MyRefera Frontend Documentation

Table of Contents

Project Overview

MyRefera is a comprehensive influencer marketing platform that connects brands with content creators and influencers. The platform facilitates seamless collaborations, campaign management, and secure payments for influencer marketing campaigns across TikTok, Instagram, and YouTube.

Key Business Features:

- **For Brands**: Discover creators, manage campaigns, track performance, and make secure payments
- **For Creators**: Showcase services, apply for campaigns, manage collaborations, and receive payments
- **For Admins**: Platform management, user oversight, and analytics

Technology Stack

Core Technologies

- **Framework**: Next.js 14.2.5 (App Router)
- **Language**: TypeScript 5.4.5
- **UI Library**: Material-UI (MUI) 5.16.7
- **State Management**: Zustand 4.5.4
- **Data Fetching**: TanStack React Query 5.59.20
- **Styling**: CSS Modules + MUI Theme
- **Authentication**: NextAuth.js 4.24.7

Key Dependencies

```
{    "dependencies": {        "@mui/material": "^5.16.7",        "@mui/icons-material": "^5.16.7",        "@mui/x-data-grid": "^8.2.0",        "@mui/x-charts": "^7.22.0",        "@tanstack/react-query": "^5.59.20",        "axios": "^1.7.5",        "formik": "^2.4.6",        "yup": "^1.4.0",        "zustand": "^4.5.4",        "socket.io-client": "^4.8.0",        "react-hot-toast": "^2.4.1"    }}
```

Development Tools

- **Linting**: ESLint with Next.js config

- **Formatting**: Prettier
- **Type Checking**: TypeScript
- **Bundle Analysis**: @next/bundle-analyzer

Project Structure

```
src/
├── app/                    # Next.js App Router
│   ├── (others)/           # Other pages
│   ├── [role]/             # Role-based routing
│   ├── api/                # API routes (131 files)
│   ├── auth/               # Authentication pages
│   ├── chat/               # Chat functionality
│   ├── go-viral/           # Go Viral feature
│   ├── helpers/            # Utility functions
│   ├── hooks/              # Custom hooks
│   ├── store/              # Global state stores
│   ├── styles/             # Global styles
│   ├── theme/              # MUI theme configuration
│   ├── types/              # TypeScript type definitions
│   └── utils/              # Utility functions
├── components/             # Reusable UI components
│   ├── auth/               # Authentication components
│   ├── campaign/           # Campaign management
│   ├── goViral/            # Go Viral feature components
│   ├── homepage/           # Landing page components
│   ├── landingPageMarch/   # March landing page
│   ├── modal/              # Modal components
│   ├── v2/                 # Version 2 components
│   └── ...                 # Other component categories
├── contexts/               # React contexts
├── features/               # Feature-specific modules
│   └── brands/
│       └── viralLibrary/   # Viral library feature
├── hooks/                  # Custom React hooks
├── models/                # Data models
├── services/              # External service integrations
├── types/                 # TypeScript interfaces
└── utilities/             # Utility functions
```

Key Features

1. User Authentication & Authorization

- **Multi-role system**: Brands, Creators, Admins
- **Authentication methods**: Email/password, Google OAuth
- **Role-based access control**: Different dashboards and permissions
- **Session management**: JWT tokens with 30-day expiration

2. Campaign Management

- **Campaign creation**: Multi-step wizard for brand campaigns
- **Campaign types**: Fixed price, gift for service, subscription, revenue sharing
- **Platform support**: TikTok, Instagram, YouTube
- **Content types**: Reels, Stories, Posts, Videos
- **Campaign lifecycle**: Draft → Published → Active → Completed

3. Marketplace

- **Service listings**: Creators can list their services
- **Service discovery**: Brands can browse and filter creator services
- **Pricing models**: Multiple pricing options per service
- **Request system**: Brands can request specific creator services

4. Messaging System

- **Real-time chat**: Socket.io integration
- **File sharing**: Support for various file types

- **Message types**: Text, images, documents
- **Chat management**: Organized by campaigns and services

5. Payment System

- **Secure payments**: Integrated payment processing
- **Multiple payment methods**: Credit cards, bank transfers
- **Escrow system**: Secure fund holding during campaigns
- **Payment tracking**: Real-time payment status updates

