
>(({ className, ...props }, ref) => (
  <p
    ref={ref}
```

```
    className={cn("text-sm text-muted-foreground", className)}
    {...props}
  />
))
CardDescription.displayName = "CardDescription"

const CardContent = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div ref={ref} className={cn("p-6 pt-0", className)} {...props} />
))
CardContent.displayName = "CardContent"

const CardFooter = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("flex items-center p-6 pt-0", className)}
    {...props}
  />
))
CardFooter.displayName = "CardFooter"

export { Card, CardHeader, CardFooter, CardTitle, CardDescription, CardContent }
```

**Data Display Components:**

Create specialized components for displaying credit and dispute data:

```
// src/components/shared/CreditScoreDisplay.tsx
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { TrendingUp, TrendingDown, Minus } from 'lucide-react'

interface CreditScoreDisplayProps {
  score: number
  previousScore?: number
  scoreModel?: string
  reportDate?: string
}

export function CreditScoreDisplay({
  score,
  previousScore,
  scoreModel = 'FICO',
  reportDate
}: CreditScoreDisplayProps) {
  const getScoreColor = (score: number) => {
```

```
  if (score >= 800) return 'text-green-600'
  if (score >= 740) return 'text-blue-600'
  if (score >= 670) return 'text-yellow-600'
  if (score >= 580) return 'text-orange-600'
  return 'text-red-600'
}

const getScoreLabel = (score: number) => {
  if (score >= 800) return 'Excellent'
  if (score >= 740) return 'Very Good'
  if (score >= 670) return 'Good'
  if (score >= 580) return 'Fair'
  return 'Poor'
}

const scoreDifference = previousScore ? score - previousScore : 0

return (
  <Card>
    <CardHeader>
      <CardTitle className="text-lg">Credit Score</CardTitle>
    </CardHeader>
    <CardContent>
      <div className="flex items-center justify-between">
        <div>
          <div className={`text-4xl font-bold ${getScoreColor(score)}`}>
            {score}
          </div>
          <div className="text-sm text-gray-600">
            {getScoreLabel(score)} • {scoreModel}
          </div>
          {reportDate && (
            <div className="text-xs text-gray-500 mt-1">
              As of {new Date(reportDate).toLocaleDateString()}
            </div>
          )}
        </div>

        {previousScore && (
          <div className="flex items-center space-x-1">
            {scoreDifference > 0 ? (
              <TrendingUp className="h-4 w-4 text-green-600" />
            ) : scoreDifference < 0 ? (
              <TrendingDown className="h-4 w-4 text-red-600" />
            ) : (
              <Minus className="h-4 w-4 text-gray-400" />
            )}
            <span className={`text-sm font-medium ${
              scoreDifference > 0 ? 'text-green-600' :
              scoreDifference < 0 ? 'text-red-600' : 'text-gray-400'
            }`}>
              {scoreDifference > 0 ? '+' : ''}{scoreDifference}
```

```
            </span>
          </div>
        ))}
      </div>

      <div className="mt-4">
        <div className="w-full bg-gray-200 rounded-full h-2">
          <div
            className={`h-2 rounded-full ${getScoreColor(score).replace('text-', 'bg-')}`}
            style={{ width: `${(score / 850) * 100}%` }}
          />
        </div>
        <div className="flex justify-between text-xs text-gray-500 mt-1">
          <span>300</span>
          <span>850</span>
        </div>
      </div>
    </CardContent>
  </Card>
 )
}

// src/components/shared/DisputeStatusBadge.tsx
import { Badge } from '@/components/ui/badge'
import {
 Clock,
 CheckCircle,
 XCircle,
 FileText,
 Send,
 MessageSquare,
 AlertCircle
} from 'lucide-react'

interface DisputeStatusBadgeProps {
 status: string
 showIcon?: boolean
}

export function DisputeStatusBadge({ status, showIcon = true }:
DisputeStatusBadgeProps) {
 const getStatusConfig = (status: string) => {
  switch (status) {
   case 'pending_review':
    return {
     label: 'Pending Review',
     variant: 'secondary' as const,
     icon: Clock,
     color: 'text-yellow-600'
    }
   case 'approved':
    return {
```

```
        label: 'Approved',
        variant: 'default' as const,
        icon: CheckCircle,
        color: 'text-green-600'
      }
    case 'rejected':
      return {
        label: 'Rejected',
        variant: 'destructive' as const,
        icon: XCircle,
        color: 'text-red-600'
      }
    case 'letter_generated':
      return {
        label: 'Letter Generated',
        variant: 'secondary' as const,
        icon: FileText,
        color: 'text-blue-600'
      }
    case 'letter_sent':
      return {
        label: 'Letter Sent',
        variant: 'default' as const,
        icon: Send,
        color: 'text-purple-600'
      }
    case 'response_received':
      return {
        label: 'Response Received',
        variant: 'secondary' as const,
        icon: MessageSquare,
        color: 'text-indigo-600'
      }
    case 'resolved':
      return {
        label: 'Resolved',
        variant: 'default' as const,
        icon: CheckCircle,
        color: 'text-green-600'
      }
    case 'unresolved':
      return {
        label: 'Unresolved',
        variant: 'destructive' as const,
        icon: AlertCircle,
        color: 'text-red-600'
      }
    default:
      return {
        label: status,
        variant: 'secondary' as const,
        icon: Clock,
```

```
        color: 'text-gray-600'
      }
    }
  }

  const config = getStatusConfig(status)
  const Icon = config.icon

  return (
    <Badge variant={config.variant} className="flex items-center space-x-1">
      {showIcon && <Icon className="h-3 w-3" />}
      <span>{config.label}</span>
    </Badge>
  )
}
```

### 5.2. Client Portal Implementation

The client portal provides the primary interface for end-users to manage their credit repair journey. Cursor AI should implement a comprehensive dashboard with intuitive navigation and clear data presentation.

**Client Dashboard:**

```
// src/app/(client)/dashboard/page.tsx
import { createServerClient } from '@/lib/supabase/server'
import { redirect } from 'next/navigation'
import { CreditScoreDisplay } from '@/components/shared/CreditScoreDisplay'
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { Button } from '@/components/ui/button'
import { DisputeStatusBadge } from '@/components/shared/DisputeStatusBadge'
import {
  FileText,
  TrendingUp,
  AlertTriangle,
  CheckCircle,
  Plus,
  Download
} from 'lucide-react'

export default async function ClientDashboard() {
  const supabase = createServerClient()

  const { data: { session } } = await supabase.auth.getSession()
  if (!session) {
    redirect('/login')
  }

  // Fetch user profile
```

```
  const { data: profile } = await supabase
    .from('profiles')
    .select('*')
    .eq('id', session.user.id)
    .single()

  // Fetch latest credit report
  const { data: latestReport } = await supabase
    .from('credit_reports')
    .select('*')
    .eq('user_id', session.user.id)
    .order('report_date', { ascending: false })
    .limit(1)
    .single()

  // Fetch credit accounts with negative items
  const { data: negativeAccounts } = await supabase
    .from('credit_accounts')
    .select('*')
    .eq('credit_report_id', latestReport?.id)
    .eq('is_negative', true)
    .order('created_at', { ascending: false })

  // Fetch active disputes
  const { data: activeDisputes } = await supabase
    .from('dispute_items')
    .select(`
      *,
      credit_accounts(*),
      credit_inquiries(*),
      public_records(*)
    `)
    .eq('user_id', session.user.id)
    .not('status', 'in', '(resolved,unresolved)')
    .order('created_at', { ascending: false })

  // Fetch recent dispute letters
  const { data: recentLetters } = await supabase
    .from('dispute_letters')
    .select(`
      *,
      dispute_items(*)
    `)
    .in('dispute_item_id', activeDisputes?.map(d => d.id) || [])
    .order('created_at', { ascending: false })
    .limit(5)

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="mb-8">
        <h1 className="text-3xl font-bold text-gray-900">
          Welcome back, {profile?.first_name}!
```

```
        </h1>
        <p className="text-gray-600 mt-2">
          Here's an overview of your credit repair progress
        </p>
      </div>

      {/* Credit Score Section */}
      <div className="grid grid-cols-1 lg:grid-cols-3 gap-6 mb-8">
        <div className="lg:col-span-1">
          {latestReport ? (
            <CreditScoreDisplay
              score={latestReport.credit_score || 0}
              scoreModel={latestReport.score_model}
              reportDate={latestReport.report_date}
            />
          ) : (
            <Card>
              <CardHeader>
                <CardTitle>Credit Score</CardTitle>
              </CardHeader>
              <CardContent>
                <div className="text-center py-8">
                  <FileText className="h-12 w-12 text-gray-400 mx-auto mb-4" />
                  <p className="text-gray-600 mb-4">No credit report imported yet</p>
                  <Button>
                    <Plus className="h-4 w-4 mr-2" />
                    Import Credit Report
                  </Button>
                </div>
              </CardContent>
            </Card>
          )}
        </div>

        <div className="lg:col-span-2">
          <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
            <Card>
              <CardHeader className="pb-2">
                <CardTitle className="text-sm font-medium text-gray-600">
                  Negative Items
                </CardTitle>
              </CardHeader>
              <CardContent>
                <div className="flex items-center">
                  <AlertTriangle className="h-8 w-8 text-red-500 mr-3" />
                  <div>
                    <div className="text-2xl font-bold text-red-600">
                      {negativeAccounts?.length || 0}
                    </div>
                    <div className="text-sm text-gray-600">Items found</div>
                  </div>
                </div>
```

```
          </CardContent>
        </Card>

        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-sm font-medium text-gray-600">
              Active Disputes
            </CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex items-center">
              <TrendingUp className="h-8 w-8 text-blue-500 mr-3" />
              <div>
                <div className="text-2xl font-bold text-blue-600">
                  {activeDisputes?.length || 0}
                </div>
                <div className="text-sm text-gray-600">In progress</div>
              </div>
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-sm font-medium text-gray-600">
              Resolved Items
            </CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex items-center">
              <CheckCircle className="h-8 w-8 text-green-500 mr-3" />
              <div>
                <div className="text-2xl font-bold text-green-600">0</div>
                <div className="text-sm text-gray-600">Completed</div>
              </div>
            </div>
          </CardContent>
        </Card>
      </div>
    </div>
  </div>

  {/* Active Disputes Section */}
  <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
    <Card>
      <CardHeader>
        <CardTitle className="flex items-center justify-between">
          Active Disputes
          <Button size="sm" variant="outline">
            <Plus className="h-4 w-4 mr-2" />
            New Dispute
          </Button>
```

```
          </CardTitle>
        </CardHeader>
        <CardContent>
          {activeDisputes && activeDisputes.length > 0 ? (
            <div className="space-y-4">
              {activeDisputes.slice(0, 5).map((dispute) => (
                <div key={dispute.id} className="flex items-center justify-between p-3
border rounded-lg">
                  <div>
                    <div className="font-medium">
                      {dispute.credit_accounts?.account_name ||
                       dispute.credit_inquiries?.creditor_name ||
                       'Personal Information'}
                    </div>
                    <div className="text-sm text-gray-600">
                      {dispute.dispute_reason}
                    </div>
                    <div className="mt-1">
                      <DisputeStatusBadge status={dispute.status} />
                    </div>
                  </div>
                  <div className="text-sm text-gray-500">
                    {new Date(dispute.created_at).toLocaleDateString()}
                  </div>
                </div>
              ))}
            </div>
          ) : (
            <div className="text-center py-8">
              <AlertTriangle className="h-12 w-12 text-gray-400 mx-auto mb-4" />
              <p className="text-gray-600">No active disputes</p>
            </div>
          )}
        </CardContent>
      </Card>

      <Card>
        <CardHeader>
          <CardTitle>Recent Letters</CardTitle>
        </CardHeader>
        <CardContent>
          {recentLetters && recentLetters.length > 0 ? (
            <div className="space-y-4">
              {recentLetters.map((letter) => (
                <div key={letter.id} className="flex items-center justify-between p-3
border rounded-lg">
                  <div>
                    <div className="font-medium">
                      Dispute Letter #{letter.id.slice(0, 8)}
                    </div>
                    <div className="text-sm text-gray-600">
                      {letter.status === 'generated' ? 'Ready to send' :
```

```
                    letter.status === 'sent' ? `Sent ${letter.sent_date}` :
                    'In progress'}
                  </div>
                </div>
                <Button size="sm" variant="outline">
                  <Download className="h-4 w-4 mr-2" />
                  Download
                </Button>
              </div>
            ))}
          </div>
        ) : (
          <div className="text-center py-8">
            <FileText className="h-12 w-12 text-gray-400 mx-auto mb-4" />
            <p className="text-gray-600">No letters generated yet</p>
          </div>
        )}
      </CardContent>
    </Card>
  </div>
 )
}
```

**Credit Report Display:**

