**📋 RESUMEN GENERAL DEL PROYECTO PLAYMI**

**🎯 DESCRIPCIÓN DEL PROYECTO:**

**PLAYMI** es una plataforma web integral para entretenimiento en transporte público que permite a empresas de buses ofrecer contenido multimedia (películas, música, juegos) a sus pasajeros a través de Raspberry Pi instalados en cada vehículo.

**🔄 FLUJO GENERAL DEL SISTEMA:**

**1. ADMINISTRACIÓN (Panel Web):**

* Los administradores gestionan empresas, contenido y configuraciones desde un panel web profesional
* Se generan paquetes personalizados de contenido para cada empresa
* Se crean códigos QR específicos para cada bus

**2. INSTALACIÓN EN BUSES:**

* Cada bus tiene un Raspberry Pi que emite su propia red WiFi
* El Pi contiene todo el contenido multimedia y el portal web localmente
* No requiere conexión a internet para funcionar (contenido offline)

**3. EXPERIENCIA DEL PASAJERO:**

* Pasajero escanea QR code pegado en el bus
* Se conecta automáticamente al WiFi del bus
* Accede a portal tipo Netflix desde su celular/tablet
* Consume películas, música y juegos durante el viaje

**4. SINCRONIZACIÓN:**

* Los Pi se sincronizan automáticamente con el servidor central cuando hay WiFi disponible
* Descargan nuevo contenido y configuraciones
* Reportan estadísticas de uso

**🏗️ ARQUITECTURA TÉCNICA:**

**SERVIDOR CENTRAL:**

* **Backend:** PHP 8+ con patrón MVC
* **Frontend Admin:** AdminLTE 3 + Bootstrap 4
* **Base de datos:** MySQL 8+
* **Servidor web:** Apache/Nginx

**RASPBERRY PI (En cada bus):**

* **SO:** Raspberry Pi OS (Debian)
* **WiFi AP:** hostapd + dnsmasq
* **Servidor web:** Apache + PHP
* **Portal:** HTML5/CSS3/JavaScript vanilla
* **Contenido:** Almacenamiento local

**FRONTEND PASAJEROS:**

* **Diseño:** Estilo Netflix responsive
* **Tecnología:** HTML5, CSS3, JavaScript vanilla
* **Video:** HTML5 Video API
* **Audio:** Web Audio API
* **Juegos:** HTML5 Canvas/WebGL en iframe sandbox

**📚 MÓDULOS PRINCIPALES:**

**1. PANEL ADMINISTRATIVO (AdminLTE 3):**

* **Dashboard:** Métricas y estadísticas en tiempo real
* **Gestión de Empresas:** CRUD completo con branding personalizado
* **Gestión de Contenido:** Upload y organización de multimedia
* **Generación de Paquetes:** Crear paquetes personalizados para Pi
* **Sistema QR:** Generar códigos QR por bus
* **Monitoreo Pi:** Estado en tiempo real de toda la flota

**2. PORTAL PASAJEROS (Estilo Netflix):**

* **Página Principal:** Hero banner + carruseles de contenido
* **Catálogo Películas:** Grid responsive con filtros
* **Reproductor Música:** Estilo Spotify con playlists
* **Catálogo Juegos:** Launcher de juegos HTML5
* **Búsqueda Global:** Búsqueda en tiempo real

**3. SISTEMA DE JUEGOS:**

* **Juegos HTML5:** Subida de archivos ZIP con index.html
* **Sandbox Seguro:** Iframe con restricciones de seguridad
* **Leaderboard:** Sistema de puntuaciones
* **Categorías:** Puzzle, Arcade, Estrategia, Cartas

**4. SINCRONIZACIÓN:**

* **APIs REST:** Comunicación servidor-Pi
* **Descarga Delta:** Solo cambios incrementales
* **Health Monitoring:** Estado de cada Pi
* **Auto-recovery:** Recuperación automática de errores

**🎨 TECNOLOGÍAS UTILIZADAS:**

**BACKEND:**

* **PHP 8+:** Lenguaje principal del servidor
* **MySQL 8+:** Base de datos relacional
* **Apache/Nginx:** Servidor web
* **Las librerías que se descarguen de forma manual para integrarlas no usar composer**

**FRONTEND ADMIN (AdminLTE 3):**

* **AdminLTE 3.2.0:** Framework de administración
* **Bootstrap 4:** Framework CSS responsive
* **jQuery 3.6+:** JavaScript para interactividad
* **DataTables:** Tablas con filtros y paginación
* **Chart.js:** Gráficos y estadísticas
* **SweetAlert2:** Modales y confirmaciones elegantes
* **Toastr:** Notificaciones toast
* **FontAwesome:** Iconografía completa

**FRONTEND PASAJEROS:**

* **HTML5:** Estructura semántica
* **CSS3:** Animations, flexbox, grid
* **JavaScript Vanilla:** Sin frameworks para performance
* **Web APIs:** Video, Audio, Canvas, LocalStorage
* **Responsive Design:** Mobile-first approach

**RASPBERRY PI:**

* **Raspberry Pi OS:** Sistema operativo
* **hostapd:** WiFi Access Point
* **dnsmasq:** DHCP y DNS server
* **Apache + PHP:** Stack web local
* **Bash Scripts:** Automatización y mantenimiento   
  inicio automatico del proyecto cada vez que tenga corriente in intervención manual

**📁 ESTRUCTURA DE ARCHIVOS:**

PLAYMI/

├── 📁 admin/ # Panel administrativo

│ ├── 📁 assets/ # AdminLTE3 + personalizaciones

│ │ ├── 📁 css/ # AdminLTE + custom CSS

│ │ ├── 📁 js/ # AdminLTE + custom JS

│ │ ├── 📁 plugins/ # Bootstrap, DataTables, etc.

│ │ └── 📁 images/ # Logos e imágenes

│ ├── 📁 config/ # Configuraciones PHP

│ ├── 📁 controllers/ # Lógica de negocio MVC

│ ├── 📁 models/ # Acceso a datos MVC

│ ├── 📁 views/ # Interfaces de usuario

│ │ ├── 📁 layouts/ # Templates base (header, sidebar, footer)

│ │ ├── 📁 companies/ # CRUD empresas

│ │ ├── 📁 content/ # Gestión contenido

│ │ ├── 📁 packages/ # Generación paquetes

│ │ └── 📁 qr-system/ # Sistema QR codes

│ └── 📁 api/ # APIs REST internas

├── 📁 passenger-portal/ # Portal tipo Netflix

│ ├── 📁 assets/ # CSS/JS del portal

│ ├── 📁 player/ # Reproductores multimedia

│ ├── 📁 games/ # Sistema de juegos

│ └── 📁 api/ # APIs del portal

├── 📁 pi-system/ # Software para Raspberry Pi

│ ├── 📁 install/ # Scripts de instalación

│ ├── 📁 config/ # Configuraciones Pi

│ └── 📁 scripts/ # Mantenimiento y sync

├── 📁 content/ # Almacén de contenido

│ ├── 📁 movies/ # Películas y videos

│ ├── 📁 music/ # Audio y música

│ └── 📁 games/ # Juegos HTML5

├── 📁 packages/ # Paquetes generados

├── 📁 companies/ # Datos por empresa

└── 📁 database/ # Schema y migraciones

**🎨 DISEÑO Y UX:**

**PANEL ADMIN (AdminLTE 3):**

* **Tema:** Profesional y moderno
* **Colores:** Azul corporativo personalizable
* **Layout:** Sidebar + contenido principal
* **Componentes:** Cards, tablas, gráficos, modales
* **Responsive:** Funciona en desktop, tablet, móvil

