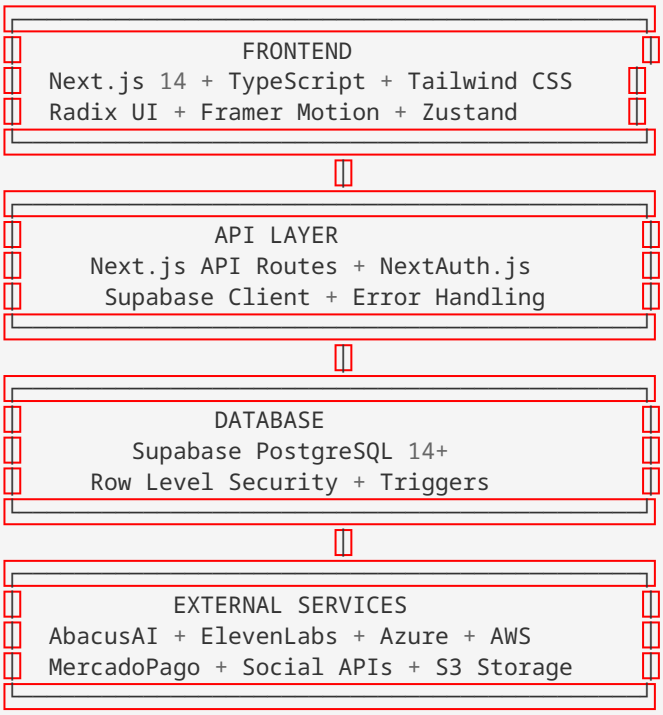
🛠️ MANUAL TÉCNICO - VIRA

Documentación técnica completa para desarrolladores y administradores del sistema.

🏗️ ARQUITECTURA DEL SISTEMA

Stack Tecnológico



Estructura de Directorios

```
VIRA/
├── 📁 app/                    # Next.js 14 App Router
│   ├── 📁 (auth)/             # Rutas de autenticación
│   ├── 📁 api/                # API endpoints
│   │   ├── 📄 generate-newscast/route.ts
│   │   ├── 📄 templates/route.ts
│   │   ├── 📄 reports/route.ts
│   │   └── 📄 auth/[...nextauth]/route.ts
│   ├── 📁 dashboard/          # Panel principal
│   ├── 📁 crear-noticiero/    # Generador de contenido
│   ├── 📁 timeline-noticiero/ # Editor interactivo
│   ├── 📁 automatizacion/     # Programación
│   └── 📄 layout.tsx          # Layout principal
├── 📁 components/             # Componentes React
│   ├── 📁 ui/                 # Componentes base (Radix)
│   ├── 📁 dashboard/          # Componentes específicos
│   └── 📁 forms/              # Formularios reutilizables
├── 📁 lib/                    # Utilidades y configuración
│   ├── 📄 supabase.ts         # Cliente y funciones DB
│   ├── 📄 auth.ts             # Configuración NextAuth
│   ├── 📄 aws-config.ts       # Configuración S3
│   ├── 📄 ai-services.ts      # Integración IA
│   └── 📄 utils.ts            # Utilidades generales
├── 📁 database/               # Schema y migraciones
│   └── 📄 supabase_schema.sql
├── 📁 types/                  # Definiciones TypeScript
└── 📁 hooks/                  # Custom React hooks
```

🔧 CONFIGURACIÓN TÉCNICA DETALLADA

Variables de Entorno

```
# === CORE SYSTEM ===
NODE_ENV=production|development
NEXTAUTH_SECRET="clave_segura_256_bits"
NEXTAUTH_URL=https://tu-dominio.com

# === DATABASE (Supabase) ===
NEXT_PUBLIC_SUPABASE_URL=https://proyecto.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
SUPABASE_SERVICE_ROLE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

# === AI SERVICES ===
ABACUSAI_API_KEY="tu_api_key_abacus"
OPENAI_API_KEY="sk-..."                    # Opcional
ANTHROPIC_API_KEY="sk-ant-api-..."         # Opcional

# === TEXT-TO-SPEECH ===
ELEVENLABS_API_KEY="tu_elevenlabs_key"
ELEVENLABS_BASE_URL=https://api.elevenlabs.io

AZURE_SPEECH_KEY="tu_azure_key"
AZURE_SPEECH_REGION=eastus

AWS_ACCESS_KEY_ID="AKIA..."
AWS_SECRET_ACCESS_KEY="tu_secret_key"
AWS_REGION=us-east-1

# === STORAGE ===
AWS_BUCKET_NAME=vira-audio-prod
AWS_FOLDER_PREFIX=production/
S3_CDN_URL=https://d1234567890.cloudfront.net

# === NEWS SCRAPING ===
EMOL_RSS_URL=https://www.emol.com/rss/rss.asp
LATERCERA_RSS_URL=https://www.latercera.com/feed/
BIOBIO_RSS_URL=https://www.biobiochile.cl/especial/rss/index.xml
OPENWEATHER_API_KEY="tu_openweather_key"

# === SOCIAL INTEGRATIONS ===
TWITTER_BEARER_TOKEN="Bearer tu_token"
TWITTER_API_KEY="tu_api_key"
TWITTER_API_SECRET="tu_api_secret"

FACEBOOK_APP_ID="123456789"
FACEBOOK_APP_SECRET="tu_app_secret"

INSTAGRAM_APP_ID="123456789"
INSTAGRAM_APP_SECRET="tu_app_secret"

SPOTIFY_CLIENT_ID="tu_client_id"
SPOTIFY_CLIENT_SECRET="tu_client_secret"

# === PAYMENTS ===
MERCADOPAGO_ACCESS_TOKEN="APP_USR-123..."
MERCADOPAGO_PUBLIC_KEY="APP_USR-123..."
MERCADOPAGO_CLIENT_ID="123456789"
MERCADOPAGO_CLIENT_SECRET="tu_client_secret"
MERCADOPAGO_WEBHOOK_SECRET="tu_webhook_secret"
MERCADOPAGO_ENVIRONMENT=sandbox|production

# === NOTIFICATIONS ===
SENDGRID_API_KEY="SG.tu_sendgrid_key"
SENDGRID_FROM_EMAIL="noreply@tu-dominio.com"
```

```
TWILIO_ACCOUNT_SID="AC123456789"
TWILIO_AUTH_TOKEN="tu_auth_token"

# === MONITORING ===
SENTRY_DSN="https://tu_sentry_dsn.ingest.sentry.io/..."
GOOGLE_ANALYTICS_ID="GA-123456789"
```

💾 BASE DE DATOS SUPABASE

Tablas Principales

👥 Autenticación

```sql
-- Usuarios del sistema
CREATE TABLE users (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    name VARCHAR(255),
    image TEXT,
    role VARCHAR(50) DEFAULT 'user', -- 'user', 'admin', 'premium'
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Cuentas OAuth vinculadas
CREATE TABLE accounts (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    type VARCHAR(255) NOT NULL,
    provider VARCHAR(255) NOT NULL,
    provider_account_id VARCHAR(255) NOT NULL,
    refresh_token TEXT,
    access_token TEXT,
    expires_at INTEGER,
    token_type VARCHAR(255),
    scope VARCHAR(255),
    id_token TEXT,
    session_state VARCHAR(255),
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(provider, provider_account_id)
);

-- Sesiones activas
CREATE TABLE sessions (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    session_token VARCHAR(255) UNIQUE NOT NULL,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    expires TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Contenido