```
// src/app/(client)/credit-reports/page.tsx
import { createServerClient } from '@/lib/supabase/server'
import { redirect } from 'next/navigation'
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { Button } from '@/components/ui/button'
import { Badge } from '@/components/ui/badge'
import { CreditScoreDisplay } from '@/components/shared/CreditScoreDisplay'
import {
  Plus,
  AlertTriangle,
  CheckCircle,
  CreditCard,
  Building,
  Car,
  GraduationCap,
  DollarSign
} from 'lucide-react'

export default async function CreditReportsPage() {
  const supabase = createServerClient()

  const { data: { session } } = await supabase.auth.getSession()
  if (!session) {
    redirect('/login')
```

```
}

// Fetch all credit reports
const { data: creditReports } = await supabase
  .from('credit_reports')
  .select('*')
  .eq('user_id', session.user.id)
  .order('report_date', { ascending: false })

// Fetch credit accounts for the latest report
const latestReport = creditReports?.[0]
const { data: creditAccounts } = await supabase
  .from('credit_accounts')
  .select('*')
  .eq('credit_report_id', latestReport?.id)
  .order('date_opened', { ascending: false })

// Fetch credit inquiries
const { data: creditInquiries } = await supabase
  .from('credit_inquiries')
  .select('*')
  .eq('credit_report_id', latestReport?.id)
  .order('inquiry_date', { ascending: false })

const getAccountIcon = (accountType: string) => {
  switch (accountType) {
    case 'credit_card':
      return CreditCard
    case 'mortgage':
      return Building
    case 'auto_loan':
      return Car
    case 'student_loan':
      return GraduationCap
    default:
      return DollarSign
  }
}

const getAccountTypeLabel = (accountType: string) => {
  return accountType.split('_').map(word =>
    word.charAt(0).toUpperCase() + word.slice(1)
  ).join(' ')
}

return (
  <div className="container mx-auto px-4 py-8">
    <div className="flex items-center justify-between mb-8">
      <div>
        <h1 className="text-3xl font-bold text-gray-900">Credit Reports</h1>
        <p className="text-gray-600 mt-2">
          View and manage your credit report data
```

```
          </p>
        </div>
        <Button>
          <Plus className="h-4 w-4 mr-2" />
          Import New Report
        </Button>
      </div>

      {latestReport ? (
        <div className="space-y-6">
          {/* Credit Score Overview */}
          <div className="grid grid-cols-1 lg:grid-cols-4 gap-6">
            <div className="lg:col-span-1">
              <CreditScoreDisplay
                score={latestReport.credit_score || 0}
                scoreModel={latestReport.score_model}
                reportDate={latestReport.report_date}
              />
            </div>
            <div className="lg:col-span-3">
              <Card>
                <CardHeader>
                  <CardTitle>Report Summary</CardTitle>
                </CardHeader>
                <CardContent>
                  <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
                    <div className="text-center">
                      <div className="text-2xl font-bold text-blue-600">
                        {creditAccounts?.length || 0}
                      </div>
                      <div className="text-sm text-gray-600">Total Accounts</div>
                    </div>
                    <div className="text-center">
                      <div className="text-2xl font-bold text-red-600">
                        {creditAccounts?.filter(acc => acc.is_negative).length || 0}
                      </div>
                      <div className="text-sm text-gray-600">Negative Items</div>
                    </div>
                    <div className="text-center">
                      <div className="text-2xl font-bold text-purple-600">
                        {creditInquiries?.length || 0}
                      </div>
                      <div className="text-sm text-gray-600">Inquiries</div>
                    </div>
                  </div>
                </CardContent>
              </Card>
            </div>
          </div>

          {/* Credit Accounts */}
          <Card>
```

```
        <CardHeader>
          <CardTitle className="flex items-center justify-between">
            Credit Accounts
            <Badge variant="secondary">
              {creditAccounts?.length || 0} accounts
            </Badge>
          </CardTitle>
        </CardHeader>
        <CardContent>
          {creditAccounts && creditAccounts.length > 0 ? (
            <div className="space-y-4">
              {creditAccounts.map((account) => {
                const Icon = getAccountIcon(account.account_type)
                return (
                  <div key={account.id} className="flex items-center justify-between
p-4 border rounded-lg">
                    <div className="flex items-center space-x-4">
                      <Icon className="h-8 w-8 text-gray-600" />
                      <div>
                        <div className="font-medium">{account.account_name}</div>
                        <div className="text-sm text-gray-600">
                          {account.creditor_name} •
{getAccountTypeLabel(account.account_type)}
                        </div>
                        <div className="flex items-center space-x-2 mt-1">
                          {account.is_negative ? (
                            <Badge variant="destructive" className="flex items-center
space-x-1">
                              <AlertTriangle className="h-3 w-3" />
                              <span>Negative</span>
                            </Badge>
                          ) : (
                            <Badge variant="default" className="flex items-center space-
x-1">
                              <CheckCircle className="h-3 w-3" />
                              <span>Positive</span>
                            </Badge>
                          )}
                          <Badge variant="outline">
                            {account.account_status}
                          </Badge>
                        </div>
                      </div>
                    </div>
                    <div className="text-right">
                      <div className="font-medium">
                        ${account.balance?.toLocaleString() || '0'}
                      </div>
                      {account.credit_limit && (
                        <div className="text-sm text-gray-600">
                          Limit: ${account.credit_limit.toLocaleString()}
                        </div>
```

```
                  )}
                  <div className="text-xs text-gray-500 mt-1">
                   Opened: {account.date_opened ?
                    new Date(account.date_opened).toLocaleDateString() :
                    'Unknown'
                   }
                  </div>
                 </div>
                </div>
               )
              })}
             </div>
            ) : (
             <div className="text-center py-8">
              <CreditCard className="h-12 w-12 text-gray-400 mx-auto mb-4" />
              <p className="text-gray-600">No credit accounts found</p>
             </div>
            )}
           </CardContent>
          </Card>

          {/* Credit Inquiries */}
          <Card>
           <CardHeader>
            <CardTitle className="flex items-center justify-between">
             Credit Inquiries
             <Badge variant="secondary">
              {creditInquiries?.length || 0} inquiries
             </Badge>
            </CardTitle>
           </CardHeader>
           <CardContent>
            {creditInquiries && creditInquiries.length > 0 ? (
             <div className="space-y-4">
              {creditInquiries.map((inquiry) => (
               <div key={inquiry.id} className="flex items-center justify-between p-4
border rounded-lg">
                <div>
                 <div className="font-medium">{inquiry.creditor_name}</div>
                 <div className="text-sm text-gray-600">
                  {inquiry.purpose && `Purpose: ${inquiry.purpose}`}
                 </div>
                 <div className="mt-1">
                  <Badge variant={inquiry.inquiry_type === 'hard' ? 'destructive' :
'secondary'}>
                   {inquiry.inquiry_type.toUpperCase()} Inquiry
                  </Badge>
                 </div>
                </div>
                <div className="text-sm text-gray-500">
                 {new Date(inquiry.inquiry_date).toLocaleDateString()}
                </div>
```

```
                </div>
              ))}
            </div>
          ) : (
            <div className="text-center py-8">
              <FileText className="h-12 w-12 text-gray-400 mx-auto mb-4" />
              <p className="text-gray-600">No credit inquiries found</p>
            </div>
          )}
        </CardContent>
      </Card>
    </div>
  ) : (
    <Card>
      <CardContent className="text-center py-12">
        <FileText className="h-16 w-16 text-gray-400 mx-auto mb-6" />
        <h2 className="text-xl font-semibold text-gray-900 mb-2">
          No Credit Reports
        </h2>
        <p className="text-gray-600 mb-6">
          Import your first credit report to get started with credit repair
        </p>
        <Button size="lg">
          <Plus className="h-5 w-5 mr-2" />
          Import Credit Report
        </Button>
      </CardContent>
    </Card>
  )}
 </div>
 )
}
```

---

**Author:** Manus AI

**Date:** July 31, 2025

**Version:** 1.0 Development Guide

**Dispute Management Interface:**

```
// src/app/(client)/disputes/page.tsx
import { createServerClient } from '@/lib/supabase/server'
import { redirect } from 'next/navigation'
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { Button } from '@/components/ui/button'
import { DisputeStatusBadge } from '@/components/shared/DisputeStatusBadge'
import { DisputeItemForm } from '@/components/client/DisputeItemForm'
```

```
import {
  Plus,
  FileText,
  Download,
  Eye,
  MessageSquare
} from 'lucide-react'

export default async function DisputesPage() {
  const supabase = createServerClient()

  const { data: { session } } = await supabase.auth.getSession()
  if (!session) {
    redirect('/login')
  }

  // Fetch all dispute items for the user
  const { data: disputeItems } = await supabase
    .from('dispute_items')
    .select(`
      *,
      credit_accounts(*),
      credit_inquiries(*),
      public_records(*),
      dispute_letters(*)
    `)
    .eq('user_id', session.user.id)
    .order('created_at', { ascending: false })

  // Fetch available items for dispute (negative items not already disputed)
  const { data: availableItems } = await supabase
    .from('credit_accounts')
    .select(`
      *,
      credit_reports!inner(user_id)
    `)
    .eq('credit_reports.user_id', session.user.id)
    .eq('is_negative', true)
    .not('id', 'in', `(${disputeItems?.map(d =>
d.credit_account_id).filter(Boolean).join(',') || 'null'})`)

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="flex items-center justify-between mb-8">
        <div>
          <h1 className="text-3xl font-bold text-gray-900">Disputes</h1>
          <p className="text-gray-600 mt-2">
            Manage your credit report disputes and track their progress
          </p>
        </div>
        <DisputeItemForm availableItems={availableItems || []}>
          <Button>
```

```
          <Plus className="h-4 w-4 mr-2" />
          New Dispute
        </Button>
      </DisputeItemForm>
    </div>

    {disputeItems && disputeItems.length > 0 ? (
      <div className="space-y-6">
        {disputeItems.map((dispute) => (
          <Card key={dispute.id}>
            <CardHeader>
              <div className="flex items-center justify-between">
                <div>
                  <CardTitle className="text-lg">
                    {dispute.credit_accounts?.account_name ||
                     dispute.credit_inquiries?.creditor_name ||
                     'Personal Information Dispute'}
                  </CardTitle>
                  <p className="text-gray-600 mt-1">
                    {dispute.dispute_reason}
                  </p>
                </div>
                <DisputeStatusBadge status={dispute.status} />
              </div>
            </CardHeader>
            <CardContent>
              <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
                <div>
                  <h4 className="font-medium mb-2">Dispute Details</h4>
                  <div className="space-y-2 text-sm">
                    <div>
                      <span className="text-gray-600">Type:</span>{' '}
                      <span className="capitalize">{dispute.dispute_type.replace('_', ' ')}</
span>
                    </div>
                    <div>
                      <span className="text-gray-600">Priority:</span>{' '}
                      <span className="capitalize">{dispute.priority}</span>
                    </div>
                    <div>
                      <span className="text-gray-600">Created:</span>{' '}
                      {new Date(dispute.created_at).toLocaleDateString()}
                    </div>
                    {dispute.resolved_at && (
                      <div>
                        <span className="text-gray-600">Resolved:</span>{' '}
                        {new Date(dispute.resolved_at).toLocaleDateString()}
                      </div>
                    )}
                  </div>

                  {dispute.client_notes && (
```

```
              <div className="mt-4">
                <h5 className="font-medium text-sm mb-1">Your Notes</h5>
                <p className="text-sm text-gray-600 bg-gray-50 p-2 rounded">
                 {dispute.client_notes}
                </p>
              </div>
             )}
            </div>

            <div>
             <h4 className="font-medium mb-2">Actions & Documents</h4>
             <div className="space-y-2">
              {dispute.dispute_letters && dispute.dispute_letters.length > 0 && (
               <div>
                <h5 className="text-sm font-medium mb-1">Dispute Letters</h5>
                {dispute.dispute_letters.map((letter: any) => (
                 <div key={letter.id} className="flex items-center justify-between
p-2 bg-gray-50 rounded">
                  <div>
                   <div className="text-sm font-medium">
                    Letter #{letter.id.slice(0, 8)}
                   </div>
                   <div className="text-xs text-gray-600">
                    {letter.status === 'generated' ? 'Ready to send' :
                     letter.status === 'sent' ? `Sent ${letter.sent_date}` :
                     'In progress'}
                   </div>
                  </div>
                  <div className="flex space-x-1">
                   <Button size="sm" variant="outline">
                    <Eye className="h-3 w-3" />
                   </Button>
                   <Button size="sm" variant="outline">
                    <Download className="h-3 w-3" />
                   </Button>
                  </div>
                 </div>
                ))}
               </div>
              )}

              <Button size="sm" variant="outline" className="w-full">
               <MessageSquare className="h-4 w-4 mr-2" />
               Contact Support
              </Button>
             </div>
            </div>
           </div>
          </CardContent>
         </Card>
        ))}
       </div>
```