Authentication System

Authentication Flow

1. **Sign Up**: Users choose role (Brand/Creator) and complete registration
2. **Email Verification**: Required for account activation
3. **Sign In**: Email/password or Google OAuth
4. **Role-based Routing**: Automatic redirection based on user type

Key Components

- `src/app/auth/sign-in/page.tsx` - Sign in page
- `src/app/auth/sign-up/page.tsx` - Sign up page
- `src/app/auth/allowAccess.tsx` - Route protection wrapper
- `src/contexts/authDataProvider.tsx` - Authentication context

Authentication Store

```
// src/app/store/store.tsinterface AuthStore {
  isAuthenticated: boolean;    typeUser: 'brand' | 'creator' | 'admin' | null;    us
erId: number | null;    userData: UserData | null;    // ... other auth-related stat
e}
```

Campaign Management

Campaign Creation Process

1. **Step 1**: Basic information (title, description, link)
2. **Step 2**: Target audience and requirements
3. **Step 3**: Content types and platforms
4. **Step 4**: Budget and pricing
5. **Step 5**: Delivery timeline
6. **Step 6**: Additional requirements
7. **Step 7**: Review and publish

Key Components

- `src/components/campaign/brandSteps/` - Campaign creation steps
- `src/components/campaign/BrandCampaignDetails/` - Campaign details
- `src/components/campaign/ManageCampaignPageHeader.tsx` - Campaign management header

Campaign States

- **Draft**: Campaign created but not published
- **Published**: Campaign is live and accepting applications
- **Active**: Campaign has accepted creators
- **Completed**: Campaign finished
- **Cancelled**: Campaign cancelled

Viral Library & Go Viral Feature

Overview

The Viral Library is a powerful feature that allows brands to discover trending content and create AI-powered campaigns based on viral videos.

Key Components

ViralCard Component (Recently Updated)

File: `src/features/brands/viralLibrary/shared/viralCard.tsx`

Recent Changes Made:

- Enhanced transcript viewing functionality
- Added “Tailor to My Brand” button for AI-powered script customization
- Improved modal management for comparison and enhanced transcript views
- Added localStorage integration for transcript storage
- Enhanced user interaction handling with button click prevention

Key Features:

```
interface Props {
    data: ViralVideos;    mode: 'edit' | 'view';    refetch?: () => Promise<any>;    si
ze?: 'small' | 'medium' | 'large';}
```

Functionality:

1. **Video Display**: Shows TikTok video thumbnails with play overlay
2. **Metrics Display**: Views, likes, and engagement rate
3. **Favorites**: Users can favorite videos for later reference
4. **Transcript Viewing**: View video transcripts with enhanced modal
5. **AI Tailoring**: Select videos for AI-powered script customization
6. **Selection Management**: Add/remove videos from campaign selection

Go Viral Page

File: `src/components/goViral/goViralPage.tsx`

Features:

- Multi-step campaign creation wizard
- Viral video discovery and selection
- AI-powered content generation

- Campaign submission and management

Viral Library Store

File: `src/components/goViral/store/goViralStore.ts`

State Management:

```
interface GoViralStore {
    step: number;    modal: string;    viralData: ViralData;    brandData: BrandDat
a;    campaignData: CampaignData;    // ... state management methods}
```

AI Integration

- **Transcript Generation**: Automatic video transcript extraction
- **Script Tailoring**: AI-powered script customization for brands
- **Content Analysis**: Video performance metrics and insights
- **Recommendation Engine**: Suggested videos based on brand preferences

API Integration

API Structure

The application uses a centralized API client with the following structure:

```
// src/app/utils/api.tsconst api = axios.create({
    baseURL: process.env.NEXT_PUBLIC_BACKEND_URL,    // ... configuratio
n});
```

Key API Endpoints

Authentication

- `POST /auth/signin` - User sign in
- `POST /auth/signup` - User registration
- `GET /auth/me` - Get current user data

Campaigns

- `GET /campaigns` - List campaigns
- `POST /campaigns` - Create campaign
- `PATCH /campaigns/:id` - Update campaign
- `DELETE /campaigns/:id` - Delete campaign