```sql
-- Plantillas de noticieros reutilizables
CREATE TABLE newscast_templates (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    region VARCHAR(100),
    duration_minutes INTEGER DEFAULT 15,
    categories JSONB DEFAULT '[]',
    ai_profile VARCHAR(50) DEFAULT 'balanced', -- 'economic', 'balanced', 'premium',
'custom'
    ai_settings JSONB DEFAULT '{}',
    ad_phrases_count INTEGER DEFAULT 5,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Reportes/noticieros generados
CREATE TABLE news_reports (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    template_id UUID REFERENCES newscast_templates(id) ON DELETE SET NULL,
    title VARCHAR(500) NOT NULL,
    region VARCHAR(100),
    total_duration_seconds INTEGER DEFAULT 0,

    -- Contenido y timeline
    timeline_data JSONB DEFAULT '[]',
    final_audio_url TEXT,
    transcript TEXT,

    -- Estado y procesamiento
    status VARCHAR(50) DEFAULT 'generating', -- 'generating', 'completed', 'failed',
'archived'
    generation_started_at TIMESTAMP DEFAULT NOW(),
    generation_completed_at TIMESTAMP,
    error_message TEXT,

    -- Métricas y costos
    total_cost DECIMAL(10,4) DEFAULT 0.00,
    tokens_used JSONB DEFAULT '{}', -- {"openai": 1500, "elevenlabs": 5000}
    processing_time_seconds INTEGER DEFAULT 0,

    -- Configuración usada
    ai_settings_used JSONB DEFAULT '{}',
    news_sources_used JSONB DEFAULT '[]',
    categories_used JSONB DEFAULT '[]',

    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Noticias extraídas automáticamente
CREATE TABLE scraped_news (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    source_name VARCHAR(255) NOT NULL,
    source_url TEXT NOT NULL,
    title VARCHAR(500) NOT NULL,
    content TEXT NOT NULL,
    original_url TEXT,
    region VARCHAR(100),
    category VARCHAR(100),
```

```
    keywords JSONB DEFAULT '[]',
    scraped_at TIMESTAMP DEFAULT NOW(),
    is_processed BOOLEAN DEFAULT false,

    -- Versiones de contenido
    original_content TEXT,
    rewritten_content TEXT,
    humanized_content TEXT,

    -- Metadata
    word_count INTEGER,
    reading_time_seconds INTEGER,
    sentiment_score DECIMAL(3,2), -- -1 to 1

    UNIQUE(source_url, title)
);
```

🎵 Multimedia

```
-- Biblioteca de audio del usuario
CREATE TABLE audio_library (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    type VARCHAR(50) NOT NULL, -- 'music', 'sfx', 'ad', 'voice'
    s3_key VARCHAR(500) NOT NULL,
    file_url TEXT NOT NULL,
    duration_seconds INTEGER,
    file_size_bytes BIGINT,
    mime_type VARCHAR(100),

    -- Metadata
    tags JSONB DEFAULT '[]',
    is_favorite BOOLEAN DEFAULT false,
    usage_count INTEGER DEFAULT 0,

    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Voces sintéticas entrenadas/clonadas
CREATE TABLE cloned_voices (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    provider VARCHAR(50) NOT NULL, -- 'elevenlabs', 'azure', 'aws'
    voice_id VARCHAR(255) NOT NULL,
    voice_settings JSONB DEFAULT '{}',
    sample_audio_url TEXT,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT NOW()
);
```

⚙️ Configuración y Automatización

```sql
-- Fuentes de noticias configuradas
CREATE TABLE news_sources (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE, -- NULL para globales
    name VARCHAR(255) NOT NULL,
    url TEXT NOT NULL,
    rss_url TEXT,
    region VARCHAR(100),
    category VARCHAR(100),
    is_active BOOLEAN DEFAULT true,
    is_global BOOLEAN DEFAULT false, -- Disponible para todos los usuarios
    last_scraped_at TIMESTAMP,
    scraping_frequency_minutes INTEGER DEFAULT 60,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Trabajos de automatización programados
CREATE TABLE automation_jobs (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    template_id UUID REFERENCES newscast_templates(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,

    -- Programación
    frequency VARCHAR(50) NOT NULL, -- 'daily', 'weekly', 'monthly', 'specific_days'
    schedule_data JSONB NOT NULL, -- días específicos, horarios, etc.
    timezone VARCHAR(50) DEFAULT 'America/Santiago',

    -- Estado
    is_active BOOLEAN DEFAULT true,
    next_run_at TIMESTAMP,
    last_run_at TIMESTAMP,
    last_run_status VARCHAR(50), -- 'success', 'failed', 'running'
    last_run_report_id UUID REFERENCES news_reports(id),

    -- Configuración
    auto_publish_settings JSONB DEFAULT '{}',
    notification_settings JSONB DEFAULT '{}',

    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Integraciones con redes sociales
CREATE TABLE social_integrations (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    platform VARCHAR(50) NOT NULL, -- 'twitter', 'facebook', 'instagram', 'spotify'

    -- Credenciales (encriptadas)
    access_token TEXT NOT NULL,
    refresh_token TEXT,
    expires_at TIMESTAMP,

    -- Configuración
    auto_publish_enabled BOOLEAN DEFAULT false,
    publish_settings JSONB DEFAULT '{}',

    -- Estado
    is_active BOOLEAN DEFAULT true,
    last_published_at TIMESTAMP,
```

```
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(user_id, platform)
);
```

📊 Métricas y Monitoreo

```
-- Uso detallado de tokens y costos
CREATE TABLE token_usage (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    report_id UUID REFERENCES news_reports(id) ON DELETE CASCADE,

    -- Servicio usado
    service VARCHAR(50) NOT NULL, -- 'openai', 'elevenlabs', 'azure', 'aws'
    operation VARCHAR(100) NOT NULL, -- 'text-generation', 'text-to-speech', 'tran-
scription'
    model_name VARCHAR(100),

    -- Consumo y costo
    tokens_used INTEGER DEFAULT 0,
    characters_processed INTEGER DEFAULT 0,
    seconds_audio DECIMAL(10,2) DEFAULT 0,
    cost_usd DECIMAL(10,6) DEFAULT 0,

    -- Metadata
    request_duration_ms INTEGER,
    response_size_bytes INTEGER,

    created_at TIMESTAMP DEFAULT NOW()
);

-- Métricas diarias agregadas del sistema
CREATE TABLE daily_metrics (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    date DATE UNIQUE NOT NULL,

    -- Métricas de actividad
    total_users_active INTEGER DEFAULT 0,
    total_reports_generated INTEGER DEFAULT 0,
    total_audio_minutes DECIMAL(10,2) DEFAULT 0,

    -- Métricas de costo
    total_cost_usd DECIMAL(10,4) DEFAULT 0,
    avg_cost_per_report DECIMAL(10,4) DEFAULT 0,
    cost_by_service JSONB DEFAULT '{}',

    -- Métricas de performance
    avg_generation_time_seconds INTEGER DEFAULT 0,
    success_rate_percentage DECIMAL(5,2) DEFAULT 100,

    -- Contenido
    most_popular_region VARCHAR(100),
    most_popular_category VARCHAR(100),

    created_at TIMESTAMP DEFAULT NOW()
);
```

Funciones SQL Útiles

```sql
-- Obtener costo total de un usuario
CREATE OR REPLACE FUNCTION get_user_total_cost(user_uuid UUID)
RETURNS DECIMAL(10,4) AS $$
BEGIN
    RETURN (
        SELECT COALESCE(SUM(total_cost), 0)
        FROM news_reports
        WHERE user_id = user_uuid
          AND status = 'completed'
    );
END;
$$ LANGUAGE plpgsql;