```
      ) : (
        <Card>
          <CardContent className="text-center py-12">
            <FileText className="h-16 w-16 text-gray-400 mx-auto mb-6" />
            <h2 className="text-xl font-semibold text-gray-900 mb-2">
              No Disputes Yet
            </h2>
            <p className="text-gray-600 mb-6">
              Start disputing negative items on your credit report to improve your score
            </p>
            {availableItems && availableItems.length > 0 ? (
              <DisputeItemForm availableItems={availableItems}>
                <Button size="lg">
                  <Plus className="h-5 w-5 mr-2" />
                  Create Your First Dispute
                </Button>
              </DisputeItemForm>
            ) : (
              <p className="text-gray-500">
                Import a credit report first to identify items for dispute
              </p>
            )}
          </CardContent>
        </Card>
      )}
    </div>
  )
}

// src/components/client/DisputeItemForm.tsx
'use client'

import { useState } from 'react'
import { useRouter } from 'next/navigation'
import { createClient } from '@/lib/supabase/client'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Label } from '@/components/ui/label'
import { Textarea } from '@/components/ui/textarea'
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from '@/
components/ui/select'
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogTrigger } from
'@/components/ui/dialog'
import { Alert, AlertDescription } from '@/components/ui/alert'
import { Loader2 } from 'lucide-react'
import { useForm } from 'react-hook-form'
import { zodResolver } from '@hookform/resolvers/zod'
import { z } from 'zod'

const disputeSchema = z.object({
  creditAccountId: z.string().uuid().optional(),
  disputeReason: z.string().min(1, 'Dispute reason is required'),
```

```
  clientNotes: z.string().max(2000, 'Notes must be less than 2000
characters').optional(),
  priority: z.enum(['low', 'medium', 'high']).default('medium')
})

type DisputeFormData = z.infer<typeof disputeSchema>

interface DisputeItemFormProps {
  children: React.ReactNode
  availableItems: any[]
}

export function DisputeItemForm({ children, availableItems }:
DisputeItemFormProps) {
  const [isOpen, setIsOpen] = useState(false)
  const [isLoading, setIsLoading] = useState(false)
  const [error, setError] = useState<string | null>(null)
  const router = useRouter()
  const supabase = createClient()

  const {
    register,
    handleSubmit,
    formState: { errors },
    setValue,
    watch,
    reset
  } = useForm<DisputeFormData>({
    resolver: zodResolver(disputeSchema),
    defaultValues: {
      priority: 'medium'
    }
  })

  const selectedAccountId = watch('creditAccountId')
  const selectedAccount = availableItems.find(item => item.id ===
selectedAccountId)

  const disputeReasons = [
    'Not my account',
    'Account paid in full',
    'Incorrect balance',
    'Incorrect payment history',
    'Account closed by consumer',
    'Duplicate account',
    'Identity theft',
    'Other'
  ]

  const onSubmit = async (data: DisputeFormData) => {
    setIsLoading(true)
    setError(null)
```

```
  try {
    const { data: { session } } = await supabase.auth.getSession()
    if (!session) throw new Error('Not authenticated')

    const { error } = await supabase
      .from('dispute_items')
      .insert({
        user_id: session.user.id,
        credit_account_id: data.creditAccountId,
        dispute_type: 'account',
        dispute_reason: data.disputeReason,
        client_notes: data.clientNotes,
        priority: data.priority,
        status: 'pending_review'
      })

    if (error) throw error

    setIsOpen(false)
    reset()
    router.refresh()
  } catch (error: any) {
    setError(error.message || 'Failed to create dispute')
  } finally {
    setIsLoading(false)
  }
}

return (
  <Dialog open={isOpen} onOpenChange={setIsOpen}>
    <DialogTrigger asChild>
      {children}
    </DialogTrigger>
    <DialogContent className="max-w-2xl">
      <DialogHeader>
        <DialogTitle>Create New Dispute</DialogTitle>
      </DialogHeader>

      {error && (
        <Alert className="border-red-200 bg-red-50">
          <AlertDescription className="text-red-800">{error}</AlertDescription>
        </Alert>
      )}

      <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
        <div>
          <Label htmlFor="creditAccountId">Select Account to Dispute</Label>
          <Select onValueChange={(value) => setValue('creditAccountId', value)}>
            <SelectTrigger>
              <SelectValue placeholder="Choose an account" />
            </SelectTrigger>
```

```
            <SelectContent>
              {availableItems.map((item) => (
                <SelectItem key={item.id} value={item.id}>
                  <div>
                    <div className="font-medium">{item.account_name}</div>
                    <div className="text-sm text-gray-600">
                      {item.creditor_name} • {item.negative_reason}
                    </div>
                  </div>
                </SelectItem>
              ))}
            </SelectContent>
          </Select>
          {errors.creditAccountId && (
            <p className="text-red-500 text-sm
mt-1">{errors.creditAccountId.message}</p>
          )}
        </div>

        {selectedAccount && (
          <div className="p-4 bg-gray-50 rounded-lg">
            <h4 className="font-medium mb-2">Account Details</h4>
            <div className="grid grid-cols-2 gap-4 text-sm">
              <div>
                <span className="text-gray-600">Account:</span>
{selectedAccount.account_name}
              </div>
              <div>
                <span className="text-gray-600">Creditor:</span>
{selectedAccount.creditor_name}
              </div>
              <div>
                <span className="text-gray-600">Balance:</span> $
{selectedAccount.balance?.toLocaleString() || '0'}
              </div>
              <div>
                <span className="text-gray-600">Status:</span>
{selectedAccount.account_status}
              </div>
            </div>
          </div>
        )}

        <div>
          <Label htmlFor="disputeReason">Dispute Reason</Label>
          <Select onValueChange={(value) => setValue('disputeReason', value)}>
            <SelectTrigger>
              <SelectValue placeholder="Select a reason" />
            </SelectTrigger>
            <SelectContent>
              {disputeReasons.map((reason) => (
                <SelectItem key={reason} value={reason}>
```

```
                {reason}
              </SelectItem>
            ))}
          </SelectContent>
        </Select>
        {errors.disputeReason && (
          <p className="text-red-500 text-sm
mt-1">{errors.disputeReason.message}</p>
        )}
      </div>

      <div>
        <Label htmlFor="priority">Priority</Label>
        <Select onValueChange={(value) => setValue('priority', value as 'low' |
'medium' | 'high')}>
          <SelectTrigger>
            <SelectValue />
          </SelectTrigger>
          <SelectContent>
            <SelectItem value="low">Low</SelectItem>
            <SelectItem value="medium">Medium</SelectItem>
            <SelectItem value="high">High</SelectItem>
          </SelectContent>
        </Select>
      </div>

      <div>
        <Label htmlFor="clientNotes">Additional Notes (Optional)</Label>
        <Textarea
          id="clientNotes"
          {...register('clientNotes')}
          placeholder="Provide any additional details that might help with your
dispute..."
          rows={4}
        />
        {errors.clientNotes && (
          <p className="text-red-500 text-sm mt-1">{errors.clientNotes.message}</
p>
        )}
      </div>

      <div className="flex justify-end space-x-3">
        <Button type="button" variant="outline" onClick={() => setIsOpen(false)}>
          Cancel
        </Button>
        <Button type="submit" disabled={isLoading}>
          {isLoading && <Loader2 className="mr-2 h-4 w-4 animate-spin" />}
          Create Dispute
        </Button>
      </div>
    </form>
  </DialogContent>
```

```
    </Dialog>
  )
}
```

### 5.3. Admin Portal Implementation

The admin portal provides comprehensive tools for credit repair professionals to manage clients, disputes, and administrative tasks. This section details the implementation of advanced admin features.

**Admin Dashboard:**

```
// src/app/(admin)/admin/dashboard/page.tsx
import { createServerClient } from '@/lib/supabase/server'
import { redirect } from 'next/navigation'
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { Button } from '@/components/ui/button'
import { Badge } from '@/components/ui/badge'
import { DisputeStatusBadge } from '@/components/shared/DisputeStatusBadge'
import {
  Users,
  FileText,
  TrendingUp,
  Clock,
  CheckCircle,
  AlertTriangle,
  BarChart3,
  Calendar
} from 'lucide-react'

export default async function AdminDashboard() {
  const supabase = createServerClient()

  const { data: { session } } = await supabase.auth.getSession()
  if (!session) {
    redirect('/login')
  }

  // Verify admin access
  const { data: profile } = await supabase
    .from('profiles')
    .select('user_type')
    .eq('id', session.user.id)
    .single()

  if (!profile || !['admin', 'super_admin'].includes(profile.user_type)) {
    redirect('/dashboard')
  }
```

```
  // Fetch dashboard statistics
  const [
    { count: totalClients },
    { count: activeDisputes },
    { count: pendingReview },
    { count: lettersGenerated },
    { count: resolvedDisputes }
  ] = await Promise.all([
    supabase.from('profiles').select('*', { count: 'exact', head: true }).eq('user_type',
'client'),
    supabase.from('dispute_items').select('*', { count: 'exact', head:
true }).not('status', 'in', '(resolved,unresolved)'),
    supabase.from('dispute_items').select('*', { count: 'exact', head:
true }).eq('status', 'pending_review'),
    supabase.from('dispute_letters').select('*', { count: 'exact', head:
true }).eq('status', 'generated'),
    supabase.from('dispute_items').select('*', { count: 'exact', head:
true }).eq('status', 'resolved')
  ])

  // Fetch recent activities
  const { data: recentDisputes } = await supabase
    .from('dispute_items')
    .select(`
      *,
      profiles!inner(first_name, last_name),
      credit_accounts(account_name, creditor_name)
    `)
    .order('created_at', { ascending: false })
    .limit(10)

  // Fetch pending tasks
  const { data: pendingTasks } = await supabase
    .from('admin_tasks')
    .select(`
      *,
      dispute_items(
        *,
        profiles!inner(first_name, last_name)
      )
    `)
    .eq('status', 'pending')
    .order('due_date', { ascending: true })
    .limit(10)

  // Fetch clients needing attention
  const { data: clientsNeedingAttention } = await supabase
    .from('profiles')
    .select(`
      *,
      dispute_items!inner(status)
    `)
```

```
    .eq('user_type', 'client')
    .eq('dispute_items.status', 'pending_review')
    .limit(5)

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="mb-8">
        <h1 className="text-3xl font-bold text-gray-900">Admin Dashboard</h1>
        <p className="text-gray-600 mt-2">
          Overview of client activities and system performance
        </p>
      </div>

      {/* Key Metrics */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-5 gap-6 mb-8">
        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-sm font-medium text-gray-600">
              Total Clients
            </CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex items-center">
              <Users className="h-8 w-8 text-blue-500 mr-3" />
              <div>
                <div className="text-2xl font-bold text-blue-600">
                  {totalClients || 0}
                </div>
                <div className="text-sm text-gray-600">Active accounts</div>
              </div>
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-sm font-medium text-gray-600">
              Active Disputes
            </CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex items-center">
              <TrendingUp className="h-8 w-8 text-purple-500 mr-3" />
              <div>
                <div className="text-2xl font-bold text-purple-600">
                  {activeDisputes || 0}
                </div>
                <div className="text-sm text-gray-600">In progress</div>
              </div>
            </div>
          </CardContent>
        </Card>
```

```
<Card>
  <CardHeader className="pb-2">
    <CardTitle className="text-sm font-medium text-gray-600">
      Pending Review
    </CardTitle>
  </CardHeader>
  <CardContent>
    <div className="flex items-center">
      <Clock className="h-8 w-8 text-yellow-500 mr-3" />
      <div>
        <div className="text-2xl font-bold text-yellow-600">
          {pendingReview || 0}
        </div>
        <div className="text-sm text-gray-600">Awaiting action</div>
      </div>
    </div>
  </CardContent>
</Card>

<Card>
  <CardHeader className="pb-2">
    <CardTitle className="text-sm font-medium text-gray-600">
      Letters Ready
    </CardTitle>
  </CardHeader>
  <CardContent>
    <div className="flex items-center">
      <FileText className="h-8 w-8 text-green-500 mr-3" />
      <div>
        <div className="text-2xl font-bold text-green-600">
          {lettersGenerated || 0}
        </div>
        <div className="text-sm text-gray-600">To be sent</div>
      </div>
    </div>
  </CardContent>
</Card>

<Card>
  <CardHeader className="pb-2">
    <CardTitle className="text-sm font-medium text-gray-600">
      Resolved
    </CardTitle>
  </CardHeader>
  <CardContent>
    <div className="flex items-center">
      <CheckCircle className="h-8 w-8 text-emerald-500 mr-3" />
      <div>
        <div className="text-2xl font-bold text-emerald-600">
          {resolvedDisputes || 0}
        </div>
```

```
            <div className="text-sm text-gray-600">This month</div>
          </div>
        </div>
      </CardContent>
    </Card>
  </div>

