ITL Deepfake Detective - Project Setup Guide

# 1 ITL Deepfake Detective - Advanced AI-Powered Detection Platform

## 1.1 Table of Contents

1. [Environment Setup](#environment-setup)
2. [Project Creation](#project-creation)
3. [Dependencies Installation](#dependencies-installation)
4. [Advanced Project Structure](#advanced-project-structure)
5. [API Configuration](#api-configuration)
6. [Core Library Implementation](#core-library-implementation)
7. [UI Components System](#ui-components-system)
8. [Advanced Features Implementation](#advanced-features-implementation)
9. [Charts & Analytics](#charts--analytics)
10. [PDF Export & Reporting](#pdf-export--reporting)
11. [Explanation System](#explanation-system)
12. [Navigation & Layout](#navigation--layout)
13. [Storage & Data Management](#storage--data-management)
14. [Running the Project](#running-the-project)
15. [Testing](#testing)
16. [Deployment](#deployment)

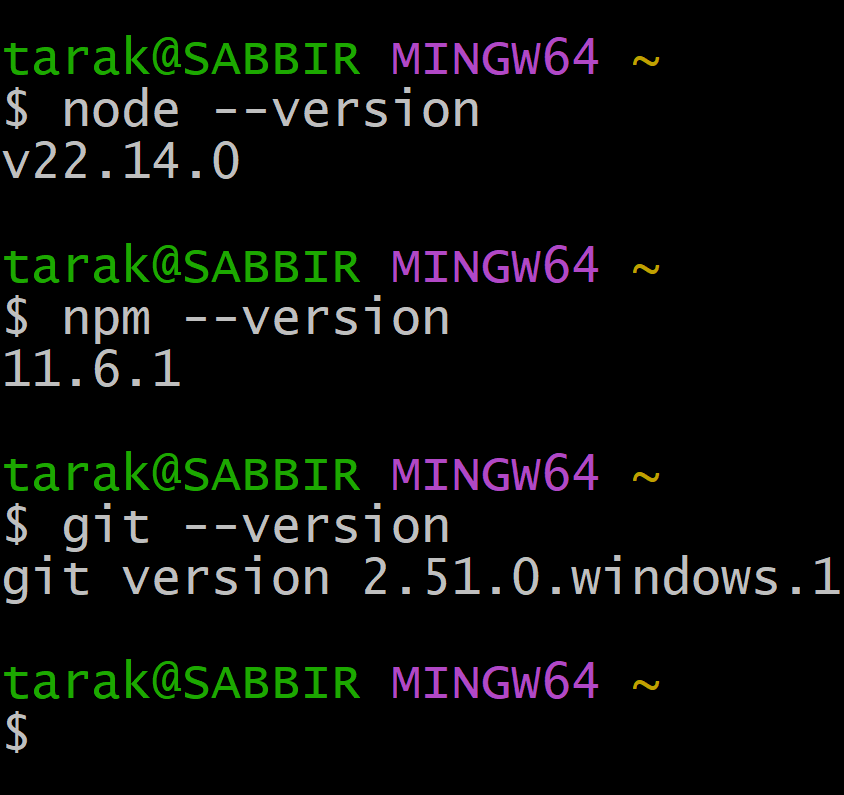
## 1.2 Environment Setup

### 1.2.1 Prerequisites Check

Download Link: <https://nodejs.org/en/download>

First, verify your development environment:

# Check Node.js version (required: Node.js 18.17 or later)  
node --version  
  
# Check npm version  
npm --version  
  
# Check Git installation  
git –version



### 1.2.2 Installing Node.js (if not installed)

**Windows:** - Download from [nodejs.org](https://nodejs.org/) - Install LTS version (20.x recommended)

**macOS:**

# Using Homebrew  
brew install node  
  
# Or download from nodejs.org

**Linux (Ubuntu/Debian):**

# Using NodeSource repository  
curl -fsSL https://deb.nodesource.com/setup\_20.x | sudo -E bash -  
sudo apt-get install -y nodejs

### 1.2.3 Development Tools

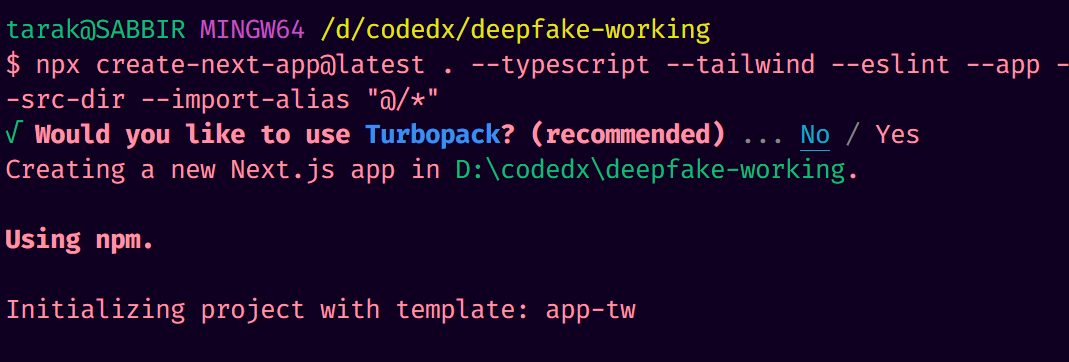
Install recommended tools:

# Install VS Code (recommended)  
# Download from https://code.visualstudio.com/  
  
# Install Git (if not installed)  
# Windows: Download from https://git-scm.com/  
# macOS: git is included with Xcode command line tools  
# Linux: sudo apt-get install git

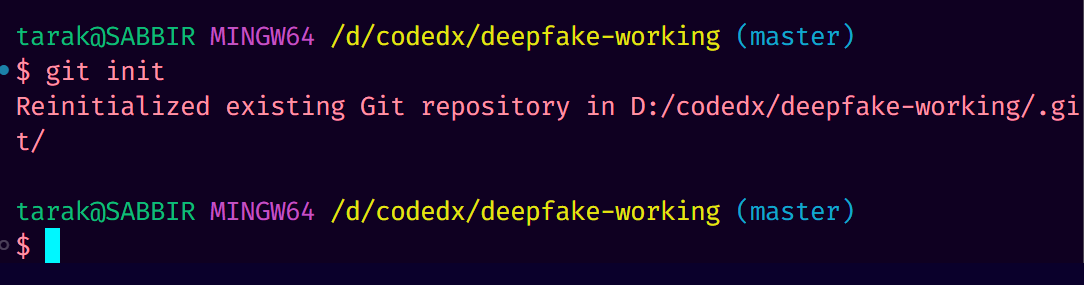
## 1.3 Project Creation

### 1.3.1 Step 1: Initialize the Project

# Create project directory  
mkdir deepfakeweb  
cd deepfakeweb  
  
# Initialize Next.js project with TypeScript in Vs Code Terminal  
npx create-next-app@latest . --typescript --tailwind --eslint --app --src-dir --import-alias "@/\*"



# Initialize Git repository (if not done automatically)  
git init



### 1.3.2 Step 2: Project Configuration Files

Create essential configuration files, this file will be automatically generated, just replace with the following code:



**next.config.ts:**

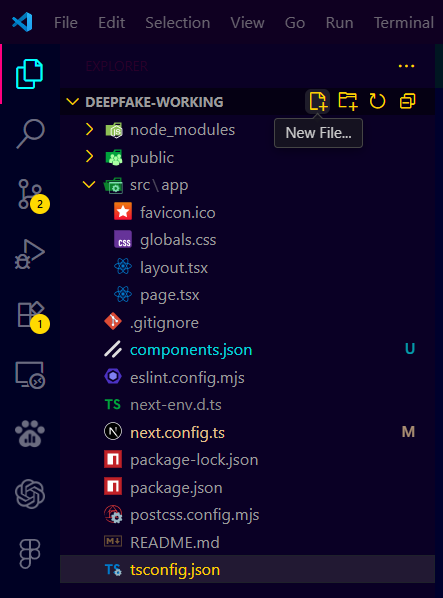
import type { NextConfig } from "next";  
  
const nextConfig: NextConfig = {  
 eslint: {  
 // Warning: This allows production builds to successfully complete even if  
 // your project has ESLint errors.  
 ignoreDuringBuilds: true,  
 },  
 typescript: {  
 // Warning: This allows production builds to successfully complete even if  
 // your project has type errors.  
 ignoreBuildErrors: false,  
 },  
};  
  
export default nextConfig;

**IMPORTANT NOTE:** This project uses **Tailwind CSS v4** which doesn’t require a separate tailwind.config.ts file. The configuration is handled directly in globals.css using the new @theme directive.

** Make sure it looks like the following code**

**postcss.config.mjs:**

const config = {  
 plugins: ["@tailwindcss/postcss"],  
};  
  
export default config;



**//Create in root dir the following file**

**components.json:**

{  
 "$schema": "https://ui.shadcn.com/schema.json",  
 "style": "new-york",  
 "rsc": true,  
 "tsx": true,  
 "tailwind": {  
 "config": "",  
 "css": "src/app/globals.css",  
 "baseColor": "neutral",  
 "cssVariables": true,  
 "prefix": ""  
 },  
 "iconLibrary": "lucide",  
 "aliases": {  
 "components": "@/components",  
 "utils": "@/lib/utils",  
 "ui": "@/components/ui",  
 "lib": "@/lib",  
 "hooks": "@/hooks"  
 },  
 "registries": {}  
}

## 1.4 Dependencies Installation in Vs Code Terminal

### 1.4.1 Core Dependencies

# Install UI and styling dependencies  
npm install @radix-ui/react-dialog @radix-ui/react-dropdown-menu @radix-ui/react-progress @radix-ui/react-separator @radix-ui/react-slot @radix-ui/react-switch @radix-ui/react-toast @radix-ui/react-toggle  
  
# Install utility libraries  
npm install class-variance-authority clsx tailwind-merge lucide-react framer-motion next-themes sonner  
  
# Install chart and visualization libraries  
npm install chart.js chartjs-adapter-date-fns recharts  
  
# Install file handling libraries  
npm install react-dropzone html2canvas jspdf @types/jspdf  
  
# Install API and environment libraries  
npm install @realitydefender/realitydefender dotenv  
  
# Install development dependencies  
npm install -D @tailwindcss/postcss tailwindcss @types/node @types/react @types/react-dom typescript tw-animate-css

### 

## 1.5 Advanced Project Structure

Create the comprehensive directory structure for the advanced deepfake detection platform:

**Complete Directory Setup:**

# Create main application directories

mkdir -p src/app/{api/rd/{result/[id],signed-url},results,upload}

mkdir -p src/components/{charts,explanation,layout,pdf,ui}

mkdir -p src/lib/types # Fixed: removed unnecessary braces

mkdir -p public

# Create all necessary files

touch src/app/{globals.css,layout.tsx,page.tsx}

touch src/app/results/page.tsx

touch src/app/upload/{loading.tsx,page.tsx}

touch src/app/api/rd/signed-url/route.ts

touch src/app/api/rd/result/[id]/route.ts

# Create component files

touch src/components/{upload-box.tsx,usage-dashboard.tsx}

touch src/components/charts/{advanced-charts.tsx,category-chart.tsx,confidence-gauge.tsx}

touch src/components/explanation/explanation-dashboard.tsx

touch src/components/layout/{navbar.tsx,sidebar.tsx}

touch src/components/pdf/{pdf-chart-components.tsx,pdf-export-dialog.tsx}

touch src/components/ui/{badge.tsx,button.tsx,card.tsx,dialog.tsx,dropdown-menu.tsx,progress.tsx,separator.tsx,sonner.tsx,switch.tsx,toggle.tsx}

# Create library files

touch src/lib/{explanation-generator.ts,pdf-generator.ts,reality-defender.ts,storage.ts,usage-tracker.ts,utils.ts}

touch src/lib/types/index.ts

**Advanced Project Structure:**

deepfakeweb/  
├── src/  
│ ├── app/  
│ │ ├── api/  
│ │ │ └── rd/  
│ │ │ ├── result/  
│ │ │ │ └── [id]/  
│ │ │ │ └── route.ts # [38 lines] - Analysis result retrieval  
│ │ │ └── signed-url/  
│ │ │ └── route.ts # [40 lines] - Secure upload URL generation  
│ │ ├── results/  
│ │ │ └── page.tsx # [628 lines] - Advanced results dashboard  
│ │ ├── upload/  
│ │ │ ├── loading.tsx # Loading state for upload  
│ │ │ └── page.tsx # Advanced upload interface  
│ │ ├── favicon.ico # App favicon  
│ │ ├── globals.css # [187 lines] - Tailwind v4 + custom styles  
│ │ ├── layout.tsx # [48 lines] - Enhanced root layout  
│ │ └── page.tsx # [242 lines] - Modern landing page  
│ ├── components/  
│ │ ├── charts/  
│ │ │ ├── advanced-charts.tsx # Advanced visualization components  
│ │ │ ├── category-chart.tsx # Category analysis charts  
│ │ │ └── confidence-gauge.tsx # Confidence visualization  
│ │ ├── explanation/  
│ │ │ └── explanation-dashboard.tsx # AI explanation system  
│ │ ├── layout/  
│ │ │ ├── navbar.tsx # Main navigation  
│ │ │ └── sidebar.tsx # Analysis history sidebar  
│ │ ├── pdf/  
│ │ │ ├── pdf-chart-components.tsx # PDF export charts  
│ │ │ └── pdf-export-dialog.tsx # PDF generation interface  
│ │ ├── ui/  
│ │ │ ├── badge.tsx # [46 lines] - Modern badge component  
│ │ │ ├── button.tsx # [58 lines] - Enhanced button component  
│ │ │ ├── card.tsx # [92 lines] - Advanced card component  
│ │ │ ├── dialog.tsx # Modal dialogs  
│ │ │ ├── dropdown-menu.tsx # Context menus  
│ │ │ ├── progress.tsx # Progress indicators  
│ │ │ ├── separator.tsx # Visual separators  
│ │ │ ├── sonner.tsx # Toast notifications  
│ │ │ ├── switch.tsx # Toggle switches  
│ │ │ └── toggle.tsx # Toggle buttons  
│ │ ├── upload-box.tsx # Advanced drag & drop upload  
│ │ └── usage-dashboard.tsx # Usage tracking dashboard  
│ └── lib/  
│ ├── types/  
│ │ └── index.ts # [311 lines] - Comprehensive TypeScript definitions  
│ ├── explanation-generator.ts # AI explanation generation  
│ ├── pdf-generator.ts # PDF report generation  
│ ├── reality-defender.ts # API integration layer  
│ ├── storage.ts # [304 lines] - Local storage management  
│ ├── usage-tracker.ts # Usage analytics & limits  
│ └── utils.ts # [6 lines] - Utility functions  
├── public/  
├── .env.local # Environment variables  
├── .env.example # Environment template  
├── .gitignore # Git ignore rules  
├── components.json # shadcn/ui configuration  
├── next.config.ts # Next.js configuration  
├── package.json # Dependencies & scripts  
├── postcss.config.mjs # PostCSS configuration  
├── tailwind.config.ts # Tailwind CSS configuration  
└── tsconfig.json # TypeScript configuration

## 1.6 API Configuration

### 1.6.1 Step 1: Environment Variables

Create .env.example in root dir:

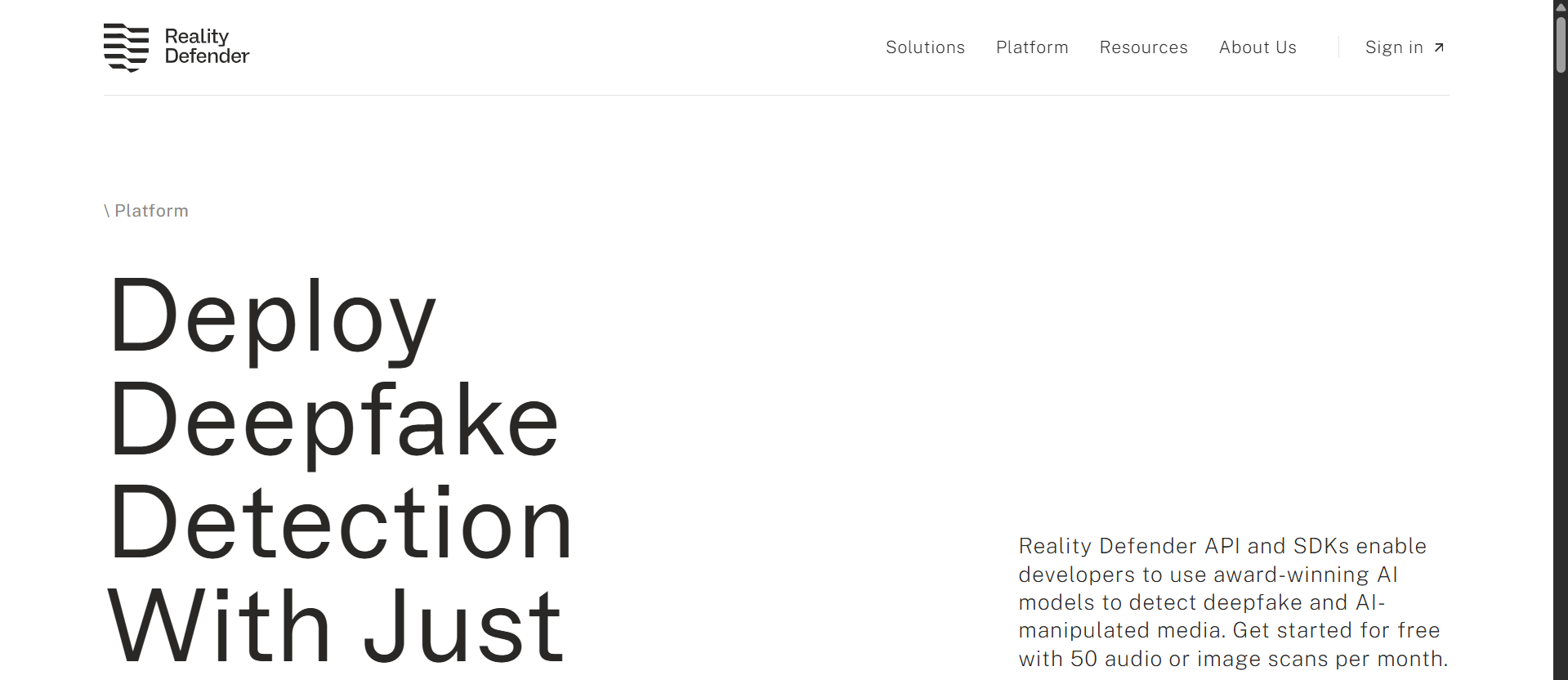
# Reality Defender API Configuration  
# Get your free API key from: https://www.realitydefender.com/platform/api  
# Free tier includes 50 audio or image scans per month  
NEXT\_PUBLIC\_RD\_API\_KEY=your\_reality\_defender\_api\_key\_here  
NEXT\_PUBLIC\_RD\_API\_URL=https://api.prd.realitydefender.xyz  
  
# App Configuration  
NEXT\_PUBLIC\_APP\_NAME="Deepfake Detective"  
NEXT\_PUBLIC\_APP\_DESCRIPTION="Advanced AI-powered deepfake detection for media files"  
  
# Real API required - configure your API key above

Create .env.local (copy from .env.example and add your actual API key):

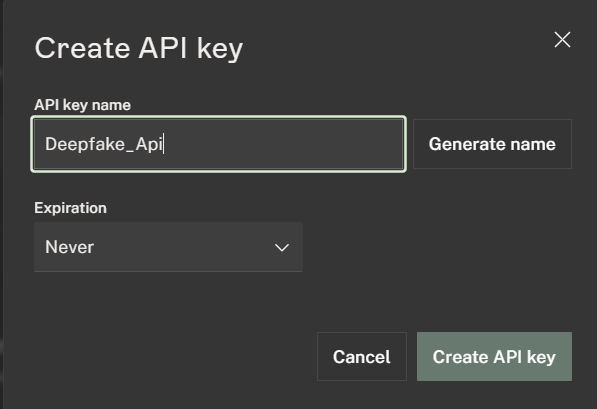
cp .env.example .env.local  
# Edit .env.local with your actual API key

### 1.6.2 Step 2: Get Reality Defender API Key

1. Visit [Reality Defender Platform](https://www.realitydefender.com/platform/api)



1. Sign up for a free account



1. Generate your API key
2. Add it to .env.local

## 1.7 Core Library Implementation

### 1.7.1 Step 1: TypeScript Type Definitions



**src/lib/types/index.ts:**

export interface FileUploadState {

  file: File | null;

  preview: string | null;

  progress: number;

  status: 'idle' | 'uploading' | 'processing' | 'completed' | 'error';

  error?: string;

}

export interface UploadProgress {

  percentage: number;

  stage: 'upload' | 'preprocessing' | 'analysis' | 'results';

  message: string;

  stagesCompleted?: string[];

  analysisDetails?: {

    activeModels: string[];

    completedModels: number;

    totalModels: number;

    modelStatuses: Record<string, string>;

    currentPhase: string;

  };

  timeElapsed?: number;

  estimatedTimeRemaining?: number;

}

export interface ChartDataPoint {

  name: string;

  value: number;

  color?: string;

}

export interface TimelineDataPoint {

  timestamp: number;

  frame?: number;

  confidence: number;

  anomalies?: string[];

}

export interface AudioSegment {

  start: number;

  end: number;

  confidence: number;

  anomalies?: string[];

}

export interface WaveformData {

  data: number[];

  duration: number;

  sampleRate: number;

}

export interface ConfidenceLevel {

  value: number;

  label: 'High' | 'Medium' | 'Low';

  color: string;

  textColor: string;

}

export interface DetectionCategory {

  name: string;

  percentage: number;

  color: string;

  description?: string;

}

export interface MediaMetadata {

  filename: string;

  fileSize: number;

  fileType: string;

  duration?: number;

  resolution?: string;

  codec?: string;

  bitrate?: number;

  createdAt?: string;

}

export interface APIStatus {

  status: 'online' | 'offline' | 'demo\_mode' | 'error';

  version?: string;

  latency?: number;

  lastCheck?: string;

}

export interface ExportOptions {

  format: 'json' | 'pdf' | 'csv';

  includeMetadata: boolean;

  includeCharts: boolean;

  includeTimeline: boolean;

}

export interface NavigationItem {

  href: string;

  label: string;

  icon: string;

  active?: boolean;

}

export interface AlertConfig {

  type: 'success' | 'error' | 'warning' | 'info';

  title: string;

  message: string;

  duration?: number;

  dismissible?: boolean;

}

*// UI Component Props Types*

export interface BaseComponentProps {

  className?: string;

  children?: React.ReactNode;

}

export interface LoadingSpinnerProps extends BaseComponentProps {

  size?: 'sm' | 'md' | 'lg';

  color?: string;

}

export interface EmptyStateProps extends BaseComponentProps {

  icon?: string;

  title: string;

  description?: string;

  action?: {

    label: string;

    onClick: () => void;

  };

}

export interface StatCardProps extends BaseComponentProps {

  title: string;

  value: string | number;

  subtitle?: string;

  icon?: React.ReactNode;

  trend?: {

    value: number;

    isPositive: boolean;

  };

}

*// Animation and Motion Types*

export interface MotionVariants {

  initial: object;

  animate: object;

  exit?: object;

}

export interface PageTransition {

  duration: number;

  ease: string;

}

*// Form and Input Types*

export interface FieldError {

  field: string;

  message: string;

}

export interface FormValidationResult {

  isValid: boolean;

  errors: FieldError[];

}

*// Utility Types*

export type Maybe<T> = T | null | undefined;

export type Optional<T, K extends keyof T> = Omit<T, K> & Partial<Pick<T, K>>;

export type RequiredFields<T, K extends keyof T> = T & Required<Pick<T, K>>;

*// Event Handler Types*

export type FileUploadHandler = (file: File) => void | Promise<void>;

export type AnalysisCompleteHandler = (result: unknown) => void;

export type ThemeChangeHandler = (theme: 'light' | 'dark' | 'system') => void;

*// API Response Types*

export interface APIResponse<T = unknown> {

  success: boolean;

  data?: T;

  error?: {

    code: string;

    message: string;

    details?: unknown;

  };

  meta?: {

    timestamp: string;

    requestId: string;

    version: string;

  };

}

*// Explainability Types*

export interface ExplanationReason {

  id: string;

  category: 'visual' | 'audio' | 'metadata' | 'model' | 'temporal' | 'technical';

  type: 'evidence' | 'anomaly' | 'pattern' | 'inconsistency' | 'artifact';

  severity: 'high' | 'medium' | 'low';

  confidence: number;

  title: string;

  description: string;

  technicalDetails?: string;

  affectedRegions?: Array<{

    startTime?: number;

    endTime?: number;

    frame?: number;

    coordinates?: { x: number; y: number; width: number; height: number; };

  }>;

  supportingEvidence?: string[];

  modelSources?: string[];

}

export interface ExplanationEvidence {

  id: string;

  type: 'visual\_artifact' | 'audio\_distortion' | 'compression\_anomaly' | 'temporal\_inconsistency' | 'statistical\_anomaly';

  location: {

    frame?: number;

    timestamp?: number;

    coordinates?: { x: number; y: number; width: number; height: number; };

    frequency?: number;

  };

  severity: number; *// 0-1 scale*

  description: string;

  visualData?: {

    heatmapData?: number[][];

    highlightRegions?: Array<{ x: number; y: number; width: number; height: number; }>;

    annotationImage?: string;

  };

}

export interface ModelInsight {

  modelName: string;

  modelType: 'deepfake\_detector' | 'face\_analysis' | 'audio\_analysis' | 'compression\_analysis' | 'ensemble';

  confidence: number;

  prediction: 'authentic' | 'manipulated' | 'inconclusive';

  keyFindings: string[];

  technicalScore: number;

  processingTime: number;

  reasoning: string;

  supportingMetrics?: Record<string, number>;

}

export interface ExplanationSummary {

  primaryReason: string;

  secondaryReasons: string[];

  overallConfidence: number;

  authenticityIndicators: Array<{

    factor: string;

    weight: number;

    contribution: 'positive' | 'negative' | 'neutral';

    explanation: string;

  }>;

  riskFactors: Array<{

    factor: string;

    severity: 'critical' | 'high' | 'medium' | 'low';

    likelihood: number;

    impact: string;

  }>;

  recommendedActions?: string[];

}

export interface DetailedExplanation {

  id: string;

  analysisId: string;

  summary: ExplanationSummary;

  reasons: ExplanationReason[];

  evidence: ExplanationEvidence[];

  modelInsights: ModelInsight[];

  temporalAnalysis?: {

    frameByFrameReasons: Array<{

      frame: number;

      timestamp: number;

      primaryConcerns: string[];

      confidenceChange?: number;

    }>;

    overallTrends: string[];

  };

  metadataAnalysis?: {

    fileProperties: Array<{

      property: string;

      expectedValue?: string | number;

      actualValue: string | number;

      assessment: 'normal' | 'suspicious' | 'anomalous';

      explanation: string;

    }>;

    processingHistory?: string[];

  };

  generatedAt: string;

  processingVersion: string;

}

export interface ExplanationConfig {

  includeModelInsights: boolean;

  includeTechnicalDetails: boolean;

  includeVisualEvidence: boolean;

  simplifyForGeneralAudience: boolean;

  maxReasonsToShow: number;

  evidenceVisualization: boolean;

}

*// Feature Flags*

export interface FeatureFlags {

  enableBetaFeatures: boolean;

  enableAdvancedAnalytics: boolean;

  enableExperimentalCharts: boolean;

  enableOfflineMode: boolean;

  enableExplanations: boolean;

  enableAdvancedExplanations: boolean;

}

export const DEFAULT\_FEATURE\_FLAGS: FeatureFlags = {

  enableBetaFeatures: false,

  enableAdvancedAnalytics: true,

  enableExperimentalCharts: false,

  enableOfflineMode: false,

  enableExplanations: true,

  enableAdvancedExplanations: true,

};

### 

### 1.7.2 Step 2: Advanced Storage Management

**src/lib/storage.ts:**

### **import type { AnalysisResult } from './reality-defender';**

### **const STORAGE\_KEYS = {**

### **ANALYSIS\_HISTORY: 'deepfake\_analysis\_history',**

### **THEME: 'deepfake\_theme',**

### **USER\_PREFERENCES: 'deepfake\_preferences',**

### **} as const;**

### **export interface StoredAnalysis extends AnalysisResult {**

### **thumbnailBlob?: string; *// Base64 encoded thumbnail***

### **}**

### **export interface UserPreferences {**

### **maxHistoryItems: number;**

### **showDetailedAnalysis: boolean;**

### **autoDownloadReports: boolean;**

### **theme: 'light' | 'dark' | 'system';**

### **}**

### **const DEFAULT\_PREFERENCES: UserPreferences = {**

### **maxHistoryItems: 10,**

### **showDetailedAnalysis: true,**

### **autoDownloadReports: false,**

### **theme: 'system',**

### **};**

### **class StorageManager {**

### **private isClient = typeof window !== 'undefined';**

### ***// Analysis History Management***

### **getAnalysisHistory(): StoredAnalysis[] {**

### **if (!*this*.isClient) return [];**

### 

### **try {**

### **const stored = localStorage.getItem(STORAGE\_KEYS.ANALYSIS\_HISTORY);**

### **return stored ? JSON.parse(stored) : [];**

### **} catch (error) {**

### **console.error('Failed to load analysis history:', error);**

### **return [];**

### **}**

### **}**

### **addAnalysisToHistory(analysis: AnalysisResult, thumbnailBlob?: string): void {**

### **if (!*this*.isClient) return;**

### **try {**

### **const history = *this*.getAnalysisHistory();**

### **const preferences = *this*.getUserPreferences();**

### 

### **const storedAnalysis: StoredAnalysis = {**

### **...analysis,**

### **thumbnailBlob,**

### **};**

### ***// Add to beginning of array (most recent first)***

### **const updatedHistory = [storedAnalysis, ...history];**

### 

### ***// Limit history size***

### **const limitedHistory = updatedHistory.slice(0, preferences.maxHistoryItems);**

### 

### **localStorage.setItem(STORAGE\_KEYS.ANALYSIS\_HISTORY, JSON.stringify(limitedHistory));**

### **} catch (error) {**

### **console.error('Failed to save analysis to history:', error);**

### **}**

### **}**

### **removeAnalysisFromHistory(analysisId: string): void {**

### **if (!*this*.isClient) return;**

### **try {**

### **const history = *this*.getAnalysisHistory();**

### **const filteredHistory = history.filter(analysis => analysis.id !== analysisId);**

### **localStorage.setItem(STORAGE\_KEYS.ANALYSIS\_HISTORY, JSON.stringify(filteredHistory));**

### **} catch (error) {**

### **console.error('Failed to remove analysis from history:', error);**

### **}**

### **}**

### **clearAnalysisHistory(): void {**

### **if (!*this*.isClient) return;**

### 

### **try {**

### **localStorage.removeItem(STORAGE\_KEYS.ANALYSIS\_HISTORY);**

### **} catch (error) {**

### **console.error('Failed to clear analysis history:', error);**

### **}**

### **}**

### **getAnalysisById(analysisId: string): StoredAnalysis | null {**

### **const history = *this*.getAnalysisHistory();**

### **return history.find(analysis => analysis.id === analysisId) || null;**

### **}**

### ***// User Preferences Management***

### **getUserPreferences(): UserPreferences {**

### **if (!*this*.isClient) return DEFAULT\_PREFERENCES;**

### **try {**

### **const stored = localStorage.getItem(STORAGE\_KEYS.USER\_PREFERENCES);**

### **if (stored) {**

### **const parsed = JSON.parse(stored);**

### ***// Merge with defaults to ensure all properties exist***

### **return { ...DEFAULT\_PREFERENCES, ...parsed };**

### **}**

### **} catch (error) {**

### **console.error('Failed to load user preferences:', error);**

### **}**

### 

### **return DEFAULT\_PREFERENCES;**

### **}**

### **updateUserPreferences(preferences: Partial<UserPreferences>): void {**

### **if (!*this*.isClient) return;**

### **try {**

### **const current = *this*.getUserPreferences();**

### **const updated = { ...current, ...preferences };**

### **localStorage.setItem(STORAGE\_KEYS.USER\_PREFERENCES, JSON.stringify(updated));**

### **} catch (error) {**

### **console.error('Failed to save user preferences:', error);**

### **}**

### **}**

### ***// Theme Management***

### **getTheme(): 'light' | 'dark' | 'system' {**

### **if (!*this*.isClient) return 'system';**

### **try {**

### **const stored = localStorage.getItem(STORAGE\_KEYS.THEME);**

### **if (stored && ['light', 'dark', 'system'].includes(stored)) {**

### **return stored as 'light' | 'dark' | 'system';**

### **}**

### **} catch (error) {**

### **console.error('Failed to load theme preference:', error);**

### **}**

### 

### **return 'system';**

### **}**

### **setTheme(theme: 'light' | 'dark' | 'system'): void {**

### **if (!*this*.isClient) return;**

### **try {**

### **localStorage.setItem(STORAGE\_KEYS.THEME, theme);**

### ***this*.updateUserPreferences({ theme });**

### **} catch (error) {**

### **console.error('Failed to save theme preference:', error);**

### **}**

### **}**

### ***// Data Export/Import***

### **exportData(): string {**

### **if (!*this*.isClient) return '';**

### **const data = {**

### **analysisHistory: *this*.getAnalysisHistory(),**

### **userPreferences: *this*.getUserPreferences(),**

### **theme: *this*.getTheme(),**

### **exportDate: new Date().toISOString(),**

### **};**

### **return JSON.stringify(data, null, 2);**

### **}**

### **importData(jsonData: string): { success: boolean; message: string } {**

### **if (!*this*.isClient) {**

### **return { success: false, message: 'Import not available on server side' };**

### **}**

### **try {**

### **const data = JSON.parse(jsonData);**

### 

### **if (data.analysisHistory) {**

### **localStorage.setItem(STORAGE\_KEYS.ANALYSIS\_HISTORY, JSON.stringify(data.analysisHistory));**

### **}**

### 

### **if (data.userPreferences) {**

### **localStorage.setItem(STORAGE\_KEYS.USER\_PREFERENCES, JSON.stringify(data.userPreferences));**

### **}**

### 

### **if (data.theme) {**

### **localStorage.setItem(STORAGE\_KEYS.THEME, data.theme);**

### **}**

### **return { success: true, message: 'Data imported successfully' };**

### **} catch (error) {**

### **console.error('Failed to import data:', error);**

### **return { success: false, message: 'Invalid data format' };**

### **}**

### **}**

### ***// Storage Statistics***

### **getStorageStats(): {**

### **historyCount: number;**

### **estimatedSize: string;**

### **lastModified?: string;**

### **} {**

### **if (!*this*.isClient) {**

### **return { historyCount: 0, estimatedSize: '0 KB' };**

### **}**

### **try {**

### **const history = *this*.getAnalysisHistory();**

### **const dataStr = localStorage.getItem(STORAGE\_KEYS.ANALYSIS\_HISTORY) || '';**

### **const sizeInBytes = new Blob([dataStr]).size;**

### **const sizeInKB = Math.round(sizeInBytes / 1024 \* 100) / 100;**

### 

### **const lastModified = history.length > 0**

### **? history[0].timestamp**

### **: undefined;**

### **return {**

### **historyCount: history.length,**

### **estimatedSize: `${sizeInKB} KB`,**

### **lastModified,**

### **};**

### **} catch (error) {**

### **console.error('Failed to calculate storage stats:', error);**

### **return { historyCount: 0, estimatedSize: '0 KB' };**

### **}**

### **}**

### **}**

### **export const storage = new StorageManager();**

### ***// Utility functions for file handling***

### **export const createThumbnail = (file: File): Promise<string> => {**

### **return new Promise((resolve, reject) => {**

### **if (file.type.startsWith('image/')) {**

### **const reader = new FileReader();**

### **reader.onload = (e) => {**

### **const img = new Image();**

### **img.onload = () => {**

### **const canvas = document.createElement('canvas');**

### **const ctx = canvas.getContext('2d');**

### 

### ***// Set thumbnail dimensions***

### **const maxSize = 200;**

### **let { width, height } = img;**

### 

### **if (width > height) {**

### **if (width > maxSize) {**

### **height = (height \* maxSize) / width;**

### **width = maxSize;**

### **}**

### **} else {**

### **if (height > maxSize) {**

### **width = (width \* maxSize) / height;**

### **height = maxSize;**

### **}**

### **}**

### 

### **canvas.width = width;**

### **canvas.height = height;**

### 

### **ctx?.drawImage(img, 0, 0, width, height);**

### **resolve(canvas.toDataURL('image/jpeg', 0.7));**

### **};**

### **img.onerror = reject;**

### **img.src = e.target?.result as string;**

### **};**

### **reader.onerror = reject;**

### **reader.readAsDataURL(file);**

### **} else if (file.type.startsWith('video/')) {**

### **const video = document.createElement('video');**

### **video.preload = 'metadata';**

### **video.onloadedmetadata = () => {**

### **video.currentTime = Math.min(video.duration \* 0.1, 5); *// 10% or 5 seconds***

### **};**

### **video.onseeked = () => {**

### **const canvas = document.createElement('canvas');**

### **const ctx = canvas.getContext('2d');**

### 

### **canvas.width = Math.min(video.videoWidth, 200);**

### **canvas.height = Math.min(video.videoHeight, 200);**

### 

### **ctx?.drawImage(video, 0, 0, canvas.width, canvas.height);**

### **resolve(canvas.toDataURL('image/jpeg', 0.7));**

### **};**

### **video.onerror = reject;**

### **video.src = URL.createObjectURL(file);**

### **} else {**

### ***// For audio and other files, return a placeholder***

### **resolve('data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iMjAwIiBoZWlnaHQ9IjIwMCIgdmlld0JveD0iMCAwIDIwMCAyMDAiIGZpbGw9Im5vbmUiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyI+PHJlY3Qgd2lkdGg9IjIwMCIgaGVpZ2h0PSIyMDAiIGZpbGw9IiNmM2Y0ZjYiLz48cGF0aCBkPSJNMTAwIDUwSDEyMFY3MEgxMDBWNTBaTTEwMCA4MEgxMjBWMTAwSDEwMFY4MFpNMTAwIDExMEgxMjBWMTMwSDEwMFYxMTBaTTEwMCAxNDBIMTIwVjE2MEgxMDBWMTQwWiIgZmlsbD0iIzZiNzI4MCIvPjwvc3ZnPg==');**