-- Obtener noticias por región
CREATE OR REPLACE FUNCTION get_news_by_region(region_name TEXT, limit_count INTEGER DE-
FAULT 10)
RETURNS SETOF scraped_news AS $$
BEGIN
    RETURN QUERY
    SELECT *
    FROM scraped_news
    WHERE region = region_name
      AND is_processed = false
    ORDER BY scraped_at DESC
    LIMIT limit_count;
END;
$$ LANGUAGE plpgsql;

-- Actualizar métricas diarias (trigger automático)
CREATE OR REPLACE FUNCTION update_daily_metrics()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO daily_metrics (date, total_reports_generated, total_cost_usd)
    VALUES (
        CURRENT_DATE,
        (SELECT COUNT(*) FROM news_reports WHERE DATE(created_at) = CURRENT_DATE),
        (SELECT COALESCE(SUM(total_cost), 0) FROM news_reports WHERE DATE(created_at)
= CURRENT_DATE)
    )
    ON CONFLICT (date) DO UPDATE SET
        total_reports_generated = EXCLUDED.total_reports_generated,
        total_cost_usd = EXCLUDED.total_cost_usd;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger para ejecutar función automáticamente
CREATE TRIGGER trigger_update_daily_metrics
    AFTER INSERT OR UPDATE ON news_reports
    FOR EACH ROW
    EXECUTE FUNCTION update_daily_metrics();
```

Vistas Útiles

```sql
-- Vista consolidada de estadísticas por usuario
CREATE VIEW user_stats AS
SELECT
    u.id,
    u.email,
    u.name,
    COUNT(nr.id) as total_reports,
    COALESCE(SUM(nr.total_cost), 0) as total_cost_usd,
    COALESCE(AVG(nr.processing_time_seconds), 0) as avg_processing_time,
    MAX(nr.created_at) as last_report_date,
    COUNT(CASE WHEN nr.status = 'completed' THEN 1 END) as successful_reports,
    COUNT(CASE WHEN nr.status = 'failed' THEN 1 END) as failed_reports
FROM users u
LEFT JOIN news_reports nr ON u.id = nr.user_id
GROUP BY u.id, u.email, u.name;

-- Vista de noticias recientes con metadatos
CREATE VIEW recent_news AS
SELECT
    sn.*,
    ns.name as source_display_name,
    ns.category as source_category,
    DATE_PART('hour', AGE(NOW(), sn.scraped_at)) as hours_since_scraped
FROM scraped_news sn
LEFT JOIN news_sources ns ON sn.source_name = ns.name
WHERE sn.scraped_at > (NOW() - INTERVAL '24 hours')
ORDER BY sn.scraped_at DESC;

-- Vista de métricas del sistema
CREATE VIEW system_metrics AS
SELECT
    date,
    total_users_active,
    total_reports_generated,
    total_audio_minutes,
    total_cost_usd,
    CASE
        WHEN total_reports_generated > 0
        THEN total_cost_usd / total_reports_generated
        ELSE 0
    END as cost_per_report,
    success_rate_percentage,
    most_popular_region,
    most_popular_category
FROM daily_metrics
ORDER BY date DESC;
```

Row Level Security (RLS)

```sql
-- Habilitar RLS en todas las tablas de usuario
ALTER TABLE newscast_templates ENABLE ROW LEVEL SECURITY;
ALTER TABLE news_reports ENABLE ROW LEVEL SECURITY;
ALTER TABLE audio_library ENABLE ROW LEVEL SECURITY;
ALTER TABLE automation_jobs ENABLE ROW LEVEL SECURITY;
ALTER TABLE social_integrations ENABLE ROW LEVEL SECURITY;
ALTER TABLE token_usage ENABLE ROW LEVEL SECURITY;

-- Política: usuarios solo ven sus propios datos
CREATE POLICY "Users can only see their own templates" ON newscast_templates
    FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can only see their own reports" ON news_reports
    FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can only see their own audio" ON audio_library
    FOR ALL USING (auth.uid() = user_id);

-- Política para administradores (acceso completo)
CREATE POLICY "Admins have full access" ON news_reports
    FOR ALL USING (
        EXISTS (
            SELECT 1 FROM users
            WHERE id = auth.uid()
            AND role = 'admin'
        )
    );
```

🔌 API ENDPOINTS DETALLADAS

Autenticación

POST `/api/auth/[...nextauth]`

Manejado por NextAuth.js automáticamente.

Providers configurados:
- Google OAuth
- GitHub OAuth
- Email/Password (opcional)

GET `/api/auth/session`

```
// Respuesta
{
  user: {
    id: "uuid",
    name: "Juan Pérez",
    email: "juan@example.com",
    image: "https://pbs.twimg.com/profile_images/1886584693795020800/zFrfh-
hgp_200x200.jpg",
    role: "user"
  },
  expires: "2024-01-01T00:00:00.000Z"
}
```

Plantillas de Noticieros

GET `/api/templates`

```
// Query params opcionales
?limit=20&offset=0

// Respuesta
{
  templates: [
    {
      id: "uuid",
      name: "Noticiero Matutino",
      region: "Santiago",
      duration_minutes: 15,
      categories: ["política", "economía"],
      ai_profile: "balanced",
      created_at: "2024-01-01T00:00:00.000Z"
    }
  ],
  total: 5,
  hasMore: false
}
```

POST `/api/templates`

```
// Request
{
  name: "Mi Plantilla",
  region: "Santiago",
  duration_minutes: 30,
  categories: ["política", "economía", "deportes"],
  ai_profile: "premium",
  ai_settings: {
    rewrite_model: "gpt-4-turbo",
    voice_provider: "elevenlabs",
    voice_id: "voice_123"
  },
  ad_phrases_count: 8
}

// Respuesta
{
  success: true,
  template: { /* plantilla creada */ }
}
```

PUT `/api/templates/[id]`

Actualizar plantilla existente.

DELETE `/api/templates/[id]`

Eliminar plantilla.

Generación de Noticieros

POST `/api/generate-newscast`

```
// Request
{
  title?: "Noticiero personalizado",
  template_id?: "uuid", // Opcional, usa plantilla existente
  region: "Santiago",
  duration_minutes: 15,
  categories: ["política", "economía"],
  ai_profile: "balanced",
  ai_settings?: {
    extraction_model: "gpt-3.5-turbo",
    rewrite_model: "gpt-4-turbo",
    humanization_model: "claude-3-sonnet",
    voice_provider: "elevenlabs",
    voice_id: "voice_123"
  },
  ad_phrases_count: 5,
  ad_phrases?: ["Frase publicitaria 1", "Frase 2"],
  auto_publish?: {
    twitter: true,
    facebook: false
  }
}

// Respuesta inmediata
{
  success: true,
  report_id: "uuid",
  estimated_completion: "2024-01-01T00:05:00.000Z",
  estimated_cost: 2.45
}

// El proceso continúa en background
// Estado se puede consultar via GET /api/reports/[id]
```

Flujo interno:

1. Validar parámetros y permisos usuario
2. Crear registro en `news_reports` con status “generating”
3. Iniciar proceso asíncrono:

- Scrapear noticias de fuentes RSS
- Extraer y categorizar contenido
- Reescribir con modelo de IA seleccionado
- Humanizar texto si está configurado
- Generar audio con TTS
- Crear timeline interactivo
- Subir archivos a S3
- Actualizar registro con resultado final

Reportes de Noticieros

GET `/api/reports`

```
// Query params opcionales
?limit=20&offset=0&status=completed&region=Santiago