  <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-8">
    {/* Recent Disputes */}
    <Card>
      <CardHeader>
        <CardTitle className="flex items-center justify-between">
          Recent Disputes
          <Button size="sm" variant="outline">
            View All
          </Button>
        </CardTitle>
      </CardHeader>
      <CardContent>
        {recentDisputes && recentDisputes.length > 0 ? (
          <div className="space-y-4">
            {recentDisputes.slice(0, 5).map((dispute) => (
              <div key={dispute.id} className="flex items-center justify-between p-3
border rounded-lg">
                <div>
                  <div className="font-medium">
                    {dispute.profiles.first_name} {dispute.profiles.last_name}
                  </div>
                  <div className="text-sm text-gray-600">
                    {dispute.credit_accounts?.account_name || dispute.dispute_reason}
                  </div>
                  <div className="mt-1">
                    <DisputeStatusBadge status={dispute.status} />
                  </div>
                </div>
                <div className="text-sm text-gray-500">
                  {new Date(dispute.created_at).toLocaleDateString()}
                </div>
              </div>
            ))}
          </div>
        ) : (
          <div className="text-center py-8">
            <FileText className="h-12 w-12 text-gray-400 mx-auto mb-4" />
            <p className="text-gray-600">No recent disputes</p>
          </div>
        )}
      </CardContent>
    </Card>

    {/* Pending Tasks */}
    <Card>
```

```
        <CardHeader>
          <CardTitle className="flex items-center justify-between">
            Pending Tasks
            <Button size="sm" variant="outline">
              View All
            </Button>
          </CardTitle>
        </CardHeader>
        <CardContent>
          {pendingTasks && pendingTasks.length > 0 ? (
            <div className="space-y-4">
              {pendingTasks.slice(0, 5).map((task) => (
                <div key={task.id} className="flex items-center justify-between p-3
border rounded-lg">
                  <div>
                    <div className="font-medium">{task.title}</div>
                    <div className="text-sm text-gray-600">
                      {task.dispute_items?.profiles?.first_name}
{task.dispute_items?.profiles?.last_name}
                    </div>
                    <div className="flex items-center space-x-2 mt-1">
                      <Badge variant={task.priority === 'high' ? 'destructive' :
                                      task.priority === 'medium' ? 'default' : 'secondary'}>
                        {task.priority}
                      </Badge>
                      <Badge variant="outline">
                        {task.task_type.replace('_', ' ')}
                      </Badge>
                    </div>
                  </div>
                  <div className="text-sm text-gray-500">
                    {task.due_date ? new Date(task.due_date).toLocaleDateString() : 'No
due date'}
                  </div>
                </div>
              ))}
            </div>
          ) : (
            <div className="text-center py-8">
              <Calendar className="h-12 w-12 text-gray-400 mx-auto mb-4" />
              <p className="text-gray-600">No pending tasks</p>
            </div>
          )}
        </CardContent>
      </Card>
    </div>

    {/* Clients Needing Attention */}
    <Card>
      <CardHeader>
        <CardTitle className="flex items-center justify-between">
          Clients Needing Attention
```

```
          <Button size="sm" variant="outline">
            View All Clients
          </Button>
        </CardTitle>
      </CardHeader>
      <CardContent>
        {clientsNeedingAttention && clientsNeedingAttention.length > 0 ? (
          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
            {clientsNeedingAttention.map((client) => (
              <div key={client.id} className="p-4 border rounded-lg">
                <div className="flex items-center justify-between mb-2">
                  <div className="font-medium">
                    {client.first_name} {client.last_name}
                  </div>
                  <Badge variant="destructive">
                    <AlertTriangle className="h-3 w-3 mr-1" />
                    Action Needed
                  </Badge>
                </div>
                <div className="text-sm text-gray-600 mb-3">
                  {client.email}
                </div>
                <Button size="sm" className="w-full">
                  Review Disputes
                </Button>
              </div>
            ))}
          </div>
        ) : (
          <div className="text-center py-8">
            <Users className="h-12 w-12 text-gray-400 mx-auto mb-4" />
            <p className="text-gray-600">All clients are up to date</p>
          </div>
        )}
      </CardContent>
    </Card>
  </div>
 )
}
```

**Client Management Interface:**

```
// src/app/(admin)/admin/clients/page.tsx
import { createServerClient } from '@/lib/supabase/server'
import { redirect } from 'next/navigation'
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card'
import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Badge } from '@/components/ui/badge'
import { ClientsTable } from '@/components/admin/ClientsTable'
```

```tsx
import { ClientDetailsModal } from '@/components/admin/ClientDetailsModal'
import {
  Search,
  Filter,
  Download,
  Users,
  UserPlus
} from 'lucide-react'

export default async function ClientsPage() {
  const supabase = createServerClient()

  const { data: { session } } = await supabase.auth.getSession()
  if (!session) {
    redirect('/login')
  }

  // Verify admin access
  const { data: profile } = await supabase
    .from('profiles')
    .select('user_type')
    .eq('id', session.user.id)
    .single()

  if (!profile || !['admin', 'super_admin'].includes(profile.user_type)) {
    redirect('/dashboard')
  }

  // Fetch all clients with their dispute counts
  const { data: clients } = await supabase
    .from('profiles')
    .select(`
      *,
      dispute_items(count),
      credit_reports(count)
    `)
    .eq('user_type', 'client')
    .order('created_at', { ascending: false })

  // Fetch client statistics
  const [
    { count: totalClients },
    { count: activeClients },
    { count: onHoldClients },
    { count: completedClients }
  ] = await Promise.all([
    supabase.from('profiles').select('*', { count: 'exact', head: true }).eq('user_type',
'client'),
    supabase.from('profiles').select('*', { count: 'exact', head: true }).eq('user_type',
'client').eq('client_status', 'active'),
    supabase.from('profiles').select('*', { count: 'exact', head: true }).eq('user_type',
'client').eq('client_status', 'on_hold'),
```

```
    supabase.from('profiles').select('*', { count: 'exact', head: true }).eq('user_type',
'client').eq('client_status', 'completed')
  ])

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="flex items-center justify-between mb-8">
        <div>
          <h1 className="text-3xl font-bold text-gray-900">Client Management</h1>
          <p className="text-gray-600 mt-2">
            Manage all client accounts and their credit repair progress
          </p>
        </div>
        <div className="flex space-x-3">
          <Button variant="outline">
            <Download className="h-4 w-4 mr-2" />
            Export
          </Button>
          <Button>
            <UserPlus className="h-4 w-4 mr-2" />
            Add Client
          </Button>
        </div>
      </div>

      {/* Client Statistics */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-6 mb-8">
        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-sm font-medium text-gray-600">
              Total Clients
            </CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex items-center">
              <Users className="h-8 w-8 text-blue-500 mr-3" />
              <div>
                <div className="text-2xl font-bold text-blue-600">
                  {totalClients || 0}
                </div>
                <div className="text-sm text-gray-600">All time</div>
              </div>
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardHeader className="pb-2">
            <CardTitle className="text-sm font-medium text-gray-600">
              Active
            </CardTitle>
          </CardHeader>
```

```
        <CardContent>
          <div className="text-2xl font-bold text-green-600">
            {activeClients || 0}
          </div>
          <div className="text-sm text-gray-600">Currently active</div>
        </CardContent>
      </Card>

      <Card>
        <CardHeader className="pb-2">
          <CardTitle className="text-sm font-medium text-gray-600">
            On Hold
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="text-2xl font-bold text-yellow-600">
            {onHoldClients || 0}
          </div>
          <div className="text-sm text-gray-600">Temporarily paused</div>
        </CardContent>
      </Card>

      <Card>
        <CardHeader className="pb-2">
          <CardTitle className="text-sm font-medium text-gray-600">
            Completed
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="text-2xl font-bold text-emerald-600">
            {completedClients || 0}
          </div>
          <div className="text-sm text-gray-600">Successfully completed</div>
        </CardContent>
      </Card>
    </div>

    {/* Search and Filters */}
    <Card className="mb-6">
      <CardContent className="pt-6">
        <div className="flex items-center space-x-4">
          <div className="flex-1">
            <div className="relative">
              <Search className="absolute left-3 top-1/2 transform -translate-y-1/2
text-gray-400 h-4 w-4" />
              <Input
                placeholder="Search clients by name, email, or phone..."
                className="pl-10"
              />
            </div>
          </div>
          <Button variant="outline">
```

```
            <Filter className="h-4 w-4 mr-2" />
            Filters
          </Button>
        </div>
      </CardContent>
    </Card>

    {/* Clients Table */}
    <Card>
      <CardHeader>
        <CardTitle>All Clients</CardTitle>
      </CardHeader>
      <CardContent>
        <ClientsTable clients={clients || []} />
      </CardContent>
    </Card>
  </div>
 )
}

// src/components/admin/ClientsTable.tsx
'use client'

import { useState } from 'react'
import { Badge } from '@/components/ui/badge'
import { Button } from '@/components/ui/button'
import {
 Table,
 TableBody,
 TableCell,
 TableHead,
 TableHeader,
 TableRow,
} from '@/components/ui/table'
import { ClientDetailsModal } from './ClientDetailsModal'
import {
 Eye,
 Edit,
 MoreHorizontal,
 Mail,
 Phone
} from 'lucide-react'

interface Client {
 id: string
 first_name: string
 last_name: string
 email: string
 phone_number?: string
 client_status: string
 created_at: string
 last_login_at?: string
```

```
  dispute_items: { count: number }[]
  credit_reports: { count: number }[]
}

interface ClientsTableProps {
  clients: Client[]
}

export function ClientsTable({ clients }: ClientsTableProps) {
  const [selectedClient, setSelectedClient] = useState<Client | null>(null)

  const getStatusBadge = (status: string) => {
    switch (status) {
      case 'active':
        return <Badge variant="default">Active</Badge>
      case 'on_hold':
        return <Badge variant="secondary">On Hold</Badge>
      case 'completed':
        return <Badge variant="outline">Completed</Badge>
      case 'suspended':
        return <Badge variant="destructive">Suspended</Badge>
      default:
        return <Badge variant="secondary">{status}</Badge>
    }
  }

  return (
    <>
      <Table>
        <TableHeader>
          <TableRow>
            <TableHead>Client</TableHead>
            <TableHead>Contact</TableHead>
            <TableHead>Status</TableHead>
            <TableHead>Disputes</TableHead>
            <TableHead>Reports</TableHead>
            <TableHead>Joined</TableHead>
            <TableHead>Last Login</TableHead>
            <TableHead>Actions</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {clients.map((client) => (
            <TableRow key={client.id}>
              <TableCell>
                <div>
                  <div className="font-medium">
                    {client.first_name} {client.last_name}
                  </div>
                  <div className="text-sm text-gray-600">
                    ID: {client.id.slice(0, 8)}...
                  </div>
```

```
          </div>
        </TableCell>
        <TableCell>
          <div className="space-y-1">
            <div className="flex items-center text-sm">
              <Mail className="h-3 w-3 mr-1 text-gray-400" />
              {client.email}
            </div>
            {client.phone_number && (
              <div className="flex items-center text-sm">
                <Phone className="h-3 w-3 mr-1 text-gray-400" />
                {client.phone_number}
              </div>
            )}
          </div>
        </TableCell>
        <TableCell>
          {getStatusBadge(client.client_status)}
        </TableCell>
        <TableCell>
          <Badge variant="outline">
            {client.dispute_items?.[0]?.count || 0}
          </Badge>
        </TableCell>
        <TableCell>
          <Badge variant="outline">
            {client.credit_reports?.[0]?.count || 0}
          </Badge>
        </TableCell>
        <TableCell>
          {new Date(client.created_at).toLocaleDateString()}
        </TableCell>
        <TableCell>
          {client.last_login_at
            ? new Date(client.last_login_at).toLocaleDateString()
            : 'Never'
          }
        </TableCell>
        <TableCell>
          <div className="flex space-x-2">
            <Button
              size="sm"
              variant="outline"
              onClick={() => setSelectedClient(client)}
            >
              <Eye className="h-3 w-3" />
            </Button>
            <Button size="sm" variant="outline">
              <Edit className="h-3 w-3" />
            </Button>
            <Button size="sm" variant="outline">
              <MoreHorizontal className="h-3 w-3" />
```

```
                </Button>
              </div>
            </TableCell>
          </TableRow>
        ))}
      </TableBody>
    </Table>

    {selectedClient && (
      <ClientDetailsModal
        client={selectedClient}
        isOpen={!!selectedClient}
        onClose={() => setSelectedClient(null)}
      />
    )}
  </>
 )
}
```

# 6. Backend API Development

## 6.1. Supabase Edge Functions

Supabase Edge Functions provide the serverless compute layer for SAINTRIX Beta, handling complex business logic, third-party integrations, and secure operations. This section details the implementation of critical Edge Functions.

**Credit Report Import Function:**

```
// supabase/functions/import-credit-report/index.ts
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"
import { createClient } from 'https://esm.sh/@supabase/supabase-js@2'

interface CreditReportRequest {
  userId: string
  provider: 'plaid' | 'finicity'
  accessToken: string
}

interface PlaidCreditResponse {
  accounts: Array<{
    account_id: string
    name: string
    type: string
    subtype: string
    balances: {
      current: number
      limit?: number
```