### **}**

### **});**

### **};**

### **export const formatFileSize = (bytes: number): string => {**

### **if (bytes === 0) return '0 Bytes';**

### 

### **const k = 1024;**

### **const sizes = ['Bytes', 'KB', 'MB', 'GB'];**

### **const i = Math.floor(Math.log(bytes) / Math.log(k));**

### 

### **return parseFloat((bytes / Math.pow(k, i)).toFixed(2)) + ' ' + sizes[i];**

### **};**

### **export const getFileIcon = (fileType: string): string => {**

### **if (fileType.startsWith('image/')) return '🖼️';**

### **if (fileType.startsWith('video/')) return '🎥';**

### **if (fileType.startsWith('audio/')) return '🎵';**

### **return '📄';**

### **};**

### 1.7.3 Step 3: Advanced Utility Functions

**src/lib/utils.ts:**

import { clsx, type ClassValue } from "clsx"  
import { twMerge } from "tailwind-merge"  
  
export function cn(...inputs: ClassValue[]) {  
 return twMerge(clsx(inputs))  
}

**src\lib\explanation-generator.ts:**

import {

  DetailedExplanation,

  ExplanationReason,

  ExplanationEvidence,

  ModelInsight,

  ExplanationSummary,

  ExplanationConfig

} from './types';

import type { AnalysisResult } from './reality-defender';

*/\*\**

*\* Generates human-readable explanations for deepfake detection results*

*\*/*

export class ExplanationGenerator {

  private config: ExplanationConfig;

  constructor(config: Partial<ExplanationConfig> = {}) {

*this*.config = {

      includeModelInsights: true,

      includeTechnicalDetails: true,

      includeVisualEvidence: true,

      simplifyForGeneralAudience: false,

      maxReasonsToShow: 10,

      evidenceVisualization: true,

      ...config

    };

  }

*/\*\**

*\* Generate complete explanation from analysis result*

*\*/*

  generateExplanation(analysis: AnalysisResult): DetailedExplanation {

    const reasons = *this*.generateReasons(analysis);

    const evidence = *this*.generateEvidence(analysis);

    const modelInsights = *this*.generateModelInsights(analysis);

    const summary = *this*.generateSummary(analysis, reasons, modelInsights);

    return {

      id: `exp\_${analysis.id}`,

      analysisId: analysis.id,

      summary,

      reasons: reasons.slice(0, *this*.config.maxReasonsToShow),

      evidence,

      modelInsights,

      temporalAnalysis: *this*.generateTemporalAnalysis(analysis),

      metadataAnalysis: *this*.generateMetadataAnalysis(analysis),

      generatedAt: new Date().toISOString(),

      processingVersion: '1.0.0'

    };

  }

*/\*\**

*\* Generate primary reasons for the classification*

*\*/*

  private generateReasons(analysis: AnalysisResult): ExplanationReason[] {

    const reasons: ExplanationReason[] = [];

    const confidence = analysis.confidence;

    const prediction = analysis.prediction;

*// Primary classification reason*

    if (prediction === 'authentic') {

      reasons.push({

        id: 'auth\_primary',

        category: 'model',

        type: 'evidence',

        severity: confidence < 0.2 ? 'low' : 'medium',

        confidence: 1 - confidence,

        title: 'Content appears authentic',

        description: `AI analysis indicates this content is likely genuine with ${Math.round((1 - confidence) \* 100)}% confidence. Multiple detection models found no significant signs of manipulation.`,

        technicalDetails: `Ensemble model score: ${confidence.toFixed(3)}, indicating low manipulation probability.`,

        modelSources: ['ensemble', 'deepfake\_detector']

      });

*// Add supporting authentic indicators*

      if (confidence < 0.1) {

        reasons.push({

          id: 'auth\_strong',

          category: 'technical',

          type: 'evidence',

          severity: 'low',

          confidence: 0.95,

          title: 'Strong authenticity indicators',

          description: 'Analysis found consistent patterns typical of genuine content with no detectable artifacts.',

          technicalDetails: 'Low variance in model predictions, consistent temporal patterns, normal compression artifacts.'

        });

      }

    }

    if (prediction === 'manipulated') {

      reasons.push({

        id: 'manip\_primary',

        category: 'model',

        type: 'anomaly',

        severity: confidence > 0.8 ? 'high' : confidence > 0.5 ? 'medium' : 'low',

        confidence,

        title: 'Potential manipulation detected',

        description: `AI analysis detected signs of possible manipulation with ${Math.round(confidence \* 100)}% confidence. Multiple indicators suggest this content may be synthetically generated or altered.`,

        technicalDetails: `Ensemble model score: ${confidence.toFixed(3)}, exceeding manipulation threshold.`,

        modelSources: ['ensemble', 'deepfake\_detector']

      });

*// Add specific manipulation indicators based on confidence level*

      if (confidence > 0.8) {

        reasons.push({

          id: 'manip\_strong',

          category: 'visual',

          type: 'artifact',

          severity: 'high',

          confidence: 0.9,

          title: 'Strong manipulation indicators',

          description: 'High-confidence detection of typical deepfake artifacts and inconsistencies.',

          technicalDetails: 'Multiple models detected convergent evidence of synthetic generation patterns.'

        });

      }

      if (confidence > 0.6) {

        reasons.push({

          id: 'manip\_patterns',

          category: 'temporal',

          type: 'pattern',

          severity: 'medium',

          confidence: 0.75,

          title: 'Suspicious temporal patterns',

          description: 'Analysis detected frame-to-frame inconsistencies common in generated content.',

          technicalDetails: 'Temporal coherence metrics indicate potential frame interpolation or synthesis.'

        });

      }

    }

    if (prediction === 'inconclusive') {

      reasons.push({

        id: 'inconcl\_primary',

        category: 'model',

        type: 'inconsistency',

        severity: 'medium',

        confidence: Math.abs(0.5 - confidence) \* 2,

        title: 'Analysis inconclusive',

        description: `The analysis could not determine with high confidence whether this content is authentic or manipulated. Confidence score: ${Math.round(confidence \* 100)}%.`,

        technicalDetails: `Score ${confidence.toFixed(3)} falls in the inconclusive range (0.3-0.7). Mixed signals from different detection models.`,

        modelSources: ['ensemble']

      });

    }

*// Add file-type specific reasons*

*this*.addFileTypeSpecificReasons(analysis, reasons);

*// Add metadata-based reasons*

*this*.addMetadataReasons(analysis, reasons);

    return reasons.sort((a, b) => b.confidence - a.confidence);

  }

  private addFileTypeSpecificReasons(analysis: AnalysisResult, reasons: ExplanationReason[]) {

    const fileType = analysis.fileType;

    if (fileType.startsWith('image/')) {

      reasons.push({

        id: 'img\_analysis',

        category: 'visual',

        type: 'evidence',

        severity: 'medium',

        confidence: 0.8,

        title: 'Image analysis completed',

        description: 'Comprehensive pixel-level analysis performed looking for manipulation artifacts, inconsistent lighting, and synthetic generation patterns.',

        technicalDetails: 'Face detection, compression analysis, and artifact detection models applied.'

      });

    }

    if (fileType.startsWith('video/')) {

      reasons.push({

        id: 'video\_analysis',

        category: 'temporal',

        type: 'evidence',

        severity: 'medium',

        confidence: 0.85,

        title: 'Video temporal analysis',

        description: 'Frame-by-frame analysis examining temporal consistency, lip-sync accuracy, and motion patterns.',

        technicalDetails: 'Temporal coherence models analyzed inter-frame relationships and motion vectors.'

      });

    }

    if (fileType.startsWith('audio/')) {

      reasons.push({

        id: 'audio\_analysis',

        category: 'audio',

        type: 'evidence',

        severity: 'medium',

        confidence: 0.8,

        title: 'Audio analysis completed',

        description: 'Spectral analysis examining voice patterns, synthetic speech indicators, and audio artifacts.',

        technicalDetails: 'Voice cloning detection and audio synthesis pattern analysis applied.'

      });

    }

  }

  private addMetadataReasons(analysis: AnalysisResult, reasons: ExplanationReason[]) {

    const metadata = analysis.details.metadata;

    if (metadata.codec) {

      const isUnusualCodec = ['h264', 'h265', 'avc1'].every(common =>

        !metadata.codec?.toLowerCase().includes(common)

      );

      if (isUnusualCodec) {

        reasons.push({

          id: 'codec\_unusual',

          category: 'metadata',

          type: 'anomaly',

          severity: 'low',

          confidence: 0.3,

          title: 'Unusual codec detected',

          description: `File uses codec '${metadata.codec}' which is less common for typical media files.`,

          technicalDetails: 'Codec analysis for synthetic generation indicators.'

        });

      }

    }

    if (metadata.bitrate && metadata.bitrate < 1000) {

      reasons.push({

        id: 'bitrate\_low',

        category: 'metadata',

        type: 'anomaly',

        severity: 'low',

        confidence: 0.25,

        title: 'Low bitrate detected',

        description: 'Unusually low bitrate may indicate heavy compression or synthetic generation.',

        technicalDetails: `Bitrate: ${metadata.bitrate} kbps - below typical range for quality media.`

      });

    }

  }

  private generateEvidence(analysis: AnalysisResult): ExplanationEvidence[] {

    const evidence: ExplanationEvidence[] = [];

*// Generate evidence based on frame analysis if available*

    if (analysis.details.frameAnalysis) {

      analysis.details.frameAnalysis.forEach((frame, index) => {

        if (frame.anomalies && frame.anomalies.length > 0) {

          evidence.push({

            id: `frame\_${frame.frame}`,

            type: 'visual\_artifact',

            location: {

              frame: frame.frame,

              timestamp: frame.timestamp

            },

            severity: frame.confidence,

            description: `Frame ${frame.frame}: ${frame.anomalies.join(', ')}`

          });

        }

      });

    }

*// Generate evidence from audio analysis*

    if (analysis.details.audioAnalysis) {

      analysis.details.audioAnalysis.segments.forEach((segment, index) => {

        if (segment.anomalies && segment.anomalies.length > 0) {

          evidence.push({

            id: `audio\_${index}`,

            type: 'audio\_distortion',

            location: {

              timestamp: segment.start

            },

            severity: segment.confidence,

            description: `Audio segment ${segment.start}s-${segment.end}s: ${segment.anomalies.join(', ')}`

          });

        }

      });

    }

    return evidence;

  }

  private generateModelInsights(analysis: AnalysisResult): ModelInsight[] {

    const insights: ModelInsight[] = [];

*// Create insights from the analysis result*

    insights.push({

      modelName: 'Reality Defender Ensemble',

      modelType: 'ensemble',

      confidence: analysis.confidence,

      prediction: analysis.prediction,

      keyFindings: *this*.generateKeyFindings(analysis),

      technicalScore: analysis.confidence,

      processingTime: analysis.processingTime,

      reasoning: *this*.generateModelReasoning(analysis)

    });

    return insights;

  }

  private generateKeyFindings(analysis: AnalysisResult): string[] {

    const findings: string[] = [];

    const confidence = analysis.confidence;

    if (analysis.prediction === 'authentic') {

      findings.push('No significant manipulation artifacts detected');

      findings.push('Consistent patterns throughout media');

      if (confidence < 0.1) findings.push('Strong authenticity indicators present');

    }

    if (analysis.prediction === 'manipulated') {

      findings.push('Synthetic generation patterns detected');

      if (confidence > 0.8) findings.push('High-confidence manipulation indicators');

      if (confidence > 0.6) findings.push('Multiple detection models agree');

    }

    if (analysis.prediction === 'inconclusive') {

      findings.push('Mixed signals from different analysis methods');

      findings.push('Requires manual review for definitive assessment');

    }

*// Add file-specific findings*

    if (analysis.fileType.startsWith('video/')) {

      findings.push('Temporal analysis completed');

    }

    if (analysis.fileType.startsWith('audio/')) {

      findings.push('Spectral analysis performed');

    }

    return findings;

  }

  private generateModelReasoning(analysis: AnalysisResult): string {

    const confidence = analysis.confidence;

    const prediction = analysis.prediction;

    if (prediction === 'authentic') {

      return `The ensemble model analyzed this content and found it to be authentic with ${Math.round((1-confidence) \* 100)}% confidence. The analysis considered multiple factors including pixel-level patterns, compression characteristics, and temporal consistency. No significant indicators of synthetic generation or manipulation were detected.`;

    }

    if (prediction === 'manipulated') {

      return `The ensemble model detected potential manipulation with ${Math.round(confidence \* 100)}% confidence. The analysis identified patterns consistent with synthetic generation, including anomalous artifacts and inconsistencies that are typical of AI-generated or heavily edited content.`;

    }

    return `The analysis resulted in an inconclusive determination. The confidence score of ${Math.round(confidence \* 100)}% falls in the uncertain range, indicating mixed signals from different detection methods. This could be due to edge cases, novel generation techniques, or ambiguous content characteristics.`;

  }

  private generateSummary(

    analysis: AnalysisResult,

    reasons: ExplanationReason[],

    insights: ModelInsight[]

  ): ExplanationSummary {

    const primaryReason = reasons[0]?.description || 'Analysis completed';

    const secondaryReasons = reasons.slice(1, 4).map(r => r.title);

    const authenticityIndicators = *this*.generateAuthenticityIndicators(analysis);

    const riskFactors = *this*.generateRiskFactors(analysis, reasons);

    const recommendedActions = *this*.generateRecommendations(analysis);

    return {

      primaryReason,

      secondaryReasons,

      overallConfidence: analysis.confidence,

      authenticityIndicators,

      riskFactors,

      recommendedActions

    };

  }

  private generateAuthenticityIndicators(analysis: AnalysisResult): Array<{

    factor: string;

    weight: number;

    contribution: 'positive' | 'negative' | 'neutral';

    explanation: string;

  }> {

    const indicators: Array<{

      factor: string;

      weight: number;

      contribution: 'positive' | 'negative' | 'neutral';

      explanation: string;

    }> = [];

    const confidence = analysis.confidence;

    indicators.push({

      factor: 'AI Model Analysis',

      weight: 0.8,

      contribution: analysis.prediction === 'authentic' ? 'positive' :

                   analysis.prediction === 'manipulated' ? 'negative' : 'neutral',

      explanation: `Primary AI detection model with ${Math.round(confidence \* 100)}% confidence`

    });

    indicators.push({

      factor: 'Technical Analysis',

      weight: 0.6,

      contribution: 'neutral',

      explanation: 'Comprehensive technical analysis of file properties and metadata'

    });

    if (analysis.details.frameAnalysis) {

      indicators.push({

        factor: 'Temporal Consistency',

        weight: 0.7,

        contribution: confidence < 0.5 ? 'positive' : 'negative',

        explanation: 'Frame-by-frame analysis for temporal consistency patterns'

      });

    }

    return indicators;

  }

  private generateRiskFactors(

    analysis: AnalysisResult,

    reasons: ExplanationReason[]

  ): Array<{

    factor: string;

    severity: 'critical' | 'high' | 'medium' | 'low';

    likelihood: number;

    impact: string;

  }> {

    const riskFactors: Array<{

      factor: string;

      severity: 'critical' | 'high' | 'medium' | 'low';

      likelihood: number;

      impact: string;

    }> = [];

    const confidence = analysis.confidence;

    if (analysis.prediction === 'manipulated') {

      riskFactors.push({

        factor: 'Synthetic Content Detection',

        severity: confidence > 0.8 ? 'critical' : confidence > 0.6 ? 'high' : 'medium',

        likelihood: confidence,

        impact: 'Content may be artificially generated or manipulated'

      });

    }

    if (analysis.prediction === 'inconclusive') {

      riskFactors.push({

        factor: 'Uncertain Analysis',

        severity: 'medium',

        likelihood: Math.abs(0.5 - confidence) \* 2,

        impact: 'Cannot determine authenticity with high confidence'

      });

    }

*// Add specific risk factors from reasons*

    const highSeverityReasons = reasons.filter(r => r.severity === 'high');

    highSeverityReasons.forEach(reason => {

      riskFactors.push({

        factor: reason.title,

        severity: 'high',

        likelihood: reason.confidence,

        impact: reason.description

      });

    });

    return riskFactors;

  }

  private generateRecommendations(analysis: AnalysisResult): string[] {

    const recommendations = [];

    const confidence = analysis.confidence;

    if (analysis.prediction === 'manipulated') {

      recommendations.push('Exercise caution when sharing or using this content');

      recommendations.push('Consider additional verification from independent sources');

      if (confidence > 0.8) {

        recommendations.push('High probability of manipulation - avoid using without verification');

      }

    }

    if (analysis.prediction === 'inconclusive') {

      recommendations.push('Seek additional expert analysis or verification');

      recommendations.push('Use caution and clearly label as unverified if sharing');

      recommendations.push('Consider analyzing with additional tools or methods');

    }

    if (analysis.prediction === 'authentic' && confidence < 0.2) {

      recommendations.push('Content appears genuine with high confidence');

    }

    recommendations.push('Keep original file metadata when possible for future reference');

    return recommendations;

  }

  private generateTemporalAnalysis(analysis: AnalysisResult) {

    if (!analysis.details.frameAnalysis) return undefined;

    const frameByFrameReasons = analysis.details.frameAnalysis.map(frame => ({

      frame: frame.frame,

      timestamp: frame.timestamp,

      primaryConcerns: frame.anomalies || [],

      confidenceChange: frame.confidence - analysis.confidence

    }));

    const overallTrends = [

      'Analyzed temporal consistency across all frames',

      frameByFrameReasons.length > 0 ? 'Some frames show anomalies' : 'No significant anomalies detected',

      'Frame-to-frame analysis completed'

    ];

    return {

      frameByFrameReasons,

      overallTrends

    };

  }

  private generateMetadataAnalysis(analysis: AnalysisResult) {

    const metadata = analysis.details.metadata;

    const fileProperties: Array<{

      property: string;

      expectedValue?: string | number;

      actualValue: string | number;

      assessment: 'normal' | 'suspicious' | 'anomalous';

      explanation: string;

    }> = [];

    if (metadata.duration) {

      fileProperties.push({

        property: 'Duration',

        actualValue: `${metadata.duration}s`,

        assessment: metadata.duration > 0 ? 'normal' : 'suspicious',

        explanation: 'Media duration within expected range'

      });

    }

    if (metadata.resolution) {

      fileProperties.push({

        property: 'Resolution',

        actualValue: metadata.resolution,

        assessment: 'normal',

        explanation: 'Standard resolution format detected'

      });

    }

    if (metadata.codec) {

      const commonCodecs = ['h264', 'h265', 'avc1', 'mp4', 'mp3'];

      const isCommon = commonCodecs.some(codec =>

        metadata.codec!.toLowerCase().includes(codec)

      );

      fileProperties.push({

        property: 'Codec',

        actualValue: metadata.codec,

        assessment: isCommon ? 'normal' : 'suspicious',

        explanation: isCommon ? 'Standard codec format' : 'Unusual codec may indicate processing'

      });

    }

    return {

      fileProperties,

      processingHistory: ['File uploaded and analyzed via Reality Defender API']

    };

  }

}

*// Export singleton instance with default config*

export const explanationGenerator = new ExplanationGenerator();

**src\lib\pdf-generator.ts**

import jsPDF from 'jspdf';

import html2canvas from 'html2canvas';

import { StoredAnalysis } from '@/lib/storage';

export interface PDFGenerationOptions {

  includeCharts?: boolean;

  includeTimeline?: boolean;

  includeAdvancedCharts?: boolean;

  includeRawData?: boolean;

  watermark?: string;

}

export class PDFReportGenerator {

  private pdf: jsPDF;

  private currentY: number = 20;

  private pageHeight: number = 297; *// A4 height in mm*

  private pageWidth: number = 210; *// A4 width in mm*

  private margin: number = 20;

  private lineHeight: number = 7;

  constructor() {

*this*.pdf = new jsPDF('p', 'mm', 'a4');

*this*.pdf.setFontSize(12);

  }

  private checkPageBreak(height: number = 10): void {

    if (*this*.currentY + height > *this*.pageHeight - *this*.margin) {

*this*.pdf.addPage();

*this*.currentY = *this*.margin;

    }

  }

  private addText(text: string, x: number = *this*.margin, fontSize: number = 12, isBold: boolean = false): void {

*this*.pdf.setFontSize(fontSize);

    if (isBold) {

*this*.pdf.setFont(undefined, 'bold');

    } else {

*this*.pdf.setFont(undefined, 'normal');

    }

*// Handle text wrapping*

    const splitText = *this*.pdf.splitTextToSize(text, *this*.pageWidth - 2 \* *this*.margin);

    for (const line of splitText) {

*this*.checkPageBreak();

*this*.pdf.text(line, x, *this*.currentY);

*this*.currentY += *this*.lineHeight;

    }

  }

  private addTitle(text: string, fontSize: number = 16): void {

*this*.checkPageBreak(15);

*this*.currentY += 5;

*this*.addText(text, *this*.margin, fontSize, true);

*this*.currentY += 5;

  }

  private addSection(title: string, content: string): void {

*this*.addTitle(title, 14);

*this*.addText(content);

*this*.currentY += 5;

  }

  private async addChartFromElement(elementId: string, title: string): Promise<void> {

    const element = document.getElementById(elementId);

    if (!element) {

      console.warn(`Element with ID ${elementId} not found for PDF export`);

*this*.addTitle(title, 12);

*this*.addText(`[Chart could not be rendered: ${title} - Element not found]`, *this*.margin, 10);

      return;

    }

    try {

*// Add title for the chart*

*this*.addTitle(title, 12);

*// Wait for charts to render completely*

      await new Promise(resolve => setTimeout(resolve, 500));

*// Temporarily make element visible for capture*

      const originalStyle = {

        position: element.style.position,

        left: element.style.left,

        opacity: element.style.opacity,

        zIndex: element.style.zIndex,

        pointerEvents: element.style.pointerEvents,

      };

      element.style.position = 'fixed';

      element.style.left = '0px';

      element.style.opacity = '1';

      element.style.zIndex = '9999';

      element.style.pointerEvents = 'auto';

*// Wait a bit more for the repositioning*

      await new Promise(resolve => setTimeout(resolve, 200));

      const canvas = await html2canvas(element, {

        backgroundColor: '#ffffff',

        scale: 1.5,

        logging: true,

        useCORS: true,

        allowTaint: true,

        width: 800,

        height: 400,

        onclone: (clonedDoc) => {

*// Ensure the cloned element is visible*

          const clonedElement = clonedDoc.getElementById(elementId);

          if (clonedElement) {

            clonedElement.style.position = 'static';

            clonedElement.style.opacity = '1';

            clonedElement.style.visibility = 'visible';

          }

        }

      });

*// Restore original styles*

      element.style.position = originalStyle.position;

      element.style.left = originalStyle.left;

      element.style.opacity = originalStyle.opacity;

      element.style.zIndex = originalStyle.zIndex;

      element.style.pointerEvents = originalStyle.pointerEvents;

      const imgData = canvas.toDataURL('image/png');

      const imgWidth = *this*.pageWidth - 2 \* *this*.margin;

      const imgHeight = (canvas.height \* imgWidth) / canvas.width;

*// Check if we need a new page for the image*

*this*.checkPageBreak(imgHeight + 10);

*this*.pdf.addImage(imgData, 'PNG', *this*.margin, *this*.currentY, imgWidth, imgHeight);

*this*.currentY += imgHeight + 10;

    } catch (error) {

      console.error('Failed to add chart to PDF:', error);

      const errorMessage = error instanceof Error ? error.message : 'Unknown error';

*this*.addText(`[Chart could not be rendered: ${title} - ${errorMessage}]`, *this*.margin, 10);

    }

  }

  private addHeader(analysis: StoredAnalysis): void {

*// Company/App header*

*this*.pdf.setFontSize(20);

*this*.pdf.setFont(undefined, 'bold');

*this*.pdf.text('ITL DeepFake Detection Report', *this*.pageWidth / 2, 15, { align: 'center' });

*this*.currentY = 30;

*// File info header*

*this*.pdf.setFontSize(16);

*this*.pdf.setFont(undefined, 'bold');

*this*.pdf.text(`Analysis Report: ${analysis.filename}`, *this*.margin, *this*.currentY);

*this*.currentY += 10;

*this*.pdf.setFontSize(10);

*this*.pdf.setFont(undefined, 'normal');

*this*.pdf.text(`Generated on: ${new Date().toLocaleDateString()} at ${new Date().toLocaleTimeString()}`, *this*.margin, *this*.currentY);

*this*.currentY += 5;

*this*.pdf.text(`Analysis Date: ${new Date(analysis.timestamp).toLocaleDateString()}`, *this*.margin, *this*.currentY);

*this*.currentY += 10;

*// Add a separator line*

*this*.pdf.line(*this*.margin, *this*.currentY, *this*.pageWidth - *this*.margin, *this*.currentY);

*this*.currentY += 10;

  }

  private addFooter(): void {

    const pageCount = *this*.pdf.internal.getNumberOfPages();

    for (let i = 1; i <= pageCount; i++) {

*this*.pdf.setPage(i);

*this*.pdf.setFontSize(8);

*this*.pdf.setFont(undefined, 'normal');

*this*.pdf.text(

        `Page ${i} of ${pageCount} | Generated by ITL DeepFake Detection System`,

*this*.pageWidth / 2,

*this*.pageHeight - 10,

        { align: 'center' }

      );

    }

  }

  private addExecutiveSummary(analysis: StoredAnalysis): void {

*this*.addTitle('Executive Summary', 16);

    const riskLevel = analysis.confidence < 0.3 ? 'LOW' : analysis.confidence < 0.7 ? 'MODERATE' : 'HIGH';

    const riskColor = analysis.confidence < 0.3 ? 'green' : analysis.confidence < 0.7 ? 'orange' : 'red';

    const summary = `

File: ${analysis.filename}

Risk Level: ${riskLevel} (${Math.round(analysis.confidence \* 100)}% confidence)

Prediction: ${analysis.prediction}

Processing Time: ${Math.round(analysis.processingTime / 1000)} seconds

File Size: ${(analysis.fileSize / (1024 \* 1024)).toFixed(2)} MB

File Type: ${analysis.fileType}

Analysis Results:

The file has been analyzed using advanced deepfake detection algorithms. Based on the analysis, this content has a ${riskLevel.toLowerCase()} probability of being artificially generated or manipulated. The confidence score of ${Math.round(analysis.confidence \* 100)}% indicates the system's certainty in this assessment.

${analysis.prediction === 'authentic'

  ? 'The content appears to be authentic with no significant signs of artificial manipulation detected.'

  : 'The content shows signs of potential artificial generation or manipulation and should be treated with caution.'

}

`;

*this*.addText(summary);

  }

  private addTechnicalDetails(analysis: StoredAnalysis): void {

*this*.addTitle('Technical Analysis Details', 16);

*// File metadata*

*this*.addSection('File Metadata', `

Type: ${analysis.fileType}

Size: ${(analysis.fileSize / (1024 \* 1024)).toFixed(2)} MB

Duration: ${analysis.details.metadata.duration ? Math.round(analysis.details.metadata.duration) + 's' : 'N/A'}

Resolution: ${analysis.details.metadata.resolution || 'N/A'}

Analysis ID: ${analysis.id}

`);

*// Processing details*

*this*.addSection('Processing Information', `

Processing Time: ${Math.round(analysis.processingTime / 1000)} seconds

Analysis Algorithm: Reality Defender API

Timestamp: ${new Date(analysis.timestamp).toLocaleString()}

`);

*// Detailed results*

    if (analysis.details.categoryBreakdown) {

*this*.addSection('Category Analysis',

        Object.entries(analysis.details.categoryBreakdown)

          .map(([category, score]) => `${category}: ${score}%`)

          .join('\n')

      );

    }

*// Frame analysis if available*

    if (analysis.details.frameAnalysis) {

*this*.addSection('Frame Analysis', `

Total Frames Analyzed: ${analysis.details.frameAnalysis.length}

Average Frame Confidence: ${(analysis.details.frameAnalysis.reduce((sum, frame) => sum + frame.confidence, 0) / analysis.details.frameAnalysis.length \* 100).toFixed(1)}%

Suspicious Frames: ${analysis.details.frameAnalysis.filter(frame => frame.confidence > 0.7).length}

`);

    }

*// Audio analysis if available*

    if (analysis.details.audioAnalysis) {

*this*.addSection('Audio Analysis', `

Audio Segments Analyzed: ${analysis.details.audioAnalysis.segments.length}

Average Audio Confidence: ${(analysis.details.audioAnalysis.segments.reduce((sum, seg) => sum + seg.confidence, 0) / analysis.details.audioAnalysis.segments.length \* 100).toFixed(1)}%

Audio Anomalies Found: ${analysis.details.audioAnalysis.segments.filter(seg => seg.anomalies && seg.anomalies.length > 0).length}

`);

    }

  }

  private addDisclaimer(): void {

*this*.addTitle('Disclaimer and Limitations', 14);

    const disclaimer = `

IMPORTANT NOTICE:

1. Accuracy Limitations: While this deepfake detection system uses advanced AI algorithms, no detection system is 100% accurate. Results should be considered as one factor in content verification, not as definitive proof.

2. Technology Evolution: Deepfake generation technology is rapidly evolving. New techniques may not be detected by current algorithms.

3. Context Matters: Consider the source, context, and other verification methods when evaluating content authenticity.

4. Legal Considerations: This report is provided for informational purposes only and should not be used as sole evidence in legal proceedings without additional verification.

5. Data Privacy: Analysis data is processed according to our privacy policy. No content is stored permanently on our servers after analysis.