// Respuesta
{
  reports: [
    {
      id: "uuid",
      title: "Noticiero Santiago - 8 Sep 2024",
      region: "Santiago",
      status: "completed",
      total_duration_seconds: 900,
      final_audio_url: "https://s3.../final_audio.mp3",
      total_cost: 2.45,
      tokens_used: {
        "openai": 1500,
        "elevenlabs": 5000
      },
      generation_completed_at: "2024-01-01T00:05:00.000Z",
      created_at: "2024-01-01T00:00:00.000Z"
    }
  ],
  total: 15,
  hasMore: true
}
```

GET `/api/reports/[id]`

```
// Respuesta
{
  id: "uuid",
  title: "Noticiero Santiago - 8 Sep 2024",
  region: "Santiago",
  status: "completed",
  total_duration_seconds: 900,

  // Timeline completo con todas las noticias
  timeline_data: [
    {
      id: "news_1",
      type: "news",
      title: "Noticia política importante",
      original_content: "Texto original extraído",
      rewritten_content: "Texto reescrito por IA",
      humanized_content: "Texto humanizado final",
      audio_url: "https://s3.../news_1_audio.mp3",
      duration_seconds: 45,
      category: "política",
      order_index: 0,
      tokens_used: {
        "rewrite": 150,
        "tts": 800
      },
      cost: 0.25
    },
    {
      id: "ad_1",
      type: "advertisement",
      content: "Frase publicitaria 1",
      audio_url: "https://s3.../ad_1.mp3",
      duration_seconds: 15,
      order_index: 1
    }
  ],

  // Metadatos
  final_audio_url: "https://s3.../final_newscast.mp3",
  transcript: "Transcripción completa...",
  total_cost: 2.45,
  tokens_used: { /*...*/ },
  processing_time_seconds: 180,
  ai_settings_used: { /*...*/ },
  created_at: "2024-01-01T00:00:00.000Z"
}
```

PUT `/api/reports/[id]`

```
// Request - Actualizar timeline
{
  timeline_data: [
    {
      id: "news_1",
      title: "Título editado",
      rewritten_content: "Contenido editado manualmente",
      regenerate_audio: true
    }
  ]
}

// Respuesta
{
  success: true,
  updated_items: ["news_1"],
  new_total_duration: 920,
  additional_cost: 0.15
}
```

Fuentes de Noticias

GET `/api/news-sources`

```
// Query params
?region=Santiago&userOnly=false

// Respuesta
{
  sources: [
    {
      id: "uuid",
      name: "Emol",
      url: "https://www.emol.com",
      rss_url: "https://www.emol.com/rss/rss.asp",
      region: "Nacional",
      category: "general",
      is_active: true,
      is_global: true,
      last_scraped_at: "2024-01-01T00:00:00.000Z"
    }
  ]
}
```

POST `/api/news-sources`

Agregar nueva fuente de noticias.

Biblioteca de Audio

GET `/api/audio-library`

```
// Query params
?type=music&tags=cortina&limit=20

// Respuesta
{
  items: [
    {
      id: "uuid",
      name: "Cortina Musical Mañana",
      type: "music",
      file_url: "https://s3.../cortina_morning.mp3",
      duration_seconds: 30,
      tags: ["cortina", "mañana", "energético"],
      is_favorite: true,
      usage_count: 45,
      created_at: "2024-01-01T00:00:00.000Z"
    }
  ]
}
```

POST `/api/audio-library/upload`

```
// Multipart form data
{
  file: File, // MP3/WAV
  name: "Mi Audio",
  type: "music", // music, sfx, ad, voice
  description?: "Descripción opcional",
  tags?: ["tag1", "tag2"]
}

// Respuesta
{
  success: true,
  audio_item: {
    id: "uuid",
    name: "Mi Audio",
    file_url: "https://s3.../uploaded_audio.mp3",
    duration_seconds: 120,
    file_size_bytes: 1920000
  }
}
```

Automatización

GET `/api/automation/jobs`

```
// Respuesta
{
  jobs: [
    {
      id: "uuid",
      name: "Noticiero Diario 7 AM",
      template_id: "uuid",
      frequency: "daily",
      schedule_data: {
        time: "07:00",
        timezone: "America/Santiago"
      },
      is_active: true,
      next_run_at: "2024-01-02T07:00:00.000Z",
      last_run_status: "success",
      created_at: "2024-01-01T00:00:00.000Z"
    }
  ]
}
```

POST `/api/automation/jobs`

```
// Request
{
  name: "Noticiero Automático",
  template_id: "uuid",
  frequency: "daily", // daily, weekly, specific_days
  schedule_data: {
    time: "07:00",
    days?: [1, 2, 3, 4, 5], // Lun-Vie para specific_days
    timezone: "America/Santiago"
  },
  auto_publish_settings: {
    twitter: true,
    facebook: false
  }
}

// Respuesta
{
  success: true,
  job: { /* trabajo creado */ },
  next_run_at: "2024-01-02T07:00:00.000Z"
}
```

Métricas y Análisis

GET `/api/metrics/dashboard`

```
// Query params
?period=7d&region=Santiago

// Respuesta
{
  summary: {
    total_reports: 25,
    total_cost_usd: 45.67,
    total_audio_minutes: 450,
    avg_cost_per_report: 1.83,
    success_rate: 96.5
  },

  daily_stats: [
    {
      date: "2024-01-01",
      reports_generated: 5,
      cost_usd: 8.45,
      audio_minutes: 90,
      most_popular_category: "política"
    }
  ],

  cost_breakdown: {
    "openai": 12.34,
    "elevenlabs": 28.90,
    "azure": 4.43
  },

  top_categories: [
    { category: "política", count: 45, percentage: 35.2 },
    { category: "economía", count: 32, percentage: 25.0 }
  ],

  regional_activity: [
    { region: "Santiago", count: 78, percentage: 62.4 },
    { region: "Valparaíso", count: 25, percentage: 20.0 }
  ]
}
```

GET `/api/metrics/costs`

```
// Query params detallados
?start_date=2024-01-01&end_date=2024-01-31&group_by=service

// Respuesta
{
  total_cost: 156.78,
  period: {
    start: "2024-01-01",
    end: "2024-01-31",
    days: 31
  },

  by_service: [
    {
      service: "elevenlabs",
      cost: 89.45,
      percentage: 57.1,
      requests: 245,
      avg_cost_per_request: 0.365
    },
    {
      service: "openai",
      cost: 45.67,
      percentage: 29.1,
      tokens_used: 45670,
      avg_cost_per_1k_tokens: 1.00
    }
  ],

  daily_breakdown: [
    {
      date: "2024-01-01",
      total_cost: 8.45,
      by_service: {
        "elevenlabs": 5.67,
        "openai": 2.78
      }
    }
  ],

  projections: {
    current_month: 178.90,
    next_month: 185.20 // basado en tendencia
  }
}
```

🎯 SERVICIOS DE IA INTEGRADOS

AbacusAI (Principal)

```
// lib/ai-services.ts

import { AbacusAI } from '@abacusai/client';

const abacus = new AbacusAI({
  apiKey: process.env.ABACUSAI_API_KEY
});

export async function extractNewsContent(rawText: string) {
  const response = await abacus.chat.completions.create({
    model: 'gpt-4-turbo',
    messages: [
      {
        role: 'system',
        content: 'Extrae y estructura las noticias del siguiente contenido RSS...'
      },
      { role: 'user', content: rawText }
    ],
    max_tokens: 2000,
    temperature: 0.3
  });

  return response.choices[0].message.content;
}

export async function rewriteNews(originalText: string, style = 'professional') {
  const response = await abacus.chat.completions.create({
    model: 'gpt-4-turbo',
    messages: [
      {
        role: 'system',
        content: `Reescribe esta noticia para radio con estilo ${style}...`
      },
      { role: 'user', content: originalText }
    ]
  });

  return response.choices[0].message.content;
}
```

ElevenLabs (Text-to-Speech Premium)