**PORTAL PASAJEROS (Estilo Netflix):**

* **Tema:** Oscuro tipo streaming
* **Colores:** Personalizables por empresa
* **Layout:** Mobile-first, táctil
* **Navegación:** Carruseles horizontales, grids
* **Interacción:** Touch gestures, animaciones fluidas

**🔧 CARACTERÍSTICAS TÉCNICAS:**

**PERFORMANCE:**

* **Admin:** < 2 segundos de carga
* **Portal:** < 3 segundos en móviles
* **Video:** Streaming inmediato < 2 segundos buffering
* **Sincronización:** Delta sync para eficiencia

**ESCALABILIDAD:**

* **Empresas:** Sin límite teórico
* **Contenido:** Limitado por espacio en disco
* **Pi concurrentes:** Cientos de Pi sincronizando
* **Usuarios por Pi:** 10-70 usuarios simultáneos

**SEGURIDAD:**

* **Autenticación:** Sesiones seguras con timeout
* **APIs:** Rate limiting y validación
* **Juegos:** Sandbox iframe con restricciones
* **Datos:** Encriptación de datos sensibles

**🚀 FLUJO DE DESARROLLO:**

**DESARROLLO SECUENCIAL POR FASES:**

1. **Fase 1:** Base de datos y backend ✅
2. **Fase 2.1:** Panel admin + login ✅
3. **Fase 2.2:** Gestión de contenido 🔄
4. **Fase 2.3:** Generación de paquetes
5. **Fase 3:** Sistema QR + WiFi
6. **Fase 4:** Portal pasajeros
7. **Fase 5:** Sistema de juegos
8. **Fase 6:** Software Raspberry Pi
9. **Fase 7:** Sincronización
10. **Fase 8:** Optimización final

**METODOLOGÍA:**

* **Una fase a la vez:** No avanzar sin completar anterior
* **Testing obligatorio:** Cada módulo debe probarse
* **Documentación clara:** Cada archivo documentado
* **Conexiones explícitas:** Saber qué llama a qué

**🎯 OBJETIVOS DEL PROYECTO:**

**PARA EMPRESAS DE TRANSPORTE:**

* Diferenciarse de la competencia
* Mejorar experiencia del pasajero
* Generar ingresos adicionales (publicidad)
* Fidelizar clientes

**PARA PASAJEROS:**

* Entretenimiento durante viajes largos
* Acceso gratuito a contenido premium
* Experiencia moderna tipo Netflix
* Sin uso de datos móviles

**PARA ADMINISTRADORES:**

* Panel intuitivo sin conocimiento técnico
* Gestión centralizada de toda la flota
* Monitoreo en tiempo real
* Actualización automática de contenido

**📱 COMPATIBILIDAD:**

**DISPOSITIVOS SOPORTADOS:**

* **Móviles:** Android 6+, iOS 12+
* **Tablets:** Android, iPad
* **Navegadores:** Chrome, Safari, Firefox, Edge
* **Raspberry Pi:** Pi 4, Pi 5,

**RESOLUCIONES:**

* **Móviles:** 320px - 767px
* **Tablets:** 768px - 1023px
* **Desktop:** 1024px+ (admin panel)

**🎯 METODOLOGÍA DE DESARROLLO**

**PRINCIPIOS FUNDAMENTALES:**

* **Una fase a la vez** - No avanzar sin completar la anterior
* **Testing obligatorio** - Cada fase debe ser probada antes de continuar
* **Documentación clara** - Cada archivo debe tener su propósito explicado
* **Conexiones explícitas** - Saber exactamente qué archivo llama a cuál

**📋 FASES DE DESARROLLO SECUENCIAL**

**FASE 1: FUNDACIÓN DEL SISTEMA**

*Objetivo: Crear la base sólida sobre la cual construir todo*

**FASE 1.1: Estructura Base y Base de Datos**

**Tiempo estimado: 2-3 horas**

**Archivos a crear:**

PLAYMI/

├── database/

│ ├── schema.sql # Estructura completa de tablas

│ ├── initial-data.sql # Datos iniciales (admin, configs)

│ └── test-connection.php # Script para probar conexión

├── admin/

│ └── config/

│ ├── database.php # Clase de conexión a BD

│ ├── system.php # Configuraciones generales

│ └── constants.php # Constantes del sistema

└── logs/

└── .gitkeep # Mantener carpeta en git

**Propósito de cada archivo:**

* **schema.sql** → Define toda la estructura de base de datos
* **initial-data.sql** → Inserta usuario admin y configuraciones básicas
* **database.php** → Maneja todas las conexiones a la base de datos
* **system.php** → Define rutas, URLs y configuraciones del sistema
* **constants.php** → Define precios, límites y valores fijos

**Conexiones entre archivos:**

* schema.sql → Se ejecuta primero en MySQL/phpMyAdmin
* initial-data.sql → Se ejecuta después de schema.sql
* database.php → Será usado por TODOS los modelos y controladores
* system.php → Será incluido en TODAS las páginas del admin
* constants.php → Será usado para cálculos y validaciones

**Criterios de éxito:**

* Base de datos "playmi" creada con todas las tablas
* Usuario admin por defecto funciona
* Script de conexión a BD funciona sin errores
* Todas las constantes están definidas
* Carpeta logs tiene permisos de escritura

**Testing requerido:**

1. Ejecutar schema.sql en phpMyAdmin

2. Ejecutar initial-data.sql en phpMyAdmin

3. Ejecutar test-connection.php desde navegador

4. Verificar que no hay errores de PHP

**FASE 1.2: Modelos de Datos (Capa de Acceso a Datos)**

**Tiempo estimado: 3-4 horas**

**Archivos a crear:**

PLAYMI/

├── admin/

│ └── models/

│ ├── BaseModel.php # Modelo padre con funciones comunes

│ ├── User.php # Modelo para usuarios admin

│ ├── Company.php # Modelo para empresas

│ ├── Content.php # Modelo para contenido multimedia

│ ├── Package.php # Modelo para paquetes generados

│ └── Notification.php # Modelo para notificaciones

**Propósito de cada archivo:**

* **BaseModel.php** → Funciones comunes (CRUD básico) para todos los modelos
* **User.php** → Autenticación, gestión de usuarios admin
* **Company.php** → CRUD de empresas, consultas específicas de empresas
* **Content.php** → CRUD de contenido, filtros por tipo (películas/música/juegos)
* **Package.php** → Gestión de paquetes generados, historial
* **Notification.php** → Gestión de notificaciones, cola de emails

**Conexiones entre archivos:**

* BaseModel.php → Heredado por TODOS los otros modelos
* User.php → Será usado por AuthController (Fase 1.3)
* Company.php → Será usado por CompanyController (Fase 1.3)
* Content.php → Será usado por ContentController (Fase 1.3)
* Package.php → Será usado por PackageController (Fase 1.3)
* Notification.php → Será usado por NotificationController (Fase 1.3)

**Funciones requeridas por modelo:**

**BaseModel:** findAll(), findById(), create(), update(), delete() **User:** authenticate(), createUser(), updatePassword() **Company:** getActiveCompanies(), getExpiringCompanies(), getTotalRevenue() **Content:** getByType(), getMovies(), getMusic(), getGames(), getTotalSize() **Package:** getByCompany(), generatePackage(), updateStatus() **Notification:** sendEmail(), getQueue(), markAsSent()

**Criterios de éxito:**

* Todos los modelos extienden BaseModel correctamente
* Cada modelo tiene sus funciones específicas implementadas
* Conexión a database.php funciona en todos los modelos
* No hay errores de sintaxis PHP