6. Technical Support: For questions about this report or the analysis methodology, contact our technical support team.

Generated by ITL DeepFake Detection System v1.0

Powered by Reality Defender API

`;

*this*.addText(disclaimer, *this*.margin, 10);

  }

  public async generateReport(

    analysis: StoredAnalysis,

    options: PDFGenerationOptions = {}

  ): Promise<Blob> {

    try {

*// Reset PDF state*

*this*.currentY = 20;

*// Add header*

*this*.addHeader(analysis);

*// Add executive summary*

*this*.addExecutiveSummary(analysis);

*// Add charts if requested*

      if (options.includeCharts !== false) {

*this*.checkPageBreak(50);

*this*.addTitle('Visual Analysis', 16);

*// Add confidence gauge*

        await *this*.addChartFromElement('confidence-gauge-pdf', 'Confidence Analysis');

*// Add category breakdown*

        if (analysis.details.categoryBreakdown) {

          await *this*.addChartFromElement('category-chart-pdf', 'Category Breakdown');

        }

      }

*// Add timeline analysis if requested*

      if (options.includeTimeline !== false) {

        await *this*.addChartFromElement('timeline-chart-pdf', 'Timeline Analysis');

      }

*// Add advanced charts if requested*

      if (options.includeAdvancedCharts) {

        await *this*.addChartFromElement('risk-heatmap-pdf', 'Risk Analysis Heatmap');

        await *this*.addChartFromElement('anomaly-scatter-pdf', 'Anomaly Detection');

        await *this*.addChartFromElement('radar-chart-pdf', 'Multidimensional Analysis');

      }

*// Add technical details*

*this*.addTechnicalDetails(analysis);

*// Add raw data if requested*

      if (options.includeRawData) {

*this*.addTitle('Raw Analysis Data', 16);

*this*.addText(JSON.stringify(analysis, null, 2), *this*.margin, 8);

      }

*// Add disclaimer*

*this*.addDisclaimer();

*// Add footer to all pages*

*this*.addFooter();

*// Generate PDF blob*

      const pdfBlob = new Blob([*this*.pdf.output('blob')], { type: 'application/pdf' });

      return pdfBlob;

    } catch (error) {

      console.error('Failed to generate PDF report:', error);

      throw new Error('Failed to generate PDF report. Please try again.');

    }

  }

  public downloadReport(analysis: StoredAnalysis, options: PDFGenerationOptions = {}): Promise<void> {

    return *this*.generateReport(analysis, options).then(blob => {

      const url = URL.createObjectURL(blob);

      const link = document.createElement('a');

      link.href = url;

      link.download = `itl-deepfake-analysis-${analysis.filename}-${new Date().toISOString().split('T')[0]}.pdf`;

      document.body.appendChild(link);

      link.click();

      document.body.removeChild(link);

      URL.revokeObjectURL(url);

    });

  }

}

*// Export a singleton instance*

export const pdfGenerator = new PDFReportGenerator();

**src\lib\reality-defender.ts**

*// For browser compatibility, we'll use direct API calls instead of the Node.js SDK*

import { usageTracker } from './usage-tracker';

import { explanationGenerator } from './explanation-generator';

import type { DetailedExplanation } from './types';

interface AnalysisResult {

  id: string;

  filename: string;

  fileType: string;

  fileSize: number;

  confidence: number;

  prediction: 'authentic' | 'manipulated' | 'inconclusive';

  details: {

    overallScore: number;

    categoryBreakdown: {

      authentic: number;

      manipulated: number;

      inconclusive: number;

    };

    frameAnalysis?: Array<{

      frame: number;

      timestamp: number;

      confidence: number;

      anomalies?: string[];

    }>;

    audioAnalysis?: {

      segments: Array<{

        start: number;

        end: number;

        confidence: number;

        anomalies?: string[];

      }>;

      waveformData?: number[];

    };

    metadata: {

      duration?: number;

      resolution?: string;

      codec?: string;

      bitrate?: number;

      modelsAnalyzed?: number;

      completedModels?: number;

    };

  };

  processingTime: number;

  timestamp: string;

  thumbnailUrl?: string;

  explanation?: DetailedExplanation;

}

interface AnalysisError {

  error: string;

  message: string;

  code?: string;

}

class RealityDefenderAPI {

  private apiKey: string;

  private baseUrl: string;

  constructor() {

*this*.apiKey = process.env.NEXT\_PUBLIC\_RD\_API\_KEY || '';

*this*.baseUrl = process.env.NEXT\_PUBLIC\_RD\_API\_URL || 'https://api.prd.realitydefender.xyz';

    if (!*this*.apiKey) {

      throw new Error('NEXT\_PUBLIC\_RD\_API\_KEY is required. Please configure your Reality Defender API key.');

    }

    console.log('Reality Defender API initialized:', {

      hasApiKey: !!*this*.apiKey,

      apiKeyPrefix: *this*.apiKey.substring(0, 10) + '...',

      baseUrl: *this*.baseUrl

    });

  }

  private getHeaders(): Record<string, string> {

    return {

      'Authorization': `Bearer ${*this*.apiKey}`,

      'Accept': 'application/json',

      'Content-Type': 'application/json',

    };

  }

  async analyzeMedia(file: File, progressCallback?: (progress: any) => void): Promise<AnalysisResult> {

    console.log('Analyzing with Reality Defender API:', file.name);

    const result = await *this*.analyzeWithDirectAPI(file, progressCallback);

    const normalizedResult = *this*.normalizeAnalysisResult(result, file);

*// Generate detailed explanation*

    try {

      const explanation = explanationGenerator.generateExplanation(normalizedResult);

      normalizedResult.explanation = explanation;

      console.log('✨ Generated detailed explanation:', explanation.summary.primaryReason);

    } catch (error) {

      console.warn('Failed to generate explanation:', error);

    }

*// Track usage for free tier monitoring*

    if (typeof window !== 'undefined') {

      usageTracker.incrementUsage(

        file.type,

        file.name,

        normalizedResult.confidence,

        normalizedResult.prediction

      );

    }

    return normalizedResult;

  }

  private async analyzeWithDirectAPI(file: File, progressCallback?: (progress: any) => void): Promise<any> {

    try {

*// Step 1: Get signed URL for upload via server-side proxy*

      progressCallback?.({

        percentage: 5,

        stage: 'upload',

        message: 'Preparing secure upload...',

        stagesCompleted: [],

      });

      const signedUrlResponse = await fetch('/api/rd/signed-url', {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify({

          fileName: file.name

        }),

      });

      if (!signedUrlResponse.ok) {

        throw new Error(`Failed to get signed URL: ${signedUrlResponse.status}`);

      }

      const signedUrlData = await signedUrlResponse.json();

      console.log('Got signed URL:', signedUrlData);

*// Step 2: Upload file to signed URL (direct to S3)*

      progressCallback?.({

        percentage: 15,

        stage: 'upload',

        message: 'Uploading to Reality Defender servers...',

        stagesCompleted: [],

      });

      const uploadResponse = await fetch(signedUrlData.response.signedUrl, {

        method: 'PUT',

        body: file,

        headers: {

          'Content-Type': file.type || 'application/octet-stream',

        },

      });

      if (!uploadResponse.ok) {

        throw new Error(`Failed to upload file: ${uploadResponse.status}`);

      }

      console.log('File uploaded successfully');

      progressCallback?.({

        percentage: 25,

        stage: 'preprocessing',

        message: 'Upload complete! Processing media...',

        stagesCompleted: ['upload'],

      });

*// Step 3: Poll for results with progress updates*

      const result = await *this*.pollForResults(signedUrlData.requestId, progressCallback);

      return result;

    } catch (error) {

      console.error('Direct API call failed:', error);

      throw error;

    }

  }

  private async pollForResults(requestId: string, progressCallback?: (progress: any) => void, maxAttempts = 60): Promise<any> {

    const startTime = Date.now();

    for (let attempt = 0; attempt < maxAttempts; attempt++) {

      try {

        const response = await fetch(`/api/rd/result/${requestId}`, {

          method: 'GET',

        });

        if (response.ok) {

          const result = await response.json();

          console.log(`Polling attempt ${attempt + 1}:`, JSON.stringify(result, null, 2));

*// Check various status indicators from Reality Defender API*

          const overallStatus = result.overallStatus || result.status || result.resultsSummary?.status || result.state;

*// Also check individual model statuses*

          const modelStatuses = result.models ? result.models.map((m: any) => m.status) : [];

          const hasAnalyzingModels = modelStatuses.some((s: any) => s === 'ANALYZING' || s === 'PROCESSING' || s === 'QUEUED');

          const hasCompletedModels = modelStatuses.some((s: any) => s !== 'ANALYZING' && s !== 'PROCESSING' && s !== 'QUEUED' && s !== 'NOT\_APPLICABLE');

*// Calculate progress based on model completion*

          const totalModels = result.models ? result.models.filter((m: any) => m.status !== 'NOT\_APPLICABLE').length : 0;

          const completedModels = result.models ? result.models.filter((m: any) =>

            m.status !== 'ANALYZING' && m.status !== 'PROCESSING' && m.status !== 'QUEUED' && m.status !== 'NOT\_APPLICABLE'

          ).length : 0;

          const analysisProgress = totalModels > 0 ? (completedModels / totalModels) \* 70 : 0; *// 70% of total progress for analysis*

          const currentProgress = 25 + analysisProgress; *// 25% was upload/preprocessing*

*// Get active models for display*

          const activeModels = result.models ? result.models

            .filter((m: any) => m.status === 'ANALYZING' || m.status === 'PROCESSING')

            .map((m: any) => m.name.replace('rd-', '').replace('-', ' '))

            .slice(0, 3) : [];

*// Update progress callback*

          if (progressCallback) {

            const timeElapsed = Date.now() - startTime;

            const estimatedTotal = totalModels > 0 ? (timeElapsed / completedModels) \* totalModels : 60000;

            const estimatedRemaining = Math.max(0, estimatedTotal - timeElapsed);

            progressCallback({

              percentage: Math.min(95, Math.round(currentProgress)),

              stage: hasCompletedModels ? 'analysis' : 'preprocessing',

              message: hasAnalyzingModels ? `AI models analyzing: ${completedModels}/${totalModels} complete` : 'Finalizing analysis...',

              stagesCompleted: ['upload', 'preprocessing'],

              analysisDetails: {

                activeModels,

                completedModels,

                totalModels,

                modelStatuses: result.models?.reduce((acc: any, m: any) => ({ ...acc, [m.name]: m.status }), {}) || {},

                currentPhase: hasAnalyzingModels ? 'Model Analysis' : 'Results Compilation'

              },

              timeElapsed: Math.round(timeElapsed / 1000),

              estimatedTimeRemaining: Math.round(estimatedRemaining / 1000)

            });

          }

          console.log(`Status check: Overall=${overallStatus}, Models=${modelStatuses}, HasCompleted=${hasCompletedModels}, Progress=${Math.round(currentProgress)}%`);

*// Return results if:*

*// 1. Overall status is not analyzing, OR*

*// 2. We have at least one completed model and no more analyzing models*

          if ((overallStatus !== 'ANALYZING' && overallStatus !== 'PROCESSING' && overallStatus !== 'QUEUED') ||

              (hasCompletedModels && !hasAnalyzingModels)) {

            console.log('Analysis complete, returning result');

            progressCallback?.({

              percentage: 100,

              stage: 'results',

              message: 'Analysis complete! Generating results...',

              stagesCompleted: ['upload', 'preprocessing', 'analysis'],

              timeElapsed: Math.round((Date.now() - startTime) / 1000)

            });

            return result;

          }

          console.log(`Status: ${overallStatus}, continuing to poll...`);

        } else {

          console.warn(`Polling failed with status ${response.status}`);

        }

*// Wait 2 seconds before next attempt*

        await new Promise(resolve => setTimeout(resolve, 2000));

      } catch (error) {

        console.warn(`Polling attempt ${attempt + 1} failed:`, error);

        if (attempt === maxAttempts - 1) throw error;

      }

    }

    throw new Error('Analysis timeout - taking longer than expected. Please try again.');

  }

  private getAnalysisType(fileType: string): string {

    if (fileType.startsWith('image/')) return 'image';

    if (fileType.startsWith('video/')) return 'video';

    if (fileType.startsWith('audio/')) return 'audio';

    return 'unknown';

  }

  private normalizeAnalysisResult(apiResult: any, file: File): AnalysisResult {

    console.log('🔍 Raw API Result for', file.name, ':', JSON.stringify(apiResult, null, 2));

    console.log('📊 Models in response:', apiResult.models?.map((m: any) => ({ name: m.name, status: m.status, score: m.normalizedPredictionNumber })));

*// Reality Defender API structure - extract from models array*

    let overallScore = 0;

    let hasCompletedAnalysis = false;

*// Check if we have completed model results*

    if (apiResult.models && Array.isArray(apiResult.models)) {

      const completedModels = apiResult.models.filter((model: any) =>

        model.status !== 'ANALYZING' &&

        model.status !== 'PROCESSING' &&

        model.status !== 'QUEUED' &&

        model.status !== 'NOT\_APPLICABLE'

      );

      if (completedModels.length > 0) {

*// Extract scores from completed models*

        const scores = completedModels

          .filter((model: any) => model.normalizedPredictionNumber !== null && model.normalizedPredictionNumber !== undefined)

          .map((model: any) => model.normalizedPredictionNumber);

        if (scores.length > 0) {

*// Use ensemble or highest confidence score*

          const ensembleModel = completedModels.find((model: any) => model.name.includes('ensemble'));

          if (ensembleModel && ensembleModel.normalizedPredictionNumber !== null) {

            overallScore = ensembleModel.normalizedPredictionNumber;

          } else {

*// Average of all model scores*

            overallScore = scores.reduce((sum: number, score: number) => sum + score, 0) / scores.length;

          }

          hasCompletedAnalysis = true;

        }

      }

    }

*// Fallback to resultsSummary if available*

    if (!hasCompletedAnalysis) {

      const results = apiResult.resultsSummary || apiResult.results || apiResult;

      overallScore = results.overallScore || results.confidence || results.score || 0;

    }

*// Convert RD score to our confidence scale (RD scores are 0-1, higher = more likely manipulated)*

*// Ensure all values are in proper 0-1 range first*

    const normalizedScore = overallScore > 1 ? overallScore / 100 : overallScore;

    let prediction: 'authentic' | 'manipulated' | 'inconclusive';

    if (normalizedScore >= 0.7) {

      prediction = 'manipulated';

    } else if (normalizedScore >= 0.3) {

      prediction = 'inconclusive';

    } else {

      prediction = 'authentic';

    }

    console.log(`🎯 Analysis Results: Score=${normalizedScore}, Prediction=${prediction}, Completed=${hasCompletedAnalysis}`);

*// Calculate category breakdown based on the normalized score*

    const authenticPercentage = Math.round(Math.max(0, (1 - normalizedScore)) \* 100);

    const manipulatedPercentage = Math.round(normalizedScore \* 100);

    const inconclusivePercentage = Math.round(Math.abs(0.5 - normalizedScore) \* 40);

    const categoryBreakdown = {

      authentic: authenticPercentage,

      manipulated: manipulatedPercentage,

      inconclusive: inconclusivePercentage,

    };

    return {

      id: apiResult.requestId || apiResult.id || `rd\_${Date.now()}`,

      filename: file.name,

      fileType: file.type,

      fileSize: file.size,

      confidence: normalizedScore,

      prediction,

      details: {

        overallScore: normalizedScore,

        categoryBreakdown,

        frameAnalysis: apiResult.frameAnalysis,

        audioAnalysis: apiResult.audioAnalysis,

        metadata: {

          ...apiResult.metadata,

          processingTime: apiResult.processingTime,

          modelsAnalyzed: apiResult.models?.length || 0,

          completedModels: hasCompletedAnalysis ? apiResult.models?.filter((m: any) => m.status !== 'ANALYZING' && m.status !== 'NOT\_APPLICABLE').length : 0,

        },

      },

      processingTime: apiResult.processingTime || 0,

      timestamp: apiResult.timestamp || new Date().toISOString(),

      thumbnailUrl: apiResult.thumbnailUrl,

    };

  }

  async checkApiStatus(): Promise<{ status: string; version?: string }> {

    try {

*// Test the actual API endpoint that we use via proxy*

      const response = await fetch('/api/rd/signed-url', {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify({

          fileName: 'status-check.jpg'

        }),

      });

      if (response.ok) {

        return { status: 'online', version: '1.0.0' };

      } else if (response.status === 401) {

        return { status: 'unauthorized' };

      }

      return { status: 'error' };

    } catch (error) {

      console.error('API status check failed:', error);

      return { status: 'offline' };

    }

  }

}

export const realityDefenderAPI = new RealityDefenderAPI();

export type { AnalysisResult, AnalysisError };

**src\lib\usage-tracker.ts**

interface UsageStats {

  totalScans: number;

  monthlyScans: number;

  currentMonth: string;

  lastReset: string;

  scanHistory: Array<{

    date: string;

    fileType: string;

    fileName: string;

    confidence: number;

    prediction: string;

  }>;

  freeTierLimit: number;

}

class UsageTracker {

  private storageKey = 'rd\_usage\_stats';

  private freeTierLimit = 50; *// Reality Defender free tier limit*

  getUsageStats(): UsageStats {

    const stored = localStorage.getItem(*this*.storageKey);

    if (!stored) {

      return *this*.createInitialStats();

    }

    const stats: UsageStats = JSON.parse(stored);

*// Check if we need to reset monthly count*

    const currentMonth = new Date().toISOString().slice(0, 7); *// YYYY-MM*

    if (stats.currentMonth !== currentMonth) {

      stats.monthlyScans = 0;

      stats.currentMonth = currentMonth;

      stats.lastReset = new Date().toISOString();

*this*.saveUsageStats(stats);

    }

    return stats;

  }

  private createInitialStats(): UsageStats {

    const currentMonth = new Date().toISOString().slice(0, 7);

    const stats: UsageStats = {

      totalScans: 0,

      monthlyScans: 0,

      currentMonth,

      lastReset: new Date().toISOString(),

      scanHistory: [],

      freeTierLimit: *this*.freeTierLimit,

    };

*this*.saveUsageStats(stats);

    return stats;

  }

  incrementUsage(fileType: string, fileName: string, confidence: number, prediction: string): UsageStats {

    const stats = *this*.getUsageStats();

    stats.totalScans += 1;

    stats.monthlyScans += 1;

    stats.scanHistory.unshift({

      date: new Date().toISOString(),

      fileType,

      fileName,

      confidence,

      prediction,

    });

*// Keep only last 100 entries to prevent storage bloat*

    if (stats.scanHistory.length > 100) {

      stats.scanHistory = stats.scanHistory.slice(0, 100);

    }

*this*.saveUsageStats(stats);

    return stats;

  }

  private saveUsageStats(stats: UsageStats): void {

    localStorage.setItem(*this*.storageKey, JSON.stringify(stats));

  }

  getRemainingScans(): number {

    const stats = *this*.getUsageStats();

    return Math.max(0, *this*.freeTierLimit - stats.monthlyScans);

  }

  getUsagePercentage(): number {

    const stats = *this*.getUsageStats();

    return Math.min(100, (stats.monthlyScans / *this*.freeTierLimit) \* 100);

  }

  canMakeScan(): boolean {

    return *this*.getRemainingScans() > 0;

  }

  getMonthlyUsageByFileType(): Record<string, number> {

    const stats = *this*.getUsageStats();

    const currentMonth = new Date().toISOString().slice(0, 7);

    const monthlyHistory = stats.scanHistory.filter(

      scan => scan.date.slice(0, 7) === currentMonth

    );

    const usage: Record<string, number> = {};

    monthlyHistory.forEach(scan => {

      const type = scan.fileType.split('/')[0]; *// Get 'image', 'audio', etc.*

      usage[type] = (usage[type] || 0) + 1;

    });

    return usage;

  }

  getConfidenceDistribution(): { range: string; count: number }[] {

    const stats = *this*.getUsageStats();

    const ranges = [

      { min: 0, max: 0.2, label: '0-20%' },

      { min: 0.2, max: 0.4, label: '20-40%' },

      { min: 0.4, max: 0.6, label: '40-60%' },

      { min: 0.6, max: 0.8, label: '60-80%' },

      { min: 0.8, max: 1, label: '80-100%' },

    ];

    return ranges.map(range => ({

      range: range.label,

      count: stats.scanHistory.filter(

        scan => scan.confidence >= range.min && scan.confidence < range.max

      ).length,

    }));

  }

  getPredictionStats(): Record<string, number> {

    const stats = *this*.getUsageStats();

    const predictions: Record<string, number> = {};

    stats.scanHistory.forEach(scan => {

      predictions[scan.prediction] = (predictions[scan.prediction] || 0) + 1;

    });

    return predictions;

  }

  getWeeklyUsage(): Array<{ day: string; count: number }> {

    const stats = *this*.getUsageStats();

    const last7Days = Array.from({ length: 7 }, (\_, i) => {

      const date = new Date();

      date.setDate(date.getDate() - i);

      return date.toISOString().slice(0, 10);

    }).reverse();

    return last7Days.map(day => ({

      day: day.slice(5), *// MM-DD format*

      count: stats.scanHistory.filter(scan => scan.date.slice(0, 10) === day).length,

    }));

  }

}

export const usageTracker = new UsageTracker();

export type { UsageStats };

## 1.8 UI Components System

### 1.8.1 Step 1: Enhanced Button Component

**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 gap-2 whitespace-nowrap rounded-md text-sm font-medium transition-all disabled:pointer-events-none disabled:opacity-50 [&\_svg]:pointer-events-none [&\_svg:not([class\*='size-'])]:size-4 shrink-0 [&\_svg]:shrink-0 outline-none focus-visible:border-ring focus-visible:ring-ring/50 focus-visible:ring-[3px] aria-invalid:ring-destructive/20 dark:aria-invalid:ring-destructive/40 aria-invalid:border-destructive",

  {

    variants: {

      variant: {

        default: "bg-primary text-primary-foreground hover:bg-primary/90",

        destructive:

          "bg-destructive text-white hover:bg-destructive/90 focus-visible:ring-destructive/20 dark:focus-visible:ring-destructive/40 dark:bg-destructive/60",

        outline:

          "border bg-background shadow-xs hover:bg-accent hover:text-accent-foreground dark:bg-input/30 dark:border-input dark:hover:bg-input/50",

        secondary:

          "bg-secondary text-secondary-foreground hover:bg-secondary/80",

        ghost:

          "hover:bg-accent hover:text-accent-foreground dark:hover:bg-accent/50",

        link: "text-primary underline-offset-4 hover:underline",

      },

      size: {

        default: "h-9 px-4 py-2 has-[>svg]:px-3",

        sm: "h-8 rounded-md gap-1.5 px-3 has-[>svg]:px-2.5",

        lg: "h-10 rounded-md px-6 has-[>svg]:px-4",

        icon: "size-9",

      },

    },

    defaultVariants: {

      variant: "default",

      size: "default",

    },

  }

)

function Button({

  className,

  variant,

  size,

  asChild = false,

  ...props

}: React.ComponentProps<"button"> &

  VariantProps<typeof buttonVariants> & {

    asChild?: boolean

  }) {

  const Comp = asChild ? Slot : "button"

  return (

    <Comp

      data-slot="button"

      className={cn(buttonVariants({ variant, size, className }))}

      {...props}

    />

  )

}

export { Button, buttonVariants }

### 1.8.2 Step 2: Modern Card Component with Enhanced Features

**src/components/ui/card.tsx:**

import \* as React from "react"  
  
import { cn } from "@/lib/utils"  
  
function Card({ className, ...props }: React.ComponentProps<"div">) {  
 return (  
 <div  
 data-slot="card"  
 className={cn(  
 "bg-card text-card-foreground flex flex-col gap-6 rounded-xl border py-6 shadow-sm",  
 className  
 )}  
 {...props}  
 />  
 )  
}  
  
function CardHeader({ className, ...props }: React.ComponentProps<"div">) {  
 return (  
 <div  
 data-slot="card-header"  
 className={cn(  
 "@container/card-header grid auto-rows-min grid-rows-[auto\_auto] items-start gap-1.5 px-6 has-data-[slot=card-action]:grid-cols-[1fr\_auto] [.border-b]:pb-6",  
 className  
 )}  
 {...props}  
 />  
 )  
}  
  
function CardTitle({ className, ...props }: React.ComponentProps<"div">) {  
 return (  
 <div  
 data-slot="card-title"  
 className={cn("leading-none font-semibold", className)}  
 {...props}  
 />  
 )  
}  
  
function CardDescription({ className, ...props }: React.ComponentProps<"div">) {  
 return (  
 <div  
 data-slot="card-description"  
 className={cn("text-muted-foreground text-sm", className)}  
 {...props}  
 />  
 )  
}  
  
function CardAction({ className, ...props }: React.ComponentProps<"div">) {  
 return (  
 <div  
 data-slot="card-action"  
 className={cn(  
 "col-start-2 row-span-2 row-start-1 self-start justify-self-end",  
 className  
 )}  
 {...props}  
 />  
 )  
}  
  
function CardContent({ className, ...props }: React.ComponentProps<"div">) {  
 return (  
 <div  
 data-slot="card-content"  
 className={cn("px-6", className)}  
 {...props}  
 />  
 )  
}  
  
function CardFooter({ className, ...props }: React.ComponentProps<"div">) {  
 return (  
 <div  
 data-slot="card-footer"  
 className={cn("flex items-center px-6 [.border-t]:pt-6", className)}  
 {...props}  
 />  
 )  
}  
  
export {  
 Card,  
 CardHeader,  
 CardFooter,  
 CardTitle,  
 CardAction,  
 CardDescription,  
 CardContent,  
}

### 1.8.3 Step 3: Enhanced Badge Component

**src/components/ui/badge.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 badgeVariants = cva(  
 "inline-flex items-center justify-center rounded-md border px-2 py-0.5 text-xs font-medium w-fit whitespace-nowrap shrink-0 [&>svg]:size-3 gap-1 [&>svg]:pointer-events-none focus-visible:border-ring focus-visible:ring-ring/50 focus-visible:ring-[3px] aria-invalid:ring-destructive/20 dark:aria-invalid:ring-destructive/40 aria-invalid:border-destructive transition-[color,box-shadow] overflow-hidden",  
 {  
 variants: {  
 variant: {  
 default:  
 "border-transparent bg-primary text-primary-foreground [a&]:hover:bg-primary/90",  
 secondary:  
 "border-transparent bg-secondary text-secondary-foreground [a&]:hover:bg-secondary/90",  
 destructive:  
 "border-transparent bg-destructive text-white [a&]:hover:bg-destructive/90 focus-visible:ring-destructive/20 dark:focus-visible:ring-destructive/40 dark:bg-destructive/60",  
 outline:  
 "text-foreground [a&]:hover:bg-accent [a&]:hover:text-accent-foreground",  
 },  
 },  
 defaultVariants: {  
 variant: "default",  
 },  
 }  
)  
  
function Badge({  
 className,  
 variant,  
 asChild = false,  
 ...props  
}: React.ComponentProps<"span"> &  
 VariantProps<typeof badgeVariants> & { asChild?: boolean }) {  
 const Comp = asChild ? Slot : "span"  
  
 return (  
 <Comp  
 data-slot="badge"  
 className={cn(badgeVariants({ variant }), className)}  
 {...props}  
 />  
 )  
}  
  
export { Badge, badgeVariants }

### 1.8.4 Step 4: Additional UI Components

You’ll need to create several additional UI components. Install shadcn/ui components:

# Install additional UI components using shadcn/ui CLI  
npx shadcn@latest add dialog dropdown-menu progress separator switch toggle

**src/components/ui/sonner.tsx:**

"use client"

import { useTheme } from "next-themes"

import { Toaster as Sonner, ToasterProps } from "sonner"

const Toaster = ({ ...props }: ToasterProps) => {

  const { theme = "system" } = useTheme()

  return (

    <Sonner

      theme={theme as ToasterProps["theme"]}

      className="toaster group"

      style={

        {

          "--normal-bg": "var(--popover)",

          "--normal-text": "var(--popover-foreground)",

          "--normal-border": "var(--border)",

        } as React.CSSProperties

      }

      {...props}

    />

  )

}

export { Toaster }

**src\components\ui\dialog.tsx**

"use client"

import \* as React from "react"

import \* as DialogPrimitive from "@radix-ui/react-dialog"

import { XIcon } from "lucide-react"

import { cn } from "@/lib/utils"

function Dialog({

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Root>) {

  return <DialogPrimitive.Root data-slot="dialog" {...props} />

}

function DialogTrigger({

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Trigger>) {

  return <DialogPrimitive.Trigger data-slot="dialog-trigger" {...props} />

}

function DialogPortal({

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Portal>) {

  return <DialogPrimitive.Portal data-slot="dialog-portal" {...props} />

}

function DialogClose({

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Close>) {

  return <DialogPrimitive.Close data-slot="dialog-close" {...props} />

}

function DialogOverlay({

  className,

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Overlay>) {

  return (

    <DialogPrimitive.Overlay

      data-slot="dialog-overlay"

      className={cn(

        "data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 fixed inset-0 z-50 bg-black/50",

        className

      )}

      {...props}

    />

  )

}

function DialogContent({

  className,

  children,

  showCloseButton = true,

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Content> & {

  showCloseButton?: boolean

}) {

  return (

    <DialogPortal data-slot="dialog-portal">

      <DialogOverlay />

      <DialogPrimitive.Content

        data-slot="dialog-content"

        className={cn(

          "bg-background data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:zoom-out-95 data-[state=open]:zoom-in-95 fixed top-[50%] left-[50%] z-50 grid w-full max-w-[calc(100%-2rem)] translate-x-[-50%] translate-y-[-50%] gap-4 rounded-lg border p-6 shadow-lg duration-200 sm:max-w-lg",

          className

        )}

        {...props}

      >

        {children}

        {showCloseButton && (

          <DialogPrimitive.Close

            data-slot="dialog-close"

            className="ring-offset-background focus:ring-ring data-[state=open]:bg-accent data-[state=open]:text-muted-foreground absolute top-4 right-4 rounded-xs opacity-70 transition-opacity hover:opacity-100 focus:ring-2 focus:ring-offset-2 focus:outline-hidden disabled:pointer-events-none [&\_svg]:pointer-events-none [&\_svg]:shrink-0 [&\_svg:not([class\*='size-'])]:size-4"

          >

            <XIcon />

            <span className="sr-only">Close</span>

          </DialogPrimitive.Close>

        )}

      </DialogPrimitive.Content>

    </DialogPortal>

  )

}

function DialogHeader({ className, ...props }: React.ComponentProps<"div">) {

  return (

    <div

      data-slot="dialog-header"

      className={cn("flex flex-col gap-2 text-center sm:text-left", className)}

      {...props}

    />

  )

}

function DialogFooter({ className, ...props }: React.ComponentProps<"div">) {

  return (

    <div

      data-slot="dialog-footer"

      className={cn(

        "flex flex-col-reverse gap-2 sm:flex-row sm:justify-end",

        className

      )}

      {...props}

    />

  )

}

function DialogTitle({

  className,

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Title>) {

  return (

    <DialogPrimitive.Title

      data-slot="dialog-title"

      className={cn("text-lg leading-none font-semibold", className)}

      {...props}

    />

  )

}

function DialogDescription({

  className,

  ...props

}: React.ComponentProps<typeof DialogPrimitive.Description>) {

  return (

    <DialogPrimitive.Description

      data-slot="dialog-description"

      className={cn("text-muted-foreground text-sm", className)}

      {...props}

    />

  )

}

export {

  Dialog,

  DialogClose,

  DialogContent,

  DialogDescription,

  DialogFooter,

  DialogHeader,

  DialogOverlay,

  DialogPortal,

  DialogTitle,

  DialogTrigger,

}

**src\components\ui\dropdown-menu.tsx**

"use client"

import \* as React from "react"

import \* as DropdownMenuPrimitive from "@radix-ui/react-dropdown-menu"

import { CheckIcon, ChevronRightIcon, CircleIcon } from "lucide-react"

import { cn } from "@/lib/utils"

function DropdownMenu({

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Root>) {

  return <DropdownMenuPrimitive.Root data-slot="dropdown-menu" {...props} />

}

function DropdownMenuPortal({

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Portal>) {

  return (

    <DropdownMenuPrimitive.Portal data-slot="dropdown-menu-portal" {...props} />

  )

}

function DropdownMenuTrigger({

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Trigger>) {

  return (

    <DropdownMenuPrimitive.Trigger

      data-slot="dropdown-menu-trigger"

      {...props}

    />

  )

}

function DropdownMenuContent({

  className,

  sideOffset = 4,

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Content>) {

  return (

    <DropdownMenuPrimitive.Portal>

      <DropdownMenuPrimitive.Content

        data-slot="dropdown-menu-content"

        sideOffset={sideOffset}

        className={cn(

          "bg-popover text-popover-foreground data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:zoom-out-95 data-[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from-top-2 data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2 data-[side=top]:slide-in-from-bottom-2 z-50 max-h-(--radix-dropdown-menu-content-available-height) min-w-[8rem] origin-(--radix-dropdown-menu-content-transform-origin) overflow-x-hidden overflow-y-auto rounded-md border p-1 shadow-md",

          className

        )}

        {...props}

      />

    </DropdownMenuPrimitive.Portal>

  )

}

function DropdownMenuGroup({

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Group>) {

  return (

    <DropdownMenuPrimitive.Group data-slot="dropdown-menu-group" {...props} />

  )

}

function DropdownMenuItem({

  className,

  inset,

  variant = "default",

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Item> & {

  inset?: boolean

  variant?: "default" | "destructive"

}) {

  return (

    <DropdownMenuPrimitive.Item

      data-slot="dropdown-menu-item"

      data-inset={inset}

      data-variant={variant}

      className={cn(

        "focus:bg-accent focus:text-accent-foreground data-[variant=destructive]:text-destructive data-[variant=destructive]:focus:bg-destructive/10 dark:data-[variant=destructive]:focus:bg-destructive/20 data-[variant=destructive]:focus:text-destructive data-[variant=destructive]:\*:[svg]:!text-destructive [&\_svg:not([class\*='text-'])]:text-muted-foreground relative flex cursor-default items-center gap-2 rounded-sm px-2 py-1.5 text-sm outline-hidden select-none data-[disabled]:pointer-events-none data-[disabled]:opacity-50 data-[inset]:pl-8 [&\_svg]:pointer-events-none [&\_svg]:shrink-0 [&\_svg:not([class\*='size-'])]:size-4",

        className

      )}

      {...props}

    />

  )

}

function DropdownMenuCheckboxItem({

  className,

  children,

  checked,

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.CheckboxItem>) {

  return (

    <DropdownMenuPrimitive.CheckboxItem

      data-slot="dropdown-menu-checkbox-item"

      className={cn(

        "focus:bg-accent focus:text-accent-foreground relative flex cursor-default items-center gap-2 rounded-sm py-1.5 pr-2 pl-8 text-sm outline-hidden select-none data-[disabled]:pointer-events-none data-[disabled]:opacity-50 [&\_svg]:pointer-events-none [&\_svg]:shrink-0 [&\_svg:not([class\*='size-'])]:size-4",

        className

      )}

      checked={checked}

      {...props}

    >

      <span className="pointer-events-none absolute left-2 flex size-3.5 items-center justify-center">

        <DropdownMenuPrimitive.ItemIndicator>

          <CheckIcon className="size-4" />

        </DropdownMenuPrimitive.ItemIndicator>

      </span>

      {children}

    </DropdownMenuPrimitive.CheckboxItem>

  )

}

function DropdownMenuRadioGroup({

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.RadioGroup>) {

  return (

    <DropdownMenuPrimitive.RadioGroup

      data-slot="dropdown-menu-radio-group"

      {...props}

    />

  )

}

function DropdownMenuRadioItem({

  className,

  children,

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.RadioItem>) {

  return (

    <DropdownMenuPrimitive.RadioItem

      data-slot="dropdown-menu-radio-item"

      className={cn(

        "focus:bg-accent focus:text-accent-foreground relative flex cursor-default items-center gap-2 rounded-sm py-1.5 pr-2 pl-8 text-sm outline-hidden select-none data-[disabled]:pointer-events-none data-[disabled]:opacity-50 [&\_svg]:pointer-events-none [&\_svg]:shrink-0 [&\_svg:not([class\*='size-'])]:size-4",

        className

      )}

      {...props}

    >

      <span className="pointer-events-none absolute left-2 flex size-3.5 items-center justify-center">

        <DropdownMenuPrimitive.ItemIndicator>

          <CircleIcon className="size-2 fill-current" />

        </DropdownMenuPrimitive.ItemIndicator>

      </span>

      {children}

    </DropdownMenuPrimitive.RadioItem>

  )

}

function DropdownMenuLabel({

  className,

  inset,

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Label> & {

  inset?: boolean

}) {

  return (

    <DropdownMenuPrimitive.Label

      data-slot="dropdown-menu-label"

      data-inset={inset}

      className={cn(

        "px-2 py-1.5 text-sm font-medium data-[inset]:pl-8",

        className

      )}

      {...props}

    />

  )

}

function DropdownMenuSeparator({

  className,

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Separator>) {

  return (

    <DropdownMenuPrimitive.Separator

      data-slot="dropdown-menu-separator"

      className={cn("bg-border -mx-1 my-1 h-px", className)}

      {...props}

    />

  )

}

function DropdownMenuShortcut({

  className,

  ...props

}: React.ComponentProps<"span">) {

  return (

    <span

      data-slot="dropdown-menu-shortcut"

      className={cn(

        "text-muted-foreground ml-auto text-xs tracking-widest",

        className

      )}

      {...props}

    />

  )

}

function DropdownMenuSub({

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.Sub>) {

  return <DropdownMenuPrimitive.Sub data-slot="dropdown-menu-sub" {...props} />

}

function DropdownMenuSubTrigger({

  className,

  inset,

  children,

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.SubTrigger> & {

  inset?: boolean

}) {

  return (

    <DropdownMenuPrimitive.SubTrigger

      data-slot="dropdown-menu-sub-trigger"

      data-inset={inset}

      className={cn(

        "focus:bg-accent focus:text-accent-foreground data-[state=open]:bg-accent data-[state=open]:text-accent-foreground flex cursor-default items-center rounded-sm px-2 py-1.5 text-sm outline-hidden select-none data-[inset]:pl-8",

        className

      )}

      {...props}

    >

      {children}

      <ChevronRightIcon className="ml-auto size-4" />

    </DropdownMenuPrimitive.SubTrigger>

  )

}

function DropdownMenuSubContent({

  className,

  ...props

}: React.ComponentProps<typeof DropdownMenuPrimitive.SubContent>) {

  return (

    <DropdownMenuPrimitive.SubContent

      data-slot="dropdown-menu-sub-content"

      className={cn(

        "bg-popover text-popover-foreground data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:zoom-out-95 data-[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from-top-2 data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2 data-[side=top]:slide-in-from-bottom-2 z-50 min-w-[8rem] origin-(--radix-dropdown-menu-content-transform-origin) overflow-hidden rounded-md border p-1 shadow-lg",

        className

      )}

      {...props}

    />

  )

}

export {

  DropdownMenu,

  DropdownMenuPortal,

  DropdownMenuTrigger,

  DropdownMenuContent,

  DropdownMenuGroup,

  DropdownMenuLabel,

  DropdownMenuItem,

  DropdownMenuCheckboxItem,

  DropdownMenuRadioGroup,

  DropdownMenuRadioItem,

  DropdownMenuSeparator,

  DropdownMenuShortcut,

  DropdownMenuSub,

  DropdownMenuSubTrigger,

  DropdownMenuSubContent,

}

**src\components\ui\progress.tsx**

"use client"

import \* as React from "react"

import \* as ProgressPrimitive from "@radix-ui/react-progress"

import { cn } from "@/lib/utils"

function Progress({

  className,

  value,

  ...props

}: React.ComponentProps<typeof ProgressPrimitive.Root>) {

  return (

    <ProgressPrimitive.Root

      data-slot="progress"

      className={cn(

        "bg-primary/20 relative h-2 w-full overflow-hidden rounded-full",

        className

      )}

      {...props}

    >

      <ProgressPrimitive.Indicator

        data-slot="progress-indicator"

        className="bg-primary h-full w-full flex-1 transition-all"

        style={{ transform: `translateX(-${100 - (value || 0)}%)` }}

      />

    </ProgressPrimitive.Root>

  )