```
    }
  }>
  credit_details: {
    accounts: Array<{
      account_id: string
      payment_history: Array<{
        date: string
        status: string
      }>
      is_negative: boolean
      negative_reasons?: string[]
    }>
  }
  credit_score: {
    score: number
    model: string
    date: string
  }
}

serve(async (req) => {
  try {
    // Verify request method
    if (req.method !== 'POST') {
      return new Response('Method not allowed', { status: 405 })
    }

    const { userId, provider, accessToken }: CreditReportRequest = await req.json()

    // Initialize Supabase client with service role key
    const supabase = createClient(
      Deno.env.get('SUPABASE_URL') ?? '',
      Deno.env.get('SUPABASE_SERVICE_ROLE_KEY') ?? ''
    )

    // Verify user authentication
    const authHeader = req.headers.get('Authorization')
    if (!authHeader) {
      return new Response('Authorization header required', { status: 401 })
    }

    const { data: { user }, error: authError } = await supabase.auth.getUser(
      authHeader.replace('Bearer ', '')
    )

    if (authError || !user || user.id !== userId) {
      return new Response('Unauthorized', { status: 401 })
    }

    // Call third-party credit API based on provider
    let creditData: PlaidCreditResponse
```

```
    if (provider === 'plaid') {
      creditData = await fetchPlaidCreditReport(accessToken)
    } else if (provider === 'finicity') {
      creditData = await fetchFinicityCreditReport(accessToken)
    } else {
      return new Response('Unsupported provider', { status: 400 })
    }

    // Create credit report record
    const { data: creditReport, error: insertError } = await supabase
      .from('credit_reports')
      .insert({
        user_id: userId,
        bureau_name: 'experian', // Default for demo
        report_date: new Date().toISOString().split('T')[0],
        credit_score: creditData.credit_score.score,
        score_model: creditData.credit_score.model,
        raw_data: creditData,
        import_status: 'processing'
      })
      .select()
      .single()

    if (insertError) {
      console.error('Error inserting credit report:', insertError)
      throw insertError
    }

    // Parse and store individual credit accounts
    await parseCreditAccounts(supabase, creditReport.id, creditData)

    // Update import status to completed
    await supabase
      .from('credit_reports')
      .update({ import_status: 'completed' })
      .eq('id', creditReport.id)

    return new Response(
      JSON.stringify({
        success: true,
        creditReportId: creditReport.id,
        accountsImported: creditData.accounts.length,
        creditScore: creditData.credit_score.score
      }),
      {
        headers: { "Content-Type": "application/json" },
        status: 200
      }
    )

  } catch (error) {
    console.error('Credit report import error:', error)
```

```
    // Update import status to failed if we have a report ID
    // This would require additional error handling logic

    return new Response(
      JSON.stringify({
        error: 'Failed to import credit report',
        details: error.message
      }),
      {
        status: 500,
        headers: { "Content-Type": "application/json" }
      }
    )
  }
})

async function fetchPlaidCreditReport(accessToken: string):
Promise<PlaidCreditResponse> {
  const plaidClientId = Deno.env.get('PLAID_CLIENT_ID')
  const plaidSecret = Deno.env.get('PLAID_SECRET')
  const plaidEnv = Deno.env.get('PLAID_ENV') || 'sandbox'

  const baseUrl = plaidEnv === 'production'
    ? 'https://production.plaid.com'
    : plaidEnv === 'development'
    ? 'https://development.plaid.com'
    : 'https://sandbox.plaid.com'

  // Fetch accounts
  const accountsResponse = await fetch(`${baseUrl}/accounts/get`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      client_id: plaidClientId,
      secret: plaidSecret,
      access_token: accessToken
    })
  })

  if (!accountsResponse.ok) {
    throw new Error(`Plaid accounts API error: ${accountsResponse.statusText}`)
  }

  const accountsData = await accountsResponse.json()

  // Fetch credit details (this is a simplified example)
  // In reality, you would need to use Plaid's Credit API endpoints
  const creditResponse = await fetch(`${baseUrl}/credit/get`, {
    method: 'POST',
```

```
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      client_id: plaidClientId,
      secret: plaidSecret,
      access_token: accessToken
    })
  })

  if (!creditResponse.ok) {
    throw new Error(`Plaid credit API error: ${creditResponse.statusText}`)
  }

  const creditData = await creditResponse.json()

  // Transform Plaid data to our standard format
  return {
    accounts: accountsData.accounts.map((account: any) => ({
      account_id: account.account_id,
      name: account.name,
      type: account.type,
      subtype: account.subtype,
      balances: {
        current: account.balances.current,
        limit: account.balances.limit
      }
    })),
    credit_details: {
      accounts: creditData.accounts?.map((account: any) => ({
        account_id: account.account_id,
        payment_history: account.payment_history || [],
        is_negative: account.is_negative || false,
        negative_reasons: account.negative_reasons || []
      })) || []
    },
    credit_score: {
      score: creditData.credit_score?.score || 0,
      model: creditData.credit_score?.model || 'FICO',
      date: new Date().toISOString()
    }
  }
}

async function fetchFinicityCreditReport(accessToken: string):
Promise<PlaidCreditResponse> {
  // Implement Finicity API integration
  // This is a placeholder implementation
  throw new Error('Finicity integration not yet implemented')
}

async function parseCreditAccounts(
```

```
  supabase: any,
  creditReportId: string,
  creditData: PlaidCreditResponse
) {
  const accountsToInsert = creditData.accounts.map(account => {
    const creditDetails = creditData.credit_details.accounts.find(
      detail => detail.account_id === account.account_id
    )

    return {
      credit_report_id: creditReportId,
      account_number_encrypted: account.account_id, // In production, encrypt this
      account_name: account.name,
      account_type: mapAccountType(account.type, account.subtype),
      creditor_name: extractCreditorName(account.name),
      account_status: 'open', // Default status
      balance: account.balances.current,
      credit_limit: account.balances.limit,
      payment_history: creditDetails?.payment_history || [],
      is_negative: creditDetails?.is_negative || false,
      negative_reason: creditDetails?.negative_reasons?.[0] || null
    }
  })

  const { error } = await supabase
    .from('credit_accounts')
    .insert(accountsToInsert)

  if (error) {
    console.error('Error inserting credit accounts:', error)
    throw error
  }
}

function mapAccountType(type: string, subtype: string): string {
  const typeMap: Record<string, string> = {
    'credit': 'credit_card',
    'loan': subtype === 'mortgage' ? 'mortgage' :
          subtype === 'auto' ? 'auto_loan' :
          subtype === 'student' ? 'student_loan' : 'personal_loan',
    'depository': 'other'
  }

  return typeMap[type] || 'other'
}

function extractCreditorName(accountName: string): string {
  // Simple extraction logic - in production, use more sophisticated parsing
  return accountName.split(' ')[0] || accountName
}
```

**Dispute Letter Generation Function:**

```
 // supabase/functions/generate-dispute-letter/index.ts
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"
import { createClient } from 'https://esm.sh/@supabase/supabase-js@2'

interface DisputeLetterRequest {
  disputeItemIds: string[]
  templateId?: string
  customizations?: {
    letterhead?: string
    signature?: string
    additionalText?: string
  }
}

interface LetterTemplate {
  id: string
  name: string
  template_content: string
  dispute_type: string
}

interface DisputeItem {
  id: string
  user_id: string
  dispute_reason: string
  client_notes?: string
  credit_accounts?: {
    account_name: string
    creditor_name: string
    account_number_encrypted: string
    balance: number
  }
  credit_inquiries?: {
    creditor_name: string
    inquiry_date: string
    inquiry_type: string
  }
  profiles: {
    first_name: string
    last_name: string
    address_line1: string
    address_line2?: string
    city: string
    state: string
    zip_code: string
  }
}

serve(async (req) => {
```

```
  try {
    if (req.method !== 'POST') {
      return new Response('Method not allowed', { status: 405 })
    }

    const { disputeItemIds, templateId, customizations }: DisputeLetterRequest =
await req.json()

    const supabase = createClient(
      Deno.env.get('SUPABASE_URL') ?? '',
      Deno.env.get('SUPABASE_SERVICE_ROLE_KEY') ?? ''
    )

    // Verify admin authentication
    const authHeader = req.headers.get('Authorization')
    if (!authHeader) {
      return new Response('Authorization header required', { status: 401 })
    }

    const { data: { user }, error: authError } = await supabase.auth.getUser(
      authHeader.replace('Bearer ', '')
    )

    if (authError || !user) {
      return new Response('Unauthorized', { status: 401 })
    }

    // Verify admin role
    const { data: profile } = await supabase
      .from('profiles')
      .select('user_type')
      .eq('id', user.id)
      .single()

    if (!profile || !['admin', 'super_admin'].includes(profile.user_type)) {
      return new Response('Forbidden', { status: 403 })
    }

    // Fetch dispute items with related data
    const { data: disputeItems, error: fetchError } = await supabase
      .from('dispute_items')
      .select(`
        *,
        profiles!inner(*),
        credit_accounts(*),
        credit_inquiries(*),
        public_records(*)
      `)
      .in('id', disputeItemIds)

    if (fetchError) {
      console.error('Error fetching dispute items:', fetchError)
```

```
    throw fetchError
  }

  if (!disputeItems || disputeItems.length === 0) {
    return new Response('No dispute items found', { status: 404 })
  }

  // Group dispute items by user
  const disputesByUser = disputeItems.reduce((acc, item) => {
    const userId = item.user_id
    if (!acc[userId]) acc[userId] = []
    acc[userId].push(item)
    return acc
  }, {} as Record<string, DisputeItem[]>)

  const generatedLetters = []

  // Generate letter for each user
  for (const [userId, userDisputes] of Object.entries(disputesByUser)) {
    // Get or use default template
    let template: LetterTemplate

    if (templateId) {
      const { data: customTemplate } = await supabase
        .from('letter_templates')
        .select('*')
        .eq('id', templateId)
        .single()

      template = customTemplate || await getDefaultTemplate(supabase,
userDisputes[0].dispute_type)
    } else {
      template = await getDefaultTemplate(supabase, userDisputes[0].dispute_type)
    }

    // Generate letter content
    const letterContent = await generateLetterContent(userDisputes, template,
customizations)

    // Generate PDF
    const pdfBuffer = await generatePDF(letterContent)

    // Upload PDF to Supabase Storage
    const fileName = `dispute-letter-${userId}-${Date.now()}.pdf`
    const { data: uploadData, error: uploadError } = await supabase.storage
      .from('dispute-letters')
      .upload(fileName, pdfBuffer, {
        contentType: 'application/pdf',
        upsert: false
      })

    if (uploadError) {
```

```
      console.error('Error uploading PDF:', uploadError)
      throw uploadError
    }

    // Save letter records to database
    for (const dispute of userDisputes) {
      const { data: letter, error: letterError } = await supabase
        .from('dispute_letters')
        .insert({
          dispute_item_id: dispute.id,
          template_id: template.id,
          letter_content: letterContent,
          letter_pdf_path: uploadData.path,
          status: 'generated',
          response_deadline: calculateResponseDeadline()
        })
        .select()
        .single()

      if (letterError) {
        console.error('Error saving letter:', letterError)
        throw letterError
      }

      generatedLetters.push(letter)

      // Update dispute item status
      await supabase
        .from('dispute_items')
        .update({ status: 'letter_generated' })
        .eq('id', dispute.id)
    }
  }

  return new Response(
    JSON.stringify({
      success: true,
      letters: generatedLetters,
      message: `Generated ${generatedLetters.length} dispute letters`
    }),
    { headers: { "Content-Type": "application/json" } }
  )

} catch (error) {
  console.error('Letter generation error:', error)
  return new Response(
    JSON.stringify({
      error: 'Failed to generate dispute letter',
      details: error.message
    }),
    {
      status: 500,
```

```
      headers: { "Content-Type": "application/json" }
    }
   )
  }
})

async function getDefaultTemplate(supabase: any, disputeType: string):
Promise<LetterTemplate> {
  const { data: template } = await supabase
   .from('letter_templates')
   .select('*')
   .eq('dispute_type', disputeType)
   .eq('is_active', true)
   .order('version', { ascending: false })
   .limit(1)
   .single()

  if (!template) {
   // Return a basic default template
   return {
    id: 'default',
    name: 'Default Template',
    template_content: getBasicTemplate(),
    dispute_type: disputeType
   }
  }

  return template
}

async function generateLetterContent(
  disputes: DisputeItem[],
  template: LetterTemplate,
  customizations?: any
): Promise<string> {
  const client = disputes[0].profiles
  const currentDate = new Date().toLocaleDateString()

  // Build dispute items list
  const disputeItemsList = disputes.map(dispute => {
   if (dispute.credit_accounts) {
    return `• ${dispute.credit_accounts.creditor_name} - $
{dispute.credit_accounts.account_name} (Reason: ${dispute.dispute_reason})`
   } else if (dispute.credit_inquiries) {
    return `• ${dispute.credit_inquiries.creditor_name} - Inquiry dated $
{dispute.credit_inquiries.inquiry_date} (Reason: ${dispute.dispute_reason})`
   } else {
    return `• Personal Information (Reason: ${dispute.dispute_reason})`
   }
  }).join('\n')