**Testing requerido:**

1. Instanciar cada modelo sin errores

2. Probar función findAll() en cada modelo

3. Probar funciones específicas de cada modelo

4. Verificar que herencia de BaseModel funciona

**FASE 1.3: Controladores (Lógica de Negocio)**

**Tiempo estimado: 4-5 horas**

**Archivos a crear:**

PLAYMI/

├── admin/

│ └── controllers/

│ ├── BaseController.php # Controlador padre

│ ├── AuthController.php # Login/logout/autenticación

│ ├── DashboardController.php # Dashboard principal

│ ├── CompanyController.php # Gestión de empresas

│ ├── ContentController.php # Gestión de contenido

│ ├── PackageController.php # Generación de paquetes

│ └── NotificationController.php # Envío de notificaciones

**Propósito de cada archivo:**

* **BaseController.php** → Funciones comunes, verificación de sesión
* **AuthController.php** → Maneja login, logout, verificación de permisos
* **DashboardController.php** → Estadísticas, gráficos, resumen general
* **CompanyController.php** → CRUD completo de empresas, subida de logos
* **ContentController.php** → CRUD de contenido, subida de archivos multimedia
* **PackageController.php** → Genera paquetes para Pi, descarga, historial
* **NotificationController.php** → Envía emails, gestiona templates

**Conexiones entre archivos:**

* BaseController.php → Heredado por TODOS los controladores
* AuthController.php → Usa User.php, será llamado por login.php
* DashboardController.php → Usa Company.php, Content.php, será llamado por index.php
* CompanyController.php → Usa Company.php, será llamado por views/companies/
* ContentController.php → Usa Content.php, será llamado por views/content/
* PackageController.php → Usa Package.php, Company.php, Content.php
* NotificationController.php → Usa Notification.php, Company.php

**Métodos requeridos por controlador:**

**BaseController:** checkAuth(), requireLogin(), redirect(), setMessage() **AuthController:** login(), logout(), checkSession() **DashboardController:** index(), getStats(), getRecentActivity() **CompanyController:** index(), create(), store(), edit(), update(), delete() **ContentController:** index(), upload(), store(), edit(), update(), delete() **PackageController:** index(), generate(), download(), history() **NotificationController:** index(), send(), templates(), configure()

**Criterios de éxito:**

* Todos los controladores extienden BaseController
* Métodos de autenticación funcionan correctamente
* CRUD completo implementado en cada controlador
* Validaciones de datos implementadas
* Manejo de errores implementado

**Testing requerido:**

1. Probar login/logout completo

2. Verificar que páginas requieren autenticación

3. Probar CRUD de empresas sin errores

4. Probar subida de archivos

5. Verificar manejo de errores

**FASE 2: INTERFAZ DE USUARIO (ADMIN PANEL)**

*Objetivo: Crear todas las pantallas del panel administrativo*

**FASE 2.1: Layout Base y Sistema de Autenticación**

**Tiempo estimado: 3-4 horas**

**Archivos a crear:**

PLAYMI/

├── admin/

│ ├── assets/

│ │ ├── css/

│ │ │ ├── adminlte.min.css # AdminLTE framework

│ │ │ ├── custom.css # Estilos personalizados

│ │ │ └── login.css # Estilos específicos del login

│ │ ├── js/

│ │ │ ├── adminlte.min.js # AdminLTE JavaScript

│ │ │ ├── jquery.min.js # jQuery

│ │ │ └── custom.js # JavaScript personalizado

│ │ └── images/

│ │ ├── logo-playmi.png # Logo principal

│ │ └── login-bg.jpg # Fondo de login

│ ├── views/

│ │ └── layouts/

│ │ ├── header.php # Header común (navbar)

│ │ ├── sidebar.php # Sidebar común (menú)

│ │ ├── footer.php # Footer común

│ │ └── auth-layout.php # Layout para login

│ ├── login.php # Página de login

│ └── logout.php # Script de logout

**Propósito de cada archivo:**

* **adminlte.min.css/js** → Framework AdminLTE para interfaz profesional
* **custom.css/js** → Personalizaciones específicas de PLAYMI
* **header.php** → Navbar superior con notificaciones y perfil
* **sidebar.php** → Menú lateral con todas las opciones del admin
* **footer.php** → Footer con información de copyright
* **login.php** → Página de autenticación con formulario

**Conexiones entre archivos:**

* login.php → Usa AuthController.php → Usa User.php
* header.php → Incluido en TODAS las páginas del admin
* sidebar.php → Incluido en TODAS las páginas del admin
* footer.php → Incluido en TODAS las páginas del admin
* logout.php → Usa AuthController.php

**Criterios de éxito:**

* AdminLTE carga correctamente sin errores 404
* Login funciona y redirige al dashboard
* Logout funciona y redirige al login
* Header, sidebar y footer se muestran correctamente
* Diseño responsive funciona en móviles