Viral Videos

- `GET /viral-videos` - Get viral videos
- `POST /viral-videos/favorite` - Toggle favorite
- `POST /viral-videos/scrape` - Scrape new videos

Services

- `GET /creators/services` - List creator services
- `POST /creators/services` - Create service
- `PATCH /creators/services/:id` - Update service

API Hooks

The application uses custom hooks for API integration:

```
// Example: src/hooks/v2/common/viralVideos.hook.tsexport const useGetViralVideos = () ⇒ {
  return useQuery({
    queryKey: ['viral-videos'],    queryFn: () ⇒ api.get('/viral-videos').then(res ⇒ res.data),  });};
```

Services Layer

External Service Integrations

AI Tailoring Service

File: `src/services/aiTailoringService.ts`

Purpose: Handles AI-powered script customization for viral videos

Key Features:

- **Script Tailoring**: Converts original video transcripts into brand-specific scripts
- **External AI API**: Integrates with `https://script-api.myrefera.com`
- **Mock Data Support**: Development mode with fallback mock responses
- **Advanced Analytics**: Provides psychological analysis and value stacking

API Interface:

```
interface TailoringRequest {
  originalTranscript: string;   productDescription: string;}
interface TailoringResponse {
  tailoredScript: string;   confidence: number;   sectionBreakdown: SectionBr
eakdown[];   sutherlandAlchemy: SutherlandAlchemy;   hormoziValueStack:
HormoziValueStack;}
```

Usage:

```
import aiTailoringService from '@/services/aiTailoringService';const result = a
wait aiTailoringService.tailorScript({
  originalTranscript: 'Hey everyone! This product is amazing!',   productNam
e: 'My Product',   productDescription: 'A revolutionary solution for...',   produc
tUrl: 'https://example.com',   brandTone: 'conversational',   targetAudience:
'young professionals',});
```

Transcript Service

File: `src/services/transcriptService.ts`

Purpose: Handles video transcript extraction from external API

Key Features:

- **Batch Processing**: Transcribe multiple videos simultaneously
- **Language Support**: Multi-language transcript extraction

- **Error Handling**: Robust error handling with fallback mechanisms
- **Health Monitoring**: API connectivity testing

API Interface:

```
interface TranscriptRequest {
   urls: string[];    lang?: string;    text?: boolean;    mode?: string;}
interface TranscriptResponse {
   success: boolean;    data?: {
      transcript: string;       language: string;       duration: number;    };    erro
r?: string;}
```

Usage:

```
import transcriptService from '@/services/transcriptService';// Single videoco
nst result = await transcriptService.getTranscriptForVideo(videoUrl);// Multipl
e videosconst batchResult = await transcriptService.getTranscriptsForVideos
([url1, url2, url3]);
```

Hybrid Storage Service

File: `src/services/hybridStorageService.ts`

Purpose: Provides unified storage interface with database/localStorage fallback

Key Features:

- **Hybrid Storage**: Automatic fallback from database to localStorage
- **Migration Support**: Migrate data from localStorage to database
- **Offline Support**: Works offline with localStorage
- **Data Synchronization**: Syncs data when connection is restored

Configuration:

```
interface HybridStorageConfig {
   useDatabase: boolean;    fallbackToLocalStorage: boolean;    autoMigrate: b
oolean;}
```

Usage:

```
import { hybridStorageService } from '@/services/hybridStorageService';// Save transcriptawait hybridStorageService.saveTranscript(videoUrl, transcriptText);// Get transcriptconst transcript = await hybridStorageService.getTranscript(videoUrl);// Migrate all dataconst results = await hybridStorageService.migrateAllToDatabase();
```

Database Services

Database Transcript Service

File: `src/services/databaseTranscriptService.ts`

Purpose: Handles transcript storage and retrieval from database

Features:

- **CRUD Operations**: Create, read, update, delete transcripts
- **URL-based Lookup**: Find transcripts by video URL
- **Batch Operations**: Handle multiple transcripts efficiently

Database Tailored Script Service

File: `src/services/databaseTailoredScriptService.ts`

Purpose: Manages AI-tailored scripts in database

Features:

- **Script Storage**: Store complete AI tailoring results
- **Metadata Management**: Track confidence, processing time, etc.
- **Search and Filter**: Find scripts by various criteria

Product Scraping Service

File: `src/services/productScrapingService.ts`

Purpose: Extracts product information from URLs

Features:

- **URL Parsing**: Extract product details from e-commerce sites
- **Metadata Extraction**: Get title, description, price, images
- **Error Handling**: Graceful handling of scraping failures

State Management

Zustand Stores

Authentication Store

```
// src/app/store/store.tsinterface AuthStore {
    isAuthenticated: boolean;    typeUser: 'brand' | 'creator' | 'admin' | null;    us
erId: number | null;    userData: UserData | null;    // ... methods}
```

Campaign Creation Store

```
// src/app/store/store.tsinterface CampaignCreationStore {
    title: string;    description: string;    platforms: string[];    budgetOption: Budg
etOption;    // ... other campaign data}
```

Go Viral Store

```
// src/components/goViral/store/goViralStore.tsinterface GoViralStore {
    step: number;    modal: string;    viralData: ViralData;    brandData: BrandDat
a;    campaignData: CampaignData;}
```

React Query Integration

- **Data Fetching**: All API calls use React Query
- **Caching**: Automatic caching and background updates
- **Optimistic Updates**: Immediate UI updates with rollback on error
- **Error Handling**: Centralized error management

UI Components

Component Architecture

The application follows a modular component architecture:

Base Components

- `src/components/base/` - Fundamental UI components
- `src/components/button/` - Button variants
- `src/components/input/` - Form input components
- `src/components/modal/` - Modal components

Feature Components

- `src/components/campaign/` - Campaign-related components
- `src/components/auth/` - Authentication components
- `src/components/goViral/` - Go Viral feature components

Layout Components

- `src/components/header/` - Navigation headers
- `src/components/footer/` - Page footers
- `src/components/sidebar/` - Sidebar navigation

Material-UI Integration

- **Theme System**: Custom MUI theme with brand colors
- **Responsive Design**: Mobile-first approach with breakpoints
- **Component Library**: Extensive use of MUI components
- **Custom Styling**: CSS modules for component-specific styles

Data Storage & Migration

LocalStorage Utilities

File: `src/utilities/localStorage.utils.ts`

Purpose: Provides structured localStorage management for transcripts and tailored scripts

TranscriptStorage Class

```
export class TranscriptStorage {
    static saveTranscript(url: string, transcript: any): boolean;    static getTransc
ript(url: string): any;    static getAllTranscripts(): { [url: string]: any };    static re
moveTranscript(url: string): void;    static clearAll(): void;}
```

Features:

- **Event-driven Updates**: Dispatches custom events for real-time updates
- **Error Handling**: Graceful error handling with console logging
- **Timestamp Tracking**: Automatic timestamp addition for data tracking

TailoredScriptStorage Class

```
export class TailoredScriptStorage {
    static saveTailoredScript(url: string, scriptData: any): boolean;    static getTa
iloredScript(url: string): any;    static getAllTailoredScripts(): { [url: string]: any
};    static removeTailoredScript(url: string): boolean;    static clearAllTailoredS
cripts(): boolean;}
```

Features:

- **Structured Data**: Organized storage for AI tailoring results
- **Metadata Tracking**: Stores generation timestamps and video URLs
- **Event System**: Real-time updates across components

Migration System

File: `src/hooks/useMigrationCheck.ts`

Purpose: Handles migration of localStorage data to database

Key Features:

- **Data Detection**: Automatically detects existing localStorage data
- **Migration Prompting**: Shows migration modal when data is found
- **Progress Tracking**: Tracks migration success/failure rates
- **User Control**: Allows users to choose when to migrate

Usage:

```
import { useMigrationCheck } from '@/hooks/useMigrationCheck';const { has
LocalData, showMigrationModal, localDataCounts, closeMigrationModal, trigg
erMigrationModal } =    useMigrationCheck();
```

Migration Process:

1. **Detection**: Check for existing localStorage data
2. **Prompting**: Show migration modal if data exists
3. **Migration**: Transfer data from localStorage to database
4. **Cleanup**: Remove localStorage data after successful migration
5. **Verification**: Confirm migration completion