```
// lib/tts-services.ts

import { ElevenLabs } from 'elevenlabs';

const elevenlabs = new ElevenLabs({
  apiKey: process.env.ELEVENLABS_API_KEY
});

export async function generateElevenLabsAudio(
  text: string,
  voiceId = 'default_voice'
) {
  try {
    const audio = await elevenlabs.generate({
      voice: voiceId,
      text: text,
      model_id: "eleven_multilingual_v2",
      voice_settings: {
        stability: 0.5,
        similarity_boost: 0.8,
        style: 0.3,
        use_speaker_boost: true
      }
    });

    // Subir a S3 y retornar URL
    const audioBuffer = Buffer.from(await audio.arrayBuffer());
    const s3Key = `audio/elevenlabs/${Date.now()}.mp3`;
    const audioUrl = await uploadAudioToS3(audioBuffer, s3Key);

    // Registrar uso para facturación
    await logTokenUsage({
      service: 'elevenlabs',
      operation: 'text-to-speech',
      characters_processed: text.length,
      cost_usd: calculateElevenLabsCost(text.length)
    });

    return {
      audioUrl,
      duration: await getAudioDuration(audioBuffer),
      cost: calculateElevenLabsCost(text.length)
    };

  } catch (error) {
    console.error('ElevenLabs Error:', error);
    throw new Error(`ElevenLabs TTS failed: ${error.message}`);
  }
}

function calculateElevenLabsCost(characterCount: number): number {
  // ElevenLabs cobra $0.18 por 1000 caracteres (modelo multilingual v2)
  return (characterCount / 1000) * 0.18;
}
```

Azure Speech (Voces Chilenas)

```
import { SpeechConfig, SpeechSynthesizer } from 'microsoft-cognitiveservices-speech-sdk';

export async function generateAzureAudio(
  text: string,
  voiceName = 'es-CL-CatalinaNeural' // Voz chilena femenina
) {
  const speechConfig = SpeechConfig.fromSubscription(
    process.env.AZURE_SPEECH_KEY!,
    process.env.AZURE_SPEECH_REGION!
  );

  speechConfig.speechSynthesisVoiceName = voiceName;
  speechConfig.speechSynthesisOutputFormat = 5; // MP3

  const synthesizer = new SpeechSynthesizer(speechConfig);

  return new Promise((resolve, reject) => {
    synthesizer.speakTextAsync(
      text,
      async (result) => {
        if (result) {
          const audioBuffer = Buffer.from(result.audioData);
          const s3Key = `audio/azure/${Date.now()}.mp3`;
          const audioUrl = await uploadAudioToS3(audioBuffer, s3Key);

          await logTokenUsage({
            service: 'azure',
            operation: 'text-to-speech',
            characters_processed: text.length,
            cost_usd: calculateAzureCost(text.length)
          });

          resolve({
            audioUrl,
            duration: await getAudioDuration(audioBuffer),
            cost: calculateAzureCost(text.length)
          });
        } else {
          reject(new Error('Azure synthesis failed'));
        }
        synthesizer.close();
      },
      (error) => {
        synthesizer.close();
        reject(error);
      }
    );
  });
}

function calculateAzureCost(characterCount: number): number {
  // Azure cobra $4 por 1 millón de caracteres
  return (characterCount / 1000000) * 4.0;
}
```

AWS Polly (Económico)

```
import { PollyClient, SynthesizeSpeechCommand } from '@aws-sdk/client-polly';

const polly = new PollyClient({
  region: process.env.AWS_REGION,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!
  }
});

export async function generatePollyAudio(
  text: string,
  voiceId = 'Conchita' // Voz en español
) {
  const command = new SynthesizeSpeechCommand({
    Text: text,
    OutputFormat: 'mp3',
    VoiceId: voiceId,
    Engine: 'neural', // Mejor calidad
    LanguageCode: 'es-ES'
  });

  try {
    const response = await polly.send(command);
    const audioBuffer = Buffer.from(await response.AudioStream!.transformToByte-
Array());

    const s3Key = `audio/polly/${Date.now()}.mp3`;
    const audioUrl = await uploadAudioToS3(audioBuffer, s3Key);

    await logTokenUsage({
      service: 'aws-polly',
      operation: 'text-to-speech',
      characters_processed: text.length,
      cost_usd: calculatePollyCost(text.length)
    });

    return {
      audioUrl,
      duration: await getAudioDuration(audioBuffer),
      cost: calculatePollyCost(text.length)
    };

  } catch (error) {
    console.error('Polly Error:', error);
    throw error;
  }
}

function calculatePollyCost(characterCount: number): number {
  // AWS Polly Neural cobra $16 por 1 millón de caracteres
  return (characterCount / 1000000) * 16.0;
}
```

📁 ALMACENAMIENTO S3

Configuración AWS

```
// lib/aws-config.ts

import { S3Client } from '@aws-sdk/client-s3';

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

export const getBucketConfig = () => ({
  bucketName: process.env.AWS_BUCKET_NAME!,
  folderPrefix: process.env.AWS_FOLDER_PREFIX || ''
});
```

Operaciones S3

```
// lib/s3.ts

import { PutObjectCommand, GetObjectCommand, DeleteObjectCommand } from '@aws-sdk/cli-
ent-s3';
import { getSignedUrl } from '@aws-sdk/s3-request-presigner';

export async function uploadFile(
  buffer: Buffer,
  fileName: string,
  contentType = 'audio/mpeg'
): Promise<string> {
  const { bucketName, folderPrefix } = getBucketConfig();
  const key = `${folderPrefix}${fileName}`;

  const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: buffer,
    ContentType: contentType,
    ACL: 'public-read' // Para acceso directo
  });

  await s3Client.send(command);

  return `https://${bucketName}.s3.amazonaws.com/${key}`;
}

export async function getSignedDownloadUrl(key: string): Promise<string> {
  const { bucketName } = getBucketConfig();

  const command = new GetObjectCommand({
    Bucket: bucketName,
    Key: key
  });

  return await getSignedUrl(s3Client, command, { expiresIn: 3600 });
}

export async function deleteFile(key: string): Promise<void> {
  const { bucketName } = getBucketConfig();

  const command = new DeleteObjectCommand({
    Bucket: bucketName,
    Key: key
  });

  await s3Client.send(command);
}

// Utilidad para subir audio y registrar en DB
export async function uploadAudioToS3(
  buffer: Buffer,
  s3Key: string,
  metadata: {
    userId: string;
    name: string;
    type: 'news' | 'ad' | 'music' | 'sfx';
    duration?: number;
  }
): Promise<string> {
  // Subir a S3
  const audioUrl = await uploadFile(buffer, s3Key, 'audio/mpeg');
```