**📋 METODOLOGÍA DE DESARROLLO COMPLETA PLAYMI**

* **🎯 CONTEXTO DEL PROYECTO:**
* Sistema web para entretenimiento en transporte público donde:
* Los administradores gestionan empresas, contenido y paquetes desde un panel web
* Los Raspberry Pi en buses emiten WiFi y hospedan contenido
* Los pasajeros escanean QR → se conectan al WiFi → acceden a portal tipo Netflix
* Contenido: películas, música (tipo YouTube) y juegos HTML5/JS
* **🏗️ ESTRUCTURA GENERAL DEL PROYECTO:**
* PLAYMI/
* ├── 📁 admin/ # Panel administrativo (FASE 1-2)
* ├── 📁 passenger-portal/ # Portal tipo Netflix para pasajeros
* ├── 📁 pi-system/ # Sistema completo para Raspberry Pi
* ├── 📁 content/ # Todo el contenido multimedia
* ├── 📁 companies/ # Datos específicos por empresa
* ├── 📁 packages/ # Paquetes generados para distribución
* ├── 📁 database/ # Base de datos y migraciones
* └── 📁 api/ # APIs compartidas
* **🚀 FASE 2.2: GESTIÓN DE CONTENIDO MULTIMEDIA**
* **Objetivo:** Crear sistema completo para subir, organizar y gestionar películas, música y juegos **Tiempo estimado:** 6-8 horas
* **ARCHIVOS A CREAR:**
* PLAYMI/
* ├── admin/
* │ ├── views/content/
* │ │ ├── index.php # Lista todo el contenido
* │ │ ├── movies.php # Gestión específica de películas
* │ │ ├── music.php # Gestión específica de música
* │ │ ├── games.php # Gestión específica de juegos
* │ │ ├── upload.php # Formulario de subida universal
* │ │ └── edit.php # Editar metadatos de contenido
* │ ├── api/content/
* │ │ ├── upload-movie.php # Subir archivo de película
* │ │ ├── upload-music.php # Subir archivo de música
* │ │ ├── upload-game.php # Subir juego HTML5 (ZIP)
* │ │ ├── process-video.php # Procesar y comprimir video
* │ │ ├── generate-thumbnail.php # Generar miniaturas automáticas
* │ │ └── delete-content.php # Eliminar contenido
* │ └── models/
* │ └── Content.php # Modelo para contenido (ya existe)
* ├── content/
* │ ├── movies/ # Almacén de películas
* │ │ ├── originals/ # Videos originales subidos
* │ │ ├── compressed/ # Videos comprimidos para streaming
* │ │ └── thumbnails/ # Miniaturas generadas
* │ ├── music/
* │ │ ├── audio/ # Archivos de audio
* │ │ ├── thumbnails/ # Carátulas/imágenes
* │ │ └── metadata/ # Información adicional
* │ └── games/
* │ ├── source/ # Archivos ZIP originales
* │ ├── extracted/ # Juegos descomprimidos
* │ └── thumbnails/ # Capturas de pantalla
* **FUNCIONALIDADES REQUERIDAS:**
* **Para Películas:**
* Subida de archivos MP4, AVI, MKV (máximo 2GB)
* Compresión automática para optimizar streaming
* Generación automática de thumbnails
* Metadatos: título, descripción, duración, género, clasificación
* Preview del video antes de publicar
* **Para Música:**
* Subida de archivos MP3, M4A, WAV
* Subida opcional de carátula/imagen
* Metadatos: título, artista, álbum, duración, género
* Reproducción previa antes de publicar
* **Para Juegos HTML5:**
* Subida de archivos ZIP que contengan index.html
* Validación automática de estructura del juego
* Generación de thumbnail desde screenshot
* Metadatos: título, descripción, categoría, instrucciones
* Preview del juego funcionando
* **CONEXIONES ENTRE ARCHIVOS:**
* views/content/\* → Usan ContentController.php → Usa Content.php
* api/content/\* → Procesan uploads y devuelven JSON
* upload-movie.php → Guarda en content/movies/ → Actualiza BD
* upload-game.php → Descomprime ZIP en content/games/extracted/
* process-video.php → Usa FFmpeg para comprimir videos
* **CRITERIOS DE ÉXITO:**
* Subida de los 3 tipos de contenido funciona sin errores
* Validaciones de formato y tamaño implementadas
* Compresión automática de videos funciona
* Generación de thumbnails automática
* Lista de contenido con filtros por tipo
* Edición de metadatos funcional
* Preview de contenido antes de publicar
* **🚀 FASE 2.3: GESTIÓN DE PAQUETES Y DISTRIBUCIÓN**
* **Objetivo:** Sistema para generar paquetes personalizados de contenido para cada empresa/Pi **Tiempo estimado:** 4-5 horas
* **ARCHIVOS A CREAR:**
* PLAYMI/
* ├── admin/
* │ ├── views/packages/
* │ │ ├── index.php # Lista de paquetes generados
* │ │ ├── generate.php # Formulario para generar paquete
* │ │ ├── customize.php # Personalizar contenido por empresa
* │ │ └── history.php # Historial de paquetes
* │ ├── api/packages/
* │ │ ├── generate-package.php # Crear paquete personalizado
* │ │ ├── download-package.php # Descargar paquete para Pi
* │ │ └── update-status.php # Actualizar estado de paquete
* │ └── models/
* │ └── Package.php # Modelo para paquetes (ya existe)
* ├── packages/
* │ ├── templates/
* │ │ ├── default-config.json # Configuración base del paquete
* │ │ ├── wifi-config.template # Template configuración WiFi
* │ │ └── portal-config.template # Template portal pasajeros
* │ └── generated/
* │ ├── [empresa-id]/ # Paquetes por empresa
* │ │ ├── content/ # Contenido seleccionado
* │ │ ├── config/ # Configuraciones específicas
* │ │ ├── portal/ # Portal web para pasajeros
* │ │ └── package.zip # Paquete final para descargar
* **FUNCIONALIDADES REQUERIDAS:**
* Seleccionar contenido específico por empresa/paquete
* Configurar branding personalizado (colores, logo)
* Generar configuración WiFi automática
* Crear QR codes personalizados por bus
* Empaquetar todo en ZIP descargable
* Seguimiento de estado: generando → listo → instalado
* Historial de paquetes por empresa
* **CONEXIONES ENTRE ARCHIVOS:**
* generate-package.php → Lee Content.php y Company.php
* generate-package.php → Copia contenido desde content/ a packages/generated/
* download-package.php → Sirve archivo ZIP desde packages/generated/
* **🚀 FASE 3: SISTEMA QR Y CONFIGURACIÓN WIFI**
* **Objetivo:** Sistema para generar QR codes y configuración WiFi automática **Tiempo estimado:** 3-4 horas
* **ARCHIVOS A CREAR:**
* PLAYMI/
* ├── admin/
* │ ├── views/qr-system/
* │ │ ├── index.php # Gestión de códigos QR
* │ │ ├── generate.php # Generar QR por bus/empresa
* │ │ └── print.php # Página para imprimir QR
* │ ├── api/qr/
* │ │ ├── generate-qr.php # API generar código QR
* │ │ └── wifi-config.php # API configuración WiFi
* │ └── libraries/
* │ └── qr-generator/ # Librería PHP para QR codes
* ├── companies/
* │ └── [empresa-id]/
* │ ├── qr-codes/ # QR codes generados
* │ └── wifi-configs/ # Configuraciones WiFi
* **FUNCIONALIDADES REQUERIDAS:**
* Generar QR codes únicos por bus/empresa
* QR contiene configuración WiFi + URL del portal
* Configuración automática de WiFi en dispositivos
* Personalización de SSID y contraseña por empresa
* Códigos QR descargables para impresión
* Seguimiento de QR codes activos
* **FLUJO DEL QR CODE:**
* Admin genera QR para empresa/bus específico
* QR contiene: WIFI:T:WPA;S:[SSID];P:[PASSWORD];H:false;; + URL
* Pasajero escanea → se conecta automáticamente al WiFi
* Browser abre automáticamente la URL del portal
* **🚀 FASE 4: PORTAL WEB PARA PASAJEROS (TIPO NETFLIX)**
* **Objetivo:** Interfaz moderna y atractiva para que pasajeros consuman contenido **Tiempo estimado:** 8-10 horas
* **ARCHIVOS A CREAR:**
* PLAYMI/
* ├── passenger-portal/
* │ ├── index.php # Página principal tipo Netflix
* │ ├── movies.php # Catálogo de películas
* │ ├── music.php # Reproductor tipo YouTube
* │ ├── games.php # Catálogo de juegos HTML5
* │ ├── player/
* │ │ ├── video-player.php # Reproductor de video
* │ │ ├── music-player.php # Reproductor de música
* │ │ └── game-launcher.php # Lanzador de juegos
* │ ├── assets/
* │ │ ├── css/
* │ │ │ ├── netflix-style.css # Estilos tipo Netflix
* │ │ │ ├── mobile-optimized.css # Optimizado para móviles
* │ │ │ └── games.css # Estilos para juegos
* │ │ ├── js/
* │ │ │ ├── portal-main.js # JavaScript principal
* │ │ │ ├── video-controls.js # Controles de video
* │ │ │ ├── music-controls.js # Controles de música
* │ │ │ └── games-manager.js # Gestión de juegos
* │ │ └── images/
* │ │ ├── hero-banners/ # Banners principales
* │ │ └── icons/ # Iconos de la interfaz
* │ ├── api/
* │ │ ├── get-content.php # API obtener contenido
* │ │ ├── track-usage.php # Seguimiento de uso
* │ │ └── company-branding.php # Obtener branding por empresa
* │ └── config/
* │ └── portal-config.php # Configuración del portal
* **CARACTERÍSTICAS DE LA INTERFAZ:**
* **Página Principal (tipo Netflix):**
* Hero banner con contenido destacado
* Carruseles por categoría: "Películas Populares", "Música Trending", "Juegos Nuevos"
* Navegación intuitiva con iconos grandes
* Branding personalizado por empresa (colores, logo)
* Optimizado para pantallas táctiles móviles
* **Reproductor de Video:**
* Controles táctiles grandes y claros
* Reproducción en pantalla completa
* Controles de volumen y progreso
* Botones de pausa/play prominentes
* **Reproductor de Música (tipo YouTube):**
* Lista de reproducción visual
* Carátulas grandes y atractivas
* Controles de siguiente/anterior
* Visualización del progreso
* **Lanzador de Juegos:**
* Grid de juegos con screenshots atractivos
* Categorías: Puzzle, Arcade, Estrategia
* Lanzamiento en iframe o ventana nueva
* Instrucciones claras de cada juego
* **🚀 FASE 5: SISTEMA DE JUEGOS HTML5**
* **Objetivo:** Infraestructura completa para ejecutar juegos web **Tiempo estimado:** 4-5 horas
* **ARCHIVOS A CREAR:**
* PLAYMI/
* ├── passenger-portal/
* │ ├── games/
* │ │ ├── launcher.php # Lanzador universal de juegos
* │ │ ├── iframe-sandbox.php # Sandbox seguro para juegos
* │ │ └── leaderboard.php # Tabla de puntuaciones
* │ └── api/games/
* │ ├── load-game.php # Cargar juego específico
* │ ├── save-score.php # Guardar puntuaciones
* │ └── get-leaderboard.php # Obtener rankings
* ├── content/games/extracted/
* │ └── [game-id]/
* │ ├── index.html # Punto de entrada del juego
* │ ├── assets/ # Recursos del juego
* │ └── game-config.json # Configuración del juego
* **ESTRUCTURA DE JUEGO HTML5:**
* Cada juego debe tener:
* index.html - Punto de entrada principal
* assets/ - Imágenes, sonidos, sprites
* game-config.json - Metadatos y configuración
* Comunicación con portal vía postMessage para puntuaciones
* **TIPOS DE JUEGOS SOPORTADOS:**
* Puzzle (Tetris, Match-3, Sudoku)
* Arcade (Snake, Breakout, Pac-Man)
* Cartas (Solitario, Memory)
* Estrategia (Chess, Checkers)
* Casual (Bubble Shooter, Word Games)
* **🚀 FASE 6: SISTEMA RASPBERRY PI**
* **Objetivo:** Software completo para instalar en Raspberry Pi **Tiempo estimado:** 6-8 horas
* **ARCHIVOS A CREAR:**
* PLAYMI/
* ├── pi-system/
* │ ├── install/
* │ │ ├── setup.sh # Script de instalación completo
* │ │ ├── wifi-ap.sh # Configurar WiFi Access Point
* │ │ ├── web-server.sh # Instalar servidor web
* │ │ └── auto-start.sh # Configurar inicio automático
* │ ├── config/
* │ │ ├── hostapd.conf # Configuración WiFi hotspot
* │ │ ├── dnsmasq.conf # Configuración DNS/DHCP
* │ │ ├── apache-config.conf # Configuración Apache
* │ │ └── company-config.json # Configuración por empresa
* │ ├── scripts/
* │ │ ├── sync-content.sh # Sincronizar contenido del servidor
* │ │ ├── update-system.sh # Actualizar sistema Pi
* │ │ └── health-check.sh # Verificar estado del sistema
* │ └── web/
* │ └── [symlink to passenger-portal] # Portal web para pasajeros
* **FUNCIONALIDADES DEL SISTEMA PI:**
* Configuración automática como WiFi Access Point
* Servidor web Apache con contenido local
* Sincronización periódica con servidor central
* Auto-inicio al encender el Pi
* Monitoreo de salud del sistema
* Logs de uso y conexiones
* **🚀 FASE 7: SISTEMA DE SINCRONIZACIÓN**
* **Objetivo:** Sincronización automática entre servidor central y Raspberry Pi **Tiempo estimado:** 4-5 horas
* **ARCHIVOS A CREAR:**
* PLAYMI/
* ├── api/sync/
* │ ├── pi-register.php # Registrar nuevo Pi
* │ ├── check-updates.php # Verificar actualizaciones
* │ ├── download-package.php # Descargar paquete actualizado
* │ └── report-status.php # Reportar estado del Pi
* ├── admin/
* │ ├── views/pi-management/
* │ │ ├── index.php # Lista de Pi registrados
* │ │ ├── monitoring.php # Monitoreo en tiempo real
* │ │ └── logs.php # Logs de sincronización
* │ └── api/pi/
* │ ├── push-update.php # Enviar actualización a Pi
* │ └── get-pi-status.php # Obtener estado de Pi específico
* **FLUJO DE SINCRONIZACIÓN:**
* Pi se registra en servidor al iniciar
* Pi verifica actualizaciones cada X minutos
* Servidor notifica cuando hay nuevo contenido
* Pi descarga y actualiza contenido automáticamente
* Pi reporta estado y estadísticas de uso
* **🚀 FASE 8: OPTIMIZACIÓN Y FINALIZACIÓN**
* **Objetivo:** Optimizar rendimiento y completar funcionalidades **Tiempo estimado:** 3-4 horas
* **OPTIMIZACIONES FINALES:**
* Compresión de assets CSS/JS
* Optimización de imágenes y videos
* Cache de contenido en Pi
* Mejoras de velocidad de carga
* Testing completo del sistema
* Documentación de instalación
* Manual de usuario para administradores
* **📋 CRITERIOS DE ÉXITO TOTAL:**
* **Al completar todas las fases:**
* ✅ Administradores pueden gestionar empresas y contenido
* ✅ Sistema genera paquetes personalizados automáticamente
* ✅ Códigos QR conectan pasajeros automáticamente al WiFi
* ✅ Portal tipo Netflix funciona fluido en móviles
* ✅ Juegos HTML5 se ejecutan sin problemas
* ✅ Raspberry Pi funciona de forma autónoma
* ✅ Sincronización automática servidor-Pi operativa
* ✅ Sistema completo listo para producción
* **🔗 CONEXIONES CRÍTICAS ENTRE FASES:**
* **Fase 2.2** alimenta contenido → **Fase 2.3** genera paquetes
* **Fase 2.3** crea configuración → **Fase 3** genera QR codes
* **Fase 3** define WiFi → **Fase 4** portal accesible vía WiFi
* **Fase 4** consume contenido → **Fase 5** ejecuta juegos
* **Fase 6** instala sistema → **Fase 7** mantiene sincronizado
* **Todas las fases** → **Fase 8** optimiza y finaliza