Database Integration

Database Schema

Transcripts Table

Purpose: Stores video transcripts with metadata

Schema:

```
CREATE TABLE transcripts (
    id SERIAL PRIMARY KEY,
    video_url VARCHAR(500) UNIQUE NOT NULL,
    video_title VARCHAR(255),
    platform VARCHAR(50) DEFAULT 'tiktok',
    transcript_text TEXT NOT NULL,
    language VARCHAR(10) DEFAULT 'en',
    duration INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE);
```

Key Features:

- **URL-based Lookup**: Unique constraint on video_url for fast retrieval
- **Multi-platform Support**: Platform field for TikTok, Instagram, YouTube
- **Language Support**: Multi-language transcript storage
- **User Association**: Links transcripts to specific users
- **Audit Trail**: Created/updated timestamps

Tailored Scripts Table

Purpose: Stores AI-generated tailored scripts with complete metadata

Schema:

```
CREATE TABLE tailored_scripts (
    id SERIAL PRIMARY KEY,
    video_url VARCHAR(500) NOT NULL,
    product_name VARCHAR(255) NOT NULL,
    product_description TEXT,
    product_url VARCHAR(500),
    brand_tone VARCHAR(50),
    target_audience TEXT,
    tailored_script TEXT NOT NULL,
    section_breakdown JSONB,
    sutherland_alchemy JSONB,
    hormozi_value_stack JSONB,
    confidence DECIMAL(3,2),
    processing_time DECIMAL(5,2),
    word_count INTEGER,
    estimated_read_time VARCHAR(20),
    api_version VARCHAR(50),
    model_used VARCHAR(100),
    improvement_areas TEXT[],
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (video_url) REFERENCES transcripts(video_url) ON DELETE C
ASCADE);
```

Key Features:

- **Complete AI Results**: Stores all AI tailoring output including psychological analysis
- **JSONB Storage**: Efficient storage of complex nested data structures
- **Performance Metrics**: Tracks confidence, processing time, word count
- **Version Tracking**: API version and model used for reproducibility
- **Foreign Key Relations**: Links to transcripts table for data integrity

Database Services Integration

Database Transcript Service

File: `src/services/databaseTranscriptService.ts`

API Endpoints:

- `POST /api/transcripts` - Save transcript
- `GET /api/transcripts/:videoUrl` - Get transcript by URL
- `PUT /api/transcripts/:id` - Update transcript
- `DELETE /api/transcripts/:id` - Delete transcript

Key Methods:

```
interface DatabaseTranscriptService {
    saveTranscript(data: TranscriptData): Promise<ApiResponse>;    getTranscr
iptByUrl(videoUrl: string): Promise<ApiResponse>;    updateTranscript(id: num
ber, data: Partial<TranscriptData>): Promise<ApiResponse>;    deleteTranscrip
t(id: number): Promise<ApiResponse>;    getUserTranscripts(userId: number):
Promise<ApiResponse>;}
```

Database Tailored Script Service

File: `src/services/databaseTailoredScriptService.ts`

API Endpoints:

- `POST /api/tailored-scripts` - Save tailored script
- `GET /api/tailored-scripts/:videoUrl` - Get script by video URL
- `GET /api/tailored-scripts/user/:userId` - Get user's scripts
- `PUT /api/tailored-scripts/:id` - Update script
- `DELETE /api/tailored-scripts/:id` - Delete script

Key Methods:

```
interface DatabaseTailoredScriptService {
    saveTailoredScript(data: TailoredScriptData): Promise<ApiResponse>;    get
TailoredScriptByUrl(videoUrl: string): Promise<ApiResponse>;    getUserTailor
edScripts(userId: number): Promise<ApiResponse>;    updateTailoredScript(i
d: number, data: Partial<TailoredScriptData>): Promise<ApiResponse>;    dele
teTailoredScript(id: number): Promise<ApiResponse>;}
```

Data Flow Architecture

Transcript Generation Flow

1. **User Action**: User clicks "View Transcript" on viral video
2. **Local Check**: Check localStorage for existing transcript
3. **Database Check**: Query database for transcript
4. **API Generation**: If not found, call external transcript API
5. **Storage**: Save to both database and localStorage