}

export { Progress }

**src\components\ui\separator.tsx**

"use client"

import \* as React from "react"

import \* as SeparatorPrimitive from "@radix-ui/react-separator"

import { cn } from "@/lib/utils"

function Separator({

  className,

  orientation = "horizontal",

  decorative = true,

  ...props

}: React.ComponentProps<typeof SeparatorPrimitive.Root>) {

  return (

    <SeparatorPrimitive.Root

      data-slot="separator"

      decorative={decorative}

      orientation={orientation}

      className={cn(

        "bg-border shrink-0 data-[orientation=horizontal]:h-px data-[orientation=horizontal]:w-full data-[orientation=vertical]:h-full data-[orientation=vertical]:w-px",

        className

      )}

      {...props}

    />

  )

}

export { Separator }

**src\components\ui\switch.tsx**

"use client"

import \* as React from "react"

import \* as SwitchPrimitive from "@radix-ui/react-switch"

import { cn } from "@/lib/utils"

function Switch({

  className,

  ...props

}: React.ComponentProps<typeof SwitchPrimitive.Root>) {

  return (

    <SwitchPrimitive.Root

      data-slot="switch"

      className={cn(

        "peer data-[state=checked]:bg-primary data-[state=unchecked]:bg-input focus-visible:border-ring focus-visible:ring-ring/50 dark:data-[state=unchecked]:bg-input/80 inline-flex h-[1.15rem] w-8 shrink-0 items-center rounded-full border border-transparent shadow-xs transition-all outline-none focus-visible:ring-[3px] disabled:cursor-not-allowed disabled:opacity-50",

        className

      )}

      {...props}

    >

      <SwitchPrimitive.Thumb

        data-slot="switch-thumb"

        className={cn(

          "bg-background dark:data-[state=unchecked]:bg-foreground dark:data-[state=checked]:bg-primary-foreground pointer-events-none block size-4 rounded-full ring-0 transition-transform data-[state=checked]:translate-x-[calc(100%-2px)] data-[state=unchecked]:translate-x-0"

        )}

      />

    </SwitchPrimitive.Root>

  )

}

export { Switch }

**src\components\ui\toggle.tsx**

"use client"

import \* as React from "react"

import \* as TogglePrimitive from "@radix-ui/react-toggle"

import { cva, type VariantProps } from "class-variance-authority"

import { cn } from "@/lib/utils"

const toggleVariants = cva(

  "inline-flex items-center justify-center gap-2 rounded-md text-sm font-medium hover:bg-muted hover:text-muted-foreground disabled:pointer-events-none disabled:opacity-50 data-[state=on]:bg-accent data-[state=on]:text-accent-foreground [&\_svg]:pointer-events-none [&\_svg:not([class\*='size-'])]:size-4 [&\_svg]:shrink-0 focus-visible:border-ring focus-visible:ring-ring/50 focus-visible:ring-[3px] outline-none transition-[color,box-shadow] aria-invalid:ring-destructive/20 dark:aria-invalid:ring-destructive/40 aria-invalid:border-destructive whitespace-nowrap",

  {

    variants: {

      variant: {

        default: "bg-transparent",

        outline:

          "border border-input bg-transparent shadow-xs hover:bg-accent hover:text-accent-foreground",

      },

      size: {

        default: "h-9 px-2 min-w-9",

        sm: "h-8 px-1.5 min-w-8",

        lg: "h-10 px-2.5 min-w-10",

      },

    },

    defaultVariants: {

      variant: "default",

      size: "default",

    },

  }

)

function Toggle({

  className,

  variant,

  size,

  ...props

}: React.ComponentProps<typeof TogglePrimitive.Root> &

  VariantProps<typeof toggleVariants>) {

  return (

    <TogglePrimitive.Root

      data-slot="toggle"

      className={cn(toggleVariants({ variant, size, className }))}

      {...props}

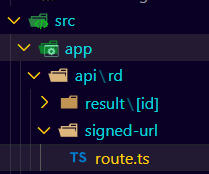
    />

  )

}

export { Toggle, toggleVariants }

### 1.8.5 Step 3: API Routes



**src/app/api/rd/signed-url/route.ts:**

import { NextResponse } from "next/server";  
  
export async function POST(req: Request) {  
 try {  
 const { fileName } = await req.json();  
  
 if (!fileName || typeof fileName !== "string") {  
 return NextResponse.json({ error: "Invalid fileName" }, { status: 400 });  
 }  
  
 const apiKey = process.env.RD\_API\_KEY || process.env.NEXT\_PUBLIC\_RD\_API\_KEY;  
 const baseUrl = process.env.NEXT\_PUBLIC\_RD\_API\_URL || "https://api.prd.realitydefender.xyz";  
  
 if (!apiKey) {  
 return NextResponse.json({ error: "API key not configured" }, { status: 500 });  
 }  
  
 const upstream = await fetch(`${baseUrl}/api/files/aws-presigned`, {  
 method: "POST",  
 headers: {  
 "X-API-KEY": apiKey,  
 "Content-Type": "application/json",  
 },  
 body: JSON.stringify({ fileName }),  
 cache: "no-store",  
 });  
  
 const data = await upstream.json().catch(() => ({}));  
  
 if (!upstream.ok) {  
 return NextResponse.json(  
 { error: "Upstream error", status: upstream.status, response: data },  
 { status: upstream.status }  
 );  
 }  
  
 return NextResponse.json(data);  
 } catch (err: unknown) {  
 return NextResponse.json({ error: err instanceof Error ? err.message : "Unexpected error" }, { status: 500 });  
 }  
}

**src/app/api/rd/result/[id]/route.ts:**

import { NextResponse } from "next/server";  
  
export async function GET(\_req: Request, context: { params: Promise<{ id: string }> }) {  
 try {  
 const { id } = await context.params || { id: '' };  
  
 if (!id) {  
 return NextResponse.json({ error: "Missing id" }, { status: 400 });  
 }  
  
 const apiKey = process.env.RD\_API\_KEY || process.env.NEXT\_PUBLIC\_RD\_API\_KEY;  
 const baseUrl = process.env.NEXT\_PUBLIC\_RD\_API\_URL || "https://api.realitydefender.com";  
  
 if (!apiKey) {  
 return NextResponse.json({ error: "API key not configured" }, { status: 500 });  
 }  
  
 const upstream = await fetch(`${baseUrl}/api/media/users/${id}`, {  
 method: "GET",  
 headers: {  
 "X-API-KEY": apiKey,  
 },  
 cache: "no-store",  
 });  
  
 const data = await upstream.json().catch(() => ({}));  
  
 if (!upstream.ok) {  
 return NextResponse.json(  
 { error: "Upstream error", status: upstream.status, response: data },  
 { status: upstream.status }  
 );  
 }  
  
 return NextResponse.json(data);  
 } catch (err: unknown) {  
 return NextResponse.json({ error: err instanceof Error ? err.message : "Unexpected error" }, { status: 500 });  
 }  
}

**src\app\upload\loading.tsx**

import { Loader2 } from "lucide-react";

export default function UploadLoading() {

  return (

    <div className="min-h-screen bg-gradient-to-br from-background via-background to-muted/20 flex items-center justify-center">

      <div className="text-center space-y-4">

        <Loader2 className="h-12 w-12 animate-spin mx-auto text-primary" />

        <div className="space-y-2">

          <h2 className="text-xl font-semibold">Loading Deepfake Detective</h2>

          <p className="text-muted-foreground">Preparing AI analysis tools...</p>

        </div>

      </div>

    </div>

  );

}

**src\app\upload\page.tsx**

'use client';

import { useState, useEffect } from 'react';

import { useRouter } from 'next/navigation';

import { UploadBox } from '@/components/upload-box';

import { Sidebar } from '@/components/layout/sidebar';

import dynamic from 'next/dynamic';

import { Suspense } from 'react';

*// Dynamic imports for heavy chart components*

const ConfidenceGauge = dynamic(() => import('@/components/charts/confidence-gauge').then(mod => ({ default: mod.ConfidenceGauge })), {

  loading: () => <div className="h-64 bg-muted/30 rounded-lg animate-pulse" />

});

const CategoryChart = dynamic(() => import('@/components/charts/category-chart').then(mod => ({ default: mod.CategoryChart })), {

  loading: () => <div className="h-64 bg-muted/30 rounded-lg animate-pulse" />

});

const UsageDashboard = dynamic(() => import('@/components/usage-dashboard').then(mod => ({ default: mod.UsageDashboard })), {

  loading: () => <div className="h-96 bg-muted/30 rounded-lg animate-pulse" />

});

const ExplanationDashboard = dynamic(() => import('@/components/explanation/explanation-dashboard').then(mod => ({ default: mod.ExplanationDashboard })), {

  loading: () => <div className="h-96 bg-muted/30 rounded-lg animate-pulse" />

});

const RiskHeatmap = dynamic(() => import('@/components/charts/advanced-charts').then(mod => ({ default: mod.RiskHeatmap })), {

  loading: () => <div className="h-64 bg-muted/30 rounded-lg animate-pulse" />

});

const AnomalyScatterPlot = dynamic(() => import('@/components/charts/advanced-charts').then(mod => ({ default: mod.AnomalyScatterPlot })), {

  loading: () => <div className="h-64 bg-muted/30 rounded-lg animate-pulse" />

});

const MultidimensionalRadar = dynamic(() => import('@/components/charts/advanced-charts').then(mod => ({ default: mod.MultidimensionalRadar })), {

  loading: () => <div className="h-64 bg-muted/30 rounded-lg animate-pulse" />

});

const ConfidenceTimelineAdvanced = dynamic(() => import('@/components/charts/advanced-charts').then(mod => ({ default: mod.ConfidenceTimelineAdvanced })), {

  loading: () => <div className="h-64 bg-muted/30 rounded-lg animate-pulse" />

});

*// Dynamic import for Recharts (heavy library)*

const ResponsiveContainer = dynamic(() => import('recharts').then(mod => ({ default: mod.ResponsiveContainer })), {

  ssr: false,

  loading: () => <div className="h-80 bg-muted/30 rounded-lg animate-pulse" />

});

const AreaChart = dynamic(() => import('recharts').then(mod => ({ default: mod.AreaChart })), {

  ssr: false

});

const Area = dynamic(() => import('recharts').then(mod => ({ default: mod.Area })), {

  ssr: false

});

const CartesianGrid = dynamic(() => import('recharts').then(mod => ({ default: mod.CartesianGrid })), {

  ssr: false

});

const XAxis = dynamic(() => import('recharts').then(mod => ({ default: mod.XAxis })), {

  ssr: false

});

const YAxis = dynamic(() => import('recharts').then(mod => ({ default: mod.YAxis })), {

  ssr: false

});

const Tooltip = dynamic(() => import('recharts').then(mod => ({ default: mod.Tooltip })), {

  ssr: false

});

const PDFExportDialog = dynamic(() => import('@/components/pdf/pdf-export-dialog').then(mod => ({ default: mod.PDFExportDialog })), {

  loading: () => <div className="w-32 h-10 bg-muted/30 rounded animate-pulse" />

});

const PDFChartComponents = dynamic(() => import('@/components/pdf/pdf-chart-components').then(mod => ({ default: mod.PDFChartComponents })), {

  ssr: false

});

import { Button } from '@/components/ui/button';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

import { Separator } from '@/components/ui/separator';

import {

  Download,

  ExternalLink,

  Clock,

  FileText,

  AlertCircle,

  Info,

  Zap,

  BarChart3,

  Activity,

  Volume2,

  Upload,

} from 'lucide-react';

import { motion, AnimatePresence } from 'framer-motion';

import { realityDefenderAPI, type AnalysisResult } from '@/lib/reality-defender';

import { storage, createThumbnail, formatFileSize } from '@/lib/storage';

import { usageTracker } from '@/lib/usage-tracker';

import type { FileUploadState, UploadProgress, DetailedExplanation } from '@/lib/types';

import { ExplanationGenerator } from '@/lib/explanation-generator';

import { toast } from 'sonner';

export default function UploadPage() {

  const [uploadState, setUploadState] = useState<FileUploadState>({

    file: null,

    preview: null,

    progress: 0,

    status: 'idle',

  });

  const [uploadProgress, setUploadProgress] = useState<UploadProgress>();

  const [analysisResult, setAnalysisResult] = useState<AnalysisResult | null>(null);

  const [explanation, setExplanation] = useState<DetailedExplanation | null>(null);

  const [showSidebar, setShowSidebar] = useState(true);

  const [activeTab, setActiveTab] = useState<'upload' | 'usage'>('upload');

  const router = useRouter();

  const handleFileSelect = async (file: File) => {

*// Check quota before processing*

    if (typeof window !== 'undefined' && !usageTracker.canMakeScan()) {

      toast.error(`You've reached your monthly limit of ${usageTracker.getUsageStats().freeTierLimit} scans. Your quota will reset next month.`);

      return;

    }

    setUploadState({

      file,

      preview: null,

      progress: 0,

      status: 'uploading',

    });

    setAnalysisResult(null);

    setExplanation(null);

    try {

      setUploadState(prev => ({ ...prev, progress: 5, status: 'processing' }));

*// Call Reality Defender API with progress callback*

      const result = await realityDefenderAPI.analyzeMedia(file, (progress) => {

        setUploadProgress(progress);

        setUploadState(prev => ({ ...prev, progress: progress.percentage }));

      });

      setUploadState(prev => ({ ...prev, progress: 100, status: 'completed' }));

*// Create thumbnail and save to storage*

      try {

        const thumbnail = await createThumbnail(file);

        storage.addAnalysisToHistory(result, thumbnail);

      } catch (error) {

        console.warn('Failed to create thumbnail:', error);

        storage.addAnalysisToHistory(result);

      }

      setAnalysisResult(result);

*// Generate explanation*

      try {

        const explanationGenerator = new ExplanationGenerator();

        const generatedExplanation = explanationGenerator.generateExplanation(result);

        setExplanation(generatedExplanation);

      } catch (error) {

        console.warn('Failed to generate explanation:', error);

        setExplanation(null);

      }

      toast.success('Analysis completed successfully!');

    } catch (error) {

      console.error('Analysis failed:', error);

      setUploadState(prev => ({

        ...prev,

        status: 'error',

        error: error instanceof Error ? error.message : 'Analysis failed',

      }));

      toast.error('Analysis failed. Please try again.');

    }

  };

  const handleFileRemove = () => {

    setUploadState({

      file: null,

      preview: null,

      progress: 0,

      status: 'idle',

    });

    setUploadProgress(undefined);

    setAnalysisResult(null);

    setExplanation(null);

  };

  const handleAnalysisSelect = (analysisId: string) => {

    router.push(`/results?id=${analysisId}`);

  };

*// Function to trigger file picker*

  const triggerFileUpload = () => {

    const fileInput = document.querySelector('input[type="file"]') as HTMLInputElement;

    if (fileInput) {

      fileInput.click();

    }

  };

*// Generate timeline data from analysis results*

  const generateTimelineData = () => {

    if (!analysisResult) return [];

    const data = [];

    const baseConfidence = analysisResult.confidence;

    const points = analysisResult.details.frameAnalysis?.length || 20;

    for (let i = 0; i < points; i++) {

      const variation = (Math.random() - 0.5) \* 0.3;

      const confidence = Math.max(0, Math.min(1, baseConfidence + variation));

      data.push({

        frame: i + 1,

        timestamp: (i / points) \* (analysisResult.details.metadata.duration || 60),

        confidence: Math.round(confidence \* 100),

        anomalies: Math.random() > 0.8 ? 1 : 0,

      });

    }

    return data;

  };

  return (

    <div className="min-h-screen bg-gradient-to-br from-background via-background to-muted/20">

      <div className="flex">

        {*/\* Sidebar \*/*}

        <AnimatePresence>

          {showSidebar && (

            <motion.div

              initial={{ width: 0, opacity: 0 }}

              animate={{ width: 280, opacity: 1 }}

              exit={{ width: 0, opacity: 0 }}

              transition={{ duration: 0.3 }}

              className="flex-shrink-0"

            >

              <Sidebar onAnalysisSelect={handleAnalysisSelect} />

            </motion.div>

          )}

        </AnimatePresence>

        {*/\* Main Content \*/*}

        <div className="flex-1 p-6">

          <div className="max-w-6xl mx-auto space-y-8">

            {*/\* Header \*/*}

            <motion.div

              initial={{ opacity: 0, y: -20 }}

              animate={{ opacity: 1, y: 0 }}

              transition={{ duration: 0.5 }}

              className="text-center space-y-4"

            >

              <h1 className="text-3xl sm:text-4xl font-bold">

                ITL Deepfake Detection Analysis

              </h1>

              <p className="text-lg text-muted-foreground max-w-2xl mx-auto">

                Upload your media file to analyze it for deepfake manipulation using advanced AI technology.

              </p>

              {*/\* Tab Navigation \*/*}

              <div className="flex items-center justify-center gap-2">

                <Button

                  variant={activeTab === 'upload' ? 'default' : 'outline'}

                  size="sm"

                  onClick={() => {

                    setActiveTab('upload');

*// If no file or analysis is complete, allow new upload*

                    if (!uploadState.file && uploadState.status === 'idle') {

                      triggerFileUpload();

                    } else if (uploadState.status === 'completed') {

*// Clear current results and trigger new upload*

                      handleFileRemove();

*// Small delay to ensure state is cleared before triggering upload*

                      setTimeout(() => triggerFileUpload(), 100);

                    }

                  }}

                  className="px-6"

                >

                  <Activity className="w-4 h-4 mr-2" />

                  Upload & Analyze

                </Button>

                <Button

                  variant={activeTab === 'usage' ? 'default' : 'outline'}

                  size="sm"

                  onClick={() => setActiveTab('usage')}

                  className="px-6"

                >

                  <BarChart3 className="w-4 h-4 mr-2" />

                  Usage Dashboard

                </Button>

              </div>

              <div className="flex items-center justify-center gap-4 text-sm text-muted-foreground">

                <div className="flex items-center gap-1">

                  <Zap className="w-4 h-4" />

                  Fast Analysis

                </div>

                <div className="flex items-center gap-1">

                  <Info className="w-4 h-4" />

                  Free Tier: Images & Audio Only

                </div>

              </div>

            </motion.div>

            {*/\* Content Tabs \*/*}

            <AnimatePresence mode="wait">

              {activeTab === 'upload' ? (

                <motion.div

                  key="upload"

                  initial={{ opacity: 0, y: 20 }}

                  animate={{ opacity: 1, y: 0 }}

                  exit={{ opacity: 0, y: -20 }}

                  transition={{ duration: 0.3 }}

                  className="space-y-8"

                >

                  {*/\* Upload Section \*/*}

                  <div>

                    <UploadBox

                      onFileSelect={handleFileSelect}

                      onFileRemove={handleFileRemove}

                      uploadState={uploadState}

                      uploadProgress={uploadProgress}

                      className="max-w-2xl mx-auto"

                    />

                  </div>

                  {*/\* Results Section \*/*}

                  <AnimatePresence>

                    {analysisResult && (

                      <motion.div

                        initial={{ opacity: 0, y: 20 }}

                        animate={{ opacity: 1, y: 0 }}

                        exit={{ opacity: 0, y: -20 }}

                        transition={{ duration: 0.5 }}

                        className="space-y-6"

                      >

                        <Separator className="my-8" />

                        {*/\* Results Header \*/*}

                        <div className="text-center space-y-2">

                          <h2 className="text-2xl font-bold">Analysis Results</h2>

                          <p className="text-muted-foreground">

                            Completed analysis for{' '}

                            <span className="font-medium">{analysisResult.filename}</span>

                          </p>

                          <div className="flex items-center justify-center gap-4 text-sm">

                            <Badge variant="outline">

                              <Clock className="w-3 h-3 mr-1" />

                              {Math.round(analysisResult.processingTime / 1000)}s processing

                            </Badge>

                            <Badge variant="outline">

                              {formatFileSize(analysisResult.fileSize)}

                            </Badge>

                          </div>

                          {*/\* Analyze New File Button \*/*}

                          <div className="pt-4">

                            <Button

                              variant="outline"

                              onClick={() => {

                                handleFileRemove();

                                setTimeout(() => triggerFileUpload(), 100);

                              }}

                              className="px-6"

                            >

                              <Upload className="w-4 h-4 mr-2" />

                              Analyze New File

                            </Button>

                          </div>

                        </div>

                        {*/\* Charts Grid \*/*}

                        <div className="grid lg:grid-cols-2 gap-6">

                          <ConfidenceGauge

                            confidence={analysisResult.confidence}

                            prediction={analysisResult.prediction}

                          />

                          <CategoryChart

                            data={analysisResult.details.categoryBreakdown}

                          />

                        </div>

                        {*/\* Why This Result Section \*/*}

                        {explanation && (

                          <motion.div

                            initial={{ opacity: 0, y: 20 }}

                            animate={{ opacity: 1, y: 0 }}

                            transition={{ duration: 0.5, delay: 0.2 }}

                          >

                            <Separator className="my-8" />

                            <div className="text-center mb-6">

                              <h3 className="text-xl font-bold">Why This Result?</h3>

                              <p className="text-muted-foreground mt-2">

                                Detailed explanation of the analysis findings and reasoning

                              </p>

                            </div>

                            <ExplanationDashboard explanation={explanation} />

                            <Separator className="my-8" />

                          </motion.div>

                        )}

                        {*/\* Metadata Card \*/*}

                        <Card>

                          <CardHeader>

                            <CardTitle className="flex items-center gap-2">

                              <FileText className="w-5 h-5" />

                              File Information

                            </CardTitle>

                          </CardHeader>

                          <CardContent className="space-y-3">

                            <div className="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">

                              <div>

                                <span className="text-muted-foreground">Type:</span>

                                <p className="font-medium capitalize">

                                  {analysisResult.fileType.split('/')[0]}

                                </p>

                              </div>

                              <div>

                                <span className="text-muted-foreground">Size:</span>

                                <p className="font-medium">

                                  {formatFileSize(analysisResult.fileSize)}

                                </p>

                              </div>

                              {analysisResult.details.metadata.duration && (

                                <div>

                                  <span className="text-muted-foreground">Duration:</span>

                                  <p className="font-medium">

                                    {Math.round(analysisResult.details.metadata.duration)}s

                                  </p>

                                </div>

                              )}

                              {analysisResult.details.metadata.resolution && (

                                <div>

                                  <span className="text-muted-foreground">Resolution:</span>

                                  <p className="font-medium">

                                    {analysisResult.details.metadata.resolution}

                                  </p>

                                </div>

                              )}

                            </div>

                          </CardContent>

                        </Card>

                        {*/\* Timeline Analysis \*/*}

                        <Card>

                          <CardHeader>

                            <CardTitle className="flex items-center gap-2">

                              <Clock className="w-5 h-5" />

                              Timeline Analysis

                            </CardTitle>

                          </CardHeader>

                          <CardContent>

                            <div className="h-80">

                              <ResponsiveContainer width="100%" height="100%">

                                <AreaChart data={generateTimelineData()}>

                                  <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

                                  <XAxis

                                    dataKey="frame"

                                    tick={{ fontSize: 12 }}

                                    axisLine={false}

                                    tickLine={false}

                                  />

                                  <YAxis

                                    domain={[0, 100]}

                                    tick={{ fontSize: 12 }}

                                    axisLine={false}

                                    tickLine={false}

                                    label={{ value: 'Confidence %', angle: -90, position: 'insideLeft' }}

                                  />

                                  <Tooltip

                                    content={({ active, payload, label }) => {

                                      if (active && payload && payload.length) {

                                        return (

                                          <div className="bg-background border border-border rounded-lg shadow-lg p-3">

                                            <p className="font-medium">Frame {label}</p>

                                            <p className="text-sm text-muted-foreground">

                                              Confidence: {payload[0].value}%

                                            </p>

                                          </div>

                                        );

                                      }

                                      return null;

                                    }}

                                  />

                                  <Area

                                    type="monotone"

                                    dataKey="confidence"

                                    stroke="#8884d8"

                                    fill="#8884d8"

                                    fillOpacity={0.2}

                                    strokeWidth={2}

                                  />

                                </AreaChart>

                              </ResponsiveContainer>

                            </div>

                          </CardContent>

                        </Card>

                        {*/\* Advanced Charts Grid \*/*}

                        <div className="space-y-6">

                          <h3 className="text-xl font-bold text-center">Advanced Analysis</h3>

                          <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">

                            <RiskHeatmap data={analysisResult} />

                            <AnomalyScatterPlot data={analysisResult} />

                            <MultidimensionalRadar data={analysisResult} />

                            <ConfidenceTimelineAdvanced data={generateTimelineData()} />

                          </div>

                        </div>

                        {*/\* Technical Details \*/*}

                        <Card>

                          <CardHeader>

                            <CardTitle className="flex items-center gap-2">

                              <FileText className="w-5 h-5" />

                              Technical Details

                            </CardTitle>

                          </CardHeader>

                          <CardContent>

                            <div className="space-y-4">

                              {*/\* Processing Stats \*/*}

                              <div className="grid grid-cols-2 md:grid-cols-4 gap-4">

                                <div className="text-center p-3 bg-muted/30 rounded">

                                  <div className="text-lg font-bold text-primary">

                                    {Math.round(analysisResult.processingTime / 1000)}s

                                  </div>

                                  <div className="text-xs text-muted-foreground">Processing Time</div>

                                </div>

                                <div className="text-center p-3 bg-muted/30 rounded">

                                  <div className="text-lg font-bold">

                                    {analysisResult.confidence > 1 ? Math.round(analysisResult.confidence) : Math.round(analysisResult.confidence \* 100)}%

                                  </div>

                                  <div className="text-xs text-muted-foreground">Risk Score</div>

                                </div>

                                <div className="text-center p-3 bg-muted/30 rounded">

                                  <div className="text-lg font-bold text-green-600">

                                    {analysisResult.details.frameAnalysis?.length || 'N/A'}

                                  </div>

                                  <div className="text-xs text-muted-foreground">Frames Analyzed</div>

                                </div>

                                <div className="text-center p-3 bg-muted/30 rounded">

                                  <div className="text-lg font-bold">

                                    {analysisResult.id.slice(-8)}

                                  </div>

                                  <div className="text-xs text-muted-foreground">Analysis ID</div>

                                </div>

                              </div>

                              {*/\* Analysis Summary in User-Friendly Format \*/*}

                              <div className="mt-6">

                                <h4 className="font-medium mb-2">Analysis Summary</h4>

                                <div className="bg-muted/30 p-4 rounded-lg space-y-3">

                                  <div className="grid grid-cols-2 gap-4 text-sm">

                                    <div>

                                      <span className="text-muted-foreground">Confidence Score:</span>

                                      <p className="font-medium">{analysisResult.confidence > 1 ? Math.round(analysisResult.confidence) : Math.round(analysisResult.confidence \* 100)}%</p>

                                    </div>

                                    <div>

                                      <span className="text-muted-foreground">Final Prediction:</span>

                                      <p className="font-medium capitalize">{analysisResult.prediction}</p>

                                    </div>

                                  </div>

                                  <div>

                                    <span className="text-muted-foreground text-sm">Category Scores:</span>

                                    <div className="grid grid-cols-3 gap-2 mt-1">

                                      <div className="text-center p-2 bg-green-50 dark:bg-green-900/20 rounded">

                                        <div className="text-sm font-semibold text-green-700 dark:text-green-400">

                                          {analysisResult.details.categoryBreakdown.authentic > 1 ? Math.round(analysisResult.details.categoryBreakdown.authentic) : Math.round(analysisResult.details.categoryBreakdown.authentic \* 100)}%

                                        </div>

                                        <div className="text-xs text-green-600 dark:text-green-400">Authentic</div>

                                      </div>

                                      <div className="text-center p-2 bg-red-50 dark:bg-red-900/20 rounded">

                                        <div className="text-sm font-semibold text-red-700 dark:text-red-400">

                                          {analysisResult.details.categoryBreakdown.manipulated > 1 ? Math.round(analysisResult.details.categoryBreakdown.manipulated) : Math.round(analysisResult.details.categoryBreakdown.manipulated \* 100)}%

                                        </div>

                                        <div className="text-xs text-red-600 dark:text-red-400">Manipulated</div>

                                      </div>

                                      <div className="text-center p-2 bg-yellow-50 dark:bg-yellow-900/20 rounded">

                                        <div className="text-sm font-semibold text-yellow-700 dark:text-yellow-400">

                                          {analysisResult.details.categoryBreakdown.inconclusive > 1 ? Math.round(analysisResult.details.categoryBreakdown.inconclusive) : Math.round(analysisResult.details.categoryBreakdown.inconclusive \* 100)}%

                                        </div>

                                        <div className="text-xs text-yellow-600 dark:text-yellow-400">Inconclusive</div>

                                      </div>

                                    </div>

                                  </div>

                                  {analysisResult.details.metadata && (

                                    <div>

                                      <span className="text-muted-foreground text-sm">Analysis Details:</span>

                                      <div className="grid grid-cols-2 gap-4 text-sm mt-1">

                                        <div>

                                          <span className="text-muted-foreground">Models Analyzed:</span>

                                          <p className="font-medium">{(analysisResult.details.metadata as any).modelsAnalyzed || 'N/A'}</p>

                                        </div>

                                        <div>

                                          <span className="text-muted-foreground">Completed Models:</span>

                                          <p className="font-medium">{(analysisResult.details.metadata as any).completedModels || 'N/A'}</p>

                                        </div>

                                      </div>

                                    </div>

                                  )}

                                </div>

                              </div>

                            </div>

                          </CardContent>

                        </Card>

                        {*/\* Export PDF Button \*/*}

                        <div className="flex justify-center">

                          <PDFExportDialog

                            analysis={{

                              ...analysisResult,

                              id: analysisResult.id,

                              timestamp: analysisResult.timestamp,

                              fileSize: analysisResult.fileSize,

                              fileType: analysisResult.fileType,

                              details: analysisResult.details,

                            }}

                            trigger={

                              <Button className="px-8">

                                <FileText className="w-4 h-4 mr-2" />

                                Export PDF Report

                              </Button>

                            }

                          />

                        </div>

                      </motion.div>

                    )}

                  </AnimatePresence>

                  {*/\* Hidden PDF Components \*/*}

                  {analysisResult && (

                    <PDFChartComponents analysis={{

                      ...analysisResult,

                      id: analysisResult.id,

                      timestamp: analysisResult.timestamp,

                      fileSize: analysisResult.fileSize,

                      fileType: analysisResult.fileType,

                      details: analysisResult.details,

                    }} />

                  )}

                </motion.div>

              ) : (

                <motion.div

                  key="usage"

                  initial={{ opacity: 0, y: 20 }}

                  animate={{ opacity: 1, y: 0 }}

                  exit={{ opacity: 0, y: -20 }}

                  transition={{ duration: 0.3 }}

                >

                  <UsageDashboard />

                </motion.div>

              )}

            </AnimatePresence>

          </div>

        </div>

      </div>

    </div>

  );

}

## 

## 1.9 Advanced Features Implementation

**src/components/layout/navbar.tsx:**

'use client';

import { useState, useEffect } from 'react';

import Link from 'next/link';

import { usePathname } from 'next/navigation';

import { Button } from '@/components/ui/button';

import { Switch } from '@/components/ui/switch';

import {

  DropdownMenu,

  DropdownMenuContent,

  DropdownMenuItem,

  DropdownMenuTrigger,

} from '@/components/ui/dropdown-menu';

import {

  Menu,

  X,

  Shield,

  Sun,

  Moon,

  Monitor,

  Settings,

  History,

} from 'lucide-react';

import { motion, AnimatePresence } from 'framer-motion';

interface NavbarProps {

  onMenuToggle?: () => void;

  showMobileMenu?: boolean;

}

export function Navbar({ onMenuToggle, showMobileMenu }: NavbarProps) {

  const [theme, setTheme] = useState<'light' | 'dark' | 'system'>('system');

  const [mounted, setMounted] = useState(false);

  const pathname = usePathname();

  useEffect(() => {

    setMounted(true);

*// Load theme from localStorage*

    const savedTheme = localStorage.getItem('deepfake\_theme') as 'light' | 'dark' | 'system';

    if (savedTheme) {

      setTheme(savedTheme);

    }

  }, []);

  useEffect(() => {

    if (!mounted) return;

    const root = document.documentElement;

    if (theme === 'system') {

      const mediaQuery = window.matchMedia('(prefers-color-scheme: dark)');

      root.classList.toggle('dark', mediaQuery.matches);

      const handler = (e: MediaQueryListEvent) => {

        root.classList.toggle('dark', e.matches);

      };

      mediaQuery.addEventListener('change', handler);

      return () => mediaQuery.removeEventListener('change', handler);

    } else {

      root.classList.toggle('dark', theme === 'dark');

    }

  }, [theme, mounted]);

  const handleThemeChange = (newTheme: 'light' | 'dark' | 'system') => {

    setTheme(newTheme);

    localStorage.setItem('deepfake\_theme', newTheme);

  };

  const navigationItems = [

    { href: '/', label: 'Home', active: pathname === '/' },

    { href: '/upload', label: 'Analyze', active: pathname === '/upload' },

    { href: '/results', label: 'Results', active: pathname === '/results' },

  ];

  const ThemeIcon = theme === 'light' ? Sun : theme === 'dark' ? Moon : Monitor;

  if (!mounted) {

    return (

      <nav className="sticky top-0 z-50 w-full border-b bg-background/95 backdrop-blur supports-[backdrop-filter]:bg-background/60">

        <div className="container flex h-16 items-center justify-between px-4">

          <div className="flex items-center space-x-2">

            <Shield className="h-6 w-6 text-primary" />

            <span className="font-semibold">ITL Deepfake Detective</span>

          </div>

          <div className="w-8 h-8" /> {*/\* Placeholder for theme toggle \*/*}

        </div>

      </nav>

    );

  }

  return (

    <nav className="sticky top-0 z-50 w-full border-b bg-background/95 backdrop-blur supports-[backdrop-filter]:bg-background/60">

      <div className="container flex h-16 items-center justify-between px-4">

        {*/\* Logo and Brand \*/*}

        <div className="flex items-center space-x-2">

          <Link href="/" className="flex items-center space-x-2 hover:opacity-80 transition-opacity">

            <Shield className="h-6 w-6 text-primary" />

            <span className="font-semibold text-foreground">ITL Deepfake Detective</span>

          </Link>

        </div>

        {*/\* Desktop Navigation \*/*}

        <div className="hidden md:flex items-center space-x-6">

          {navigationItems.map((item) => (

            <Link

              key={item.href}

              href={item.href}

              className={`text-sm font-medium transition-colors hover:text-primary ${

                item.active

                  ? 'text-primary border-b-2 border-primary pb-1'

                  : 'text-muted-foreground'

              }`}

            >

              {item.label}

            </Link>

          ))}

        </div>

        {*/\* Actions \*/*}

        <div className="flex items-center space-x-2">

          {*/\* Theme Toggle \*/*}

          <DropdownMenu>

            <DropdownMenuTrigger asChild>

              <Button variant="ghost" size="sm" className="w-9 px-0">

                <ThemeIcon className="h-4 w-4" />

                <span className="sr-only">Toggle theme</span>

              </Button>

            </DropdownMenuTrigger>

            <DropdownMenuContent align="end">

              <DropdownMenuItem onClick={() => handleThemeChange('light')}>

                <Sun className="mr-2 h-4 w-4" />

                Light

              </DropdownMenuItem>

              <DropdownMenuItem onClick={() => handleThemeChange('dark')}>

                <Moon className="mr-2 h-4 w-4" />

                Dark

              </DropdownMenuItem>

              <DropdownMenuItem onClick={() => handleThemeChange('system')}>

                <Monitor className="mr-2 h-4 w-4" />

                System

              </DropdownMenuItem>

            </DropdownMenuContent>

          </DropdownMenu>

          {*/\* Mobile Menu Toggle \*/*}

          <Button

            variant="ghost"

            size="sm"

            className="md:hidden w-9 px-0"

            onClick={onMenuToggle}

          >

            {showMobileMenu ? (

              <X className="h-4 w-4" />

            ) : (

              <Menu className="h-4 w-4" />

            )}

            <span className="sr-only">Toggle menu</span>

          </Button>

        </div>

      </div>

      {*/\* Mobile Navigation \*/*}

      <AnimatePresence>

        {showMobileMenu && (

          <motion.div

            initial={{ height: 0, opacity: 0 }}

            animate={{ height: 'auto', opacity: 1 }}

            exit={{ height: 0, opacity: 0 }}

            transition={{ duration: 0.2 }}

            className="md:hidden border-t overflow-hidden"