**📋 DOCUMENTACIÓN DETALLADA MODULAR - FASE 2.2: GESTIÓN DE CONTENIDO**

**🎯 DIVISIÓN EN MÓDULOS ESPECÍFICOS:**

**FASE 2.2 se divide en 12 MÓDULOS independientes:**

**📚 MÓDULO 2.2.1: admin/views/content/index.php**

**Propósito:** Lista principal de todo el contenido multimedia

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de visualización con tabla interactiva
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::index()
* **Modelo:** Content::findAll(), Content::getStats()
* **Layout:** layouts/base.php
* **APIs llamadas:** api/content/delete-content.php, api/content/update-status.php
* **JavaScript:** DataTables, SweetAlert2, funciones AJAX
* **CSS:** Estilos para thumbnails, badges de estado

**📊 DATOS A MOSTRAR:**

* Thumbnail del contenido (imagen pequeña)
* Título del contenido
* Tipo (película/música/juego) con badge de color
* Duración (para videos/audio) o tamaño (para juegos)
* Estado (activo/inactivo/procesando)
* Fecha de subida
* Acciones (ver/editar/eliminar/preview)

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Tabla con DataTables (paginación, ordenamiento, búsqueda)
* Filtros por tipo de contenido (películas/música/juegos)
* Filtros por estado (activo/inactivo/procesando)
* Búsqueda por título
* Botones de exportación (Excel/PDF)
* Confirmación antes de eliminar
* Cambio de estado con dropdown
* Preview de contenido en modal/ventana nueva