6. **Display**: Show transcript in modal

AI Tailoring Flow

1. **User Action**: User clicks "Tailor to My Brand"
2. **Data Collection**: Gather product info and original transcript
3. **AI Processing**: Send to external AI API for script generation
4. **Result Processing**: Parse and format AI response
5. **Storage**: Save complete result to database and localStorage
6. **Display**: Show tailored script in comparison modal

Migration Flow

1. **Detection**: Check for localStorage data on app load
2. **User Prompt**: Show migration modal if data exists
3. **Batch Migration**: Transfer all localStorage data to database
4. **Verification**: Confirm successful migration
5. **Cleanup**: Remove localStorage data after successful migration
6. **Fallback**: Keep localStorage as backup if database fails

Error Handling & Fallback Strategy

Database Unavailable

- **Automatic Fallback**: Switch to localStorage when database is down
- **Retry Logic**: Attempt to reconnect to database periodically
- **User Notification**: Inform users about offline mode
- **Data Sync**: Sync localStorage data when database comes back online

API Failures

- **Graceful Degradation**: Show cached data when APIs fail
- **User Feedback**: Clear error messages with retry options

- **Fallback Data**: Use mock data in development mode
- **Retry Mechanisms**: Automatic retry with exponential backoff

Performance Optimizations

Caching Strategy

- **React Query**: Automatic caching of API responses
- **LocalStorage**: Persistent caching for offline access
- **Database Indexing**: Optimized queries with proper indexes
- **CDN Integration**: Static assets served from CDN

Data Loading

- **Lazy Loading**: Load transcripts and scripts on demand
- **Batch Operations**: Process multiple items simultaneously
- **Pagination**: Handle large datasets efficiently
- **Background Sync**: Sync data in background when possible

AI Tailoring System

AI Service Integration

External AI API

Endpoint: `https://script-api.myrefera.com`

Key Features:

- **Advanced Psychology**: Uses Sutherland Alchemy and Hormozi Value Stack principles
- **Script Analysis**: Breaks down scripts into psychological sections
- **Value Reframing**: Transforms product features into emotional benefits
- **Confidence Scoring**: Provides confidence metrics for generated content

AI Request/Response Structure

Request Format:

```
interface TailoringRequest {
    originalTranscript: string;    productDescription: string;}
```

Response Format:

```
interface TailoringResponse {
    tailoredScript: string;    confidence: number;    processingTime: number;    wordCount: number;    estimatedReadTime: string;    sectionBreakdown: SectionBreakdown[];    sutherlandAlchemy: SutherlandAlchemy;    hormoziValueStack: HormoziValueStack;}
```

Section Breakdown Analysis

```
interface SectionBreakdown {
    sectionName: string; // "Hook", "Transformation Story", "CTA"    triggerEmotionalState: string; // "Curiosity Gap + Social Proof"    originalQuote: string; // Original transcript text    rewrittenVersion: string; // AI-rewritten version    sceneDescription: string; // Visual direction for filming    psychologicalPrinciples: string[]; // Applied psychology principles    timestamp: string; // Video timestamp}
```

Sutherland Alchemy (Value Reframing)

```
interface SutherlandAlchemy {
    explanation: string; // How the reframing works    valueReframing: ValueReframing[]; // Specific reframing examples    identityShifts: string[]; // Identity transformations}
interface ValueReframing {
    original: string; // Original framing    reframed: string; // New framing    psychologyBehind: string; // Psychological explanation}
```

Hormozi Value Stack

```
interface HormoziValueStack {
    coreOffer: string; // Main product offer    valueElements: ValueElement[]; // I
ndividual value components    totalStack: {
        totalPerceivedValue: string; // Total perceived value        actualPrice: strin
g; // Actual price        valueMultiplier: string; // Value multiplier (e.g., "7x+ valu
e")    };    grandSlamElements: string[]; // High-impact value elements}
```

AI Processing Pipeline

1. Input Validation

- **Transcript Validation**: Ensure transcript is valid and sufficient length
- **Product Info Validation**: Validate product name, description, URL
- **Brand Tone Validation**: Ensure brand tone is from allowed values
- **Target Audience Validation**: Validate audience description length