```
  // Registrar en base de datos
  const { data, error } = await supabase
    .from('audio_library')
    .insert({
      user_id: metadata.userId,
      name: metadata.name,
      type: metadata.type,
      s3_key: s3Key,
      file_url: audioUrl,
      duration_seconds: metadata.duration,
      file_size_bytes: buffer.length,
      mime_type: 'audio/mpeg'
    })
    .select()
    .single();

  if (error) {
    console.error('Error saving to audio library:', error);
    // No fallar el upload por error en DB
  }

  return audioUrl;
}
```

🔄 SCRAPING DE NOTICIAS

Scraper Principal

```
// lib/news-scraper.ts

import Parser from 'rss-parser';
import { JSDOM } from 'jsdom';

const parser = new Parser({
  timeout: 10000,
  headers: {
    'User-Agent': 'VIRA-NewsBot/1.0'
  }
});

export async function scrapeAllSources(region?: string): Promise<ScrapedNews[]> {
  const sources = await getActiveNewsSources(region);
  const allNews: ScrapedNews[] = [];

  for (const source of sources) {
    try {
      const newsItems = await scrapeSource(source);
      allNews.push(...newsItems);
    } catch (error) {
      console.error(`Error scraping ${source.name}:`, error);
      // Continuar con otras fuentes
    }
  }

  // Deduplicar y ordenar por relevancia
  return deduplicateNews(allNews)
    .sort((a, b) => b.scraped_at.getTime() - a.scraped_at.getTime())
    .slice(0, 50); // Top 50 noticias más recientes
}

async function scrapeSource(source: NewsSource): Promise<ScrapedNews[]> {
  if (source.rss_url) {
    return await scrapeRSSFeed(source);
  } else {
    return await scrapeWebPage(source);
  }
}

async function scrapeRSSFeed(source: NewsSource): Promise<ScrapedNews[]> {
  const feed = await parser.parseURL(source.rss_url!);
  const newsItems: ScrapedNews[] = [];

  for (const item of feed.items.slice(0, 20)) { // Top 20 por fuente
    try {
      // Extraer contenido completo del artículo
      const fullContent = await extractArticleContent(item.link!);

      const newsItem: ScrapedNews = {
        source_name: source.name,
        source_url: item.link!,
        title: item.title || '',
        content: fullContent || item.contentSnippet || '',
        original_url: item.link!,
        region: source.region,
        category: source.category || categorizeContent(item.title + ' ' + item.contentS
nippet),
        keywords: extractKeywords(item.title + ' ' + fullContent),
        scraped_at: new Date(),
        original_content: fullContent,
        word_count: fullContent ? fullContent.split(' ').length : 0,
```

```
        reading_time_seconds: calculateReadingTime(fullContent || ''),
        sentiment_score: await analyzeSentiment(fullContent || item.contentSnippet || '
')
      };

      newsItems.push(newsItem);

    } catch (error) {
      console.error(`Error processing item ${item.link}:`, error);
    }
  }

  return newsItems;
}

async function extractArticleContent(url: string): Promise<string | null> {
  try {
    const response = await fetch(url, {
      headers: { 'User-Agent': 'VIRA-NewsBot/1.0' },
      timeout: 15000
    });

    const html = await response.text();
    const dom = new JSDOM(html);
    const document = dom.window.document;

    // Estrategias múltiples para extraer contenido
    let content = '';

    // 1. Buscar por selectores comunes
    const selectors = [
      'article p',
      '.article-content p',
      '.post-content p',
      '.entry-content p',
      '.content p',
      'main p'
    ];

    for (const selector of selectors) {
      const elements = document.querySelectorAll(selector);
      if (elements.length > 3) { // Al menos 3 párrafos
        content = Array.from(elements)
          .map(el => el.textContent?.trim())
          .filter(text => text && text.length > 50) // Párrafos significativos
          .join('\n\n');
        break;
      }
    }

    // 2. Si no encuentra nada, usar heurística
    if (!content) {
      const paragraphs = document.querySelectorAll('p');
      const validParagraphs = Array.from(paragraphs)
        .map(p => p.textContent?.trim())
        .filter(text => text && text.length > 100) // Párrafos largos
        .slice(0, 10); // Primeros 10 párrafos

      content = validParagraphs.join('\n\n');
    }

    return content || null;
```

```
  } catch (error) {
    console.error(`Error extracting content from ${url}:`, error);
    return null;
  }
}

function categorizeContent(text: string): string {
  const categories = {
    'política': ['gobierno', 'presidente', 'congreso', 'elecciones', 'política', 'min-
istro'],
    'economía': ['economía', 'dólar', 'inflación', 'empresa', 'mercado', 'financiero'],
    'internacional': ['mundial', 'internacional', 'países', 'exterior', 'diplomacia'],
    'deportes': ['fútbol', 'deporte', 'copa', 'campeonato', 'equipo', 'jugador'],
    'tecnología': ['tecnología', 'internet', 'digital', 'app', 'software',
'innovación'],
    'salud': ['salud', 'medicina', 'hospital', 'enfermedad', 'tratamiento', 'médico'],
    'cultura': ['cultura', 'arte', 'música', 'cine', 'teatro', 'festival'],
    'sociedad': ['social', 'comunidad', 'familia', 'educación', 'seguridad']
  };

  const textLower = text.toLowerCase();

  for (const [category, keywords] of Object.entries(categories)) {
    if (keywords.some(keyword => textLower.includes(keyword))) {
      return category;
    }
  }

  return 'general';
}

function extractKeywords(text: string): string[] {
  // Palabras comunes a ignorar
  const stopWords = new Set([
    'el', 'la', 'de', 'que', 'y', 'a', 'en', 'un', 'es', 'se', 'no', 'te', 'lo',
    'le', 'da', 'su', 'por', 'son', 'con', 'para', 'al', 'del', 'los', 'las',
    'muy', 'más', 'pero', 'como', 'sin', 'entre', 'hasta', 'desde'
  ]);

  const words = text
    .toLowerCase()
    .replace(/[^\w\sáéíóúñ]/g, ' ')
    .split(/\s+/)
    .filter(word => word.length > 3 && !stopWords.has(word));

  // Contar frecuencia
  const wordCount: Record<string, number> = {};
  words.forEach(word => {
    wordCount[word] = (wordCount[word] || 0) + 1;
  });

  // Retornar las 10 palabras más frecuentes
  return Object.entries(wordCount)
    .sort(([,a], [,b]) => b - a)
    .slice(0, 10)
    .map(([word]) => word);
}

function calculateReadingTime(text: string): number {
  // Promedio 200 palabras por minuto en español
  const wordCount = text.split(' ').length;
  const minutes = wordCount / 200;
  return Math.round(minutes * 60); // Convertir a segundos
```

```
}

async function analyzeSentiment(text: string): Promise<number> {
  // Análisis básico de sentimientos
  // En producción, usar servicio especializado como AWS Comprehend

  const positiveWords = ['bueno', 'excelente', 'positivo', 'éxito', 'crecimiento', 'me-
jora'];
  const negativeWords = ['malo', 'crisis', 'problema', 'error', 'fallo', 'deterioro'];

  const textLower = text.toLowerCase();
  let score = 0;

  positiveWords.forEach(word => {
    const matches = (textLower.match(new RegExp(word, 'g')) || []).length;
    score += matches * 0.1;
  });

  negativeWords.forEach(word => {
    const matches = (textLower.match(new RegExp(word, 'g')) || []).length;
    score -= matches * 0.1;
  });

  // Normalizar entre -1 y 1
  return Math.max(-1, Math.min(1, score));
}

function deduplicateNews(newsArray: ScrapedNews[]): ScrapedNews[] {
  const seen = new Set<string>();
  return newsArray.filter(news => {
    const key = news.title.toLowerCase().replace(/[^\w]/g, '');
    if (seen.has(key)) {
      return false;
    }
    seen.add(key);
    return true;
  });
}