**📈 ESTADÍSTICAS A MOSTRAR:**

* Total de contenido
* Total por tipo (películas/música/juegos)
* Espacio ocupado total
* Contenido agregado este mes

**🔧 ELEMENTOS DE INTERFAZ:**

* Cards de estadísticas superiores (4 info-boxes)
* Card de filtros con formulario
* Card principal con tabla DataTables
* Botones de acción en cada fila
* Modales para confirmaciones

**🎯 CRITERIOS DE ÉXITO:**

* Tabla carga sin errores
* Filtros funcionan correctamente
* Eliminación con confirmación funciona
* Cambio de estado funciona
* Preview abre correctamente
* Exportación PDF/Excel funciona
* Responsive en móviles

**📚 MÓDULO 2.2.2: admin/views/content/upload.php**

**Propósito:** Formulario universal para subir todo tipo de contenido

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Formulario con subida de archivos
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET (mostrar) / POST (enviar a API)
* **Tiempo estimado:** 1 hora

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::upload()
* **APIs destino:** api/content/upload-movie.php, api/content/upload-music.php, api/content/upload-game.php
* **Layout:** layouts/base.php
* **JavaScript:** Validación, progress bars, tipo de archivo dinámico
* **CSS:** Estilos para drag-and-drop, progress bars

**📝 CAMPOS DEL FORMULARIO:**

**Comunes para todos:**

* Selector de tipo (película/música/juego)
* Título (requerido)
* Descripción
* Tags/categorías

**Específicos por tipo:**

* **Películas:** Género, clasificación, director, año
* **Música:** Artista, álbum, género, año
* **Juegos:** Categoría, instrucciones, controles

**📁 SUBIDA DE ARCHIVOS:**

* **Películas:** MP4, AVI, MKV (máximo 2GB)
* **Música:** MP3, M4A, WAV + imagen opcional
* **Juegos:** ZIP que contenga index.html

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Drag and drop para archivos
* Progress bar durante subida
* Validación de formato y tamaño
* Preview antes de enviar
* Campos dinámicos según tipo seleccionado
* Validación frontend y envío AJAX
* Redirección después de éxito

**⚠️ VALIDACIONES NECESARIAS:**

* Tamaño máximo por tipo
* Formatos permitidos
* Campos requeridos por tipo
* Estructura de ZIP para juegos
* Duplicados de título

**🎯 CRITERIOS DE ÉXITO:**

* Subida funciona para los 3 tipos
* Validaciones funcionan correctamente
* Progress bar se muestra
* Errores se muestran claramente
* Redirección después de éxito

**📚 MÓDULO 2.2.3: admin/views/content/movies.php**

**Propósito:** Gestión específica de películas con funciones especializadas

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página especializada para películas
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::movies()
* **Modelo:** Content::getMovies(), Content::getMovieStats()
* **APIs:** api/content/process-video.php, api/content/generate-thumbnail.php
* **Layout:** layouts/base.php

**📊 DATOS ESPECÍFICOS DE PELÍCULAS:**

* Thumbnail generado automáticamente
* Título, director, año, género
* Duración del video
* Resolución y calidad
* Tamaño del archivo original vs comprimido
* Estado de procesamiento
* Número de visualizaciones

**🎬 FUNCIONALIDADES ESPECÍFICAS:**

* Recomprimir video para optimizar
* Regenerar thumbnail automático
* Ver detalles técnicos del video
* Player de preview integrado
* Gestión de géneros
* Clasificación por edades

**📈 ESTADÍSTICAS ESPECÍFICAS:**

* Total de películas por género
* Espacio ocupado por películas
* Películas más vistas
* Películas agregadas por mes

**🎯 CRITERIOS DE ÉXITO:**

* Lista filtrada solo películas
* Funciones específicas funcionan
* Compresión de video funciona
* Player de preview funciona
* Estadísticas específicas correctas

**📚 MÓDULO 2.2.4: admin/views/content/music.php**

**Propósito:** Gestión específica de música con reproductor integrado

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página especializada para música
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::music()
* **Modelo:** Content::getMusic(), Content::getMusicStats()
* **Layout:** layouts/base.php
* **JavaScript:** Reproductor de audio personalizado

**🎵 DATOS ESPECÍFICOS DE MÚSICA:**

* Carátula del álbum
* Título, artista, álbum
* Duración del audio
* Género musical
* Calidad de audio (bitrate)
* Tamaño del archivo

**🎛️ FUNCIONALIDADES ESPECÍFICAS:**

* Reproductor de audio integrado
* Gestión de álbumes
* Edición de metadatos ID3
* Subida masiva de álbumes completos
* Organización por artista/álbum

**📈 ESTADÍSTICAS ESPECÍFICAS:**

* Total por género musical
* Artistas con más canciones
* Álbumes completos vs canciones sueltas
* Música más reproducida

**🎯 CRITERIOS DE ÉXITO:**

* Reproductor funciona correctamente
* Gestión de metadatos funciona
* Organización por álbum funciona
* Estadísticas musicales correctas

**📚 MÓDULO 2.2.5: admin/views/content/games.php**

**Propósito:** Gestión específica de juegos HTML5

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página especializada para juegos
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 50 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::games()
* **Modelo:** Content::getGames(), Content::getGameStats()
* **APIs:** api/content/validate-game.php, api/content/extract-game.php
* **Layout:** layouts/base.php

**🎮 DATOS ESPECÍFICOS DE JUEGOS:**

* Screenshot del juego
* Título, categoría, dificultad
* Instrucciones de juego
* Controles necesarios
* Tamaño del archivo ZIP
* Estado de extracción

**🎛️ FUNCIONALIDADES ESPECÍFICAS:**

* Validación de estructura HTML5
* Extracción automática de ZIP
* Preview del juego en iframe
* Generación de screenshot automático
* Gestión de categorías de juegos
* Testing de funcionalidad

**📈 ESTADÍSTICAS ESPECÍFICAS:**

* Total por categoría de juego
* Juegos más jugados
* Juegos con errores
* Tamaño promedio de juegos

**🎯 CRITERIOS DE ÉXITO:**

* Validación de juegos funciona
* Extracción automática funciona
* Preview en iframe funciona
* Categorización funciona correctamente

**📚 MÓDULO 2.2.6: admin/views/content/edit.php**

**Propósito:** Editar metadatos de contenido existente

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Formulario de edición
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET (mostrar) / POST (actualizar)
* **Tiempo estimado:** 35 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::edit(), ContentController::update()
* **Modelo:** Content::findById(), Content::update()
* **API:** api/content/update-content.php
* **Layout:** layouts/base.php

**📝 CAMPOS EDITABLES:**

* Metadatos básicos (título, descripción)
* Metadatos específicos por tipo
* Estado del contenido
* Tags y categorías
* Thumbnail/carátula (cambiar archivo)

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Cargar datos existentes en formulario
* Validación antes de actualizar
* Cambio de thumbnail/carátula
* Preview de cambios
* Historial de modificaciones

**🎯 CRITERIOS DE ÉXITO:**

* Formulario carga con datos existentes
* Actualización funciona correctamente
* Validaciones funcionan
* Cambio de imagen funciona

**📚 MÓDULO 2.2.7: api/content/upload-movie.php**

**Propósito:** API para procesar subida de películas

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 1 hora

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::storeMovie()
* **Modelo:** Content::create()
* **Utilidades:** FFmpeg para procesamiento
* **Destino:** content/movies/originals/, content/movies/compressed/