          >

            <div className="container px-4 py-4 space-y-3">

              {navigationItems.map((item) => (

                <Link

                  key={item.href}

                  href={item.href}

                  className={`block px-3 py-2 text-sm font-medium rounded-md transition-colors ${

                    item.active

                      ? 'bg-primary text-primary-foreground'

                      : 'text-muted-foreground hover:text-foreground hover:bg-accent'

                  }`}

                >

                  {item.label}

                </Link>

              ))}

              <div className="pt-3 border-t">

                <div className="flex items-center justify-between px-3 py-2 text-sm">

                  <span className="text-muted-foreground">Theme</span>

                  <div className="flex items-center space-x-2">

                    <Button

                      variant={theme === 'light' ? 'default' : 'outline'}

                      size="sm"

                      onClick={() => handleThemeChange('light')}

                      className="w-8 h-8 p-0"

                    >

                      <Sun className="h-3 w-3" />

                    </Button>

                    <Button

                      variant={theme === 'dark' ? 'default' : 'outline'}

                      size="sm"

                      onClick={() => handleThemeChange('dark')}

                      className="w-8 h-8 p-0"

                    >

                      <Moon className="h-3 w-3" />

                    </Button>

                    <Button

                      variant={theme === 'system' ? 'default' : 'outline'}

                      size="sm"

                      onClick={() => handleThemeChange('system')}

                      className="w-8 h-8 p-0"

                    >

                      <Monitor className="h-3 w-3" />

                    </Button>

                  </div>

                </div>

              </div>

            </div>

          </motion.div>

        )}

      </AnimatePresence>

    </nav>

  );

}

**src/components/layout/sidebar.tsx:**

'use client';

import { useState, useEffect } from 'react';

import Link from 'next/link';

import { Button } from '@/components/ui/button';

import { Badge } from '@/components/ui/badge';

import { Separator } from '@/components/ui/separator';

import {

  Card,

  CardContent,

  CardDescription,

  CardHeader,

  CardTitle,

} from '@/components/ui/card';

import {

  History,

  FileImage,

  FileVideo,

  FileAudio,

  File,

  Trash2,

  Download,

  AlertTriangle,

  CheckCircle,

  XCircle,

  Clock,

  ChevronRight,

} from 'lucide-react';

import { motion, AnimatePresence } from 'framer-motion';

import { storage, type StoredAnalysis } from '@/lib/storage';

import { formatFileSize, getFileIcon } from '@/lib/storage';

interface SidebarProps {

  className?: string;

  onAnalysisSelect?: (analysisId: string) => void;

}

export function Sidebar({ className, onAnalysisSelect }: SidebarProps) {

  const [analyses, setAnalyses] = useState<StoredAnalysis[]>([]);

  const [isCollapsed, setIsCollapsed] = useState(false);

  useEffect(() => {

    loadAnalyses();

*// Listen for storage changes*

    const handleStorageChange = () => {

      loadAnalyses();

    };

    window.addEventListener('storage', handleStorageChange);

    return () => window.removeEventListener('storage', handleStorageChange);

  }, []);

  const loadAnalyses = () => {

    const history = storage.getAnalysisHistory();

    setAnalyses(history);

  };

  const handleDeleteAnalysis = (analysisId: string) => {

    storage.removeAnalysisFromHistory(analysisId);

    loadAnalyses();

  };

  const getConfidenceColor = (confidence: number) => {

    if (confidence < 0.3) return 'text-green-600 dark:text-green-400';

    if (confidence < 0.7) return 'text-yellow-600 dark:text-yellow-400';

    return 'text-red-600 dark:text-red-400';

  };

  const getConfidenceIcon = (confidence: number) => {

    if (confidence < 0.3) return CheckCircle;

    if (confidence < 0.7) return AlertTriangle;

    return XCircle;

  };

  const formatTimestamp = (timestamp: string) => {

    const date = new Date(timestamp);

    const now = new Date();

    const diff = now.getTime() - date.getTime();

    const minutes = Math.floor(diff / (1000 \* 60));

    const hours = Math.floor(diff / (1000 \* 60 \* 60));

    const days = Math.floor(diff / (1000 \* 60 \* 60 \* 24));

    if (minutes < 1) return 'Just now';

    if (minutes < 60) return `${minutes}m ago`;

    if (hours < 24) return `${hours}h ago`;

    if (days < 7) return `${days}d ago`;

    return date.toLocaleDateString();

  };

  if (isCollapsed) {

    return (

      <motion.div

        initial={{ width: 280 }}

        animate={{ width: 60 }}

        transition={{ duration: 0.3 }}

        className={`bg-card border-r h-screen sticky top-16 overflow-hidden ${className}`}

      >

        <div className="p-3">

          <Button

            variant="ghost"

            size="sm"

            onClick={() => setIsCollapsed(false)}

            className="w-full p-2"

          >

            <History className="h-5 w-5" />

          </Button>

        </div>

      </motion.div>

    );

  }

  return (

    <motion.div

      initial={{ width: 60 }}

      animate={{ width: 280 }}

      transition={{ duration: 0.3 }}

      className={`bg-card border-r h-screen sticky top-16 overflow-hidden ${className}`}

    >

      <div className="flex flex-col h-full">

        {*/\* Header \*/*}

        <div className="p-4 border-b">

          <div className="flex items-center justify-between">

            <h2 className="font-semibold text-sm flex items-center gap-2">

              <History className="h-4 w-4" />

              Recent Results

            </h2>

            <Button

              variant="ghost"

              size="sm"

              onClick={() => setIsCollapsed(true)}

              className="h-6 w-6 p-0"

            >

              <ChevronRight className="h-3 w-3" />

            </Button>

          </div>

          <p className="text-xs text-muted-foreground mt-1">

            {analyses.length} {analyses.length === 1 ? 'analysis' : 'analyses'}

          </p>

        </div>

        {*/\* Content \*/*}

        <div className="flex-1 overflow-y-auto p-2">

          <AnimatePresence>

            {analyses.length === 0 ? (

              <motion.div

                initial={{ opacity: 0, y: 20 }}

                animate={{ opacity: 1, y: 0 }}

                className="text-center py-8 px-4"

              >

                <History className="h-8 w-8 text-muted-foreground mx-auto mb-3" />

                <p className="text-sm text-muted-foreground mb-2">No recent analyses</p>

                <p className="text-xs text-muted-foreground">

                  Upload and analyze files to see results here

                </p>

              </motion.div>

            ) : (

              <div className="space-y-2">

                {analyses.map((analysis, index) => {

                  const ConfidenceIcon = getConfidenceIcon(analysis.confidence);

                  return (

                    <motion.div

                      key={analysis.id}

                      initial={{ opacity: 0, x: -20 }}

                      animate={{ opacity: 1, x: 0 }}

                      transition={{ delay: index \* 0.1 }}

                    >

                      <Card

                        className="cursor-pointer hover:bg-accent/50 transition-colors group"

                        onClick={() => onAnalysisSelect?.(analysis.id)}

                      >

                        <CardContent className="p-3">

                          <div className="flex items-start gap-2">

                            {*/\* File Preview/Icon \*/*}

                            <div className="w-10 h-10 rounded bg-muted flex items-center justify-center flex-shrink-0">

                              {analysis.thumbnailBlob ? (

                                <img

                                  src={analysis.thumbnailBlob}

                                  alt={analysis.filename}

                                  className="w-full h-full object-cover rounded"

                                />

                              ) : (

                                <span className="text-lg">{getFileIcon(analysis.fileType)}</span>

                              )}

                            </div>

                            {*/\* Content \*/*}

                            <div className="flex-1 min-w-0">

                              <div className="flex items-start justify-between gap-1">

                                <h4 className="text-xs font-medium truncate" title={analysis.filename}>

                                  {analysis.filename}

                                </h4>

                                <Badge

                                  variant="secondary"

                                  className={`text-xs px-1 py-0 ${getConfidenceColor(analysis.confidence)}`}

                                >

                                  {analysis.confidence > 1 ? Math.round(analysis.confidence) : Math.round(analysis.confidence \* 100)}%

                                </Badge>

                              </div>

                              <div className="flex items-center gap-1 mt-1">

                                <ConfidenceIcon className={`h-3 w-3 ${getConfidenceColor(analysis.confidence)}`} />

                                <span className="text-xs text-muted-foreground capitalize">

                                  {analysis.prediction}

                                </span>

                              </div>

                              <div className="flex items-center justify-between mt-2">

                                <span className="text-xs text-muted-foreground">

                                  {formatFileSize(analysis.fileSize)}

                                </span>

                                <span className="text-xs text-muted-foreground flex items-center gap-1">

                                  <Clock className="h-2.5 w-2.5" />

                                  {formatTimestamp(analysis.timestamp)}

                                </span>

                              </div>

                            </div>

                          </div>

                          {*/\* Actions \*/*}

                          <div className="flex items-center justify-end gap-1 mt-2 opacity-0 group-hover:opacity-100 transition-opacity">

                            <Link href={`/results?id=${analysis.id}`}>

                              <Button

                                variant="ghost"

                                size="sm"

                                className="h-6 w-6 p-0"

                                onClick={(e) => e.stopPropagation()}

                              >

                                <ChevronRight className="h-3 w-3" />

                              </Button>

                            </Link>

                            <Button

                              variant="ghost"

                              size="sm"

                              className="h-6 w-6 p-0 hover:text-destructive"

                              onClick={(e) => {

                                e.stopPropagation();

                                handleDeleteAnalysis(analysis.id);

                              }}

                            >

                              <Trash2 className="h-3 w-3" />

                            </Button>

                          </div>

                        </CardContent>

                      </Card>

                    </motion.div>

                  );

                })}

              </div>

            )}

          </AnimatePresence>

        </div>

        {*/\* Footer \*/*}

        {analyses.length > 0 && (

          <div className="p-3 border-t">

            <div className="flex gap-2">

              <Button

                variant="outline"

                size="sm"

                className="flex-1 text-xs h-7"

                onClick={() => {

                  const data = storage.exportData();

                  const blob = new Blob([data], { type: 'application/json' });

                  const url = URL.createObjectURL(blob);

                  const a = document.createElement('a');

                  a.href = url;

                  a.download = `deepfake-analysis-${new Date().toISOString().split('T')[0]}.json`;

                  a.click();

                  URL.revokeObjectURL(url);

                }}

              >

                <Download className="h-3 w-3 mr-1" />

                Export

              </Button>

              <Button

                variant="outline"

                size="sm"

                className="flex-1 text-xs h-7 hover:text-destructive"

                onClick={() => {

                  if (confirm('Clear all analysis history?')) {

                    storage.clearAnalysisHistory();

                    loadAnalyses();

                  }

                }}

              >

                <Trash2 className="h-3 w-3 mr-1" />

                Clear

              </Button>

            </div>

          </div>

        )}

      </div>

    </motion.div>

  );

}

### 1.9.2 Step 1: Enhanced Root Layout with Advanced Features

**src/app/layout.tsx:**

import type { Metadata } from "next";  
import { Geist, Geist\_Mono } from "next/font/google";  
import "./globals.css";  
import { Navbar } from "@/components/layout/navbar";  
import { Toaster } from "@/components/ui/sonner";  
  
const geistSans = Geist({  
 variable: "--font-geist-sans",  
 subsets: ["latin"],  
});  
  
const geistMono = Geist\_Mono({  
 variable: "--font-geist-mono",  
 subsets: ["latin"],  
});  
  
export const metadata: Metadata = {  
 title: "ITL Deepfake Detective | AI-Powered Media Authentication",  
 description: "Advanced deepfake detection for images, videos, and audio files using AI technology. Verify media authenticity instantly.",  
 keywords: "deepfake detection, AI, media verification, fake media detection, video analysis, image authentication",  
 authors: [{ name: "ITL Deepfake Detective Team" }],  
};  
  
export const viewport = {  
 width: 'device-width',  
 initialScale: 1,  
};  
  
export default function RootLayout({  
 children,  
}: Readonly<{  
 children: React.ReactNode;  
}>) {  
 return (  
 <html lang="en" suppressHydrationWarning>  
 <body  
 className={`${geistSans.variable} ${geistMono.variable} antialiased`}  
 suppressHydrationWarning  
 >  
 <div className="relative flex min-h-screen flex-col">  
 <Navbar />  
 <main className="flex-1">{children}</main>  
 </div>  
 <Toaster />  
 </body>  
 </html>  
 );  
}

### 1.9.3 Step 2: Advanced Tailwind CSS v4 Styling

**src/app/globals.css:**

@import "tailwindcss";  
@import "tw-animate-css";  
  
@custom-variant dark (&:is(.dark \*));  
  
@theme inline {  
 --color-background: var(--background);  
 --color-foreground: var(--foreground);  
 --font-sans: var(--font-geist-sans);  
 --font-mono: var(--font-geist-mono);  
 --color-sidebar-ring: var(--sidebar-ring);  
 --color-sidebar-border: var(--sidebar-border);  
 --color-sidebar-accent-foreground: var(--sidebar-accent-foreground);  
 --color-sidebar-accent: var(--sidebar-accent);  
 --color-sidebar-primary-foreground: var(--sidebar-primary-foreground);  
 --color-sidebar-primary: var(--sidebar-primary);  
 --color-sidebar-foreground: var(--sidebar-foreground);  
 --color-sidebar: var(--sidebar);  
 --color-chart-5: var(--chart-5);  
 --color-chart-4: var(--chart-4);  
 --color-chart-3: var(--chart-3);  
 --color-chart-2: var(--chart-2);  
 --color-chart-1: var(--chart-1);  
 --color-ring: var(--ring);  
 --color-input: var(--input);  
 --color-border: var(--border);  
 --color-destructive: var(--destructive);  
 --color-accent-foreground: var(--accent-foreground);  
 --color-accent: var(--accent);  
 --color-muted-foreground: var(--muted-foreground);  
 --color-muted: var(--muted);  
 --color-secondary-foreground: var(--secondary-foreground);  
 --color-secondary: var(--secondary);  
 --color-primary-foreground: var(--primary-foreground);  
 --color-primary: var(--primary);  
 --color-popover-foreground: var(--popover-foreground);  
 --color-popover: var(--popover);  
 --color-card-foreground: var(--card-foreground);  
 --color-card: var(--card);  
 --radius-sm: calc(var(--radius) - 4px);  
 --radius-md: calc(var(--radius) - 2px);  
 --radius-lg: var(--radius);  
 --radius-xl: calc(var(--radius) + 4px);  
}  
  
:root {  
 --radius: 0.625rem;  
 --background: #ffffff;  
 --foreground: #242424;  
 --card: #ffffff;  
 --card-foreground: #242424;  
 --popover: #ffffff;  
 --popover-foreground: #242424;  
 --primary: #343434;  
 --primary-foreground: #fafafa;  
 --secondary: #f7f7f7;  
 --secondary-foreground: #343434;  
 --muted: #f7f7f7;  
 --muted-foreground: #8e8e8e;  
 --accent: #f7f7f7;  
 --accent-foreground: #343434;  
 --destructive: #dc2626;  
 --border: #e5e5e5;  
 --input: #e5e5e5;  
 --ring: #b4b4b4;  
 --chart-1: #f97316;  
 --chart-2: #06b6d4;  
 --chart-3: #6366f1;  
 --chart-4: #84cc16;  
 --chart-5: #eab308;  
 --sidebar: #fafafa;  
 --sidebar-foreground: #242424;  
 --sidebar-primary: #343434;  
 --sidebar-primary-foreground: #fafafa;  
 --sidebar-accent: #f7f7f7;  
 --sidebar-accent-foreground: #343434;  
 --sidebar-border: #e5e5e5;  
 --sidebar-ring: #b4b4b4;  
}  
  
.dark {  
 --background: #0a0a0a;  
 --foreground: #fafafa;  
 --card: #141414;  
 --card-foreground: #fafafa;  
 --popover: #141414;  
 --popover-foreground: #fafafa;  
 --primary: #ffffff;  
 --primary-foreground: #0a0a0a;  
 --secondary: #1a1a1a;  
 --secondary-foreground: #fafafa;  
 --muted: #1a1a1a;  
 --muted-foreground: #a1a1aa;  
 --accent: #1a1a1a;  
 --accent-foreground: #fafafa;  
 --destructive: #ef4444;  
 --border: rgba(255, 255, 255, 0.1);  
 --input: rgba(255, 255, 255, 0.05);  
 --ring: #404040;  
 --chart-1: #8b5cf6;  
 --chart-2: #10b981;  
 --chart-3: #eab308;  
 --chart-4: #a855f7;  
 --chart-5: #f59e0b;  
 --sidebar: #141414;  
 --sidebar-foreground: #fafafa;  
 --sidebar-primary: #8b5cf6;  
 --sidebar-primary-foreground: #fafafa;  
 --sidebar-accent: #1a1a1a;  
 --sidebar-accent-foreground: #fafafa;  
 --sidebar-border: rgba(255, 255, 255, 0.1);  
 --sidebar-ring: #404040;  
}  
  
@layer base {  
 \* {  
 @apply border-border outline-ring/50;  
 }  
 body {  
 @apply bg-background text-foreground;  
 }  
}  
  
@layer components {  
 .gradient-bg {  
 @apply bg-gradient-to-br from-blue-50 via-white to-purple-50 dark:from-black dark:via-gray-950 dark:to-gray-900;  
 }  
   
 .glass-morphism {  
 @apply bg-white/10 dark:bg-black/10 backdrop-blur-md border border-white/20 dark:border-white/10;  
 }  
   
 .confidence-high {  
 @apply text-green-600 dark:text-green-400;  
 }  
   
 .confidence-medium {  
 @apply text-yellow-600 dark:text-yellow-400;  
 }  
   
 .confidence-low {  
 @apply text-red-600 dark:text-red-400;  
 }  
   
 .upload-zone {  
 @apply border-2 border-dashed border-muted-foreground/25 hover:border-primary/50 transition-colors duration-200;  
 }  
   
 .upload-zone.active {  
 @apply border-primary bg-primary/5;  
 }  
}  
  
@layer utilities {  
 .animate-pulse-slow {  
 animation: pulse 3s cubic-bezier(0.4, 0, 0.6, 1) infinite;  
 }  
   
 .animate-bounce-subtle {  
 animation: bounce-subtle 2s infinite;  
 }  
   
 .animate-scan {  
 animation: scan 2s ease-in-out infinite;  
 }  
}  
  
@keyframes bounce-subtle {  
 0%, 100% {  
 transform: translateY(0);  
 }  
 50% {  
 transform: translateY(-4px);  
 }  
}  
  
@keyframes scan {  
 0% {  
 transform: translateX(-100%);  
 }  
 50% {  
 transform: translateX(100%);  
 }  
 100% {  
 transform: translateX(-100%);  
 }  
}

### 1.9.4 Step 3: Modern Landing Page with Advanced Features

**src/app/page.tsx:**

'use client';  
  
import Link from "next/link";  
import { Button } from "@/components/ui/button";  
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card";  
import { Badge } from "@/components/ui/badge";  
import {  
 Shield,  
 Zap,  
 Eye,  
 Upload,  
 Brain,  
 Lock,  
 FileImage,  
 FileVideo,  
 FileAudio,  
 ArrowRight,  
 Github,  
 Star,  
} from "lucide-react";  
import { motion } from "framer-motion";  
  
export default function Home() {  
 const features = [  
 {  
 icon: Brain,  
 title: "AI-Powered Detection",  
 description: "Advanced machine learning algorithms analyze media for manipulation patterns and deepfake signatures.",  
 },  
 {  
 icon: Zap,  
 title: "Lightning Fast",  
 description: "Get results in seconds with our optimized processing pipeline and cloud-based analysis.",  
 },  
 {  
 icon: Lock,  
 title: "Secure & Private",  
 description: "Your files are processed securely and never stored permanently. Complete privacy guaranteed.",  
 },  
 {  
 icon: Eye,  
 title: "Detailed Analysis",  
 description: "Comprehensive reports with confidence scores, frame-by-frame analysis, and visual breakdowns.",  
 },  
 ];  
  
 const supportedFormats = [  
 { icon: FileImage, name: "Images", formats: "JPG, PNG, GIF, WebP" },  
 { icon: FileVideo, name: "Videos", formats: "MP4, WebM, MOV, AVI" },  
 { icon: FileAudio, name: "Audio", formats: "MP3, WAV, FLAC, AAC" },  
 ];  
  
 const stats = [  
 { value: "99.2%", label: "Accuracy Rate" },  
 { value: "<3s", label: "Average Analysis" },  
 { value: "50M+", label: "Files Analyzed" },  
 { value: "24/7", label: "Availability" },  
 ];  
  
 return (  
 <div className="min-h-screen gradient-bg">  
 {/\* Hero Section \*/}  
 <section className="relative overflow-hidden py-20 sm:py-32">  
 <div className="absolute inset-0 bg-grid-white/[0.02] bg-[size:60px\_60px]" />  
 <div className="relative container mx-auto px-4 sm:px-6 lg:px-8">  
 <motion.div  
 initial={{ opacity: 0, y: 20 }}  
 animate={{ opacity: 1, y: 0 }}  
 transition={{ duration: 0.8 }}  
 className="text-center max-w-4xl mx-auto"  
 >  
 <div className="flex items-center justify-center gap-2 mb-6">  
 <Badge variant="outline" className="px-3 py-1">  
 <Star className="w-3 h-3 mr-1 fill-current" />  
 AI-Powered Detection  
 </Badge>  
 </div>  
   
 <h1 className="text-4xl sm:text-6xl lg:text-7xl font-bold tracking-tight mb-6">  
 Upload Media.{" "}  
 <span className="text-transparent bg-clip-text bg-gradient-to-r from-blue-600 to-purple-600">  
 Detect Deepfakes.  
 </span>  
 </h1>  
   
 <p className="text-xl text-muted-foreground mb-8 max-w-2xl mx-auto">  
 Advanced AI-powered deepfake detection for images, videos, and audio files.   
 Verify media authenticity with enterprise-grade accuracy in seconds.  
 </p>  
   
 <div className="flex flex-col sm:flex-row gap-4 justify-center items-center mb-12">  
 <Link href="/upload" prefetch={true}>  
 <Button size="lg" className="font-semibold px-8 py-6 text-base">  
 <Upload className="mr-2 h-5 w-5" />  
 Try Detection  
 <ArrowRight className="ml-2 h-4 w-4" />  
 </Button>  
 </Link>  
 <Button variant="outline" size="lg" className="font-semibold px-8 py-6 text-base">  
 <Github className="mr-2 h-5 w-5" />  
 View Demo  
 </Button>  
 </div>  
  
 {/\* Stats \*/}  
 <div className="grid grid-cols-2 md:grid-cols-4 gap-8 max-w-2xl mx-auto">  
 {stats.map((stat, index) => (  
 <motion.div  
 key={stat.label}  
 initial={{ opacity: 0, y: 20 }}  
 animate={{ opacity: 1, y: 0 }}  
 transition={{ duration: 0.5, delay: 0.2 + index \* 0.1 }}  
 className="text-center"  
 >  
 <div className="text-2xl sm:text-3xl font-bold text-primary mb-1">  
 {stat.value}  
 </div>  
 <div className="text-sm text-muted-foreground">{stat.label}</div>  
 </motion.div>  
 ))}  
 </div>  
 </motion.div>  
 </div>  
 </section>  
  
 {/\* Features Section \*/}  
 <section className="py-20 bg-muted/30">  
 <div className="container mx-auto px-4 sm:px-6 lg:px-8">  
 <motion.div  
 initial={{ opacity: 0, y: 20 }}  
 whileInView={{ opacity: 1, y: 0 }}  
 viewport={{ once: true }}  
 transition={{ duration: 0.6 }}  
 className="text-center mb-16"  
 >  
 <h2 className="text-3xl sm:text-4xl font-bold mb-4">  
 Why Choose ITL Deepfake Detective?  
 </h2>  
 <p className="text-xl text-muted-foreground max-w-2xl mx-auto">  
 Built with cutting-edge AI technology to provide the most accurate and reliable deepfake detection available.  
 </p>  
 </motion.div>  
   
 <div className="grid md:grid-cols-2 lg:grid-cols-4 gap-8">  
 {features.map((feature, index) => (  
 <motion.div  
 key={feature.title}  
 initial={{ opacity: 0, y: 20 }}  
 whileInView={{ opacity: 1, y: 0 }}  
 viewport={{ once: true }}  
 transition={{ duration: 0.5, delay: index \* 0.1 }}  
 >  
 <Card className="h-full hover:shadow-lg transition-shadow">  
 <CardHeader>  
 <feature.icon className="h-10 w-10 text-primary mb-4" />  
 <CardTitle className="text-lg">{feature.title}</CardTitle>  
 </CardHeader>  
 <CardContent>  
 <CardDescription className="text-sm leading-relaxed">  
 {feature.description}  
 </CardDescription>  
 </CardContent>  
 </Card>  
 </motion.div>  
 ))}  
 </div>  
 </div>  
 </section>  
  
 {/\* Supported Formats \*/}  
 <section className="py-20">  
 <div className="container mx-auto px-4 sm:px-6 lg:px-8">  
 <motion.div  
 initial={{ opacity: 0, y: 20 }}  
 whileInView={{ opacity: 1, y: 0 }}  
 viewport={{ once: true }}  
 transition={{ duration: 0.6 }}  
 className="text-center mb-16"  
 >  
 <h2 className="text-3xl sm:text-4xl font-bold mb-4">  
 Supports All Media Types  
 </h2>  
 <p className="text-xl text-muted-foreground max-w-2xl mx-auto">  
 Comprehensive analysis for images, videos, and audio files with detailed insights and frame-by-frame detection.  
 </p>  
 </motion.div>  
   
 <div className="grid md:grid-cols-3 gap-8 max-w-4xl mx-auto">  
 {supportedFormats.map((format, index) => (  
 <motion.div  
 key={format.name}  
 initial={{ opacity: 0, y: 20 }}  
 whileInView={{ opacity: 1, y: 0 }}  
 viewport={{ once: true }}  
 transition={{ duration: 0.5, delay: index \* 0.1 }}  
 >  
 <Card className="text-center hover:shadow-lg transition-shadow">  
 <CardHeader>  
 <format.icon className="h-16 w-16 text-primary mx-auto mb-4" />  
 <CardTitle className="text-xl">{format.name}</CardTitle>  
 <CardDescription className="text-sm">  
 {format.formats}  
 </CardDescription>  
 </CardHeader>  
 </Card>  
 </motion.div>  
 ))}  
 </div>  
 </div>  
 </section>  
  
 {/\* CTA Section \*/}  
 <section className="py-20 bg-muted/30">  
 <div className="container mx-auto px-4 sm:px-6 lg:px-8">  
 <motion.div  
 initial={{ opacity: 0, y: 20 }}  
 whileInView={{ opacity: 1, y: 0 }}  
 viewport={{ once: true }}  
 transition={{ duration: 0.6 }}  
 className="text-center max-w-3xl mx-auto"  
 >  
 <Shield className="h-16 w-16 text-primary mx-auto mb-6" />  
 <h2 className="text-3xl sm:text-4xl font-bold mb-4">  
 Ready to Verify Your Media?  
 </h2>  
 <p className="text-xl text-muted-foreground mb-8">  
 Start detecting deepfakes with our advanced AI technology.   
 Upload your file and get results in seconds.  
 </p>  
 <Link href="/upload" prefetch={true}>  
 <Button size="lg" className="font-semibold px-8 py-6 text-base">  
 <Upload className="mr-2 h-5 w-5" />  
 Start Free Analysis  
 <ArrowRight className="ml-2 h-4 w-4" />  
 </Button>  
 </Link>  
 </motion.div>  
 </div>  
 </section>  
 </div>  
 );  
}

### 1.9.5 Step 6: Results Page

**src/app/results/page.tsx:**

'use client';

import { useState, useEffect, Suspense } from 'react';

import { useSearchParams, useRouter } from 'next/navigation';

import { ConfidenceGauge } from '@/components/charts/confidence-gauge';

import { CategoryChart } from '@/components/charts/category-chart';

import { Sidebar } from '@/components/layout/sidebar';

import { Button } from '@/components/ui/button';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

import { Separator } from '@/components/ui/separator';

import { Progress } from '@/components/ui/progress';

import {

  Download,

  ArrowLeft,

  Clock,

  FileText,

  Activity,

  Image as ImageIcon,

  Video,

  Volume2,

  AlertTriangle,

  CheckCircle,

  XCircle,

  Share,

  Trash2,

  BarChart3,

  Lightbulb,

} from 'lucide-react';

import { motion, AnimatePresence } from 'framer-motion';

import { LineChart, Line, XAxis, YAxis, CartesianGrid, ResponsiveContainer, Tooltip, AreaChart, Area } from 'recharts';

import { storage, formatFileSize, getFileIcon, type StoredAnalysis } from '@/lib/storage';

import { usageTracker } from '@/lib/usage-tracker';

import {

  RiskHeatmap,

  AnomalyScatterPlot,

  MultidimensionalRadar,

  ConfidenceTimelineAdvanced,

  ComparativeAnalysis,

  FileTypeTreemap,

} from '@/components/charts/advanced-charts';

import { ExplanationDashboard } from '@/components/explanation/explanation-dashboard';

import { toast } from 'sonner';

import { PDFExportDialog } from '@/components/pdf/pdf-export-dialog';

import { PDFChartComponents } from '@/components/pdf/pdf-chart-components';

function ResultsPageContent() {

  const [analysis, setAnalysis] = useState<StoredAnalysis | null>(null);

  const [loading, setLoading] = useState(true);

  const [activeTab, setActiveTab] = useState<'overview' | 'explanation' | 'timeline' | 'advanced' | 'details'>('overview');

  const searchParams = useSearchParams();

  const router = useRouter();

  const analysisId = searchParams.get('id');

  useEffect(() => {

    if (!analysisId) {

      router.push('/upload');

      return;

    }

    const result = storage.getAnalysisById(analysisId);

    if (result) {

      setAnalysis(result);

    } else {

      toast.error('Analysis not found');

      router.push('/upload');

    }

    setLoading(false);

  }, [analysisId, router]);

  const handleDelete = () => {

    if (!analysis) return;

    if (confirm('Are you sure you want to delete this analysis?')) {

      storage.removeAnalysisFromHistory(analysis.id);

      toast.success('Analysis deleted successfully');

      router.push('/upload');

    }

  };

  const downloadReport = () => {

    if (!analysis) return;

    const report = {

      id: analysis.id,

      filename: analysis.filename,

      analysis: {

        confidence: analysis.confidence,

        prediction: analysis.prediction,

        timestamp: analysis.timestamp,

        processingTime: analysis.processingTime,

      },

      details: analysis.details,

      explanation: analysis.explanation ? {

        summary: analysis.explanation.summary,

        reasons: analysis.explanation.reasons,

        evidence: analysis.explanation.evidence,

        modelInsights: analysis.explanation.modelInsights,

        temporalAnalysis: analysis.explanation.temporalAnalysis,

        metadataAnalysis: analysis.explanation.metadataAnalysis,

        generatedAt: analysis.explanation.generatedAt,

        processingVersion: analysis.explanation.processingVersion

      } : null,

      metadata: {

        fileSize: analysis.fileSize,

        fileType: analysis.fileType,

        exportedAt: new Date().toISOString(),

        includesExplanation: !!analysis.explanation

      },

    };

    const blob = new Blob([JSON.stringify(report, null, 2)], {

      type: 'application/json',

    });

    const url = URL.createObjectURL(blob);

    const a = document.createElement('a');

    a.href = url;

    const suffix = analysis.explanation ? '-with-explanation' : '';

    a.download = `deepfake-analysis-${analysis.filename}-${new Date().toISOString().split('T')[0]}${suffix}.json`;

    a.click();

    URL.revokeObjectURL(url);

    const message = analysis.explanation ?