// Función para guardar noticias en base de datos
export async function saveScrapedNews(newsArray: ScrapedNews[]): Promise<void> {
  const { error } = await supabase
    .from('scraped_news')
    .upsert(newsArray, {
      onConflict: 'source_url,title',
      ignoreDuplicates: true
    });

  if (error) {
    console.error('Error saving scraped news:', error);
  }
}
```

🔄 PROCESAMIENTO ASÍNCRONO

Queue de Trabajos

```
// lib/job-queue.ts

import { createClient } from '@supabase/supabase-js';

export class NewscastQueue {
  private supabase = createClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.SUPABASE_SERVICE_ROLE_KEY!
  );

  async addJob(reportId: string, config: NewscastConfig) {
    // Agregar trabajo a cola
    const { error } = await this.supabase
      .from('background_jobs')
      .insert({
        type: 'generate_newscast',
        report_id: reportId,
        config: config,
        status: 'queued',
        priority: config.priority || 1
      });

    if (error) throw error;

    // Procesar inmediatamente en desarrollo
    if (process.env.NODE_ENV === 'development') {
      this.processJob(reportId, config);
    }
  }

  private async processJob(reportId: string, config: NewscastConfig) {
    try {
      // Actualizar estado
      await this.updateJobStatus(reportId, 'processing');

      // 1. Scraping
      console.log(`[${reportId}] Starting news scraping...`);
      const scrapedNews = await scrapeAllSources(config.region);
      await saveScrapedNews(scrapedNews);

      // 2. Selección y filtrado
      console.log(`[${reportId}] Selecting relevant news...`);
      const selectedNews = await selectNewsForNewscast(
        scrapedNews,
        config.categories,
        config.duration_minutes
      );

      // 3. Reescritura con IA
      console.log(`[${reportId}] Rewriting with AI...`);
      const rewrittenNews = await Promise.all(
        selectedNews.map(news => rewriteNewsItem(news, config.ai_settings))
      );

      // 4. Generación de audio
      console.log(`[${reportId}] Generating audio...`);
      const audioItems = await Promise.all(
        rewrittenNews.map(news => generateNewsAudio(news, config.ai_settings))
      );

      // 5. Crear timeline
      console.log(`[${reportId}] Building timeline...`);
```

```
      const timeline = await buildNewscastTimeline(
        audioItems,
        config.ad_phrases_count,
        config.duration_minutes
      );

      // 6. Audio final (opcional)
      let finalAudioUrl = null;
      if (config.generate_final_audio) {
        console.log(`[${reportId}] Combining final audio...`);
        finalAudioUrl = await combineAudioItems(timeline);
      }

      // 7. Guardar resultado
      const totalCost = calculateTotalCost(audioItems);
      await this.updateNewsReport(reportId, {
        timeline_data: timeline,
        final_audio_url: finalAudioUrl,
        total_cost: totalCost,
        status: 'completed',
        generation_completed_at: new Date()
      });

      console.log(`[${reportId}] Newscast generation completed!`);

    } catch (error) {
      console.error(`[${reportId}] Error generating newscast:`, error);
      await this.updateJobStatus(reportId, 'failed', error.message);
    }
  }

  private async updateJobStatus(
    reportId: string,
    status: string,
    errorMessage?: string
  ) {
    const { error } = await this.supabase
      .from('news_reports')
      .update({
        status,
        error_message: errorMessage,
        updated_at: new Date()
      })
      .eq('id', reportId);

    if (error) console.error('Error updating job status:', error);
  }

  private async updateNewsReport(reportId: string, updates: any) {
    const { error } = await this.supabase
      .from('news_reports')
      .update({
        ...updates,
        updated_at: new Date()
      })
      .eq('id', reportId);

    if (error) throw error;
  }
}

// Instancia global
export const newscastQueue = new NewscastQueue();
```

Lógica de Selección de Noticias

```
// lib/news-selection.ts

export async function selectNewsForNewscast(
  allNews: ScrapedNews[],
  categories: string[],
  durationMinutes: number
): Promise<ScrapedNews[]> {
  // Calcular distribución de tiempo
  const totalSeconds = durationMinutes * 60;
  const adTimeSeconds = Math.floor(totalSeconds * 0.15); // 15% para publicidad
  const newsTimeSeconds = totalSeconds - adTimeSeconds;

  // Tiempo promedio por noticia (45 segundos)
  const avgNewsTime = 45;
  const maxNewsCount = Math.floor(newsTimeSeconds / avgNewsTime);

  // Filtrar por categorías seleccionadas
  let filteredNews = allNews.filter(news =>
    categories.length === 0 || categories.includes(news.category)
  );

  // Si no hay categorías específicas, tomar las más relevantes
  if (categories.length === 0) {
    filteredNews = filteredNews.slice(0, maxNewsCount);
  } else {
    // Distribuir equitativamente entre categorías
    const newsPerCategory = Math.floor(maxNewsCount / categories.length);
    const selected: ScrapedNews[] = [];

    for (const category of categories) {
      const categoryNews = filteredNews
        .filter(news => news.category === category)
        .slice(0, newsPerCategory);
      selected.push(...categoryNews);
    }

    // Completar con noticias adicionales si sobra espacio
    const remaining = maxNewsCount - selected.length;
    if (remaining > 0) {
      const additionalNews = filteredNews
        .filter(news => !selected.find(s => s.id === news.id))
        .slice(0, remaining);
      selected.push(...additionalNews);
    }

    filteredNews = selected;
  }

  // Ordenar por relevancia y diversidad
  return diversifyNewsBySource(filteredNews).slice(0, maxNewsCount);
}

function diversifyNewsBySource(news: ScrapedNews[]): ScrapedNews[] {
  // Asegurar diversidad de fuentes
  const sourceCount: Record<string, number> = {};
  const diversified: ScrapedNews[] = [];

  // Primer pase: una noticia por fuente
  for (const item of news) {
    if (!sourceCount[item.source_name]) {
      sourceCount[item.source_name] = 1;
      diversified.push(item);
```

```
    }
  }

  // Segundo pase: completar con noticias restantes
  for (const item of news) {
    if (diversified.length >= 20) break; // Máximo 20 noticias

    if (!diversified.find(d => d.id === item.id) &&
        sourceCount[item.source_name] < 3) { // Max 3 por fuente
      sourceCount[item.source_name]++;
      diversified.push(item);
    }
  }

  return diversified;
}
```

🎵 PROCESAMIENTO DE AUDIO

Combinación de Audio Final

```typescript
// lib/audio-processor.ts

import ffmpeg from 'fluent-ffmpeg';
import { Readable } from 'stream';

export async function combineAudioItems(
  timeline: TimelineItem[]
): Promise<string> {
  return new Promise(async (resolve, reject) => {
    try {
      // Crear lista de archivos temporales
      const tempFiles: string[] = [];
      const outputPath = `/tmp/newscast_${Date.now()}.mp3`;

      // Descargar todos los audios
      for (const item of timeline) {
        if (item.audio_url) {
          const tempPath = `/tmp/audio_${item.id}.mp3`;
          await downloadFile(item.audio_url, tempPath);
          tempFiles.push(tempPath);
        }
      }

      // Combinar con FFmpeg
      const command = ffmpeg();

      tempFiles.forEach(file => {
        command.input(file);
      });

      command
        .on('end', async () => {
          // Subir audio final a S3
          const finalBuffer = await fs.readFile(outputPath);
          const s3Key = `newscast-final/${Date.now()}.mp3`;
          const finalUrl = await uploadFile(finalBuffer, s3Key);

          // Limpiar archivos temporales
          [...tempFiles, outputPath].forEach(file => {
            fs.unlink(file).catch(console.error);
          });

          resolve(finalUrl);
        })
        .on('error', reject)
        .mergeToFile(outputPath, '/tmp/');