**⚙️ PROCESO DE SUBIDA:**

1. Validar archivo (formato, tamaño)
2. Guardar archivo original
3. Generar thumbnail automático
4. Comprimir video para streaming
5. Extraer metadatos (duración, resolución)
6. Guardar en base de datos
7. Retornar respuesta JSON

**📁 ESTRUCTURA DE ALMACENAMIENTO:**

content/movies/

├── originals/[movie-id].mp4

├── compressed/[movie-id]\_compressed.mp4

└── thumbnails/[movie-id]\_thumb.jpg

**🎯 CRITERIOS DE ÉXITO:**

* Subida completa sin errores
* Compresión automática funciona
* Thumbnail se genera correctamente
* Metadatos se extraen bien
* Base de datos se actualiza

**📚 MÓDULO 2.2.8: api/content/upload-music.php**

**Propósito:** API para procesar subida de música

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::storeMusic()
* **Modelo:** Content::create()
* **Utilidades:** Librería para metadatos ID3
* **Destino:** content/music/audio/, content/music/thumbnails/

**⚙️ PROCESO DE SUBIDA:**

1. Validar archivo de audio
2. Extraer metadatos ID3 (artista, álbum, etc.)
3. Procesar carátula si existe
4. Guardar archivo de audio
5. Optimizar calidad si es necesario
6. Guardar en base de datos
7. Retornar respuesta JSON

**🎯 CRITERIOS DE ÉXITO:**

* Metadatos ID3 se extraen correctamente
* Carátula se procesa bien
* Audio se optimiza si es necesario
* Base de datos se actualiza correctamente

**📚 MÓDULO 2.2.9: api/content/upload-game.php**

**Propósito:** API para procesar subida de juegos HTML5

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 1 hora 15 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::storeGame()
* **Modelo:** Content::create()
* **Utilidades:** ZipArchive para extracción
* **Destino:** content/games/source/, content/games/extracted/

**⚙️ PROCESO DE SUBIDA:**

1. Validar archivo ZIP
2. Extraer temporalmente para validar estructura
3. Verificar que existe index.html
4. Guardar ZIP original
5. Extraer a carpeta definitiva
6. Generar screenshot automático
7. Validar que el juego funciona
8. Guardar en base de datos

**🎯 CRITERIOS DE ÉXITO:**

* Validación de estructura funciona
* Extracción completa sin errores
* Screenshot se genera automáticamente
* Juego funciona en iframe
* Base de datos se actualiza

**📚 MÓDULO 2.2.10: api/content/delete-content.php**

**Propósito:** API para eliminar contenido y archivos asociados

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 30 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::destroy()
* **Modelo:** Content::delete(), Content::findById()
* **Sistema:** Eliminación física de archivos

**⚙️ PROCESO DE ELIMINACIÓN:**

1. Verificar que contenido existe
2. Verificar permisos de eliminación
3. Eliminar archivos físicos asociados
4. Eliminar registros de base de datos
5. Limpiar referencias en otras tablas
6. Registrar en log de actividad
7. Retornar confirmación

**🎯 CRITERIOS DE ÉXITO:**

* Archivos físicos se eliminan completamente
* Registros de BD se eliminan
* No quedan referencias huérfanas
* Log de actividad se registra

**📚 MÓDULO 2.2.11: api/content/process-video.php**

**Propósito:** API para recomprimir videos existentes

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint (proceso pesado)
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::processVideo()
* **Utilidades:** FFmpeg
* **Modelo:** Content::updateProcessingStatus()

**⚙️ PROCESO DE COMPRESIÓN:**

1. Verificar video existe
2. Marcar estado como "procesando"
3. Ejecutar FFmpeg para compresión
4. Generar múltiples calidades si necesario
5. Actualizar estado a "completado"
6. Notificar resultado vía WebSocket/polling

**🎯 CRITERIOS DE ÉXITO:**

* Compresión se ejecuta sin errores
* Estados se actualizan correctamente
* Múltiples calidades se generan
* Notificación de completado funciona

**📚 MÓDULO 2.2.12: api/content/generate-thumbnail.php**

**Propósito:** API para regenerar thumbnails automáticamente

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 25 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** ContentController::generateThumbnail()
* **Utilidades:** FFmpeg (videos), GD/ImageMagick (imágenes)
* **Modelo:** Content::updateThumbnail()

**⚙️ PROCESO DE GENERACIÓN:**

1. Identificar tipo de contenido
2. Generar thumbnail según tipo:
   * Videos: Frame del segundo 10
   * Música: Carátula o imagen por defecto
   * Juegos: Screenshot de iframe
3. Optimizar imagen generada
4. Guardar en directorio correcto
5. Actualizar base de datos

**🎯 CRITERIOS DE ÉXITO:**

* Thumbnail se genera para todos los tipos
* Calidad de imagen es adecuada
* Archivos se guardan correctamente
* Base de datos se actualiza

**🔄 FLUJO ENTRE MÓDULOS:**

1. **Usuario accede** → index.php (Módulo 2.2.1)
2. **Quiere subir** → upload.php (Módulo 2.2.2) → APIs de subida (Módulos 2.2.7-2.2.9)
3. **Gestión específica** → movies.php/music.php/games.php (Módulos 2.2.3-2.2.5)
4. **Editar contenido** → edit.php (Módulo 2.2.6)
5. **Eliminar** → delete-content.php (Módulo 2.2.10)
6. **Procesar** → APIs de procesamiento (Módulos 2.2.11-2.2.12)

**✅ CRITERIOS DE ÉXITO TOTAL DE FASE 2.2:**

* Todos los 12 módulos funcionan independientemente
* Flujo completo de subida funciona
* Gestión específica por tipo funciona
* APIs procesan correctamente
* Archivos se almacenan organizadamente
* Base de datos mantiene integridad

**📋 DOCUMENTACIÓN COMPLETA DETALLADA - PROYECTO PLAYMI**

**🚀 FASE 2.3: GESTIÓN DE PAQUETES Y DISTRIBUCIÓN**

**📋 DIVISIÓN EN MÓDULOS:**

**FASE 2.3 se divide en 8 MÓDULOS específicos:**

**📚 MÓDULO 2.3.1: admin/views/packages/index.php**

**Propósito:** Lista principal de paquetes generados para empresas

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de visualización con tabla interactiva
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PackageController::index()
* **Modelo:** Package::findAll(), Package::getStats()
* **Layout:** layouts/base.php
* **APIs:** api/packages/update-status.php, api/packages/delete-package.php

**📊 DATOS A MOSTRAR:**

* Nombre del paquete
* Empresa asociada
* Versión del paquete
* Fecha de creación
* Estado (generando/listo/instalado/error)
* Tamaño del paquete
* Número de archivos incluidos
* Fecha de última descarga

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Tabla con DataTables (paginación, ordenamiento, búsqueda)
* Filtros por empresa
* Filtros por estado del paquete
* Descarga directa de paquetes listos
* Regenerar paquete con contenido actualizado
* Eliminar paquetes obsoletos
* Ver log de instalación

**📈 ESTADÍSTICAS A MOSTRAR:**

* Total de paquetes generados
* Paquetes por estado
* Empresas con paquetes activos
* Tamaño total de todos los paquetes

**🎯 CRITERIOS DE ÉXITO:**

* Lista carga correctamente
* Filtros funcionan
* Descarga de paquetes funciona
* Regeneración funciona
* Estados se actualizan correctamente

**📚 MÓDULO 2.3.2: admin/views/packages/generate.php**

**Propósito:** Formulario para generar nuevos paquetes personalizados

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Formulario multi-paso
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET (mostrar) / POST (generar)
* **Tiempo estimado:** 1 hora 15 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PackageController::generate()
* **Modelo:** Company::findAll(), Content::findAll()
* **API:** api/packages/generate-package.php
* **JavaScript:** Wizard multi-paso, selección de contenido