      'Complete report with explanations downloaded!' :

      'Report downloaded successfully!';

    toast.success(message);

  };

  const getConfidenceColor = (confidence: number) => {

    if (confidence < 0.3) return 'text-green-600 dark:text-green-400';

    if (confidence < 0.7) return 'text-yellow-600 dark:text-yellow-400';

    return 'text-red-600 dark:text-red-400';

  };

  const getConfidenceIcon = (confidence: number) => {

    if (confidence < 0.3) return CheckCircle;

    if (confidence < 0.7) return AlertTriangle;

    return XCircle;

  };

*// Generate timeline data from analysis results*

  const generateTimelineData = () => {

    if (!analysis) return [];

    const data = [];

    const baseConfidence = analysis.confidence;

    const points = analysis.details.frameAnalysis?.length || 20;

    for (let i = 0; i < points; i++) {

      const variation = (Math.random() - 0.5) \* 0.3;

      const confidence = Math.max(0, Math.min(1, baseConfidence + variation));

      data.push({

        frame: i + 1,

        timestamp: (i / points) \* (analysis.details.metadata.duration || 60),

        confidence: Math.round(confidence \* 100),

        anomalies: Math.random() > 0.8 ? 1 : 0,

      });

    }

    return data;

  };

  const timelineData = generateTimelineData();

  if (loading) {

    return (

      <div className="min-h-screen flex items-center justify-center">

        <div className="text-center">

          <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-primary mx-auto mb-4"></div>

          <p className="text-muted-foreground">Loading analysis...</p>

        </div>

      </div>

    );

  }

  if (!analysis) {

    return (

      <div className="min-h-screen flex items-center justify-center">

        <div className="text-center">

          <AlertTriangle className="h-12 w-12 text-muted-foreground mx-auto mb-4" />

          <h2 className="text-xl font-semibold mb-2">Analysis Not Found</h2>

          <p className="text-muted-foreground mb-4">The requested analysis could not be found.</p>

          <Button onClick={() => router.push('/upload')}>

            <ArrowLeft className="w-4 h-4 mr-2" />

            Back to Upload

          </Button>

        </div>

      </div>

    );

  }

  const ConfidenceIcon = getConfidenceIcon(analysis.confidence);

  return (

    <div className="min-h-screen bg-gradient-to-br from-background via-background to-muted/20">

      <div className="flex">

        {*/\* Sidebar \*/*}

        <Sidebar onAnalysisSelect={(id) => router.push(`/results?id=${id}`)} />

        {*/\* Main Content \*/*}

        <div className="flex-1 p-6">

          <div className="max-w-6xl mx-auto space-y-6">

            {*/\* Header \*/*}

            <motion.div

              initial={{ opacity: 0, y: -20 }}

              animate={{ opacity: 1, y: 0 }}

              transition={{ duration: 0.5 }}

              className="flex items-center justify-between"

            >

              <div className="flex items-center gap-4">

                <Button

                  variant="outline"

                  size="sm"

                  onClick={() => router.back()}

                >

                  <ArrowLeft className="w-4 h-4 mr-2" />

                  Back

                </Button>

                <div>

                  <h1 className="text-2xl font-bold flex items-center gap-2">

                    <span className="text-2xl">{getFileIcon(analysis.fileType)}</span>

                    {analysis.filename}

                  </h1>

                  <p className="text-muted-foreground">

                    Analyzed on {new Date(analysis.timestamp).toLocaleDateString()}

                  </p>

                </div>

              </div>

              <div className="flex items-center gap-2">

                <Badge variant="outline" className={getConfidenceColor(analysis.confidence)}>

                  <ConfidenceIcon className="w-3 h-3 mr-1" />

                  {analysis.confidence > 1 ? Math.round(analysis.confidence) : Math.round(analysis.confidence \* 100)}% Risk

                </Badge>

                <Button variant="outline" size="sm" onClick={downloadReport}>

                  <Download className="w-4 h-4 mr-2" />

                  {analysis.explanation ? 'Download Full Report' : 'Download JSON'}

                </Button>

                {*/\* PDF Export Dialog Trigger \*/*}

                <PDFExportDialog

                  analysis={analysis}

                  trigger={

                    <Button variant="outline" size="sm">

                      <FileText className="w-4 h-4 mr-2" />

                      Export PDF

                    </Button>

                  }

                />

                <Button variant="outline" size="sm" onClick={handleDelete}>

                  <Trash2 className="w-4 h-4 mr-2" />

                  Delete

                </Button>

              </div>

            </motion.div>

            {*/\* Tab Navigation \*/*}

            <motion.div

              initial={{ opacity: 0, y: 20 }}

              animate={{ opacity: 1, y: 0 }}

              transition={{ duration: 0.5, delay: 0.1 }}

            >

              <div className="border-b">

                <nav className="-mb-px flex space-x-8">

                  {[

                    { id: 'overview', label: 'Overview', icon: Activity },

                    { id: 'explanation', label: 'Why This Result?', icon: Lightbulb },

                    { id: 'timeline', label: 'Timeline Analysis', icon: Clock },

                    { id: 'advanced', label: 'Advanced Charts', icon: BarChart3 },

                    { id: 'details', label: 'Technical Details', icon: FileText },

                  ].map((tab) => (

                    <button

                      key={tab.id}

                      onClick={() => setActiveTab(tab.id as any)}

                      className={`py-2 px-1 border-b-2 font-medium text-sm flex items-center gap-2 ${

                        activeTab === tab.id

                          ? 'border-primary text-primary'

                          : 'border-transparent text-muted-foreground hover:text-foreground hover:border-border'

                      }`}

                    >

                      <tab.icon className="w-4 h-4" />

                      {tab.label}

                    </button>

                  ))}

                </nav>

              </div>

            </motion.div>

            {*/\* Tab Content \*/*}

            <AnimatePresence mode="wait">

              {activeTab === 'overview' && (

                <motion.div

                  key="overview"

                  initial={{ opacity: 0, y: 20 }}

                  animate={{ opacity: 1, y: 0 }}

                  exit={{ opacity: 0, y: -20 }}

                  transition={{ duration: 0.3 }}

                  className="space-y-6"

                >

                  {*/\* Main Charts \*/*}

                  <div className="grid lg:grid-cols-2 gap-6">

                    <ConfidenceGauge

                      confidence={analysis.confidence}

                      prediction={analysis.prediction}

                    />

                    <CategoryChart

                      data={analysis.details.categoryBreakdown}

                    />

                  </div>

                  {*/\* File Info & Processing Stats \*/*}

                  <div className="grid md:grid-cols-2 gap-6">

                    <Card>

                      <CardHeader>

                        <CardTitle className="flex items-center gap-2">

                          <FileText className="w-5 h-5" />

                          File Information

                        </CardTitle>

                      </CardHeader>

                      <CardContent className="space-y-4">

                        <div className="grid grid-cols-2 gap-4 text-sm">

                          <div>

                            <span className="text-muted-foreground">Type:</span>

                            <p className="font-medium capitalize">

                              {analysis.fileType.split('/')[0]}

                            </p>

                          </div>

                          <div>

                            <span className="text-muted-foreground">Size:</span>

                            <p className="font-medium">

                              {formatFileSize(analysis.fileSize)}

                            </p>

                          </div>

                          {analysis.details.metadata.duration && (

                            <div>

                              <span className="text-muted-foreground">Duration:</span>

                              <p className="font-medium">

                                {Math.round(analysis.details.metadata.duration)}s

                              </p>

                            </div>

                          )}

                          {analysis.details.metadata.resolution && (

                            <div>

                              <span className="text-muted-foreground">Resolution:</span>

                              <p className="font-medium">

                                {analysis.details.metadata.resolution}

                              </p>

                            </div>

                          )}

                        </div>

                      </CardContent>

                    </Card>

                    <Card>

                      <CardHeader>

                        <CardTitle className="flex items-center gap-2">

                          <Activity className="w-5 h-5" />

                          Processing Stats

                        </CardTitle>

                      </CardHeader>

                      <CardContent className="space-y-4">

                        <div className="space-y-3">

                          <div className="flex justify-between items-center text-sm">

                            <span className="text-muted-foreground">Processing Time:</span>

                            <span className="font-medium">

                              {Math.round(analysis.processingTime / 1000)}s

                            </span>

                          </div>

                          <div className="flex justify-between items-center text-sm">

                            <span className="text-muted-foreground">Analysis ID:</span>

                            <span className="font-mono text-xs bg-muted px-2 py-1 rounded">

                              {analysis.id.slice(-8)}

                            </span>

                          </div>

                          <div className="flex justify-between items-center text-sm">

                            <span className="text-muted-foreground">Status:</span>

                            <Badge variant="secondary">Complete</Badge>

                          </div>

                        </div>

                      </CardContent>

                    </Card>

                  </div>

                </motion.div>

              )}

              {activeTab === 'explanation' && (

                <motion.div

                  key="explanation"

                  initial={{ opacity: 0, y: 20 }}

                  animate={{ opacity: 1, y: 0 }}

                  exit={{ opacity: 0, y: -20 }}

                  transition={{ duration: 0.3 }}

                  className="space-y-6"

                >

                  {analysis.explanation ? (

                    <ExplanationDashboard explanation={analysis.explanation} />

                  ) : (

                    <Card>

                      <CardContent className="text-center py-8">

                        <Lightbulb className="w-12 h-12 mx-auto text-muted-foreground mb-4" />

                        <h3 className="font-medium mb-2">Explanation Not Available</h3>

                        <p className="text-sm text-muted-foreground">

                          Detailed explanations are not available for this analysis. This may be due to an older analysis or processing limitations.

                        </p>

                        <Button

                          variant="outline"

                          className="mt-4"

                          onClick={() => {

*// Re-analyze to get explanation*

                            toast.info('Re-analysis feature coming soon');

                          }}

                        >

                          <Lightbulb className="w-4 h-4 mr-2" />

                          Generate Explanation

                        </Button>

                      </CardContent>

                    </Card>

                  )}

                </motion.div>

              )}

              {activeTab === 'timeline' && (

                <motion.div

                  key="timeline"

                  initial={{ opacity: 0, y: 20 }}

                  animate={{ opacity: 1, y: 0 }}

                  exit={{ opacity: 0, y: -20 }}

                  transition={{ duration: 0.3 }}

                  className="space-y-6"

                >

                  <Card>

                    <CardHeader>

                      <CardTitle className="flex items-center gap-2">

                        <Clock className="w-5 h-5" />

                        Frame-by-Frame Analysis

                      </CardTitle>

                    </CardHeader>

                    <CardContent>

                      <div className="h-80">

                        <ResponsiveContainer width="100%" height="100%">

                          <AreaChart data={timelineData}>

                            <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

                            <XAxis

                              dataKey="frame"

                              tick={{ fontSize: 12 }}

                              axisLine={false}

                              tickLine={false}

                            />

                            <YAxis

                              domain={[0, 100]}

                              tick={{ fontSize: 12 }}

                              axisLine={false}

                              tickLine={false}

                              label={{ value: 'Confidence %', angle: -90, position: 'insideLeft' }}

                            />

                            <Tooltip

                              content={({ active, payload, label }) => {

                                if (active && payload && payload.length) {

                                  return (

                                    <div className="bg-background border border-border rounded-lg shadow-lg p-3">

                                      <p className="font-medium">Frame {label}</p>

                                      <p className="text-sm text-muted-foreground">

                                        Confidence: {payload[0].value}%

                                      </p>

                                    </div>

                                  );

                                }

                                return null;

                              }}

                            />

                            <Area

                              type="monotone"

                              dataKey="confidence"

                              stroke="#8884d8"

                              fill="#8884d8"

                              fillOpacity={0.2}

                              strokeWidth={2}

                            />

                          </AreaChart>

                        </ResponsiveContainer>

                      </div>

                    </CardContent>

                  </Card>

                  {*/\* Audio Analysis if applicable \*/*}

                  {analysis.details.audioAnalysis && (

                    <Card>

                      <CardHeader>

                        <CardTitle className="flex items-center gap-2">

                          <Volume2 className="w-5 h-5" />

                          Audio Analysis

                        </CardTitle>

                      </CardHeader>

                      <CardContent>

                        <div className="space-y-4">

                          <div className="h-32 bg-muted/30 rounded flex items-center justify-center">

                            <BarChart3 className="w-8 h-8 text-muted-foreground" />

                            <span className="ml-2 text-muted-foreground">Waveform Visualization</span>

                          </div>

                          <div className="grid grid-cols-2 md:grid-cols-3 gap-4 text-sm">

                            <div>

                              <span className="text-muted-foreground">Segments Analyzed:</span>

                              <p className="font-medium">{analysis.details.audioAnalysis.segments.length}</p>

                            </div>

                            <div>

                              <span className="text-muted-foreground">Avg Confidence:</span>

                              <p className="font-medium">

                                {Math.round(analysis.details.audioAnalysis.segments.reduce((sum, seg) => sum + seg.confidence, 0) / analysis.details.audioAnalysis.segments.length \* 100)}%

                              </p>

                            </div>

                            <div>

                              <span className="text-muted-foreground">Anomalies Found:</span>

                              <p className="font-medium">

                                {analysis.details.audioAnalysis.segments.filter(seg => seg.anomalies && seg.anomalies.length > 0).length}

                              </p>

                            </div>

                          </div>

                        </div>

                      </CardContent>

                    </Card>

                  )}

                </motion.div>

              )}

              {activeTab === 'advanced' && (

                <motion.div

                  key="advanced"

                  initial={{ opacity: 0, y: 20 }}

                  animate={{ opacity: 1, y: 0 }}

                  exit={{ opacity: 0, y: -20 }}

                  transition={{ duration: 0.3 }}

                  className="space-y-6"

                >

                  <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">

                    <RiskHeatmap data={analysis} />

                    <AnomalyScatterPlot data={analysis} />

                    <MultidimensionalRadar data={analysis} />

                    <ConfidenceTimelineAdvanced data={timelineData} />

                  </div>

                  {*/\* Comparative Analysis if there are multiple analyses \*/*}

                  {storage.getAnalysisHistory().length > 1 && (

                    <ComparativeAnalysis analyses={storage.getAnalysisHistory()} />

                  )}

                </motion.div>

              )}

              {activeTab === 'details' && (

                <motion.div

                  key="details"

                  initial={{ opacity: 0, y: 20 }}

                  animate={{ opacity: 1, y: 0 }}

                  exit={{ opacity: 0, y: -20 }}

                  transition={{ duration: 0.3 }}

                  className="space-y-6"

                >

                  {*/\* Usage Statistics Card \*/*}

                  <Card>

                    <CardHeader>

                      <CardTitle className="flex items-center gap-2">

                        <BarChart3 className="w-5 h-5" />

                        Free Tier Usage Statistics

                      </CardTitle>

                    </CardHeader>

                    <CardContent>

                      <div className="space-y-4">

                        <div className="grid grid-cols-2 md:grid-cols-4 gap-4">

                          <div className="text-center p-3 bg-muted/30 rounded">

                            <div className="text-lg font-bold text-primary">{usageTracker.getUsageStats().monthlyScans}</div>

                            <div className="text-xs text-muted-foreground">This Month</div>

                          </div>

                          <div className="text-center p-3 bg-muted/30 rounded">

                            <div className="text-lg font-bold text-green-600">{usageTracker.getRemainingScans()}</div>

                            <div className="text-xs text-muted-foreground">Remaining</div>

                          </div>

                          <div className="text-center p-3 bg-muted/30 rounded">

                            <div className="text-lg font-bold">{usageTracker.getUsageStats().totalScans}</div>

                            <div className="text-xs text-muted-foreground">Total</div>

                          </div>

                          <div className="text-center p-3 bg-muted/30 rounded">

                            <div className={`text-lg font-bold ${usageTracker.getUsagePercentage() > 80 ? 'text-red-600' : 'text-primary'}`}>

                              {usageTracker.getUsagePercentage().toFixed(1)}%

                            </div>

                            <div className="text-xs text-muted-foreground">Used</div>

                          </div>

                        </div>

                        {*/\* File Type Usage Distribution \*/*}

                        <FileTypeTreemap usageData={usageTracker.getMonthlyUsageByFileType()} />

                      </div>

                    </CardContent>

                  </Card>

                  <Card>

                    <CardHeader>

                      <CardTitle>Raw Analysis Data</CardTitle>

                    </CardHeader>

                    <CardContent>

                      <pre className="bg-muted/30 p-4 rounded-lg overflow-auto text-xs">

                        {JSON.stringify(analysis, null, 2)}

                      </pre>

                    </CardContent>

                  </Card>

                </motion.div>

              )}

            </AnimatePresence>

            {*/\* Hidden PDF Components for Export \*/*}

            <PDFChartComponents analysis={analysis} />

          </div>

        </div>

      </div>

    </div>

  );

}

export default function ResultsPage() {

  return (

    <Suspense fallback={

      <div className="min-h-screen flex items-center justify-center">

        <div className="text-center">

          <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-primary mx-auto mb-4"></div>

          <p className="text-muted-foreground">Loading analysis...</p>

        </div>

      </div>

    }>

      <ResultsPageContent />

    </Suspense>

  );

}

## **src\components\charts\advanced-charts.tsx**

'use client';

import React from 'react';

import {

  RadarChart,

  Radar,

  PolarGrid,

  PolarAngleAxis,

  PolarRadiusAxis,

  ScatterChart,

  Scatter,

  XAxis,

  YAxis,

  CartesianGrid,

  Tooltip,

  ResponsiveContainer,

  ComposedChart,

  Line,

  Area,

  Bar,

  Legend,

  LineChart,

  Treemap,

  Cell

} from 'recharts';

import { motion } from 'framer-motion';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

*// Risk Heatmap Component*

export function RiskHeatmap({ data }: { data: any }) {

  const heatmapData = [

    { name: 'Facial Analysis', authentic: 85, manipulated: 15, risk: 'low' },

    { name: 'Voice Patterns', authentic: 70, manipulated: 30, risk: 'medium' },

    { name: 'Temporal Consistency', authentic: 90, manipulated: 10, risk: 'low' },

    { name: 'Metadata Analysis', authentic: 95, manipulated: 5, risk: 'low' },

    { name: 'Compression Artifacts', authentic: 80, manipulated: 20, risk: 'low' },

  ];

  const getRiskColor = (risk: string) => {

    switch (risk) {

      case 'low': return '#10b981';

      case 'medium': return '#f59e0b';

      case 'high': return '#ef4444';

      default: return '#6b7280';

    }

  };

  return (

    <Card>

      <CardHeader>

        <CardTitle className="flex items-center gap-2">

          Risk Analysis Heatmap

          <Badge variant="secondary">Multi-Layer Detection</Badge>

        </CardTitle>

      </CardHeader>

      <CardContent>

        <div className="space-y-4">

          {heatmapData.map((item, index) => (

            <motion.div

              key={item.name}

              initial={{ opacity: 0, x: -20 }}

              animate={{ opacity: 1, x: 0 }}

              transition={{ delay: index \* 0.1 }}

              className="flex items-center justify-between p-3 rounded-lg border"

            >

              <div className="flex-1">

                <div className="font-medium text-sm">{item.name}</div>

                <div className="flex items-center gap-2 mt-1">

                  <div className="flex-1 bg-muted rounded-full h-2 overflow-hidden">

                    <div

                      className="h-full transition-all duration-1000"

                      style={{

                        width: `${item.authentic}%`,

                        backgroundColor: getRiskColor(item.risk)

                      }}

                    />

                  </div>

                  <div className="text-xs text-muted-foreground min-w-[3rem]">

                    {item.authentic}%

                  </div>

                </div>

              </div>

              <Badge

                variant={item.risk === 'low' ? 'default' : item.risk === 'medium' ? 'secondary' : 'destructive'}

                className="ml-2"

              >

                {item.risk.toUpperCase()}

              </Badge>

            </motion.div>

          ))}

        </div>

      </CardContent>

    </Card>

  );

}

*// Anomaly Detection Scatter Plot*

export function AnomalyScatterPlot({ data }: { data: any }) {

  const scatterData = Array.from({ length: 50 }, (\_, i) => ({

    x: Math.random() \* 100,

    y: Math.random() \* 100,

    confidence: Math.random(),

    anomaly: Math.random() > 0.7,

  }));

  const CustomTooltip = ({ active, payload }: any) => {

    if (active && payload && payload.length) {

      const data = payload[0].payload;

      return (

        <div className="bg-background border rounded-lg shadow-lg p-3">

          <p className="font-medium">Frame Analysis</p>

          <p className="text-sm">Position: ({data.x.toFixed(1)}, {data.y.toFixed(1)})</p>

          <p className="text-sm">Confidence: {(data.confidence \* 100).toFixed(1)}%</p>

          <p className="text-sm">

            Status: <span className={data.anomaly ? 'text-red-500' : 'text-green-500'}>

              {data.anomaly ? 'Anomaly Detected' : 'Normal'}

            </span>

          </p>

        </div>

      );

    }

    return null;

  };

  return (

    <Card>

      <CardHeader>

        <CardTitle>Anomaly Detection Analysis</CardTitle>

      </CardHeader>

      <CardContent>

        <div className="h-64">

          <ResponsiveContainer width="100%" height="100%">

            <ScatterChart data={scatterData}>

              <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

              <XAxis

                type="number"

                dataKey="x"

                domain={[0, 100]}

                tick={{ fontSize: 12 }}

              />

              <YAxis

                type="number"

                dataKey="y"

                domain={[0, 100]}

                tick={{ fontSize: 12 }}

              />

              <Tooltip content={<CustomTooltip />} />

              <Scatter

                dataKey="confidence"

                fill="#8884d8"

                shape={(props: any) => {

                  const { cx, cy, payload } = props;

                  return (

                    <circle

                      cx={cx}

                      cy={cy}

                      r={payload.anomaly ? 6 : 3}

                      fill={payload.anomaly ? '#ef4444' : '#10b981'}

                      opacity={0.7}

                    />

                  );

                }}

              />

            </ScatterChart>

          </ResponsiveContainer>

        </div>

        <div className="flex items-center gap-4 mt-2 text-xs text-muted-foreground">

          <div className="flex items-center gap-1">

            <div className="w-3 h-3 rounded-full bg-green-500" />

            Normal Frames

          </div>

          <div className="flex items-center gap-1">

            <div className="w-3 h-3 rounded-full bg-red-500" />

            Anomalies Detected

          </div>

        </div>

      </CardContent>

    </Card>

  );

}

*// Radar Chart for Multi-Dimensional Analysis*

export function MultidimensionalRadar({ data }: { data: any }) {

  const radarData = [

    {

      metric: 'Facial Consistency',

      score: 85,

      fullMark: 100,

    },

    {

      metric: 'Temporal Flow',

      score: 78,

      fullMark: 100,

    },

    {

      metric: 'Lighting Analysis',

      score: 92,

      fullMark: 100,

    },

    {

      metric: 'Compression Patterns',

      score: 88,

      fullMark: 100,

    },

    {

      metric: 'Metadata Integrity',

      score: 95,

      fullMark: 100,

    },

    {

      metric: 'Pixel Artifacts',

      score: 76,

      fullMark: 100,

    },

  ];

  return (

    <Card>

      <CardHeader>

        <CardTitle>Multi-Dimensional Analysis</CardTitle>

      </CardHeader>

      <CardContent>

        <div className="h-64">

          <ResponsiveContainer width="100%" height="100%">

            <RadarChart data={radarData}>

              <PolarGrid />

              <PolarAngleAxis

                dataKey="metric"

                tick={{ fontSize: 10 }}

              />

              <PolarRadiusAxis

                angle={90}

                domain={[0, 100]}

                tick={{ fontSize: 10 }}

              />

              <Radar

                name="Analysis Score"

                dataKey="score"

                stroke="#8884d8"

                fill="#8884d8"

                fillOpacity={0.3}

                strokeWidth={2}

              />

            </RadarChart>

          </ResponsiveContainer>

        </div>

      </CardContent>

    </Card>

  );

}

*// Confidence Timeline with Anomaly Markers*

export function ConfidenceTimelineAdvanced({ data }: { data: any }) {

  const timelineData = Array.from({ length: 30 }, (\_, i) => ({

    frame: i + 1,

    confidence: 65 + Math.sin(i / 5) \* 15 + (Math.random() - 0.5) \* 10,

    anomaly: Math.random() > 0.8 ? 100 : null,

    threshold: 70,

  }));

  return (

    <Card>

      <CardHeader>

        <CardTitle>Confidence Timeline Analysis</CardTitle>

      </CardHeader>

      <CardContent>

        <div className="h-64">

          <ResponsiveContainer width="100%" height="100%">

            <ComposedChart data={timelineData}>

              <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

              <XAxis

                dataKey="frame"

                tick={{ fontSize: 12 }}

              />

              <YAxis

                domain={[0, 100]}

                tick={{ fontSize: 12 }}

              />

              <Tooltip

                content={({ active, payload, label }) => {

                  if (active && payload && payload.length) {

                    return (

                      <div className="bg-background border rounded-lg shadow-lg p-3">

                        <p className="font-medium">Frame {label}</p>

                        <p className="text-sm">

                          Confidence: {payload[0]?.value?.toFixed(1)}%

                        </p>

                        {payload[1]?.value && (

                          <p className="text-sm text-red-500">⚠️ Anomaly Detected</p>

                        )}

                      </div>

                    );

                  }

                  return null;

                }}

              />

              <Area

                type="monotone"

                dataKey="confidence"

                stroke="#8884d8"

                fill="#8884d8"

                fillOpacity={0.3}

                strokeWidth={2}

              />

              <Line

                type="monotone"

                dataKey="threshold"

                stroke="#f59e0b"

                strokeDasharray="5 5"

                dot={false}

              />

              <Scatter

                dataKey="anomaly"

                fill="#ef4444"

                shape="diamond"

              />

              <Legend />

            </ComposedChart>

          </ResponsiveContainer>

        </div>

      </CardContent>

    </Card>

  );

}

*// Comparative Analysis Chart*

export function ComparativeAnalysis({ analyses }: { analyses: any[] }) {

  const comparisonData = analyses.slice(0, 5).map((analysis, index) => ({

    name: `Analysis ${index + 1}`,

    confidence: analysis.confidence \* 100,

    authentic: (1 - analysis.confidence) \* 100,

    processing\_time: analysis.processingTime / 1000,

  }));

  return (

    <Card>

      <CardHeader>

        <CardTitle>Comparative Analysis</CardTitle>

      </CardHeader>

      <CardContent>

        <div className="h-64">

          <ResponsiveContainer width="100%" height="100%">

            <ComposedChart data={comparisonData}>

              <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

              <XAxis dataKey="name" />

              <YAxis />

              <Tooltip />

              <Legend />

              <Bar dataKey="confidence" fill="#ef4444" name="Manipulation Risk %" />

              <Bar dataKey="authentic" fill="#10b981" name="Authenticity %" />

              <Line

                type="monotone"

                dataKey="processing\_time"

                stroke="#f59e0b"

                name="Processing Time (s)"

                yAxisId="right"

              />

            </ComposedChart>

          </ResponsiveContainer>

        </div>

      </CardContent>

    </Card>

  );

}

*// File Type Usage Treemap*

export function FileTypeTreemap({ usageData }: { usageData: Record<string, number> }) {

  const treemapData = Object.entries(usageData).map(([type, count]) => ({

    name: type.charAt(0).toUpperCase() + type.slice(1),

    value: count,

    percentage: (count / Object.values(usageData).reduce((a, b) => a + b, 0) \* 100).toFixed(1),

  }));

  const COLORS = ['#8884d8', '#82ca9d', '#ffc658', '#ff7300', '#8dd1e1'];

  return (

    <Card>

      <CardHeader>

        <CardTitle>File Type Analysis Distribution</CardTitle>

      </CardHeader>

      <CardContent>

        <div className="h-64">

          <ResponsiveContainer width="100%" height="100%">

            <Treemap

              data={treemapData}

              dataKey="value"

              stroke="#fff"

              content={(props: any) => {

                const { x, y, width, height, index, name, value, percentage } = props;

                return (

                  <g>

                    <rect

                      x={x}

                      y={y}

                      width={width}

                      height={height}

                      fill={COLORS[index % COLORS.length]}

                      opacity={0.8}

                    />

                    {width > 60 && height > 30 && (

                      <>

                        <text

                          x={x + width / 2}

                          y={y + height / 2 - 6}

                          textAnchor="middle"

                          fill="#fff"

                          fontSize="12"

                          fontWeight="bold"

                        >

                          {name}

                        </text>

                        <text

                          x={x + width / 2}

                          y={y + height / 2 + 6}

                          textAnchor="middle"

                          fill="#fff"

                          fontSize="10"

                        >

                          {value} ({percentage}%)

                        </text>

                      </>

                    )}

                  </g>

                );

              }}

            />

          </ResponsiveContainer>

        </div>

      </CardContent>

    </Card>

  );

}

**src\components\charts\category-chart.tsx**

'use client';

import { BarChart, Bar, XAxis, YAxis, CartesianGrid, ResponsiveContainer, PieChart, Pie, Cell, Tooltip } from 'recharts';

import { motion } from 'framer-motion';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

import { Button } from '@/components/ui/button';

*// Toggle group removed - using buttons instead*

import { BarChart3, PieChart as PieChartIcon } from 'lucide-react';

import { useState } from 'react';

interface CategoryData {

  authentic: number;

  manipulated: number;

  inconclusive: number;

}

interface CategoryChartProps {

  data: CategoryData;

  className?: string;

}

const COLORS = {

  authentic: '#10b981',

  manipulated: '#ef4444',

  inconclusive: '#f59e0b',

};

export function CategoryChart({ data, className }: CategoryChartProps) {

  const [viewType, setViewType] = useState<'pie' | 'bar'>('pie');

*// Transform data for charts - handle both 0-1 and 0-100 ranges safely*

  const normalizeValue = (value: number) => value > 1 ? Math.round(value) : Math.round(value \* 100);

  const chartData = [

    {

      name: 'Authentic',

      value: normalizeValue(data.authentic),

      percentage: `${normalizeValue(data.authentic)}%`,

      color: COLORS.authentic,

    },

    {

      name: 'Manipulated',

      value: normalizeValue(data.manipulated),

      percentage: `${normalizeValue(data.manipulated)}%`,

      color: COLORS.manipulated,

    },

    {

      name: 'Inconclusive',

      value: normalizeValue(data.inconclusive),

      percentage: `${normalizeValue(data.inconclusive)}%`,

      color: COLORS.inconclusive,

    },

  ];

  const CustomTooltip = ({ active, payload }: { active?: boolean; payload?: Array<{ name: string; value: number; color: string }> }) => {

    if (active && payload && payload.length) {

      const data = payload[0];

      return (

        <div className="bg-background border border-border rounded-lg shadow-lg p-3">

          <p className="font-medium">{data.name}</p>

          <p className="text-sm text-muted-foreground">

            <span style={{ color: data.color }}>●</span> {data.value}%

          </p>

        </div>

      );

    }

    return null;

  };

  const renderCustomizedLabel = (entry: any) => {

    return `${entry.percent?.toFixed(0) || 0}%`;

  };

  return (

    <Card className={className}>

      <CardHeader className="pb-4">

        <div className="flex items-center justify-between">

          <CardTitle>Category Breakdown</CardTitle>

          <div className="flex items-center gap-2">

            <Button

              variant={viewType === 'pie' ? 'default' : 'outline'}

              size="sm"

              onClick={() => setViewType('pie')}

              className="h-8 w-8 p-0"

            >

              <PieChartIcon className="h-4 w-4" />

            </Button>

            <Button

              variant={viewType === 'bar' ? 'default' : 'outline'}

              size="sm"

              onClick={() => setViewType('bar')}

              className="h-8 w-8 p-0"

            >

              <BarChart3 className="h-4 w-4" />

            </Button>

          </div>

        </div>

      </CardHeader>

      <CardContent>

        <motion.div

          key={viewType}

          initial={{ opacity: 0, scale: 0.95 }}

          animate={{ opacity: 1, scale: 1 }}

          transition={{ duration: 0.3 }}

        >

          {viewType === 'pie' ? (

            <div className="space-y-6">

              <div className="h-64">

                <ResponsiveContainer width="100%" height="100%">

                  <PieChart>

                    <Pie

                      data={chartData}

                      cx="50%"

                      cy="50%"

                      labelLine={false}

                      label={renderCustomizedLabel}

                      outerRadius={80}

                      fill="#8884d8"

                      dataKey="value"

                      animationBegin={0}

                      animationDuration={1000}

                    >

                      {chartData.map((entry, index) => (

                        <Cell key={`cell-${index}`} fill={entry.color} />

                      ))}

                    </Pie>

                    <Tooltip content={<CustomTooltip />} />

                  </PieChart>

                </ResponsiveContainer>

              </div>

              {*/\* Legend \*/*}

              <div className="grid grid-cols-1 gap-3">

                {chartData.map((item, index) => (

                  <motion.div

                    key={item.name}

                    initial={{ opacity: 0, x: -20 }}

                    animate={{ opacity: 1, x: 0 }}

                    transition={{ delay: index \* 0.1 + 0.5 }}

                    className="flex items-center justify-between p-3 bg-muted/30 rounded-lg"

                  >

                    <div className="flex items-center gap-3">

                      <div

                        className="w-4 h-4 rounded-full"

                        style={{ backgroundColor: item.color }}

                      />

                      <span className="font-medium text-sm">{item.name}</span>

                    </div>

                    <Badge variant="secondary" className="font-semibold">

                      {item.percentage}

                    </Badge>

                  </motion.div>

                ))}

              </div>

            </div>

          ) : (

            <div className="space-y-6">

              <div className="h-64">

                <ResponsiveContainer width="100%" height="100%">

                  <BarChart data={chartData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

                    <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

                    <XAxis

                      dataKey="name"

                      tick={{ fontSize: 12 }}

                      axisLine={false}

                      tickLine={false}

                    />

                    <YAxis

                      tick={{ fontSize: 12 }}

                      axisLine={false}

                      tickLine={false}

                      domain={[0, 100]}

                    />

                    <Tooltip content={<CustomTooltip />} />

                    <Bar

                      dataKey="value"

                      fill="#8884d8"

                      radius={[4, 4, 0, 0]}

                      animationBegin={0}

                      animationDuration={1000}

                    >

                      {chartData.map((entry, index) => (

                        <Cell key={`cell-${index}`} fill={entry.color} />

                      ))}

                    </Bar>

                  </BarChart>

                </ResponsiveContainer>

              </div>

              {*/\* Statistics \*/*}

              <div className="grid grid-cols-3 gap-4">

                {chartData.map((item, index) => (

                  <motion.div

                    key={item.name}

                    initial={{ opacity: 0, y: 20 }}

                    animate={{ opacity: 1, y: 0 }}

                    transition={{ delay: index \* 0.1 + 0.5 }}

                    className="text-center p-4 border rounded-lg"

                  >

                    <div

                      className="text-2xl font-bold mb-1"

                      style={{ color: item.color }}

                    >

                      {item.percentage}

                    </div>

                    <div className="text-sm text-muted-foreground font-medium">

                      {item.name}

                    </div>

                  </motion.div>

                ))}

              </div>

            </div>

          )}

        </motion.div>

        {*/\* Summary \*/*}

        <div className="mt-6 p-4 bg-muted/50 rounded-lg">

          <h4 className="font-medium mb-2">Analysis Summary</h4>

          <p className="text-sm text-muted-foreground">

            {data.authentic > data.manipulated && data.authentic > data.inconclusive &&

              'The analysis indicates this file is likely authentic with low signs of manipulation.'}

            {data.manipulated > data.authentic && data.manipulated > data.inconclusive &&

              'The analysis suggests this file may be manipulated or artificially generated.'}

            {data.inconclusive >= data.authentic && data.inconclusive >= data.manipulated &&

              'The analysis results are inconclusive, requiring further investigation.'}

          </p>

        </div>

      </CardContent>

    </Card>

  );

}

**src\components\charts\confidence-gauge.tsx**

'use client';

import { PieChart, Pie, Cell, ResponsiveContainer } from 'recharts';

import { motion } from 'framer-motion';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

import { AlertTriangle, CheckCircle, XCircle } from 'lucide-react';

interface ConfidenceGaugeProps {

  confidence: number;

  prediction: 'authentic' | 'manipulated' | 'inconclusive';

  className?: string;

}

export function ConfidenceGauge({

  confidence,

  prediction,

  className,

}: ConfidenceGaugeProps) {

*// Handle both 0-1 range (0.99) and 0-100 range (99) safely*

  const percentage = confidence > 1 ? Math.round(confidence) : Math.round(confidence \* 100);

*// Calculate gauge data*

  const gaugeData = [

    { name: 'confidence', value: percentage },

    { name: 'remaining', value: 100 - percentage },

  ];

*// Get color based on confidence level*

  const getConfidenceColor = () => {

    if (confidence < 0.3) return '#10b981'; *// green*

    if (confidence < 0.7) return '#f59e0b'; *// yellow*

    return '#ef4444'; *// red*

  };

*// Get icon based on prediction*

  const getPredictionIcon = () => {

    switch (prediction) {

      case 'authentic':

        return <CheckCircle className="h-5 w-5 text-green-600 dark:text-green-400" />;

      case 'manipulated':

        return <XCircle className="h-5 w-5 text-red-600 dark:text-red-400" />;

      case 'inconclusive':

        return <AlertTriangle className="h-5 w-5 text-yellow-600 dark:text-yellow-400" />;

    }

  };

  const getConfidenceLabel = () => {

    if (confidence < 0.3) return 'Low Risk';

    if (confidence < 0.7) return 'Medium Risk';

    return 'High Risk';

  };

  const getBadgeVariant = () => {

    switch (prediction) {

      case 'authentic':

        return 'secondary';

      case 'manipulated':

        return 'destructive';

      case 'inconclusive':

        return 'outline';

    }

  };

  return (

    <Card className={className}>

      <CardHeader className="pb-4">

        <CardTitle className="flex items-center justify-between">

          <span>Confidence Score</span>

          <Badge variant={getBadgeVariant()} className="capitalize">

            {getPredictionIcon()}

            <span className="ml-1">{prediction}</span>

          </Badge>

        </CardTitle>

      </CardHeader>

      <CardContent>

        <div className="relative">

          {*/\* Gauge Chart \*/*}

          <div className="relative mx-auto w-48 h-48">

            <ResponsiveContainer width="100%" height="100%">

              <PieChart>

                <Pie

                  data={gaugeData}

                  cx="50%"

                  cy="50%"

                  startAngle={180}

                  endAngle={0}

                  innerRadius={60}

                  outerRadius={80}

                  paddingAngle={0}

                  dataKey="value"

                >

                  <Cell fill={getConfidenceColor()} />

                  <Cell fill="transparent" />

                </Pie>

              </PieChart>

            </ResponsiveContainer>

            {*/\* Center Content \*/*}

            <div className="absolute inset-0 flex flex-col items-center justify-center">

              <motion.div

                initial={{ scale: 0 }}

                animate={{ scale: 1 }}

                transition={{ delay: 0.5, type: "spring", stiffness: 200 }}

                className="text-center"

              >

                <div

                  className="text-3xl font-bold mb-1"

                  style={{ color: getConfidenceColor() }}

                >

                  {percentage}%

                </div>

                <div className="text-sm text-muted-foreground font-medium">

                  {getConfidenceLabel()}

                </div>

              </motion.div>

            </div>

            {*/\* Animated Background Ring \*/*}

            <div className="absolute inset-0">

              <svg className="w-full h-full -rotate-90" viewBox="0 0 192 192">

                <circle

                  cx="96"

                  cy="96"

                  r="70"

                  fill="none"

                  stroke="currentColor"

                  strokeWidth="4"

                  className="text-muted/20"