    } catch (error) {
      reject(error);
    }
  });
}

async function downloadFile(url: string, localPath: string): Promise<void> {
  const response = await fetch(url);
  const buffer = await response.arrayBuffer();
  await fs.writeFile(localPath, Buffer.from(buffer));
}

export async function getAudioDuration(buffer: Buffer): Promise<number> {
  return new Promise((resolve, reject) => {
    const tempPath = `/tmp/duration_check_${Date.now()}.mp3`;
```

```
    fs.writeFile(tempPath, buffer)
      .then(() => {
        ffmpeg(tempPath)
          .ffprobe((err, metadata) => {
            fs.unlink(tempPath).catch(console.error);

            if (err) {
              reject(err);
            } else {
              resolve(Math.round(metadata.format.duration || 0));
            }
          });
      })
      .catch(reject);
  });
}
```

🔐 SEGURIDAD Y AUTENTICACIÓN

Middleware de Autenticación

```
// lib/auth-middleware.ts

import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth/next';
import { authOptions } from './auth';

export async function withAuth(
  request: NextRequest,
  handler: (req: NextRequest, user: User) => Promise<NextResponse>,
  options: { requireRole?: string } = {}
) {
  try {
    const session = await getServerSession(authOptions);

    if (!session?.user) {
      return NextResponse.json(
        { error: 'Unauthorized - Login required' },
        { status: 401 }
      );
    }

    if (options.requireRole && session.user.role !== options.requireRole) {
      return NextResponse.json(
        { error: 'Forbidden - Insufficient permissions' },
        { status: 403 }
      );
    }

    return await handler(request, session.user);

  } catch (error) {
    console.error('Auth middleware error:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}

// Uso en API routes
export async function GET(request: NextRequest) {
  return withAuth(request, async (req, user) => {
    // Lógica del endpoint aquí
    const data = await getUserData(user.id);
    return NextResponse.json({ data });
  });
}
```

Validación de Inputs

```
// lib/validation.ts

import { z } from 'zod';

export const NewscastConfigSchema = z.object({
  title: z.string().min(1).max(500).optional(),
  region: z.string().min(1).max(100),
  duration_minutes: z.number().int().min(5).max(60),
  categories: z.array(z.string()).max(8),
  ai_profile: z.enum(['economic', 'balanced', 'premium', 'custom']),
  ai_settings: z.object({
    extraction_model: z.string().optional(),
    rewrite_model: z.string().optional(),
    humanization_model: z.string().optional(),
    voice_provider: z.string().optional(),
    voice_id: z.string().optional()
  }).optional(),
  ad_phrases_count: z.number().int().min(0).max(30),
  template_id: z.string().uuid().optional()
});

export const TemplateSchema = z.object({
  name: z.string().min(1).max(255),
  region: z.string().min(1).max(100),
  duration_minutes: z.number().int().min(5).max(60),
  categories: z.array(z.string()).max(8),
  ai_profile: z.enum(['economic', 'balanced', 'premium', 'custom']),
  ai_settings: z.object({}).optional(),
  ad_phrases_count: z.number().int().min(0).max(30)
});

export function validateInput<T>(schema: z.ZodSchema<T>, input: unknown): T {
  try {
    return schema.parse(input);
  } catch (error) {
    if (error instanceof z.ZodError) {
      throw new Error(`Validation error: ${error.errors.map(e => e.message).join(', ')}
`);
    }
    throw error;
  }
}
```

📊 MONITOREO Y LOGGING

Sistema de Logs

```typescript
// lib/logger.ts

import { supabase } from './supabase';

export enum LogLevel {
  ERROR = 'error',
  WARN = 'warn',
  INFO = 'info',
  DEBUG = 'debug'
}

export class Logger {
  static async log(
    level: LogLevel,
    message: string,
    metadata?: any,
    userId?: string
  ) {
    const logEntry = {
      level,
      message,
      metadata: metadata ? JSON.stringify(metadata) : null,
      user_id: userId,
      timestamp: new Date(),
      service: 'vira-app'
    };

    // Log a consola en desarrollo
    if (process.env.NODE_ENV === 'development') {
      console.log(`[${level.toUpperCase()}] ${message}`, metadata || '');
    }

    // Guardar en base de datos
    try {
      await supabase.from('system_logs').insert(logEntry);
    } catch (error) {
      console.error('Failed to save log to database:', error);
    }

    // Enviar a Sentry en producción para errores
    if (level === LogLevel.ERROR && process.env.SENTRY_DSN) {
      // Integrar con Sentry aquí
    }
  }

  static error(message: string, metadata?: any, userId?: string) {
    return this.log(LogLevel.ERROR, message, metadata, userId);
  }

  static warn(message: string, metadata?: any, userId?: string) {
    return this.log(LogLevel.WARN, message, metadata, userId);
  }

  static info(message: string, metadata?: any, userId?: string) {
    return this.log(LogLevel.INFO, message, metadata, userId);
  }

  static debug(message: string, metadata?: any, userId?: string) {
    return this.log(LogLevel.DEBUG, message, metadata, userId);
  }
}
```

```
// Middleware para logging automático de requests
export function withLogging(handler: Function) {
  return async (req: NextRequest) => {
    const startTime = Date.now();
    const method = req.method;
    const url = req.url;

    try {
      const response = await handler(req);
      const duration = Date.now() - startTime;

      Logger.info(`${method} ${url} - ${response.status}`, {
        duration,
        status: response.status,
        userAgent: req.headers.get('user-agent')
      });

      return response;

    } catch (error) {
      const duration = Date.now() - startTime;

      Logger.error(`${method} ${url} - ERROR`, {
        duration,
        error: error.message,
        stack: error.stack
      });

      throw error;
    }
  };
}
```

Métricas en Tiempo Real

```typescript
// lib/metrics.ts

export class MetricsCollector {
  private static metrics: Record<string, number> = {};

  static increment(metric: string, value = 1) {
    this.metrics[metric] = (this.metrics[metric] || 0) + value;
  }

  static gauge(metric: string, value: number) {
    this.metrics[metric] = value;
  }

  static async flush() {
    if (Object.keys(this.metrics).length === 0) return;

    const timestamp = new Date();
    const entries = Object.entries(this.metrics).map(([name, value]) => ({
      metric_name: name,
      metric_value: value,
      timestamp,
      service: 'vira-app'
    }));

    try {
      await supabase.from('metrics').insert(entries);
      this.metrics = {}; // Reset after flush
    } catch (error) {
      console.error('Failed to flush metrics:', error);
    }
  }
}

// Middleware para métricas automáticas
export function withMetrics(handler: Function) {
  return async (req: NextRequest) => {
    MetricsCollector.increment('http_requests_total');
    MetricsCollector.increment(`http_requests_${req.method.toLowerCase()}`);

    const startTime = Date.now();

    try {
      const response = await handler(req);
      MetricsCollector.increment(`http_responses_${response.status}`);
      MetricsCollector.gauge('http_request_duration_ms', Date.now() - startTime);
      return response;
    } catch (error) {
      MetricsCollector.increment('http_errors_total');
      throw error;
    }
  };
}

// Flush métricas cada minuto
if (typeof setInterval !== 'undefined') {
  setInterval(() => {
    MetricsCollector.flush().catch(console.error);
  }, 60000);
}
```

Este manual técnico proporciona toda la información necesaria para entender, mantener y extender el sistema VIRA. Para documentación adicional, consulta los otros manuales en la carpeta `DOCUMENTACION/`.