  // Replace template placeholders
```

```
  let content = template.template_content
    .replace(/{{client_name}}/g, `${client.first_name} ${client.last_name}`)
    .replace(/{{client_first_name}}/g, client.first_name)
    .replace(/{{client_last_name}}/g, client.last_name)
    .replace(/{{client_address}}/g, formatAddress(client))
    .replace(/{{current_date}}/g, currentDate)
    .replace(/{{dispute_items}}/g, disputeItemsList)
    .replace(/{{dispute_count}}/g, disputes.length.toString())

  // Apply customizations
  if (customizations?.letterhead) {
    content = customizations.letterhead + '\n\n' + content
  }

  if (customizations?.additionalText) {
    content = content + '\n\n' + customizations.additionalText
  }

  if (customizations?.signature) {
    content = content + '\n\nSincerely,\n\n' + customizations.signature
  }

  return content
}

async function generatePDF(content: string): Promise<Uint8Array> {
  // For this example, we'll use a simple HTML to PDF conversion
  // In production, you might want to use a more sophisticated PDF library

  const htmlContent = `
    <!DOCTYPE html>
    <html>
    <head>
      <meta charset="utf-8">
      <style>
        body {
          font-family: Arial, sans-serif;
          line-height: 1.6;
          margin: 40px;
          color: #333;
        }
        .header {
          text-align: center;
          margin-bottom: 30px;
        }
        .content {
          white-space: pre-line;
        }
        .footer {
          margin-top: 30px;
          text-align: center;
          font-size: 12px;
```

```
        color: #666;
      }
    </style>
  </head>
  <body>
    <div class="header">
      <h1>Credit Dispute Letter</h1>
    </div>
    <div class="content">
      ${content.replace(/\n/g, '<br>')}
    </div>
    <div class="footer">
      <p>Generated by SAINTRIX Beta Credit Repair System</p>
    </div>
  </body>
  </html>
`

  // Use Deno's built-in capabilities or a PDF generation service
  // This is a simplified example - in production, integrate with a proper PDF service
  const encoder = new TextEncoder()
  return encoder.encode(htmlContent) // This would be actual PDF bytes in
production
}

function formatAddress(client: any): string {
  let address = client.address_line1
  if (client.address_line2) {
    address += `, ${client.address_line2}`
  }
  address += `\n${client.city}, ${client.state} ${client.zip_code}`
  return address
}

function calculateResponseDeadline(): string {
  const deadline = new Date()
  deadline.setDate(deadline.getDate() + 30) // 30 days from now
  return deadline.toISOString().split('T')[0]
}

function getBasicTemplate(): string {
  return `{{current_date}}

Credit Reporting Agency
P.O. Box [Bureau Address]

Re: Request for Investigation of Credit Report Information

Dear Sir or Madam,

I am writing to dispute the following information in my file. I have circled the items
I dispute on the attached copy of the report I received.
```

```
{{client_name}}
{{client_address}}

The following items are inaccurate or incomplete:

{{dispute_items}}

I am requesting that the items be removed or corrected to update my credit profile. Enclosed are copies of supporting documents. Please investigate these matters and delete or correct the disputed items as soon as possible.

Sincerely,

{{client_name}}`
}
```

### 6.2. Next.js API Routes

Next.js API routes serve as the middleware layer, providing additional security, validation, and business logic between the frontend and Supabase Edge Functions.

**Credit Report Import API Route:**

```
// src/app/api/credit/import/route.ts
import { NextRequest, NextResponse } from 'next/server'
import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs'
import { cookies } from 'next/headers'
import { z } from 'zod'

const importSchema = z.object({
  provider: z.enum(['plaid', 'finicity']),
  accessToken: z.string().min(1),
  publicToken: z.string().optional()
})

export async function POST(request: NextRequest) {
  try {
    const supabase = createRouteHandlerClient({ cookies })

    // Verify user authentication
    const { data: { session }, error: authError } = await supabase.auth.getSession()

    if (authError || !session) {
      return NextResponse.json(
        { error: 'Unauthorized' },
        { status: 401 }
      )
    }
```

```
    // Validate request body
    const body = await request.json()
    const validationResult = importSchema.safeParse(body)

    if (!validationResult.success) {
      return NextResponse.json(
        { error: 'Invalid request data', details: validationResult.error.errors },
        { status: 400 }
      )
    }

    const { provider, accessToken } = validationResult.data

    // Check if user already has a recent import
    const { data: recentImport } = await supabase
      .from('credit_reports')
      .select('created_at')
      .eq('user_id', session.user.id)
      .gte('created_at', new Date(Date.now() - 24 * 60 * 60 * 1000).toISOString()) //
Last 24 hours
      .single()

    if (recentImport) {
      return NextResponse.json(
        { error: 'Credit report already imported within the last 24 hours' },
        { status: 429 }
      )
    }

    // Call Supabase Edge Function
    const { data, error } = await supabase.functions.invoke('import-credit-report', {
      body: {
        userId: session.user.id,
        provider,
        accessToken
      }
    })

    if (error) {
      console.error('Edge function error:', error)
      return NextResponse.json(
        { error: 'Failed to import credit report', details: error.message },
        { status: 500 }
      )
    }

    // Log the successful import
    await supabase
      .from('audit_logs')
      .insert({
        user_id: session.user.id,
```

```
        action: 'CREDIT_REPORT_IMPORTED',
        table_name: 'credit_reports',
        record_id: data.creditReportId,
        new_values: { provider, accounts_imported: data.accountsImported }
      })

    return NextResponse.json({
      success: true,
      creditReportId: data.creditReportId,
      accountsImported: data.accountsImported,
      creditScore: data.creditScore
    })

  } catch (error) {
    console.error('API route error:', error)
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    )
  }
}

// Handle preflight requests for CORS
export async function OPTIONS(request: NextRequest) {
  return new NextResponse(null, {
    status: 200,
    headers: {
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Methods': 'POST, OPTIONS',
      'Access-Control-Allow-Headers': 'Content-Type, Authorization',
    },
  })
}
```

**Dispute Management API Routes:**

```
// src/app/api/disputes/route.ts
import { NextRequest, NextResponse } from 'next/server'
import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs'
import { cookies } from 'next/headers'
import { z } from 'zod'

const createDisputeSchema = z.object({
  creditAccountId: z.string().uuid().optional(),
  creditInquiryId: z.string().uuid().optional(),
  publicRecordId: z.string().uuid().optional(),
  disputeType: z.enum(['account', 'inquiry', 'public_record', 'personal_info']),
  disputeReason: z.string().min(1).max(500),
  clientNotes: z.string().max(2000).optional(),
  priority: z.enum(['low', 'medium', 'high']).default('medium')
```

```
}).refine(
  (data) => {
    const refs = [data.creditAccountId, data.creditInquiryId, data.publicRecordId]
    return refs.filter(Boolean).length === 1
  },
  {
    message: "Exactly one reference ID must be provided"
  }
)

// GET /api/disputes - Fetch user's disputes
export async function GET(request: NextRequest) {
  try {
    const supabase = createRouteHandlerClient({ cookies })

    const { data: { session }, error: authError } = await supabase.auth.getSession()

    if (authError || !session) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 })
    }

    const { searchParams } = new URL(request.url)
    const status = searchParams.get('status')
    const limit = parseInt(searchParams.get('limit') || '50')
    const offset = parseInt(searchParams.get('offset') || '0')

    let query = supabase
      .from('dispute_items')
      .select(`
        *,
        credit_accounts(*),
        credit_inquiries(*),
        public_records(*),
        dispute_letters(*)
      `)
      .eq('user_id', session.user.id)
      .order('created_at', { ascending: false })
      .range(offset, offset + limit - 1)

    if (status) {
      query = query.eq('status', status)
    }

    const { data: disputes, error } = await query

    if (error) {
      console.error('Error fetching disputes:', error)
      return NextResponse.json(
        { error: 'Failed to fetch disputes' },
        { status: 500 }
      )
    }
```

```
    return NextResponse.json({ disputes })

  } catch (error) {
    console.error('API route error:', error)
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    )
  }
}

// POST /api/disputes - Create new dispute
export async function POST(request: NextRequest) {
  try {
    const supabase = createRouteHandlerClient({ cookies })

    const { data: { session }, error: authError } = await supabase.auth.getSession()

    if (authError || !session) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 })
    }

    const body = await request.json()
    const validationResult = createDisputeSchema.safeParse(body)

    if (!validationResult.success) {
      return NextResponse.json(
        { error: 'Invalid request data', details: validationResult.error.errors },
        { status: 400 }
      )
    }

    const disputeData = validationResult.data

    // Check if item is already being disputed
    const existingDispute = await supabase
      .from('dispute_items')
      .select('id')
      .eq('user_id', session.user.id)
      .eq(
        disputeData.creditAccountId ? 'credit_account_id' :
        disputeData.creditInquiryId ? 'credit_inquiry_id' : 'public_record_id',
        disputeData.creditAccountId || disputeData.creditInquiryId ||
disputeData.publicRecordId
      )
      .not('status', 'in', '(resolved,unresolved)')
      .single()

    if (existingDispute.data) {
      return NextResponse.json(
        { error: 'This item is already being disputed' },
```

```
        { status: 409 }
      )
    }

    // Create the dispute
    const { data: dispute, error } = await supabase
      .from('dispute_items')
      .insert({
        user_id: session.user.id,
        credit_account_id: disputeData.creditAccountId,
        credit_inquiry_id: disputeData.creditInquiryId,
        public_record_id: disputeData.publicRecordId,
        dispute_type: disputeData.disputeType,
        dispute_reason: disputeData.disputeReason,
        client_notes: disputeData.clientNotes,
        priority: disputeData.priority,
        status: 'pending_review'
      })
      .select()
      .single()

    if (error) {
      console.error('Error creating dispute:', error)
      return NextResponse.json(
        { error: 'Failed to create dispute' },
        { status: 500 }
      )
    }

    // Log the dispute creation
    await supabase
      .from('audit_logs')
      .insert({
        user_id: session.user.id,
        action: 'DISPUTE_CREATED',
        table_name: 'dispute_items',
        record_id: dispute.id,
        new_values: disputeData
      })

    return NextResponse.json({ dispute }, { status: 201 })

  } catch (error) {
    console.error('API route error:', error)
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    )
  }
}

// src/app/api/admin/disputes/[id]/route.ts
```

```typescript
import { NextRequest, NextResponse } from 'next/server'
import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs'
import { cookies } from 'next/headers'
import { z } from 'zod'

const updateDisputeSchema = z.object({
  status: z.enum([
    'pending_review', 'approved', 'rejected', 'letter_generated',
    'letter_sent', 'response_received', 'resolved', 'unresolved'
  ]).optional(),
  adminNotes: z.string().max(2000).optional(),
  priority: z.enum(['low', 'medium', 'high', 'urgent']).optional()
})

// PUT /api/admin/disputes/[id] - Update dispute (admin only)
export async function PUT(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    const supabase = createRouteHandlerClient({ cookies })

    const { data: { session }, error: authError } = await supabase.auth.getSession()

    if (authError || !session) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 })
    }

    // Verify admin role
    const { data: profile } = await supabase
      .from('profiles')
      .select('user_type')
      .eq('id', session.user.id)
      .single()

    if (!profile || !['admin', 'super_admin'].includes(profile.user_type)) {
      return NextResponse.json({ error: 'Forbidden' }, { status: 403 })
    }

    const body = await request.json()
    const validationResult = updateDisputeSchema.safeParse(body)

    if (!validationResult.success) {
      return NextResponse.json(
        { error: 'Invalid request data', details: validationResult.error.errors },
        { status: 400 }
      )
    }

    const updateData = validationResult.data

    // Add resolved timestamp if status is being set to resolved
```

```
    if (updateData.status === 'resolved') {
      updateData.resolved_at = new Date().toISOString()
    }

    const { data: dispute, error } = await supabase
      .from('dispute_items')
      .update({
        ...updateData,
        admin_notes: updateData.adminNotes,
        updated_at: new Date().toISOString()
      })
      .eq('id', params.id)
      .select()
      .single()

    if (error) {
      console.error('Error updating dispute:', error)
      return NextResponse.json(
        { error: 'Failed to update dispute' },
        { status: 500 }
      )
    }

    return NextResponse.json({ dispute })

  } catch (error) {
    console.error('API route error:', error)
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    )
  }
}
```

# 7. Testing and Quality Assurance

## 7.1. Testing Strategy

SAINTRIX Beta requires comprehensive testing to ensure the security, reliability, and functionality of the credit repair platform. This section outlines the testing approach and implementation.

**Testing Framework Setup:**