                />

                <motion.circle

                  cx="96"

                  cy="96"

                  r="70"

                  fill="none"

                  stroke={getConfidenceColor()}

                  strokeWidth="4"

                  strokeLinecap="round"

                  strokeDasharray={`${2 \* Math.PI \* 70}`}

                  initial={{ strokeDashoffset: 2 \* Math.PI \* 70 }}

                  animate={{ strokeDashoffset: 2 \* Math.PI \* 70 \* (1 - percentage / 100) }}

                  transition={{ duration: 1.5, ease: "easeInOut" }}

                  style={{

                    filter: 'drop-shadow(0 0 6px currentColor)',

                    opacity: 0.8,

                  }}

                />

              </svg>

            </div>

          </div>

          {*/\* Risk Level Indicators \*/*}

          <div className="mt-6 space-y-3">

            <div className="flex items-center justify-between text-sm">

              <span className="flex items-center gap-2">

                <div className="w-3 h-3 rounded-full bg-green-500"></div>

                Low Risk

              </span>

              <span className="text-muted-foreground">0-30%</span>

            </div>

            <div className="flex items-center justify-between text-sm">

              <span className="flex items-center gap-2">

                <div className="w-3 h-3 rounded-full bg-yellow-500"></div>

                Medium Risk

              </span>

              <span className="text-muted-foreground">30-70%</span>

            </div>

            <div className="flex items-center justify-between text-sm">

              <span className="flex items-center gap-2">

                <div className="w-3 h-3 rounded-full bg-red-500"></div>

                High Risk

              </span>

              <span className="text-muted-foreground">70-100%</span>

            </div>

          </div>

          {*/\* Additional Info \*/*}

          <div className="mt-6 p-4 bg-muted/50 rounded-lg">

            <div className="flex items-center justify-between mb-2">

              <span className="text-sm font-medium">Detection Result</span>

              {getPredictionIcon()}

            </div>

            <p className="text-xs text-muted-foreground leading-relaxed">

              {prediction === 'authentic' &&

                'This file appears to be authentic with low likelihood of manipulation. AI analysis found consistent patterns typical of genuine content.'}

              {prediction === 'manipulated' &&

                'This file shows signs of potential manipulation or deepfake generation. Multiple AI models detected anomalous patterns consistent with synthetic content.'}

              {prediction === 'inconclusive' &&

                'The analysis could not determine with high confidence whether this file is authentic or manipulated. Mixed signals from different detection methods require further investigation.'}

            </p>

            <div className="mt-3 pt-2 border-t border-muted">

              <div className="flex items-center gap-2 text-xs text-muted-foreground">

                <span className="font-medium">💡 Tip:</span>

                <span>Check the "Why This Result?" tab for detailed explanations and AI reasoning.</span>

              </div>

            </div>

          </div>

        </div>

      </CardContent>

    </Card>

  );

}

**src\components\explanation\explanation-dashboard.tsx**

'use client';

import { useState } from 'react';

import { motion, AnimatePresence } from 'framer-motion';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

import { Button } from '@/components/ui/button';

import { Separator } from '@/components/ui/separator';

import { Progress } from '@/components/ui/progress';

import {

  Brain,

  Eye,

  Volume2,

  FileText,

  Clock,

  AlertTriangle,

  CheckCircle,

  XCircle,

  Info,

  ChevronRight,

  ChevronDown,

  Lightbulb,

  Shield,

  TrendingUp,

  TrendingDown,

  Minus,

  ZoomIn

} from 'lucide-react';

import type { DetailedExplanation, ExplanationReason } from '@/lib/types';

interface ExplanationDashboardProps {

  explanation: DetailedExplanation;

  className?: string;

}

export function ExplanationDashboard({ explanation, className }: ExplanationDashboardProps) {

  const [activeTab, setActiveTab] = useState<'summary' | 'reasons' | 'insights' | 'evidence'>('summary');

  const [expandedReasons, setExpandedReasons] = useState<Set<string>>(new Set());

  const toggleReasonExpansion = (reasonId: string) => {

    const newExpanded = new Set(expandedReasons);

    if (newExpanded.has(reasonId)) {

      newExpanded.delete(reasonId);

    } else {

      newExpanded.add(reasonId);

    }

    setExpandedReasons(newExpanded);

  };

  const getSeverityColor = (severity: 'high' | 'medium' | 'low') => {

    switch (severity) {

      case 'high': return 'text-red-600 dark:text-red-400 bg-red-100 dark:bg-red-900/20';

      case 'medium': return 'text-yellow-600 dark:text-yellow-400 bg-yellow-100 dark:bg-yellow-900/20';

      case 'low': return 'text-green-600 dark:text-green-400 bg-green-100 dark:bg-green-900/20';

    }

  };

  const getCategoryIcon = (category: ExplanationReason['category']) => {

    switch (category) {

      case 'visual': return Eye;

      case 'audio': return Volume2;

      case 'model': return Brain;

      case 'temporal': return Clock;

      case 'metadata': return FileText;

      case 'technical': return Info;

      default: return Info;

    }

  };

  const getContributionIcon = (contribution: 'positive' | 'negative' | 'neutral') => {

    switch (contribution) {

      case 'positive': return TrendingUp;

      case 'negative': return TrendingDown;

      case 'neutral': return Minus;

    }

  };

  return (

    <div className={className}>

      {*/\* Tab Navigation \*/*}

      <div className="mb-6">

        <div className="border-b">

          <nav className="-mb-px flex space-x-8">

            {[

              { id: 'summary', label: 'Summary', icon: Lightbulb },

              { id: 'reasons', label: 'Detailed Reasons', icon: Brain },

              { id: 'insights', label: 'AI Insights', icon: Eye },

              { id: 'evidence', label: 'Evidence', icon: Shield },

            ].map((tab) => (

              <button

                key={tab.id}

                onClick={() => setActiveTab(tab.id as any)}

                className={`py-2 px-1 border-b-2 font-medium text-sm flex items-center gap-2 transition-colors ${

                  activeTab === tab.id

                    ? 'border-primary text-primary'

                    : 'border-transparent text-muted-foreground hover:text-foreground hover:border-border'

                }`}

              >

                <tab.icon className="w-4 h-4" />

                {tab.label}

              </button>

            ))}

          </nav>

        </div>

      </div>

      {*/\* Tab Content \*/*}

      <AnimatePresence mode="wait">

        {activeTab === 'summary' && (

          <motion.div

            key="summary"

            initial={{ opacity: 0, y: 20 }}

            animate={{ opacity: 1, y: 0 }}

            exit={{ opacity: 0, y: -20 }}

            transition={{ duration: 0.3 }}

            className="space-y-6"

          >

            {*/\* Primary Reason \*/*}

            <Card>

              <CardHeader>

                <CardTitle className="flex items-center gap-2">

                  <Lightbulb className="w-5 h-5" />

                  Why This Classification?

                </CardTitle>

              </CardHeader>

              <CardContent>

                <p className="text-lg leading-relaxed mb-4">

                  {explanation.summary.primaryReason}

                </p>

                {explanation.summary.secondaryReasons.length > 0 && (

                  <div>

                    <h4 className="font-medium mb-2 text-muted-foreground">Additional factors:</h4>

                    <ul className="space-y-1">

                      {explanation.summary.secondaryReasons.map((reason, index) => (

                        <li key={index} className="flex items-center gap-2 text-sm">

                          <ChevronRight className="w-3 h-3 text-muted-foreground" />

                          {reason}

                        </li>

                      ))}

                    </ul>

                  </div>

                )}

              </CardContent>

            </Card>

            {*/\* Authenticity Indicators \*/*}

            <Card>

              <CardHeader>

                <CardTitle className="flex items-center gap-2">

                  <Shield className="w-5 h-5" />

                  Authenticity Indicators

                </CardTitle>

              </CardHeader>

              <CardContent>

                <div className="space-y-4">

                  {explanation.summary.authenticityIndicators.map((indicator, index) => {

                    const ContributionIcon = getContributionIcon(indicator.contribution);

                    return (

                      <div key={index} className="flex items-center justify-between p-3 bg-muted/30 rounded-lg">

                        <div className="flex items-center gap-3">

                          <ContributionIcon className={`w-4 h-4 ${

                            indicator.contribution === 'positive' ? 'text-green-600' :

                            indicator.contribution === 'negative' ? 'text-red-600' :

                            'text-muted-foreground'

                          }`} />

                          <div>

                            <div className="font-medium">{indicator.factor}</div>

                            <div className="text-sm text-muted-foreground">{indicator.explanation}</div>

                          </div>

                        </div>

                        <div className="text-right">

                          <div className="text-sm font-medium">

                            Weight: {Math.round(indicator.weight \* 100)}%

                          </div>

                        </div>

                      </div>

                    );

                  })}

                </div>

              </CardContent>

            </Card>

            {*/\* Risk Factors \*/*}

            {explanation.summary.riskFactors.length > 0 && (

              <Card>

                <CardHeader>

                  <CardTitle className="flex items-center gap-2">

                    <AlertTriangle className="w-5 h-5" />

                    Risk Assessment

                  </CardTitle>

                </CardHeader>

                <CardContent>

                  <div className="space-y-4">

                    {explanation.summary.riskFactors.map((risk, index) => (

                      <div key={index} className="border-l-4 border-yellow-500 pl-4 py-2">

                        <div className="flex items-center gap-2 mb-1">

                          <Badge variant="outline" className={

                            risk.severity === 'critical' ? 'border-red-500 text-red-700' :

                            risk.severity === 'high' ? 'border-orange-500 text-orange-700' :

                            risk.severity === 'medium' ? 'border-yellow-500 text-yellow-700' :

                            'border-blue-500 text-blue-700'

                          }>

                            {risk.severity.toUpperCase()}

                          </Badge>

                          <span className="font-medium">{risk.factor}</span>

                          <span className="text-sm text-muted-foreground">

                            ({Math.round(risk.likelihood \* 100)}% likelihood)

                          </span>

                        </div>

                        <p className="text-sm text-muted-foreground">{risk.impact}</p>

                      </div>

                    ))}

                  </div>

                </CardContent>

              </Card>

            )}

            {*/\* Recommendations \*/*}

            {explanation.summary.recommendedActions && explanation.summary.recommendedActions.length > 0 && (

              <Card>

                <CardHeader>

                  <CardTitle className="flex items-center gap-2">

                    <Info className="w-5 h-5" />

                    Recommendations

                  </CardTitle>

                </CardHeader>

                <CardContent>

                  <ul className="space-y-2">

                    {explanation.summary.recommendedActions.map((action, index) => (

                      <li key={index} className="flex items-start gap-2">

                        <CheckCircle className="w-4 h-4 mt-0.5 text-green-600 flex-shrink-0" />

                        <span className="text-sm">{action}</span>

                      </li>

                    ))}

                  </ul>

                </CardContent>

              </Card>

            )}

          </motion.div>

        )}

        {activeTab === 'reasons' && (

          <motion.div

            key="reasons"

            initial={{ opacity: 0, y: 20 }}

            animate={{ opacity: 1, y: 0 }}

            exit={{ opacity: 0, y: -20 }}

            transition={{ duration: 0.3 }}

            className="space-y-4"

          >

            {explanation.reasons.map((reason, index) => {

              const CategoryIcon = getCategoryIcon(reason.category);

              const isExpanded = expandedReasons.has(reason.id);

              return (

                <Card key={reason.id}>

                  <CardContent className="p-0">

                    <button

                      onClick={() => toggleReasonExpansion(reason.id)}

                      className="w-full p-4 text-left hover:bg-muted/50 transition-colors"

                    >

                      <div className="flex items-center justify-between">

                        <div className="flex items-center gap-3">

                          <CategoryIcon className="w-5 h-5 text-muted-foreground" />

                          <div>

                            <h3 className="font-medium">{reason.title}</h3>

                            <div className="flex items-center gap-2 mt-1">

                              <Badge className={getSeverityColor(reason.severity)}>

                                {reason.severity}

                              </Badge>

                              <span className="text-sm text-muted-foreground capitalize">

                                {reason.category} • {reason.type}

                              </span>

                            </div>

                          </div>

                        </div>

                        <div className="flex items-center gap-2">

                          <div className="text-right">

                            <div className="text-sm font-medium">

                              {Math.round(reason.confidence \* 100)}% confidence

                            </div>

                          </div>

                          {isExpanded ? (

                            <ChevronDown className="w-4 h-4" />

                          ) : (

                            <ChevronRight className="w-4 h-4" />

                          )}

                        </div>

                      </div>

                    </button>

                    <AnimatePresence>

                      {isExpanded && (

                        <motion.div

                          initial={{ height: 0, opacity: 0 }}

                          animate={{ height: 'auto', opacity: 1 }}

                          exit={{ height: 0, opacity: 0 }}

                          transition={{ duration: 0.2 }}

                          className="overflow-hidden"

                        >

                          <div className="px-4 pb-4 border-t">

                            <div className="pt-4 space-y-3">

                              <div>

                                <h4 className="font-medium mb-1">Description</h4>

                                <p className="text-sm text-muted-foreground leading-relaxed">

                                  {reason.description}

                                </p>

                              </div>

                              {reason.technicalDetails && (

                                <div>

                                  <h4 className="font-medium mb-1">Technical Details</h4>

                                  <p className="text-xs font-mono bg-muted/50 p-2 rounded">

                                    {reason.technicalDetails}

                                  </p>

                                </div>

                              )}

                              {reason.modelSources && reason.modelSources.length > 0 && (

                                <div>

                                  <h4 className="font-medium mb-1">Source Models</h4>

                                  <div className="flex flex-wrap gap-1">

                                    {reason.modelSources.map((model, idx) => (

                                      <Badge key={idx} variant="secondary" className="text-xs">

                                        {model}

                                      </Badge>

                                    ))}

                                  </div>

                                </div>

                              )}

                              {reason.supportingEvidence && reason.supportingEvidence.length > 0 && (

                                <div>

                                  <h4 className="font-medium mb-1">Supporting Evidence</h4>

                                  <ul className="text-sm text-muted-foreground space-y-1">

                                    {reason.supportingEvidence.map((evidence, idx) => (

                                      <li key={idx} className="flex items-start gap-2">

                                        <ZoomIn className="w-3 h-3 mt-0.5 flex-shrink-0" />

                                        {evidence}

                                      </li>

                                    ))}

                                  </ul>

                                </div>

                              )}

                            </div>

                          </div>

                        </motion.div>

                      )}

                    </AnimatePresence>

                  </CardContent>

                </Card>

              );

            })}

          </motion.div>

        )}

        {activeTab === 'insights' && (

          <motion.div

            key="insights"

            initial={{ opacity: 0, y: 20 }}

            animate={{ opacity: 1, y: 0 }}

            exit={{ opacity: 0, y: -20 }}

            transition={{ duration: 0.3 }}

            className="space-y-6"

          >

            {explanation.modelInsights.map((insight, index) => (

              <Card key={index}>

                <CardHeader>

                  <CardTitle className="flex items-center gap-2">

                    <Brain className="w-5 h-5" />

                    {insight.modelName}

                  </CardTitle>

                </CardHeader>

                <CardContent className="space-y-4">

                  <div className="flex items-center justify-between">

                    <Badge variant="outline" className="capitalize">

                      {insight.modelType.replace('\_', ' ')}

                    </Badge>

                    <div className="text-right">

                      <div className="text-sm text-muted-foreground">Confidence</div>

                      <div className="font-bold">{Math.round(insight.confidence \* 100)}%</div>

                    </div>

                  </div>

                  <div>

                    <h4 className="font-medium mb-2">AI Reasoning</h4>

                    <p className="text-sm text-muted-foreground leading-relaxed">

                      {insight.reasoning}

                    </p>

                  </div>

                  <div>

                    <h4 className="font-medium mb-2">Key Findings</h4>

                    <ul className="space-y-1">

                      {insight.keyFindings.map((finding, idx) => (

                        <li key={idx} className="flex items-start gap-2 text-sm">

                          <CheckCircle className="w-3 h-3 mt-0.5 text-green-600 flex-shrink-0" />

                          {finding}

                        </li>

                      ))}

                    </ul>

                  </div>

                  <div className="grid grid-cols-2 gap-4 pt-2 border-t">

                    <div className="text-center">

                      <div className="text-2xl font-bold text-primary">

                        {insight.technicalScore.toFixed(3)}

                      </div>

                      <div className="text-xs text-muted-foreground">Technical Score</div>

                    </div>

                    <div className="text-center">

                      <div className="text-2xl font-bold text-muted-foreground">

                        {Math.round(insight.processingTime / 1000)}s

                      </div>

                      <div className="text-xs text-muted-foreground">Processing Time</div>

                    </div>

                  </div>

                </CardContent>

              </Card>

            ))}

          </motion.div>

        )}

        {activeTab === 'evidence' && (

          <motion.div

            key="evidence"

            initial={{ opacity: 0, y: 20 }}

            animate={{ opacity: 1, y: 0 }}

            exit={{ opacity: 0, y: -20 }}

            transition={{ duration: 0.3 }}

            className="space-y-6"

          >

            {explanation.evidence.length > 0 ? (

              <div className="space-y-4">

                {explanation.evidence.map((evidence) => (

                  <Card key={evidence.id}>

                    <CardContent className="p-4">

                      <div className="flex items-start justify-between mb-3">

                        <div>

                          <Badge variant="outline" className="mb-2 capitalize">

                            {evidence.type.replace('\_', ' ')}

                          </Badge>

                          <p className="text-sm">{evidence.description}</p>

                        </div>

                        <div className="text-right">

                          <div className="text-sm text-muted-foreground">Severity</div>

                          <Progress

                            value={evidence.severity \* 100}

                            className="w-20 mt-1"

                          />

                        </div>

                      </div>

                      {evidence.location && (

                        <div className="flex gap-4 text-xs text-muted-foreground">

                          {evidence.location.frame && (

                            <span>Frame: {evidence.location.frame}</span>

                          )}

                          {evidence.location.timestamp && (

                            <span>Time: {evidence.location.timestamp.toFixed(2)}s</span>

                          )}

                          {evidence.location.frequency && (

                            <span>Frequency: {evidence.location.frequency}Hz</span>

                          )}

                        </div>

                      )}

                    </CardContent>

                  </Card>

                ))}

              </div>

            ) : (

              <Card>

                <CardContent className="text-center py-8">

                  <Shield className="w-12 h-12 mx-auto text-muted-foreground mb-4" />

                  <h3 className="font-medium mb-2">No Specific Evidence Found</h3>

                  <p className="text-sm text-muted-foreground">

                    The analysis did not identify specific anomalies or artifacts that require detailed examination.

                  </p>

                </CardContent>

              </Card>

            )}

            {*/\* Metadata Analysis \*/*}

            {explanation.metadataAnalysis && (

              <Card>

                <CardHeader>

                  <CardTitle className="flex items-center gap-2">

                    <FileText className="w-5 h-5" />

                    File Properties Analysis

                  </CardTitle>

                </CardHeader>

                <CardContent>

                  <div className="space-y-3">

                    {explanation.metadataAnalysis.fileProperties.map((prop, index) => (

                      <div key={index} className="flex items-center justify-between p-2 bg-muted/30 rounded">

                        <div>

                          <span className="font-medium">{prop.property}:</span>

                          <span className="ml-2">{prop.actualValue}</span>

                        </div>

                        <Badge variant={prop.assessment === 'normal' ? 'secondary' : 'destructive'}>

                          {prop.assessment}

                        </Badge>

                      </div>

                    ))}

                  </div>

                </CardContent>

              </Card>

            )}

          </motion.div>

        )}

      </AnimatePresence>

    </div>

  );

}

**src\components\pdf\pdf-chart-components.tsx**

'use client';

import { useEffect, useRef } from 'react';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

import { StoredAnalysis } from '@/lib/storage';

import { ConfidenceGauge } from '@/components/charts/confidence-gauge';

import { CategoryChart } from '@/components/charts/category-chart';

import {

  LineChart,

  Line,

  XAxis,

  YAxis,

  CartesianGrid,

  ResponsiveContainer,

  Tooltip,

  AreaChart,

  Area,

  PieChart,

  Pie,

  Cell,

} from 'recharts';

import {

  CheckCircle,

  AlertTriangle,

  XCircle,

  BarChart3,

} from 'lucide-react';

interface PDFChartComponentsProps {

  analysis: StoredAnalysis;

}

*// Generate timeline data from analysis results*

const generateTimelineData = (analysis: StoredAnalysis) => {

  if (!analysis) return [];

  const data = [];

  const baseConfidence = analysis.confidence;

  const points = analysis.details.frameAnalysis?.length || 20;

  for (let i = 0; i < points; i++) {

    const variation = (Math.random() - 0.5) \* 0.3;

    const confidence = Math.max(0, Math.min(1, baseConfidence + variation));

    data.push({

      frame: i + 1,

      timestamp: (i / points) \* (analysis.details.metadata.duration || 60),

      confidence: Math.round(confidence \* 100),

      anomalies: Math.random() > 0.8 ? 1 : 0,

    });

  }

  return data;

};

const getConfidenceColor = (confidence: number) => {

  if (confidence < 0.3) return 'text-green-600';

  if (confidence < 0.7) return 'text-yellow-600';

  return 'text-red-600';

};

const getConfidenceIcon = (confidence: number) => {

  if (confidence < 0.3) return CheckCircle;

  if (confidence < 0.7) return AlertTriangle;

  return XCircle;

};

export function PDFChartComponents({ analysis }: PDFChartComponentsProps) {

  const timelineData = generateTimelineData(analysis);

  const ConfidenceIcon = getConfidenceIcon(analysis.confidence);

*// Components positioned for PDF capture*

  return (

    <div

      className="fixed top-0 left-[-1500px] w-[800px] bg-white z-[9998]"

      style={{

        opacity: 0.01, *// Barely visible but not completely hidden*

        pointerEvents: 'none',

        visibility: 'visible'

      }}

    >

      {*/\* Confidence Gauge for PDF \*/*}

      <div id="confidence-gauge-pdf" className="p-6 bg-white">

        <div style={{ border: 'none', boxShadow: 'none', backgroundColor: '#ffffff', padding: '16px', borderRadius: '8px' }}>

          <div style={{ textAlign: 'center', paddingBottom: '8px' }}>

            <h3 style={{ display: 'flex', alignItems: 'center', justifyContent: 'center', gap: '8px', fontSize: '18px', margin: '0', color: '#1f2937' }}>

              ITL Deepfake Detection Results

            </h3>

          </div>

          <div style={{ padding: '16px' }}>

            {*/\* Confidence Circle \*/*}

            <div style={{ display: 'flex', alignItems: 'center', justifyContent: 'center' }}>

              <div style={{ position: 'relative', width: '128px', height: '128px' }}>

                <svg style={{ width: '128px', height: '128px', transform: 'rotate(-90deg)' }} viewBox="0 0 100 100">

                  <circle

                    cx="50"

                    cy="50"

                    r="40"

                    stroke="#e5e7eb"

                    strokeWidth="8"

                    fill="transparent"

                  />

                  <circle

                    cx="50"

                    cy="50"

                    r="40"

                    stroke={analysis.confidence < 0.3 ? '#10b981' : analysis.confidence < 0.7 ? '#f59e0b' : '#ef4444'}

                    strokeWidth="8"

                    fill="transparent"

                    strokeDasharray={`${2 \* Math.PI \* 40 \* analysis.confidence} ${2 \* Math.PI \* 40}`}

                    strokeLinecap="round"

                  />

                </svg>

                <div style={{ position: 'absolute', top: '0', left: '0', right: '0', bottom: '0', display: 'flex', alignItems: 'center', justifyContent: 'center' }}>

                  <div style={{ textAlign: 'center' }}>

                    <div style={{ fontSize: '24px', fontWeight: 'bold', color: analysis.confidence < 0.3 ? '#10b981' : analysis.confidence < 0.7 ? '#f59e0b' : '#ef4444' }}>

                      {analysis.confidence > 1 ? Math.round(analysis.confidence) : Math.round(analysis.confidence \* 100)}%

                    </div>

                    <div style={{ fontSize: '14px', color: '#6b7280' }}>Risk</div>

                  </div>

                </div>

              </div>

            </div>

            <div style={{ textAlign: 'center', marginTop: '16px' }}>

              <div style={{

                display: 'inline-block',

                padding: '4px 8px',

                borderRadius: '4px',

                backgroundColor: analysis.confidence < 0.3 ? '#3b82f6' : analysis.confidence < 0.7 ? '#6b7280' : '#ef4444',

                color: '#ffffff',

                fontSize: '14px',

                fontWeight: '500'

              }}>

                {analysis.prediction.toUpperCase()}

              </div>

              <p style={{ fontSize: '14px', color: '#4b5563', marginTop: '8px', margin: '8px 0 0 0' }}>

                {analysis.confidence < 0.3

                  ? 'Content appears authentic with low risk of manipulation'

                  : analysis.confidence < 0.7

                  ? 'Moderate risk detected - further verification recommended'

                  : 'High risk of artificial generation or manipulation detected'

                }

              </p>

            </div>

          </div>

        </div>

      </div>

      {*/\* Category Chart for PDF \*/*}

      {analysis.details.categoryBreakdown && (

        <div id="category-chart-pdf" style={{ padding: '24px', backgroundColor: '#ffffff' }}>

          <div style={{ border: 'none', boxShadow: 'none', backgroundColor: '#ffffff', borderRadius: '8px' }}>

            <div style={{ padding: '16px 16px 8px 16px' }}>

              <h3 style={{ fontSize: '18px', fontWeight: '600', margin: '0', color: '#1f2937' }}>Analysis Categories</h3>

            </div>

            <div style={{ padding: '16px' }}>

              <div style={{ height: '300px', width: '100%' }}>

                {*/\* Simple bar chart representation \*/*}

                <div style={{ display: 'flex', flexDirection: 'column', gap: '16px', height: '100%', justifyContent: 'center' }}>

                  {Object.entries(analysis.details.categoryBreakdown).map(([category, score], index) => {

                    const colors = ['#3b82f6', '#10b981', '#f59e0b', '#ef4444', '#8b5cf6'];

                    const color = colors[index % colors.length];

                    return (

                      <div key={category} style={{ display: 'flex', alignItems: 'center', gap: '12px' }}>

                        <div style={{ minWidth: '120px', fontSize: '14px', fontWeight: '500', color: '#374151', textTransform: 'capitalize' }}>

                          {category}

                        </div>

                        <div style={{ flex: '1', backgroundColor: '#f3f4f6', height: '20px', borderRadius: '10px', overflow: 'hidden' }}>

                          <div

                            style={{

                              height: '100%',

                              backgroundColor: color,

                              width: `${Math.max(score > 1 ? score : score \* 100, 2)}%`,

                              borderRadius: '10px',

                              transition: 'width 0.3s ease'

                            }}

                          />

                        </div>

                        <div style={{ minWidth: '50px', fontSize: '14px', fontWeight: '600', color: '#1f2937', textAlign: 'right' }}>

                          {score > 1 ? Math.round(score) : Math.round(score \* 100)}%

                        </div>

                      </div>

                    );

                  })}

                </div>

              </div>

            </div>

          </div>

        </div>

      )}

      {*/\* Timeline Chart for PDF \*/*}

      <div id="timeline-chart-pdf" style={{ padding: '24px', backgroundColor: '#ffffff' }}>

        <div style={{ border: 'none', boxShadow: 'none', backgroundColor: '#ffffff', borderRadius: '8px' }}>

          <div style={{ padding: '16px 16px 8px 16px' }}>

            <h3 style={{ fontSize: '18px', fontWeight: '600', margin: '0', color: '#1f2937' }}>Frame-by-Frame Analysis Timeline</h3>

          </div>

          <div style={{ padding: '16px' }}>

            <div style={{ height: '256px', width: '100%' }}>

              <ResponsiveContainer width="100%" height="100%">

                <AreaChart data={timelineData}>

                  <CartesianGrid strokeDasharray="3 3" stroke="#e5e7eb" />

                  <XAxis

                    dataKey="frame"

                    tick={{ fontSize: 12, fill: '#6b7280' }}

                    axisLine={false}

                    tickLine={false}

                  />

                  <YAxis

                    domain={[0, 100]}

                    tick={{ fontSize: 12, fill: '#6b7280' }}

                    axisLine={false}

                    tickLine={false}

                    label={{ value: 'Confidence %', angle: -90, position: 'insideLeft', style: { textAnchor: 'middle', fill: '#6b7280' } }}

                  />

                  <Tooltip

                    contentStyle={{

                      backgroundColor: '#ffffff',

                      border: '1px solid #d1d5db',

                      borderRadius: '8px',

                      boxShadow: '0 4px 6px -1px rgba(0, 0, 0, 0.1)',

                      padding: '12px'

                    }}

                    labelStyle={{ fontWeight: '500', color: '#1f2937' }}

                    itemStyle={{ color: '#4b5563', fontSize: '14px' }}

                  />

                  <Area

                    type="monotone"

                    dataKey="confidence"

                    stroke="#3b82f6"

                    fill="#3b82f6"

                    fillOpacity={0.2}

                    strokeWidth={2}

                  />

                </AreaChart>

              </ResponsiveContainer>

            </div>

          </div>

        </div>

      </div>

      {*/\* Risk Heatmap for PDF \*/*}

      <div id="risk-heatmap-pdf" style={{ padding: '24px', backgroundColor: '#ffffff' }}>

        <div style={{ border: 'none', boxShadow: 'none', backgroundColor: '#ffffff', borderRadius: '8px' }}>

          <div style={{ padding: '16px 16px 8px 16px' }}>

            <h3 style={{ fontSize: '18px', fontWeight: '600', margin: '0', color: '#1f2937' }}>Risk Analysis Heatmap</h3>

          </div>

          <div style={{ padding: '16px' }}>

            <div style={{ display: 'flex', flexDirection: 'column', gap: '16px' }}>

              {*/\* Mock heatmap using colored bars \*/*}

              <div style={{ display: 'grid', gridTemplateColumns: 'repeat(8, 1fr)', gap: '4px' }}>

                {Array.from({ length: 64 }).map((\_, i) => {

                  const hue = 120 - (analysis.confidence \* 120); *// Green to red based on confidence*

                  const lightness = 50 + (Math.sin(i \* 0.5) \* 20); *// Vary lightness for visual interest*

                  return (

                    <div

                      key={i}

                      style={{

                        aspectRatio: '1',

                        borderRadius: '2px',

                        backgroundColor: `hsl(${hue}, 60%, ${Math.max(30, Math.min(70, lightness))}%)`,

                      }}

                    />

                  );

                })}

              </div>

              <div style={{ display: 'flex', justifyContent: 'space-between', fontSize: '14px', color: '#6b7280' }}>

                <span>Low Risk</span>

                <span>High Risk</span>

              </div>

            </div>

          </div>

        </div>

      </div>

      {*/\* Anomaly Scatter Plot for PDF \*/*}

      <div id="anomaly-scatter-pdf" style={{ padding: '24px', backgroundColor: '#ffffff' }}>

        <div style={{ border: 'none', boxShadow: 'none', backgroundColor: '#ffffff', borderRadius: '8px' }}>

          <div style={{ padding: '16px 16px 8px 16px' }}>

            <h3 style={{ fontSize: '18px', fontWeight: '600', margin: '0', color: '#1f2937' }}>Anomaly Detection Plot</h3>

          </div>

          <div style={{ padding: '16px' }}>

            <div style={{ height: '256px', width: '100%' }}>

              <ResponsiveContainer width="100%" height="100%">

                <LineChart

                  data={timelineData.map((point, i) => ({

                    ...point,

                    anomaly: Math.random() \* 100,

                    x: i,

                    y: Math.random() \* 100,

                  }))}

                >

                  <CartesianGrid strokeDasharray="3 3" stroke="#e5e7eb" />

                  <XAxis dataKey="x" tick={{ fontSize: 12, fill: '#6b7280' }} />

                  <YAxis tick={{ fontSize: 12, fill: '#6b7280' }} />

                  <Tooltip

                    contentStyle={{

                      backgroundColor: '#ffffff',

                      border: '1px solid #d1d5db',

                      borderRadius: '8px',

                      padding: '12px'

                    }}

                  />

                  <Line

                    type="monotone"

                    dataKey="y"

                    stroke="#ef4444"

                    strokeWidth={0}

                    dot={{ r: 3, fill: '#ef4444' }}

                  />

                </LineChart>

              </ResponsiveContainer>

            </div>

          </div>

        </div>

      </div>

      {*/\* Radar Chart for PDF \*/*}

      <div id="radar-chart-pdf" style={{ padding: '24px', backgroundColor: '#ffffff' }}>

        <div style={{ border: 'none', boxShadow: 'none', backgroundColor: '#ffffff', borderRadius: '8px' }}>

          <div style={{ padding: '16px 16px 8px 16px' }}>

            <h3 style={{ fontSize: '18px', fontWeight: '600', margin: '0', color: '#1f2937' }}>Multidimensional Analysis</h3>

          </div>

          <div style={{ padding: '16px' }}>

            <div style={{ display: 'flex', flexDirection: 'column', gap: '16px' }}>

              {*/\* Mock radar chart using progress bars \*/*}

              {[

                { name: 'Visual Quality', score: Math.round(analysis.confidence \* 80 + 20) },

                { name: 'Temporal Consistency', score: Math.round(analysis.confidence \* 75 + 25) },

                { name: 'Audio Synchronization', score: Math.round(analysis.confidence \* 85 + 15) },

                { name: 'Compression Artifacts', score: Math.round(analysis.confidence \* 70 + 30) },

                { name: 'Facial Features', score: Math.round(analysis.confidence \* 90 + 10) },

              ].map((metric) => (

                <div key={metric.name} style={{ display: 'flex', flexDirection: 'column', gap: '8px' }}>

                  <div style={{ display: 'flex', justifyContent: 'space-between', fontSize: '14px' }}>

                    <span style={{ color: '#374151' }}>{metric.name}</span>

                    <span style={{ fontWeight: '500', color: '#1f2937' }}>{metric.score}%</span>

                  </div>

                  <div style={{ width: '100%', backgroundColor: '#e5e7eb', borderRadius: '9999px', height: '8px' }}>

                    <div

                      style={{

                        backgroundColor: '#2563eb',

                        height: '8px',

                        borderRadius: '9999px',

                        width: `${metric.score}%`,

                        transition: 'width 0.3s ease'

                      }}

                    />

                  </div>

                </div>

              ))}

            </div>

          </div>

        </div>

      </div>

    </div>

  );

}

*// Hook to mount PDF components when needed*

export function usePDFComponents() {

  const mountedRef = useRef(false);

  const mountPDFComponents = (analysis: StoredAnalysis) => {

    if (mountedRef.current) return;

    const container = document.createElement('div');

    container.id = 'pdf-components-container';

    document.body.appendChild(container);

*// This would need React to render the components*

*// For now, we'll use a different approach in the PDF generator*

    mountedRef.current = true;

  };

  const unmountPDFComponents = () => {

    const container = document.getElementById('pdf-components-container');

    if (container) {

      document.body.removeChild(container);

      mountedRef.current = false;

    }

  };

  return { mountPDFComponents, unmountPDFComponents };

}

**src\components\pdf\pdf-export-dialog.tsx**

'use client';

import { useState } from 'react';

import { Button } from '@/components/ui/button';

import {

  Dialog,

  DialogContent,

  DialogDescription,

  DialogFooter,

  DialogHeader,

  DialogTitle,

  DialogTrigger,

} from '@/components/ui/dialog';

import { Switch } from '@/components/ui/switch';

import { Progress } from '@/components/ui/progress';

import { Badge } from '@/components/ui/badge';

import {

  FileText,

  Download,

  Settings,

  BarChart3,

  Clock,

  Database,

  Loader2,

  CheckCircle

} from 'lucide-react';

import { toast } from 'sonner';

import { pdfGenerator, PDFGenerationOptions } from '@/lib/pdf-generator';

import { StoredAnalysis } from '@/lib/storage';

interface PDFExportDialogProps {

  analysis: StoredAnalysis;

  trigger?: React.ReactNode;

}

export function PDFExportDialog({ analysis, trigger }: PDFExportDialogProps) {

  const [isOpen, setIsOpen] = useState(false);

  const [isGenerating, setIsGenerating] = useState(false);

  const [progress, setProgress] = useState(0);

  const [options, setOptions] = useState<PDFGenerationOptions>({

    includeCharts: true,

    includeTimeline: true,

    includeAdvancedCharts: false,

    includeRawData: false,

  });

  const handleExport = async () => {

    setIsGenerating(true);

    setProgress(0);

    try {

*// Simulate progress updates*

      const progressInterval = setInterval(() => {

        setProgress(prev => {

          if (prev >= 90) {

            clearInterval(progressInterval);

            return 90;

          }

          return prev + 10;

        });

      }, 200);

*// Generate PDF*

      await pdfGenerator.downloadReport(analysis, options);

      clearInterval(progressInterval);

      setProgress(100);

      toast.success('PDF report generated successfully!', {

        description: 'The report has been downloaded to your device.',

      });

      setTimeout(() => {

        setIsOpen(false);

        setProgress(0);

      }, 1000);

    } catch (error) {

      toast.error('Failed to generate PDF report', {

        description: 'Please try again or contact support if the issue persists.',

      });

      console.error('PDF export error:', error);

    } finally {

      setIsGenerating(false);

    }

  };

  const getEstimatedSize = () => {

    let size = 0.5; *// Base size in MB*

    if (options.includeCharts) size += 1.0;

    if (options.includeTimeline) size += 0.5;

    if (options.includeAdvancedCharts) size += 1.5;

    if (options.includeRawData) size += 0.3;

    return size.toFixed(1);

  };

  const getEstimatedPages = () => {

    let pages = 3; *// Base pages (header, summary, details, disclaimer)*

    if (options.includeCharts) pages += 1;

    if (options.includeTimeline) pages += 1;

    if (options.includeAdvancedCharts) pages += 2;

    if (options.includeRawData) pages += 1;

    return pages;

  };

  return (

    <Dialog open={isOpen} onOpenChange={setIsOpen}>

      <DialogTrigger asChild>

        {trigger || (

          <Button variant="outline" size="sm">

            <FileText className="w-4 h-4 mr-2" />

            Export PDF

          </Button>

        )}

      </DialogTrigger>

      <DialogContent className="sm:max-w-[500px]">

        <DialogHeader>

          <DialogTitle className="flex items-center gap-2">

            <FileText className="w-5 h-5" />

            Export PDF Report

          </DialogTitle>

          <DialogDescription>

            Customize your PDF report options. The report will include a comprehensive analysis

            of {analysis.filename}.