2. AI Processing

- **Script Analysis**: Break down original transcript into sections
- **Psychology Application**: Apply psychological principles to each section
- **Value Stacking**: Build comprehensive value proposition
- **Reframing**: Transform features into emotional benefits

3. Output Processing

- **Timestamp Formatting**: Convert AI output to proper timestamp format
- **Quality Scoring**: Calculate confidence and processing metrics
- **Metadata Generation**: Generate word count, read time, etc.
- **Error Handling**: Handle API failures gracefully

4. Storage & Caching

- **Database Storage**: Save complete AI results to database
- **LocalStorage Backup**: Cache results locally for offline access
- **Event Dispatch**: Notify components of new data
- **Migration Support**: Support migration from localStorage to database

Development & Testing

Mock Data Support

```
// Development mode with mock dataconst useMockData = process.env.NODE_ENV === 'development' && process.env.NEXT_PUBLIC_USE_MOCK_AI === 'true';
```

Mock Response Features:

- **Realistic Data**: Mock responses match real API structure
- **Configurable Delays**: Simulate API processing time
- **Error Simulation**: Test error handling scenarios
- **Development Fallback**: Automatic fallback when API unavailable

Debug Tools

```
// Console debugging helperswindow.debugAITailoring = () ⇒ aiTailoringService.debugConnectivity();window.clearTailoredScripts = () ⇒ aiTailoringService.clearStoredTailoredScripts();
```

Debug Features:

- **Connectivity Testing**: Test API endpoint availability
- **Request/Response Logging**: Detailed logging of API calls
- **Error Analysis**: Comprehensive error reporting
- **Performance Monitoring**: Track processing times and success rates

User Experience Features

Loading States

- **Processing Indicators**: Show loading states during AI processing
- **Progress Tracking**: Display processing progress when possible
- **Timeout Handling**: Handle long-running requests gracefully
- **Retry Options**: Allow users to retry failed requests

Error Handling

- **User-Friendly Messages**: Clear, actionable error messages
- **Fallback Options**: Offer alternatives when AI fails
- **Offline Support**: Work with cached data when offline
- **Recovery Suggestions**: Suggest solutions for common issues

Performance Optimization

- **Request Debouncing**: Prevent duplicate requests
- **Caching Strategy**: Cache results to avoid redundant API calls
- **Background Processing**: Process requests in background when possible
- **Progressive Enhancement**: Enhance experience when AI is available

Recent Changes

ViralCard Component Updates

File: `src/features/brands/viralLibrary/shared/viralCard.tsx`

Changes Made:

1. **Enhanced Transcript Functionality**:
    - Added `handleViewTranscript()` function for viewing video transcripts
    - Integrated with `EnhancedTranscriptModal` component
    - Added localStorage integration for transcript storage
2. **AI Tailoring Integration**:

    - Added `handleTailorScript()` function for AI-powered script customization
    - Integrated with `ComparisonModal` for script comparison
    - Added state management for tailoring process
3. **Improved User Interaction**:
    - Added `isButtonClicked` state to prevent unwanted video clicks
    - Enhanced event handling with proper event prevention
    - Added loading states for better UX
4. **Modal Management**:
    - Added state for `showComparisonModal` and `showEnhancedTranscriptModal`
    - Integrated with existing modal system
    - Added proper cleanup and state reset
5. **Transcript Storage**:
    - Added `transcriptions` state for localStorage integration
    - Added event listeners for transcript updates
    - Implemented `getTranscriptText()` helper function

Key Features Added:

- **View Transcript Button**: Allows users to view video transcripts
- **Tailor to My Brand Button**: Enables AI-powered script customization
- **Enhanced Modal System**: Better modal management and user experience
- **Local Storage Integration**: Persistent transcript storage

Services Integration

Recent additions to the services layer:

1. **AI Tailoring Service**: Complete integration with external AI API
2. **Hybrid Storage Service**: Unified storage with database/localStorage fallback
3. **Migration System**: Automatic data migration from localStorage to database

4. **Enhanced Error Handling**: Robust error handling across all services
5. **Offline Support**: Full offline functionality with localStorage fallback