```
// jest.config.js
const nextJest = require('next/jest')

const createJestConfig = nextJest({
```

```
  dir: './',
})

const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  moduleNameMapping: {
    '^@/components/(.*)$': '<rootDir>/src/components/$1',
    '^@/lib/(.*)$': '<rootDir>/src/lib/$1',
    '^@/hooks/(.*)$': '<rootDir>/src/hooks/$1',
  },
  testEnvironment: 'jest-environment-jsdom',
  collectCoverageFrom: [
    'src/**/*.{js,jsx,ts,tsx}',
    '!src/**/*.d.ts',
    '!src/app/layout.tsx',
    '!src/app/globals.css',
  ],
  coverageThreshold: {
    global: {
      branches: 70,
      functions: 70,
      lines: 70,
      statements: 70,
    },
  },
}

module.exports = createJestConfig(customJestConfig)

// jest.setup.js
import '@testing-library/jest-dom'
import { TextEncoder, TextDecoder } from 'util'

global.TextEncoder = TextEncoder
global.TextDecoder = TextDecoder

// Mock Supabase client
jest.mock('@/lib/supabase/client', () => ({
  createClient: jest.fn(() => ({
    auth: {
      getSession: jest.fn(),
      signInWithPassword: jest.fn(),
      signUp: jest.fn(),
      signOut: jest.fn(),
    },
    from: jest.fn(() => ({
      select: jest.fn().mockReturnThis(),
      insert: jest.fn().mockReturnThis(),
      update: jest.fn().mockReturnThis(),
      delete: jest.fn().mockReturnThis(),
      eq: jest.fn().mockReturnThis(),
      single: jest.fn(),
```

```
    }),
  }),
})
```

**Component Testing:**

```
// src/components/__tests__/CreditScoreDisplay.test.tsx
import { render, screen } from '@testing-library/react'
import { CreditScoreDisplay } from '@/components/shared/CreditScoreDisplay'

describe('CreditScoreDisplay', () => {
  it('renders credit score correctly', () => {
    render(
      <CreditScoreDisplay
        score={750}
        scoreModel="FICO"
        reportDate="2024-01-15"
      />
    )

    expect(screen.getByText('750')).toBeInTheDocument()
    expect(screen.getByText('Very Good • FICO')).toBeInTheDocument()
    expect(screen.getByText('As of 1/15/2024')).toBeInTheDocument()
  })

  it('displays correct color for different score ranges', () => {
    const { rerender } = render(<CreditScoreDisplay score={850} />)
    expect(screen.getByText('850')).toHaveClass('text-green-600')

    rerender(<CreditScoreDisplay score={500} />)
    expect(screen.getByText('500')).toHaveClass('text-red-600')
  })

  it('shows score improvement when previous score is provided', () => {
    render(
      <CreditScoreDisplay
        score={750}
        previousScore={720}
      />
    )

    expect(screen.getByText('+30')).toBeInTheDocument()
    expect(screen.getByText('+30')).toHaveClass('text-green-600')
  })

  it('shows score decline when previous score is higher', () => {
    render(
      <CreditScoreDisplay
        score={700}
        previousScore={750}
```

```
      />
    )

    expect(screen.getByText('-50')).toBeInTheDocument()
    expect(screen.getByText('-50')).toHaveClass('text-red-600')
  })
})

// src/components/__tests__/DisputeStatusBadge.test.tsx
import { render, screen } from '@testing-library/react'
import { DisputeStatusBadge } from '@/components/shared/DisputeStatusBadge'

describe('DisputeStatusBadge', () => {
  it('renders pending review status correctly', () => {
    render(<DisputeStatusBadge status="pending_review" />)

    expect(screen.getByText('Pending Review')).toBeInTheDocument()
  })

  it('renders approved status with correct styling', () => {
    render(<DisputeStatusBadge status="approved" />)

    const badge = screen.getByText('Approved')
    expect(badge).toBeInTheDocument()
  })

  it('hides icon when showIcon is false', () => {
    render(<DisputeStatusBadge status="resolved" showIcon={false} />)

    expect(screen.getByText('Resolved')).toBeInTheDocument()
    // Icon should not be present
    expect(screen.queryByRole('img')).not.toBeInTheDocument()
  })
})
```

**API Route Testing:**

```
 // src/app/api/__tests__/disputes.test.ts
import { NextRequest } from 'next/server'
import { GET, POST } from '@/app/api/disputes/route'
import { createMocks } from 'node-mocks-http'

// Mock Supabase
jest.mock('@supabase/auth-helpers-nextjs', () => ({
  createRouteHandlerClient: jest.fn(() => ({
    auth: {
      getSession: jest.fn(),
    },
    from: jest.fn(() => ({
      select: jest.fn().mockReturnThis(),
```

```
      insert: jest.fn().mockReturnThis(),
      eq: jest.fn().mockReturnThis(),
      order: jest.fn().mockReturnThis(),
      range: jest.fn().mockReturnThis(),
      single: jest.fn(),
    })),
  })),
}))

describe('/api/disputes', () => {
  describe('GET', () => {
    it('returns unauthorized when no session', async () => {
      const { createRouteHandlerClient } = require('@supabase/auth-helpers-nextjs')
      const mockSupabase = createRouteHandlerClient()
      mockSupabase.auth.getSession.mockResolvedValue({
        data: { session: null },
        error: null,
      })

      const request = new NextRequest('http://localhost:3000/api/disputes')
      const response = await GET(request)
      const data = await response.json()

      expect(response.status).toBe(401)
      expect(data.error).toBe('Unauthorized')
    })

    it('returns disputes for authenticated user', async () => {
      const { createRouteHandlerClient } = require('@supabase/auth-helpers-nextjs')
      const mockSupabase = createRouteHandlerClient()

      mockSupabase.auth.getSession.mockResolvedValue({
        data: { session: { user: { id: 'user-123' } } },
        error: null,
      })

      const mockDisputes = [
        {
          id: 'dispute-1',
          user_id: 'user-123',
          dispute_reason: 'Not my account',
          status: 'pending_review',
        },
      ]

      mockSupabase.from().select().eq().order().range.mockResolvedValue({
        data: mockDisputes,
        error: null,
      })

      const request = new NextRequest('http://localhost:3000/api/disputes')
      const response = await GET(request)
```

```
    const data = await response.json()

    expect(response.status).toBe(200)
    expect(data.disputes).toEqual(mockDisputes)
  })
})

describe('POST', () => {
  it('creates dispute with valid data', async () => {
    const { createRouteHandlerClient } = require('@supabase/auth-helpers-nextjs')
    const mockSupabase = createRouteHandlerClient()

    mockSupabase.auth.getSession.mockResolvedValue({
      data: { session: { user: { id: 'user-123' } } },
      error: null,
    })

    // Mock no existing dispute
    mockSupabase.from().select().eq().not().single.mockResolvedValue({
      data: null,
      error: null,
    })

    const newDispute = {
      id: 'dispute-new',
      user_id: 'user-123',
      dispute_reason: 'Incorrect balance',
      status: 'pending_review',
    }

    mockSupabase.from().insert().select().single.mockResolvedValue({
      data: newDispute,
      error: null,
    })

    const requestBody = {
      creditAccountId: 'account-123',
      disputeType: 'account',
      disputeReason: 'Incorrect balance',
      priority: 'medium',
    }

    const request = new NextRequest('http://localhost:3000/api/disputes', {
      method: 'POST',
      body: JSON.stringify(requestBody),
    })

    const response = await POST(request)
    const data = await response.json()

    expect(response.status).toBe(201)
    expect(data.dispute).toEqual(newDispute)
```

```
    })

    it('validates request data', async () => {
      const { createRouteHandlerClient } = require('@supabase/auth-helpers-nextjs')
      const mockSupabase = createRouteHandlerClient()

      mockSupabase.auth.getSession.mockResolvedValue({
        data: { session: { user: { id: 'user-123' } } },
        error: null,
      })

      const invalidRequestBody = {
        disputeType: 'invalid-type',
        disputeReason: '',
      }

      const request = new NextRequest('http://localhost:3000/api/disputes', {
        method: 'POST',
        body: JSON.stringify(invalidRequestBody),
      })

      const response = await POST(request)
      const data = await response.json()

      expect(response.status).toBe(400)
      expect(data.error).toBe('Invalid request data')
    })
  })
})
```

**End-to-End Testing:**

```
// tests/e2e/auth.spec.ts
import { test, expect } from '@playwright/test'

test.describe('Authentication Flow', () => {
  test('user can sign up and log in', async ({ page }) => {
    // Navigate to registration page
    await page.goto('/register')

    // Fill out registration form
    await page.fill('[data-testid="first-name"]', 'John')
    await page.fill('[data-testid="last-name"]', 'Doe')
    await page.fill('[data-testid="email"]', 'john.doe@example.com')
    await page.fill('[data-testid="password"]', 'SecurePassword123!')
    await page.fill('[data-testid="confirm-password"]', 'SecurePassword123!')
    await page.check('[data-testid="terms-accepted"]')

    // Submit registration
    await page.click('[data-testid="register-button"]')
```

```
    // Should show success message
    await expect(page.locator('[data-testid="success-message"]')).toBeVisible()

    // Navigate to login page
    await page.goto('/login')

    // Fill out login form
    await page.fill('[data-testid="email"]', 'john.doe@example.com')
    await page.fill('[data-testid="password"]', 'SecurePassword123!')

    // Submit login
    await page.click('[data-testid="login-button"]')

    // Should redirect to dashboard
    await expect(page).toHaveURL('/dashboard')
    await expect(page.locator('h1')).toContainText('Welcome back, John!')
  })

  test('shows error for invalid credentials', async ({ page }) => {
    await page.goto('/login')

    await page.fill('[data-testid="email"]', 'invalid@example.com')
    await page.fill('[data-testid="password"]', 'wrongpassword')

    await page.click('[data-testid="login-button"]')

    await expect(page.locator('[data-testid="error-message"]')).toBeVisible()
  })
})

// tests/e2e/disputes.spec.ts
import { test, expect } from '@playwright/test'

test.describe('Dispute Management', () => {
  test.beforeEach(async ({ page }) => {
    // Login as test user
    await page.goto('/login')
    await page.fill('[data-testid="email"]', 'testuser@example.com')
    await page.fill('[data-testid="password"]', 'TestPassword123!')
    await page.click('[data-testid="login-button"]')
    await expect(page).toHaveURL('/dashboard')
  })

  test('user can create a new dispute', async ({ page }) => {
    // Navigate to disputes page
    await page.goto('/disputes')

    // Click new dispute button
    await page.click('[data-testid="new-dispute-button"]')

    // Fill out dispute form
```

```
    await page.selectOption('[data-testid="credit-account-select"]', 'account-123')
    await page.selectOption('[data-testid="dispute-reason-select"]', 'Not my
account')
    await page.fill('[data-testid="client-notes"]', 'This account does not belong to
me.')
    await page.selectOption('[data-testid="priority-select"]', 'high')

    // Submit dispute
    await page.click('[data-testid="create-dispute-button"]')

    // Should show success and redirect
    await expect(page.locator('[data-testid="dispute-item"]')).toBeVisible()
    await expect(page.locator('[data-testid="dispute-
status"]')).toContainText('Pending Review')
  })

  test('admin can review and approve disputes', async ({ page }) => {
    // Login as admin
    await page.goto('/login')
    await page.fill('[data-testid="email"]', 'admin@example.com')
    await page.fill('[data-testid="password"]', 'AdminPassword123!')
    await page.click('[data-testid="login-button"]')

    // Navigate to admin disputes
    await page.goto('/admin/disputes')

    // Find pending dispute
    const disputeRow = page.locator('[data-testid="dispute-row"]').first()
    await disputeRow.click()

    // Update status to approved
    await page.selectOption('[data-testid="status-select"]', 'approved')
    await page.fill('[data-testid="admin-notes"]', 'Dispute approved for processing')
    await page.click('[data-testid="update-dispute-button"]')

    // Should show updated status
    await expect(page.locator('[data-testid="dispute-
status"]')).toContainText('Approved')
  })
})
```

### 7.2. Security Testing

```
 // tests/security/auth.test.ts
import { test, expect } from '@playwright/test'

test.describe('Security Tests', () => {
  test('prevents SQL injection in search', async ({ page }) => {
    await page.goto('/login')
    // Login as admin
```

```
    await page.fill('[data-testid="email"]', 'admin@example.com')
    await page.fill('[data-testid="password"]', 'AdminPassword123!')
    await page.click('[data-testid="login-button"]')

    await page.goto('/admin/clients')

    // Attempt SQL injection
    const maliciousInput = "'; DROP TABLE profiles; --"
    await page.fill('[data-testid="search-input"]', maliciousInput)
    await page.press('[data-testid="search-input"]', 'Enter')

    // Should not crash or expose database errors
    await expect(page.locator('[data-testid="error-message"]')).not.toBeVisible()
    await expect(page.locator('body')).not.toContainText('database error')
  })

  test('enforces rate limiting on login attempts', async ({ page }) => {
    await page.goto('/login')

    // Attempt multiple failed logins
    for (let i = 0; i < 6; i++) {
      await page.fill('[data-testid="email"]', 'test@example.com')
      await page.fill('[data-testid="password"]', 'wrongpassword')
      await page.click('[data-testid="login-button"]')
      await page.waitForTimeout(1000)
    }