          </DialogDescription>

        </DialogHeader>

        {isGenerating ? (

          <div className="space-y-6 py-6">

            <div className="text-center space-y-4">

              {progress < 100 ? (

                <Loader2 className="w-12 h-12 animate-spin mx-auto text-primary" />

              ) : (

                <CheckCircle className="w-12 h-12 mx-auto text-green-600" />

              )}

              <div className="space-y-2">

                <p className="text-sm font-medium">

                  {progress < 100 ? 'Generating PDF report...' : 'Report generated successfully!'}

                </p>

                <Progress value={progress} className="w-full" />

                <p className="text-xs text-muted-foreground">

                  {progress}% complete

                </p>

              </div>

            </div>

          </div>

        ) : (

          <div className="space-y-6">

            {*/\* Report Preview Info \*/*}

            <div className="bg-muted/30 p-4 rounded-lg space-y-2">

              <div className="flex items-center justify-between text-sm">

                <span className="text-muted-foreground">Estimated size:</span>

                <Badge variant="secondary">{getEstimatedSize()} MB</Badge>

              </div>

              <div className="flex items-center justify-between text-sm">

                <span className="text-muted-foreground">Estimated pages:</span>

                <Badge variant="secondary">{getEstimatedPages()} pages</Badge>

              </div>

              <div className="flex items-center justify-between text-sm">

                <span className="text-muted-foreground">Format:</span>

                <Badge variant="secondary">PDF (A4)</Badge>

              </div>

            </div>

            {*/\* Export Options \*/*}

            <div className="space-y-4">

              <h4 className="font-medium text-sm">Report Content</h4>

              <div className="space-y-3">

                <div className="flex items-center justify-between">

                  <div className="flex items-center space-x-3">

                    <BarChart3 className="w-4 h-4 text-muted-foreground" />

                    <div>

                      <p className="text-sm font-medium">Include Charts</p>

                      <p className="text-xs text-muted-foreground">

                        Confidence gauge and category breakdown

                      </p>

                    </div>

                  </div>

                  <Switch

                    checked={options.includeCharts}

                    onCheckedChange={(checked) =>

                      setOptions(prev => ({ ...prev, includeCharts: checked }))

                    }

                  />

                </div>

                <div className="flex items-center justify-between">

                  <div className="flex items-center space-x-3">

                    <Clock className="w-4 h-4 text-muted-foreground" />

                    <div>

                      <p className="text-sm font-medium">Timeline Analysis</p>

                      <p className="text-xs text-muted-foreground">

                        Frame-by-frame confidence timeline

                      </p>

                    </div>

                  </div>

                  <Switch

                    checked={options.includeTimeline}

                    onCheckedChange={(checked) =>

                      setOptions(prev => ({ ...prev, includeTimeline: checked }))

                    }

                  />

                </div>

                <div className="flex items-center justify-between">

                  <div className="flex items-center space-x-3">

                    <Settings className="w-4 h-4 text-muted-foreground" />

                    <div>

                      <p className="text-sm font-medium">Advanced Charts</p>

                      <p className="text-xs text-muted-foreground">

                        Heatmaps, scatter plots, and radar charts

                      </p>

                    </div>

                  </div>

                  <Switch

                    checked={options.includeAdvancedCharts}

                    onCheckedChange={(checked) =>

                      setOptions(prev => ({ ...prev, includeAdvancedCharts: checked }))

                    }

                  />

                </div>

                <div className="flex items-center justify-between">

                  <div className="flex items-center space-x-3">

                    <Database className="w-4 h-4 text-muted-foreground" />

                    <div>

                      <p className="text-sm font-medium">Raw Data</p>

                      <p className="text-xs text-muted-foreground">

                        Complete JSON analysis data

                      </p>

                    </div>

                  </div>

                  <Switch

                    checked={options.includeRawData}

                    onCheckedChange={(checked) =>

                      setOptions(prev => ({ ...prev, includeRawData: checked }))

                    }

                  />

                </div>

              </div>

            </div>

            {*/\* Report Content Summary \*/*}

            <div className="bg-blue-50 dark:bg-blue-950/20 p-3 rounded-lg">

              <p className="text-xs text-blue-700 dark:text-blue-300">

                💡 <strong>Note:</strong> The PDF report will always include an executive summary,

                technical details, and disclaimer regardless of the options selected above.

              </p>

            </div>

          </div>

        )}

        <DialogFooter>

          {!isGenerating && (

            <>

              <Button variant="outline" onClick={() => setIsOpen(false)}>

                Cancel

              </Button>

              <Button onClick={handleExport} className="min-w-[120px]">

                <Download className="w-4 h-4 mr-2" />

                Generate PDF

              </Button>

            </>

          )}

        </DialogFooter>

      </DialogContent>

    </Dialog>

  );

}

**src\components\upload-box.tsx**

'use client';

import { useState, useCallback, useRef } from 'react';

import { useDropzone } from 'react-dropzone';

import { Button } from '@/components/ui/button';

import { Progress } from '@/components/ui/progress';

import { Badge } from '@/components/ui/badge';

import {

  Card,

  CardContent,

  CardDescription,

  CardHeader,

  CardTitle,

} from '@/components/ui/card';

import {

  Upload,

  File,

  FileImage,

  FileVideo,

  FileAudio,

  X,

  AlertCircle,

  CheckCircle2,

  Loader2,

} from 'lucide-react';

import { motion, AnimatePresence } from 'framer-motion';

import { formatFileSize, getFileIcon } from '@/lib/storage';

import { usageTracker } from '@/lib/usage-tracker';

import type { FileUploadState, UploadProgress } from '@/lib/types';

interface UploadBoxProps {

  onFileSelect: (file: File) => void;

  onFileRemove: () => void;

  uploadState: FileUploadState;

  uploadProgress?: UploadProgress;

  className?: string;

  accept?: Record<string, string[]>;

  maxSize?: number;

  disabled?: boolean;

}

*// Free tier only supports images and audio - no video*

const DEFAULT\_ACCEPT = {

  'image/\*': ['.png', '.jpg', '.jpeg', '.gif', '.webp'],

  'audio/\*': ['.mp3', '.wav', '.flac', '.aac', '.ogg', '.m4a'],

*// Video support requires Growth plan ($399/month)*

*// 'video/\*': ['.mp4', '.webm', '.mov', '.avi', '.mkv'],*

*// 'application/pdf': ['.pdf'],*

};

*// Reality Defender limits: Images 10MB, Audio 100MB*

const DEFAULT\_MAX\_SIZE = 100 \* 1024 \* 1024; *// 100MB (will validate per file type)*

export function UploadBox({

  onFileSelect,

  onFileRemove,

  uploadState,

  uploadProgress,

  className,

  accept = DEFAULT\_ACCEPT,

  maxSize = DEFAULT\_MAX\_SIZE,

  disabled = false,

}: UploadBoxProps) {

  const [preview, setPreview] = useState<string | null>(null);

  const onDrop = useCallback(

    (acceptedFiles: File[], rejectedFiles: any[]) => {

      if (disabled || uploadState.status === 'processing') return;

*// Check free tier quota*

      if (typeof window !== 'undefined' && !usageTracker.canMakeScan()) {

        console.error('Free tier quota exceeded');

        alert(`You've reached your monthly limit of ${usageTracker.getUsageStats().freeTierLimit} scans. Your quota will reset next month.`);

        return;

      }

      if (rejectedFiles.length > 0) {

        const error = rejectedFiles[0].errors[0];

        console.error('File rejection:', error);

        return;

      }

      if (acceptedFiles.length > 0) {

        const file = acceptedFiles[0];

*// Validate file size based on Reality Defender limits*

        const maxImageSize = 10 \* 1024 \* 1024; *// 10MB for images*

        const maxAudioSize = 100 \* 1024 \* 1024; *// 100MB for audio*

        if (file.type.startsWith('image/') && file.size > maxImageSize) {

          alert('Image files must be smaller than 10MB');

          return;

        }

        if (file.type.startsWith('audio/') && file.size > maxAudioSize) {

          alert('Audio files must be smaller than 100MB');

          return;

        }

        onFileSelect(file);

*// Create preview for images and videos*

        if (file.type.startsWith('image/')) {

          const reader = new FileReader();

          reader.onload = () => setPreview(reader.result as string);

          reader.readAsDataURL(file);

        } else if (file.type.startsWith('video/')) {

          const url = URL.createObjectURL(file);

          setPreview(url);

        } else {

          setPreview(null);

        }

      }

    },

    [onFileSelect, disabled, uploadState.status]

  );

  const { getRootProps, getInputProps, isDragActive, isDragReject } = useDropzone({

    onDrop,

    accept,

    maxSize,

    multiple: false,

    disabled: disabled || uploadState.status === 'processing',

    onDropRejected: (rejectedFiles) => {

      const firstError = rejectedFiles[0]?.errors[0];

      console.error('Upload rejected:', firstError?.message);

    },

  });

  const handleRemoveFile = () => {

    onFileRemove();

    setPreview(null);

    if (preview && preview.startsWith('blob:')) {

      URL.revokeObjectURL(preview);

    }

  };

  const getStatusIcon = () => {

    switch (uploadState.status) {

      case 'uploading':

      case 'processing':

        return <Loader2 className="h-6 w-6 animate-spin text-primary" />;

      case 'completed':

        return <CheckCircle2 className="h-6 w-6 text-green-600" />;

      case 'error':

        return <AlertCircle className="h-6 w-6 text-red-600" />;

      default:

        return <Upload className="h-6 w-6 text-muted-foreground" />;

    }

  };

  const getStatusText = () => {

    if (uploadProgress) {

      return uploadProgress.message;

    }

    switch (uploadState.status) {

      case 'uploading':

        return 'Uploading file to secure servers...';

      case 'processing':

        return 'AI models analyzing for manipulation patterns...';

      case 'completed':

        return 'Analysis completed! View detailed results.';

      case 'error':

        return uploadState.error || 'Upload failed';

      default:

        return 'Drag & drop your file here, or click to browse';

    }

  };

  const getDetailedProgress = () => {

    if (!uploadProgress) return null;

    const stages = [

      { key: 'upload', label: 'File Upload', icon: Upload },

      { key: 'preprocessing', label: 'Media Processing', icon: Loader2 },

      { key: 'analysis', label: 'AI Analysis', icon: CheckCircle2 },

      { key: 'results', label: 'Results Ready', icon: CheckCircle2 }

    ];

    return (

      <div className="space-y-3">

        <div className="flex items-center justify-between text-sm">

          <span className="font-medium">{uploadProgress.message}</span>

          <span className="text-muted-foreground">{uploadProgress.percentage}%</span>

        </div>

        <Progress value={uploadProgress.percentage} className="h-2" />

        {*/\* Analysis Stages \*/*}

        <div className="grid grid-cols-2 gap-2 mt-4">

          {stages.map((stage, index) => {

            const isActive = uploadProgress.stage === stage.key;

            const isCompleted = uploadProgress.stagesCompleted?.includes(stage.key);

            const Icon = stage.icon;

            return (

              <div

                key={stage.key}

                className={`flex items-center gap-2 p-2 rounded-lg text-xs transition-colors ${

                  isActive

                    ? 'bg-primary/10 text-primary border border-primary/20'

                    : isCompleted

                    ? 'bg-green-50 text-green-700 dark:bg-green-900/20 dark:text-green-400'

                    : 'bg-muted/50 text-muted-foreground'

                }`}

              >

                <Icon className={`h-3 w-3 ${

                  isActive ? 'animate-spin' : ''

                }`} />

                <span className="font-medium">{stage.label}</span>

                {isCompleted && <CheckCircle2 className="h-3 w-3 ml-auto" />}

              </div>

            );

          })}

        </div>

        {*/\* Real-time Hints \*/*}

        <div className="mt-3 p-3 bg-blue-50 dark:bg-blue-900/20 rounded-lg border border-blue-200 dark:border-blue-800">

          <div className="flex items-start gap-2">

            <div className="w-2 h-2 bg-blue-500 rounded-full mt-2 animate-pulse" />

            <div className="text-xs text-blue-700 dark:text-blue-300">

              <div className="font-medium mb-1">What's happening now:</div>

              <div>{getAnalysisHint(uploadProgress)}</div>

            </div>

          </div>

        </div>

      </div>

    );

  };

  const getAnalysisHint = (progress: UploadProgress) => {

    switch (progress.stage) {

      case 'upload':

        return 'Securely uploading your file to Reality Defender servers for analysis.';

      case 'preprocessing':

        return 'Extracting frames, audio tracks, and metadata for comprehensive analysis.';

      case 'analysis':

        if (progress.analysisDetails) {

          const { activeModels, completedModels, totalModels } = progress.analysisDetails;

          return `AI models analyzing: ${completedModels}/${totalModels} completed. Active: ${activeModels?.join(', ') || 'Multiple models'}.`;

        }

        return 'Multiple AI models examining your media for manipulation patterns and deepfake signatures.';

      case 'results':

        return 'Compiling results from all AI models and generating detailed analysis report.';

      default:

        return 'Processing your media file...';

    }

  };

  const getAcceptedFormats = () => {

    const formats = Object.values(accept).flat();

    const unique = [...new Set(formats)];

    return unique.slice(0, 6).join(', ') + (unique.length > 6 ? '...' : '');

  };

  return (

    <Card className={className}>

      <CardHeader>

        <CardTitle className="flex items-center gap-2">

          {getStatusIcon()}

          Upload Media File

        </CardTitle>

        <CardDescription>

          {uploadState.file ? uploadState.file.name : 'Select a file to analyze for deepfake detection'}

        </CardDescription>

      </CardHeader>

      <CardContent>

        <AnimatePresence mode="wait">

          {uploadState.file ? (

            <motion.div

              key="file-selected"

              initial={{ opacity: 0, scale: 0.95 }}

              animate={{ opacity: 1, scale: 1 }}

              exit={{ opacity: 0, scale: 0.95 }}

              transition={{ duration: 0.2 }}

            >

              <div className="space-y-4">

                {*/\* File Preview \*/*}

                <div className="flex items-center gap-4 p-4 bg-muted/50 rounded-lg">

                  <div className="w-16 h-16 rounded-lg bg-background flex items-center justify-center flex-shrink-0 overflow-hidden">

                    {preview ? (

                      uploadState.file?.type.startsWith('image/') ? (

                        <img

                          src={preview}

                          alt="Preview"

                          className="w-full h-full object-cover rounded-lg"

                        />

                      ) : uploadState.file?.type.startsWith('video/') ? (

                        <video

                          src={preview}

                          className="w-full h-full object-cover rounded-lg"

                          muted

                        />

                      ) : null

                    ) : (

                      <span className="text-2xl">

                        {getFileIcon(uploadState.file?.type || '')}

                      </span>

                    )}

                  </div>

                  <div className="flex-1 min-w-0">

                    <h4 className="font-medium truncate" title={uploadState.file.name}>

                      {uploadState.file.name}

                    </h4>

                    <p className="text-sm text-muted-foreground">

                      {formatFileSize(uploadState.file.size)}

                    </p>

                    <div className="flex items-center gap-2 mt-1">

                      <Badge variant="outline" className="text-xs">

                        {uploadState.file.type.split('/')[0]}

                      </Badge>

                      {uploadState.status === 'completed' && (

                        <Badge variant="secondary" className="text-xs text-green-600">

                          <CheckCircle2 className="w-3 h-3 mr-1" />

                          Analyzed

                        </Badge>

                      )}

                    </div>

                  </div>

                  {uploadState.status !== 'processing' && uploadState.status !== 'uploading' && (

                    <Button

                      variant="ghost"

                      size="sm"

                      onClick={handleRemoveFile}

                      className="flex-shrink-0"

                    >

                      <X className="h-4 w-4" />

                    </Button>

                  )}

                </div>

                {*/\* Progress Bar \*/*}

                {(uploadState.status === 'uploading' || uploadState.status === 'processing') && (

                  <div className="space-y-2">

                    {uploadProgress ? (

                      getDetailedProgress()

                    ) : (

                      <>

                        <Progress value={uploadState.progress} className="w-full" />

                        <div className="flex justify-between text-sm">

                          <span className="text-muted-foreground">{getStatusText()}</span>

                          <span className="text-muted-foreground">{uploadState.progress}%</span>

                        </div>

                      </>

                    )}

                  </div>

                )}

                {*/\* Error Message \*/*}

                {uploadState.status === 'error' && uploadState.error && (

                  <div className="flex items-center gap-2 p-3 bg-red-50 dark:bg-red-900/20 border border-red-200 dark:border-red-800 rounded-lg">

                    <AlertCircle className="h-4 w-4 text-red-600 dark:text-red-400 flex-shrink-0" />

                    <p className="text-sm text-red-700 dark:text-red-300">{uploadState.error}</p>

                  </div>

                )}

                {*/\* Success Message \*/*}

                {uploadState.status === 'completed' && (

                  <div className="flex items-center gap-2 p-3 bg-green-50 dark:bg-green-900/20 border border-green-200 dark:border-green-800 rounded-lg">

                    <CheckCircle2 className="h-4 w-4 text-green-600 dark:text-green-400 flex-shrink-0" />

                    <p className="text-sm text-green-700 dark:text-green-300">

                      File analyzed successfully! Check the results below.

                    </p>

                  </div>

                )}

              </div>

            </motion.div>

          ) : (

            <motion.div

              key="upload-area"

              initial={{ opacity: 0, scale: 0.95 }}

              animate={{ opacity: 1, scale: 1 }}

              exit={{ opacity: 0, scale: 0.95 }}

              transition={{ duration: 0.2 }}

            >

              <div

                {...getRootProps()}

                className={`

                  upload-zone

                  relative min-h-[200px] p-8 rounded-lg transition-all duration-200 cursor-pointer

                  flex flex-col items-center justify-center text-center space-y-4

                  ${isDragActive ? 'active border-primary bg-primary/5' : ''}

                  ${isDragReject ? 'border-red-500 bg-red-50 dark:bg-red-900/20' : ''}

                  ${disabled ? 'opacity-50 cursor-not-allowed' : 'hover:border-primary/50'}

                `}

              >

                <input {...getInputProps()} />

                <motion.div

                  animate={{

                    y: isDragActive ? -4 : 0,

                    scale: isDragActive ? 1.05 : 1,

                  }}

                  transition={{ duration: 0.2 }}

                  className="flex flex-col items-center space-y-3"

                >

                  <div className={`

                    w-16 h-16 rounded-full flex items-center justify-center

                    ${isDragActive ? 'bg-primary text-primary-foreground' : 'bg-muted'}

                    ${isDragReject ? 'bg-red-100 text-red-600 dark:bg-red-900/40' : ''}

                  `}>

                    {isDragReject ? (

                      <AlertCircle className="h-8 w-8" />

                    ) : (

                      <Upload className={`h-8 w-8 ${isDragActive ? 'animate-bounce-subtle' : ''}`} />

                    )}

                  </div>

                  <div>

                    <h3 className="font-medium mb-1">

                      {isDragReject

                        ? 'Invalid file type or size'

                        : isDragActive

                        ? 'Drop your file here'

                        : 'Drag & drop your file here'

                      }

                    </h3>

                    <p className="text-sm text-muted-foreground">

                      {isDragReject

                        ? 'Please select a valid media file'

                        : 'or click to browse your files'

                      }

                    </p>

                  </div>

                  {!isDragReject && (

                    <Button variant="outline" size="sm" disabled={disabled}>

                      Choose File

                    </Button>

                  )}

                </motion.div>

                {*/\* Scan line animation when processing \*/*}

                {uploadState.status === 'processing' && (

                  <div className="absolute inset-0 overflow-hidden rounded-lg pointer-events-none">

                    <div className="animate-scan absolute inset-y-0 w-1 bg-gradient-to-b from-transparent via-primary to-transparent opacity-50" />

                  </div>

                )}

              </div>

              {*/\* File Type Info \*/*}

              <div className="mt-4 text-center">

                <p className="text-sm text-muted-foreground mb-2">Supported formats:</p>

                <p className="text-xs text-muted-foreground">

                  {getAcceptedFormats()}

                </p>

                <p className="text-xs text-muted-foreground mt-1">

                  Max size: {Math.round(maxSize / (1024 \* 1024))}MB

                </p>

              </div>

            </motion.div>

          )}

        </AnimatePresence>

      </CardContent>

    </Card>

  );

}

**src\components\usage-dashboard.tsx**

'use client';

import { useState, useEffect, useMemo } from 'react';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Progress } from '@/components/ui/progress';

import { Badge } from '@/components/ui/badge';

import {

  BarChart,

  Bar,

  LineChart,

  Line,

  XAxis,

  YAxis,

  CartesianGrid,

  Tooltip,

  ResponsiveContainer,

  PieChart,

  Pie,

  Cell

} from 'recharts';

import {

  Zap,

  TrendingUp,

  AlertTriangle,

  CheckCircle,

  FileImage,

  Calendar,

  Target,

} from 'lucide-react';

import { motion } from 'framer-motion';

import { usageTracker, UsageStats } from '@/lib/usage-tracker';

export function UsageDashboard() {

  const [stats, setStats] = useState<UsageStats | null>(null);

  const [isClient, setIsClient] = useState(false);

  useEffect(() => {

    setIsClient(true);

    if (typeof window !== 'undefined') {

      setStats(usageTracker.getUsageStats());

    }

  }, []);

  const trackerData = useMemo(() => {

    if (!isClient || !stats) return null;

    return {

      remainingScans: usageTracker.getRemainingScans(),

      usagePercentage: usageTracker.getUsagePercentage(),

      weeklyUsage: usageTracker.getWeeklyUsage(),

      fileTypeUsage: usageTracker.getMonthlyUsageByFileType(),

      confidenceDistribution: usageTracker.getConfidenceDistribution(),

      predictionStats: usageTracker.getPredictionStats(),

    };

  }, [isClient, stats]);

  if (!isClient || !stats || !trackerData) {

    return <div>Loading usage statistics...</div>;

  }

  const { remainingScans, usagePercentage, weeklyUsage, fileTypeUsage, confidenceDistribution, predictionStats } = trackerData;

  const getUsageColor = (percentage: number) => {

    if (percentage >= 90) return 'text-red-600 dark:text-red-400';

    if (percentage >= 70) return 'text-yellow-600 dark:text-yellow-400';

    return 'text-green-600 dark:text-green-400';

  };

  const getUsageBadgeVariant = (percentage: number) => {

    if (percentage >= 90) return 'destructive';

    if (percentage >= 70) return 'secondary';

    return 'default';

  };

  const COLORS = ['#10b981', '#f59e0b', '#ef4444', '#8b5cf6', '#06b6d4'];

  return (

    <div className="space-y-6">

      {*/\* Usage Overview \*/*}

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">

        <motion.div

          initial={{ opacity: 0, y: 20 }}

          animate={{ opacity: 1, y: 0 }}

          transition={{ duration: 0.5 }}

        >

          <Card>

            <CardHeader className="pb-2">

              <CardTitle className="text-sm font-medium">Monthly Usage</CardTitle>

            </CardHeader>

            <CardContent>

              <div className="flex items-center justify-between mb-2">

                <span className={`text-2xl font-bold ${getUsageColor(usagePercentage)}`}>

                  {stats.monthlyScans}

                </span>

                <Badge variant={getUsageBadgeVariant(usagePercentage)}>

                  {usagePercentage.toFixed(1)}%

                </Badge>

              </div>

              <Progress value={usagePercentage} className="h-2" />

              <p className="text-xs text-muted-foreground mt-1">

                {remainingScans} scans remaining

              </p>

            </CardContent>

          </Card>

        </motion.div>

        <motion.div

          initial={{ opacity: 0, y: 20 }}

          animate={{ opacity: 1, y: 0 }}

          transition={{ duration: 0.5, delay: 0.1 }}

        >

          <Card>

            <CardHeader className="pb-2">

              <CardTitle className="text-sm font-medium">Free Tier Limit</CardTitle>

            </CardHeader>

            <CardContent>

              <div className="flex items-center gap-2 mb-2">

                <Target className="h-4 w-4 text-primary" />

                <span className="text-2xl font-bold">{stats.freeTierLimit}</span>

              </div>

              <p className="text-xs text-muted-foreground">

                Audio & Image files per month

              </p>

              <div className="mt-2">

                <Badge variant="outline" className="text-xs">

                  Reality Defender Free

                </Badge>

              </div>

            </CardContent>

          </Card>

        </motion.div>

        <motion.div

          initial={{ opacity: 0, y: 20 }}

          animate={{ opacity: 1, y: 0 }}

          transition={{ duration: 0.5, delay: 0.2 }}

        >

          <Card>

            <CardHeader className="pb-2">

              <CardTitle className="text-sm font-medium">Total Scans</CardTitle>

            </CardHeader>

            <CardContent>

              <div className="flex items-center gap-2 mb-2">

                <CheckCircle className="h-4 w-4 text-green-500" />

                <span className="text-2xl font-bold">{stats.totalScans}</span>

              </div>

              <p className="text-xs text-muted-foreground">

                All-time analyses completed

              </p>

            </CardContent>

          </Card>

        </motion.div>

        <motion.div

          initial={{ opacity: 0, y: 20 }}

          animate={{ opacity: 1, y: 0 }}

          transition={{ duration: 0.5, delay: 0.3 }}

        >

          <Card>

            <CardHeader className="pb-2">

              <CardTitle className="text-sm font-medium">Next Reset</CardTitle>

            </CardHeader>

            <CardContent>

              <div className="flex items-center gap-2 mb-2">

                <Calendar className="h-4 w-4 text-primary" />

                <span className="text-lg font-bold">

                  {new Date(new Date().getFullYear(), new Date().getMonth() + 1, 1)

                    .toLocaleDateString('en-US', { month: 'short', day: 'numeric' })}

                </span>

              </div>

              <p className="text-xs text-muted-foreground">

                Monthly quota resets

              </p>

            </CardContent>

          </Card>

        </motion.div>

      </div>

      {*/\* Usage Alerts \*/*}

      {usagePercentage >= 80 && (

        <motion.div

          initial={{ opacity: 0, scale: 0.95 }}

          animate={{ opacity: 1, scale: 1 }}

          transition={{ duration: 0.3 }}

        >

          <Card className="border-yellow-200 bg-yellow-50 dark:border-yellow-800 dark:bg-yellow-950">

            <CardContent className="pt-6">

              <div className="flex items-center gap-3">

                <AlertTriangle className="h-5 w-5 text-yellow-600" />

                <div>

                  <h3 className="font-medium text-yellow-800 dark:text-yellow-200">

                    Usage Alert: {usagePercentage.toFixed(1)}% of monthly quota used

                  </h3>

                  <p className="text-sm text-yellow-700 dark:text-yellow-300">

                    You have {remainingScans} scans remaining this month. Consider upgrading for unlimited access.

                  </p>

                </div>

              </div>

            </CardContent>

          </Card>

        </motion.div>

      )}

      {*/\* Charts Grid \*/*}

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">

        {*/\* Weekly Usage Chart \*/*}

        <Card>

          <CardHeader>

            <CardTitle className="flex items-center gap-2">

              <TrendingUp className="h-4 w-4" />

              Weekly Usage Trend

            </CardTitle>

          </CardHeader>

          <CardContent>

            <div className="h-64">

              <ResponsiveContainer width="100%" height="100%">

                <LineChart data={weeklyUsage}>

                  <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

                  <XAxis dataKey="day" tick={{ fontSize: 12 }} />

                  <YAxis tick={{ fontSize: 12 }} />

                  <Tooltip

                    content={({ active, payload, label }) => {

                      if (active && payload && payload.length) {

                        return (

                          <div className="bg-background border rounded-lg shadow-lg p-3">

                            <p className="font-medium">Date: {label}</p>

                            <p className="text-sm">Scans: {payload[0].value}</p>

                          </div>

                        );

                      }

                      return null;

                    }}

                  />

                  <Line

                    type="monotone"

                    dataKey="count"

                    stroke="#8884d8"

                    strokeWidth={2}

                    dot={{ fill: '#8884d8', strokeWidth: 2, r: 4 }}

                  />

                </LineChart>

              </ResponsiveContainer>

            </div>

          </CardContent>

        </Card>

        {*/\* File Type Distribution \*/*}

        <Card>

          <CardHeader>

            <CardTitle className="flex items-center gap-2">

              <FileImage className="h-4 w-4" />

              File Type Distribution

            </CardTitle>

          </CardHeader>

          <CardContent>

            <div className="h-64">

              {Object.keys(fileTypeUsage).length > 0 ? (

                <ResponsiveContainer width="100%" height="100%">

                  <PieChart>

                    <Pie

                      data={Object.entries(fileTypeUsage).map(([type, count]) => ({

                        name: type.charAt(0).toUpperCase() + type.slice(1),

                        value: count,

                        percentage: ((count / stats.monthlyScans) \* 100).toFixed(1)

                      }))}

                      cx="50%"

                      cy="50%"

                      labelLine={false}

                      label={({ name, percentage }) => `${name} (${percentage}%)`}

                      outerRadius={80}

                      fill="#8884d8"

                      dataKey="value"

                    >

                      {Object.entries(fileTypeUsage).map((\_, index) => (

                        <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />

                      ))}

                    </Pie>

                    <Tooltip />

                  </PieChart>

                </ResponsiveContainer>

              ) : (

                <div className="flex items-center justify-center h-full text-muted-foreground">

                  <div className="text-center">

                    <FileImage className="h-8 w-8 mx-auto mb-2 opacity-50" />

                    <p className="text-sm">No usage data yet</p>

                  </div>

                </div>

              )}

            </div>

          </CardContent>

        </Card>

        {*/\* Confidence Distribution \*/*}

        <Card>

          <CardHeader>

            <CardTitle>Confidence Score Distribution</CardTitle>

          </CardHeader>

          <CardContent>

            <div className="h-64">

              <ResponsiveContainer width="100%" height="100%">

                <BarChart data={confidenceDistribution}>

                  <CartesianGrid strokeDasharray="3 3" className="opacity-30" />

                  <XAxis dataKey="range" tick={{ fontSize: 12 }} />

                  <YAxis tick={{ fontSize: 12 }} />

                  <Tooltip />

                  <Bar

                    dataKey="count"

                    fill="#8884d8"

                    radius={[4, 4, 0, 0]}

                  />

                </BarChart>

              </ResponsiveContainer>

            </div>

          </CardContent>

        </Card>

        {*/\* Prediction Statistics \*/*}

        <Card>

          <CardHeader>

            <CardTitle>Detection Results Summary</CardTitle>

          </CardHeader>

          <CardContent>

            <div className="space-y-3">

              {Object.entries(predictionStats).map(([prediction, count], index) => (

                <motion.div

                  key={prediction}

                  initial={{ opacity: 0, x: -20 }}

                  animate={{ opacity: 1, x: 0 }}

                  transition={{ delay: index \* 0.1 }}

                  className="flex items-center justify-between p-3 rounded-lg border"

                >

                  <div className="flex items-center gap-2">

                    <div

                      className="w-3 h-3 rounded-full"

                      style={{

                        backgroundColor: prediction === 'authentic' ? '#10b981' :

                                        prediction === 'manipulated' ? '#ef4444' : '#f59e0b'

                      }}

                    />

                    <span className="font-medium text-sm capitalize">

                      {prediction}

                    </span>

                  </div>

                  <div className="text-right">

                    <div className="font-bold">{count}</div>

                    <div className="text-xs text-muted-foreground">

                      {((count / stats.totalScans) \* 100).toFixed(1)}%

                    </div>

                  </div>

                </motion.div>

              ))}

            </div>

            {stats.totalScans === 0 && (

              <div className="text-center py-8 text-muted-foreground">

                <CheckCircle className="h-8 w-8 mx-auto mb-2 opacity-50" />

                <p className="text-sm">Upload files to see detection statistics</p>

              </div>

            )}

          </CardContent>

        </Card>

      </div>

      {*/\* Optimization Tips \*/*}

      <Card>

        <CardHeader>

          <CardTitle className="flex items-center gap-2">

            <Zap className="h-4 w-4" />

            Free Tier Optimization Tips

          </CardTitle>

        </CardHeader>

        <CardContent>

          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">

            <div className="p-4 rounded-lg bg-muted/30">

              <h4 className="font-medium text-sm mb-2">📊 Batch Analysis</h4>

              <p className="text-xs text-muted-foreground">

                Upload multiple files at once to make the most of your 50 monthly scans.

              </p>

            </div>

            <div className="p-4 rounded-lg bg-muted/30">

              <h4 className="font-medium text-sm mb-2">🎯 Focus on Suspicious Content</h4>

              <p className="text-xs text-muted-foreground">

                Save scans by manually reviewing files before uploading obviously authentic content.

              </p>

            </div>

            <div className="p-4 rounded-lg bg-muted/30">

              <h4 className="font-medium text-sm mb-2">📅 Track Your Usage</h4>

              <p className="text-xs text-muted-foreground">

                Monitor your monthly progress to pace your usage throughout the month.

              </p>

            </div>

          </div>

        </CardContent>

      </Card>

    </div>

  );

}

## 1.10 Running the Project

### 1.10.1 Development Server

# Install dependencies  
npm install  
  
# Start development server  
npm run dev  
  
# Open http://localhost:3000 in your browser

### 1.10.2 Build for Production

# Build the application  
npm run build  
  
# Start production server  
npm start

### 1.10.3 Environment Setup

1. Copy environment template:

cp .env.example .env.local

1. Edit .env.local and add your Reality Defender API key:

NEXT\_PUBLIC\_RD\_API\_KEY=your\_actual\_api\_key\_here