    // Should show rate limit error
    await expect(page.locator('[data-testid="error-message"]')).toContainText('rate
limit')
  })

  test('prevents unauthorized access to admin routes', async ({ page }) => {
    // Try to access admin page without authentication
    await page.goto('/admin/dashboard')

    // Should redirect to login
    await expect(page).toHaveURL('/login')

    // Login as regular user
    await page.fill('[data-testid="email"]', 'user@example.com')
    await page.fill('[data-testid="password"]', 'UserPassword123!')
    await page.click('[data-testid="login-button"]')

    // Try to access admin page as regular user
    await page.goto('/admin/dashboard')

    // Should redirect to user dashboard
    await expect(page).toHaveURL('/dashboard')
  })
})
```

**Author:** Manus AI

**Date:** July 31, 2025

**Version:** 1.0 Development Guide

# 8. Deployment and Production Setup

## 8.1. Environment Configuration

Proper environment configuration is crucial for the secure deployment of SAINTRIX Beta. This section provides detailed instructions for setting up production environments.

**Production Environment Variables:**

```
 # .env.production
# Supabase Configuration
NEXT_PUBLIC_SUPABASE_URL=https://your-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_production_anon_key
SUPABASE_SERVICE_ROLE_KEY=your_production_service_role_key

# Third-party API Configuration
PLAID_CLIENT_ID=your_production_plaid_client_id
PLAID_SECRET=your_production_plaid_secret
PLAID_ENV=production

# Email Service
RESEND_API_KEY=your_production_resend_key

# Security
NEXTAUTH_SECRET=your_nextauth_secret_key
NEXTAUTH_URL=https://your-domain.com

# Monitoring and Analytics
VERCEL_ANALYTICS_ID=your_vercel_analytics_id
SENTRY_DSN=your_sentry_dsn
```

**Vercel Deployment Configuration:**

```
 // vercel.json
{
  "framework": "nextjs",
  "buildCommand": "npm run build",
  "devCommand": "npm run dev",
  "installCommand": "npm install",
```

```
  "functions": {
    "src/app/api/**/*.ts": {
      "maxDuration": 30
    }
  },
  "headers": [
    {
      "source": "/api/(.*)",
      "headers": [
        {
          "key": "Access-Control-Allow-Origin",
          "value": "*"
        },
        {
          "key": "Access-Control-Allow-Methods",
          "value": "GET, POST, PUT, DELETE, OPTIONS"
        },
        {
          "key": "Access-Control-Allow-Headers",
          "value": "Content-Type, Authorization"
        }
      ]
    }
  ],
  "rewrites": [
    {
      "source": "/admin/:path*",
      "destination": "/admin/:path*"
    }
  ]
}
```

**Database Migration Script:**

```
 -- migrations/001_initial_schema.sql
-- This script should be run in Supabase SQL editor for production setup

-- Enable required extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";

-- Create all tables (copy from previous database schema section)
-- ... (include all table creation statements from section 3.1)

-- Create indexes
-- ... (include all index creation statements from section 3.2)

-- Set up Row Level Security
-- ... (include all RLS policies from section 3.3)
```

```
-- Create functions and triggers
-- ... (include all functions and triggers from section 3.4)

-- Insert default data
INSERT INTO letter_templates (name, description, template_content,
dispute_type, is_active, version) VALUES
('Default Account Dispute', 'Standard template for disputing credit account
information',
'{{current_date}}

Credit Reporting Agency
P.O. Box [Bureau Address]

Re: Request for Investigation of Credit Report Information

Dear Sir or Madam,

I am writing to dispute the following information in my file. I have circled the items
I dispute on the attached copy of the report I received.

{{client_name}}
{{client_address}}

The following account information is inaccurate or incomplete:

{{dispute_items}}

I am requesting that the items be removed or corrected to update my credit
profile. Enclosed are copies of supporting documents. Please investigate these
matters and delete or correct the disputed items as soon as possible.

Sincerely,

{{client_name}}', 'account', true, 1),

('Default Inquiry Dispute', 'Standard template for disputing credit inquiries',
'{{current_date}}

Credit Reporting Agency
P.O. Box [Bureau Address]

Re: Request for Investigation of Credit Report Information

Dear Sir or Madam,

I am writing to dispute the following credit inquiries in my file:

{{client_name}}
{{client_address}}

The following inquiries are inaccurate or unauthorized:
```

```
{{dispute_items}}

I did not authorize these inquiries and request that they be removed from my credit report immediately.

Sincerely,

{{client_name}}', 'inquiry', true, 1);
```

### 8.2. Supabase Edge Functions Deployment

Deploy the Edge Functions to Supabase for production use:

```
# Install Supabase CLI
npm install -g supabase

# Login to Supabase
supabase login

# Link to your project
supabase link --project-ref your-project-ref

# Deploy Edge Functions
supabase functions deploy import-credit-report
supabase functions deploy generate-dispute-letter

# Set environment variables for Edge Functions
supabase secrets set PLAID_CLIENT_ID=your_production_plaid_client_id
supabase secrets set PLAID_SECRET=your_production_plaid_secret
supabase secrets set PLAID_ENV=production
supabase secrets set RESEND_API_KEY=your_production_resend_key
```

### 8.3. Vercel Deployment

Deploy the Next.js application to Vercel:

```
# Install Vercel CLI
npm install -g vercel

# Deploy to Vercel
vercel --prod

# Set environment variables in Vercel dashboard or CLI
vercel env add NEXT_PUBLIC_SUPABASE_URL production
vercel env add NEXT_PUBLIC_SUPABASE_ANON_KEY production
vercel env add SUPABASE_SERVICE_ROLE_KEY production
vercel env add PLAID_CLIENT_ID production
vercel env add PLAID_SECRET production
```

```
vercel env add PLAID_ENV production
vercel env add RESEND_API_KEY production
```

### 8.4. Domain and SSL Configuration

Configure custom domain and SSL certificates:

```
# Add custom domain in Vercel dashboard
# Configure DNS records:
# Type: CNAME
# Name: www (or @)
# Value: cname.vercel-dns.com

# SSL certificates are automatically provisioned by Vercel
```

### 8.5. Monitoring and Analytics Setup

Implement comprehensive monitoring for production:

```
// src/lib/monitoring.ts
import * as Sentry from '@sentry/nextjs'

// Initialize Sentry for error tracking
Sentry.init({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV,
  tracesSampleRate: 0.1,
  beforeSend(event) {
    // Filter out sensitive data
    if (event.request?.data) {
      delete event.request.data.password
      delete event.request.data.ssn
      delete event.request.data.accessToken
    }
    return event
  }
})

// Custom analytics tracking
export function trackEvent(eventName: string, properties?: Record<string, any>) {
  if (typeof window !== 'undefined' && window.gtag) {
    window.gtag('event', eventName, properties)
  }
}

// Performance monitoring
export function measurePerformance(name: string, fn: () => Promise<any>) {
  const start = performance.now()
```

```
  return fn().finally(() => {
    const duration = performance.now() - start
    trackEvent('performance_measure', { name, duration })
  })
}
```

# 9. Maintenance and Updates

## 9.1. Regular Maintenance Tasks

Establish a maintenance schedule for SAINTRIX Beta:

**Daily Tasks:**
- Monitor system health and error rates
- Review new user registrations and disputes
- Check for failed credit report imports
- Monitor database performance metrics

**Weekly Tasks:**
- Review and update dispute letter templates
- Analyze user engagement metrics
- Update security patches and dependencies
- Backup critical data and configurations

**Monthly Tasks:**
- Conduct security audits and penetration testing
- Review and optimize database queries
- Update third-party API integrations
- Generate compliance and audit reports

## 9.2. Update Procedures

Implement safe update procedures for production:

```
 # Update procedure script
#!/bin/bash

# 1. Create backup
echo "Creating database backup..."
supabase db dump --file backup-$(date +%Y%m%d).sql

# 2. Run tests
echo "Running test suite..."
npm run test:all
```

```
# 3. Deploy to staging
echo "Deploying to staging..."
vercel --target staging

# 4. Run integration tests
echo "Running integration tests..."
npm run test:e2e:staging

# 5. Deploy to production
echo "Deploying to production..."
vercel --prod

# 6. Run smoke tests
echo "Running production smoke tests..."
npm run test:smoke:production

echo "Deployment complete!"
```

### 9.3. Scaling Considerations

Plan for scaling SAINTRIX Beta as user base grows:

**Database Scaling:**
- Implement read replicas for improved performance
- Set up connection pooling for high concurrency
- Consider database sharding for large datasets
- Implement caching strategies for frequently accessed data

**Application Scaling:**
- Utilize Vercel's automatic scaling capabilities
- Implement CDN for static assets
- Consider implementing a queue system for heavy operations
- Monitor and optimize Edge Function performance

**Third-party API Management:**
- Implement rate limiting and retry logic
- Set up fallback providers for critical services
- Monitor API usage and costs
- Implement caching for API responses where appropriate

# 10. Compliance and Security

## 10.1. Data Privacy Compliance

Ensure SAINTRIX Beta meets data privacy requirements:

**GDPR Compliance:**
- Implement data export functionality
- Provide data deletion capabilities
- Maintain consent records
- Implement privacy by design principles

**CCPA Compliance:**
- Provide clear privacy notices
- Implement opt-out mechanisms
- Maintain data processing records
- Ensure third-party vendor compliance

## 10.2. Financial Data Security

Implement industry-standard security measures:

**PCI DSS Considerations:**
- Avoid storing sensitive payment data
- Use tokenization for payment processing
- Implement secure data transmission
- Regular security assessments

**SOC 2 Compliance:**
- Implement access controls and monitoring
- Maintain audit logs and documentation
- Regular security training for team members
- Incident response procedures

## 10.3. Regular Security Audits

Establish a security audit schedule:

```
// Security audit checklist
const securityAuditChecklist = {
  authentication: [
    'Multi-factor authentication enabled',
    'Password policies enforced',
```

```
    'Session management secure',
    'Account lockout mechanisms'
  ],
  authorization: [
    'Role-based access control implemented',
    'Principle of least privilege followed',
    'Regular access reviews conducted',
    'Privileged account monitoring'
  ],
  dataProtection: [
    'Data encryption at rest and in transit',
    'Secure key management',
    'Data backup and recovery tested',
    'Data retention policies enforced'
  ],
  infrastructure: [
    'Security patches up to date',
    'Network security controls',
    'Monitoring and alerting configured',
    'Incident response plan tested'
  ]
}
```

# 11. Conclusion

This comprehensive development guide provides Cursor AI with all the necessary information to build SAINTRIX Beta from start to finish. The guide covers every aspect of the development process, from initial setup and database design to frontend implementation, backend development, testing, and deployment.

## 11.1. Key Success Factors

To ensure successful implementation of SAINTRIX Beta, focus on these critical areas:

**Security First:** Given the sensitive nature of financial data, security must be the top priority throughout development. Implement encryption, access controls, and audit logging from the beginning.

**User Experience:** Create intuitive interfaces that make credit repair accessible to users while providing powerful tools for administrators. Focus on clear navigation, helpful feedback, and responsive design.

**Scalability:** Design the system to handle growth in users and data volume. Leverage serverless technologies and implement efficient database queries and caching strategies.

**Compliance:** Ensure the system meets all relevant regulatory requirements for handling financial and personal data. Implement proper consent management and data protection measures.

**Testing:** Comprehensive testing is essential for a financial application. Implement unit tests, integration tests, and end-to-end tests to ensure reliability and security.

## 11.2. Next Steps

After implementing the core functionality outlined in this guide, consider these enhancements for future versions:

1. **Mobile Application:** Develop native mobile apps for iOS and Android
2. **Advanced Analytics:** Implement machine learning for credit score prediction and optimization recommendations
3. **Integration Expansion:** Add support for additional credit bureaus and financial data providers
4. **Automated Workflows:** Implement more sophisticated automation for dispute processing and follow-up
5. **Educational Content:** Add comprehensive credit education modules and resources

## 11.3. Support and Resources

For ongoing development and maintenance:

- **Documentation:** Maintain comprehensive documentation for all code and processes
- **Community:** Engage with the Supabase and Next.js communities for support and best practices
- **Monitoring:** Implement robust monitoring and alerting to quickly identify and resolve issues
- **Training:** Ensure team members are trained on security best practices and compliance requirements

SAINTRIX Beta represents a significant opportunity to revolutionize the credit repair industry through technology. By following this guide and maintaining focus on security, user experience, and compliance, Cursor AI can build a platform that truly empowers individuals to take control of their financial health.

The combination of modern technologies like Supabase, Next.js, and Vercel provides a solid foundation for building a scalable, secure, and user-friendly credit repair platform. With careful implementation of the features and best practices outlined in this guide,

SAINTRIX Beta will be well-positioned to succeed in the competitive financial services market.

---

**Author:** Manus AI

**Date:** July 31, 2025

**Version:** 1.0 Complete Development Guide