**📝 PASOS DEL FORMULARIO:**

1. **Paso 1:** Seleccionar empresa
2. **Paso 2:** Configurar branding (colores, logo)
3. **Paso 3:** Seleccionar contenido (películas/música/juegos)
4. **Paso 4:** Configurar WiFi (SSID, contraseña)
5. **Paso 5:** Configurar portal (nombre, configuraciones)
6. **Paso 6:** Revisar y generar

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Wizard con navegación entre pasos
* Validación en cada paso
* Preview de selecciones
* Estimación de tamaño del paquete
* Guardado de borrador
* Generación en background con progress bar

**⚙️ CONFIGURACIONES INCLUIDAS:**

* Branding personalizado por empresa
* Selección específica de contenido
* Configuración WiFi automática
* Settings del portal pasajeros
* Configuración de QR codes

**🎯 CRITERIOS DE ÉXITO:**

* Wizard navega correctamente
* Validaciones funcionan en cada paso
* Preview se muestra correctamente
* Generación se ejecuta en background
* Progress bar funciona

**📚 MÓDULO 2.3.3: admin/views/packages/customize.php**

**Propósito:** Personalizar contenido específico por empresa

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Interfaz de personalización avanzada
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET (mostrar) / POST (guardar)
* **Tiempo estimado:** 50 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PackageController::customize()
* **Modelo:** Company::findById(), Content::getByCompany()
* **APIs:** api/packages/save-customization.php
* **JavaScript:** Drag and drop, preview en tiempo real

**🎨 OPCIONES DE PERSONALIZACIÓN:**

* Orden de contenido en portal
* Contenido destacado en hero banner
* Colores del portal según empresa
* Logo de empresa en portal
* Nombre personalizado del servicio
* Configuración de categorías

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Drag and drop para reordenar contenido
* Preview en tiempo real del portal
* Color picker para branding
* Upload de logo personalizado
* Guardado automático de cambios
* Restaurar configuración por defecto

**👁️ PREVIEW EN TIEMPO REAL:**

* Vista previa del portal con cambios aplicados
* Simulación en dispositivo móvil
* Vista de diferentes secciones del portal

**🎯 CRITERIOS DE ÉXITO:**

* Personalización se aplica correctamente
* Preview funciona en tiempo real
* Guardado automático funciona
* Restaurar defaults funciona

**📚 MÓDULO 2.3.4: admin/views/packages/history.php**

**Propósito:** Historial completo de paquetes por empresa

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de historial con timeline
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 35 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PackageController::history()
* **Modelo:** Package::getHistory(), PackageLog::getByPackage()
* **Layout:** layouts/base.php

**📊 DATOS DEL HISTORIAL:**

* Todas las versiones de paquetes por empresa
* Fechas de generación e instalación
* Cambios entre versiones
* Logs de errores o problemas
* Estadísticas de uso por versión
* Tiempo de generación de cada paquete

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Timeline visual del historial
* Comparación entre versiones
* Descarga de versiones anteriores
* Ver logs detallados de cada versión
* Filtros por empresa y fecha
* Exportar historial a PDF

**📈 MÉTRICAS MOSTRADAS:**

* Frecuencia de actualizaciones por empresa
* Tiempo promedio de generación
* Errores más comunes
* Empresas más activas

**🎯 CRITERIOS DE ÉXITO:**

* Timeline se muestra correctamente
* Comparación entre versiones funciona
* Logs se muestran detalladamente
* Filtros funcionan correctamente

**📚 MÓDULO 2.3.5: api/packages/generate-package.php**

**Propósito:** API para generar paquetes completos en background

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint (proceso pesado)
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 1 hora 30 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PackageController::generatePackage()
* **Modelos:** Package::create(), Company::findById(), Content::findByIds()
* **Utilidades:** ZipArchive, file operations
* **Templates:** packages/templates/

**⚙️ PROCESO DE GENERACIÓN:**

1. **Validar datos** de entrada (empresa, contenido, configuraciones)
2. **Crear estructura** de directorios del paquete
3. **Copiar contenido** seleccionado a directorio temporal
4. **Generar configuraciones** (WiFi, portal, QR codes)
5. **Personalizar portal** con branding de empresa
6. **Crear archivos** de configuración JSON
7. **Comprimir todo** en archivo ZIP
8. **Mover a directorio** final
9. **Actualizar base de datos** con información del paquete
10. **Limpiar archivos** temporales

**📁 ESTRUCTURA DEL PAQUETE GENERADO:**

packages/generated/[empresa-id]/[timestamp]/

├── content/

│ ├── movies/

│ ├── music/

│ └── games/

├── config/

│ ├── wifi-config.json

│ ├── portal-config.json

│ ├── company-branding.json

│ └── qr-codes.json

├── portal/

│ ├── [todos los archivos del portal]

│ └── assets/[personalizados]

├── install/

│ ├── install.sh

│ └── setup-pi.sh

└── package-info.json

**🎯 CRITERIOS DE ÉXITO:**

* Paquete se genera sin errores
* Estructura es correcta y completa
* Configuraciones se aplican bien
* ZIP se crea correctamente
* Base de datos se actualiza
* Archivos temporales se limpian

**📚 MÓDULO 2.3.6: api/packages/download-package.php**

**Propósito:** API para descargar paquetes generados

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint de descarga
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** GET
* **Tiempo estimado:** 20 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PackageController::download()
* **Modelo:** Package::findById(), Package::updateDownloadCount()
* **Sistema:** Headers de descarga, stream de archivos

**⚙️ PROCESO DE DESCARGA:**

1. **Validar permisos** de descarga
2. **Verificar que archivo** existe
3. **Registrar descarga** en base de datos
4. **Configurar headers** para descarga
5. **Stream del archivo** al cliente
6. **Log de actividad**

**🔒 SEGURIDAD:**

* Validación de permisos por empresa
* Rate limiting para descargas
* Verificación de integridad del archivo
* Log de todas las descargas

**🎯 CRITERIOS DE ÉXITO:**

* Descarga inicia correctamente
* Archivo se descarga completo
* Contadores se actualizan
* Logs se registran

**📚 MÓDULO 2.3.7: api/packages/update-status.php**

**Propósito:** API para actualizar estado de paquetes

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 15 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PackageController::updateStatus()
* **Modelo:** Package::updateStatus(), PackageLog::create()

**📝 ESTADOS POSIBLES:**

* **generando:** Paquete en proceso de creación
* **listo:** Paquete completado y disponible
* **descargado:** Paquete descargado por empresa
* **instalado:** Paquete instalado en Raspberry Pi
* **activo:** Paquete funcionando en producción
* **error:** Error en generación o instalación

**⚙️ PROCESO DE ACTUALIZACIÓN:**

1. **Validar nuevo estado** es válido
2. **Verificar permisos** de cambio
3. **Actualizar estado** en base de datos
4. **Registrar cambio** en log
5. **Notificar cambio** si es necesario

**🎯 CRITERIOS DE ÉXITO:**

* Estado se actualiza correctamente
* Logs se registran
* Notificaciones se envían si aplica

**📚 MÓDULO 2.3.8: packages/templates/default-config.json**

**Propósito:** Template base para configuración de paquetes

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Archivo de configuración JSON
* **Propósito:** Template base que se personaliza por empresa
* **Tiempo estimado:** 25 minutos

**🔗 ARCHIVOS RELACIONADOS:**

* **Usado por:** api/packages/generate-package.php
* **Personalizado con:** Datos de Company y configuraciones específicas

**📝 ESTRUCTURA DEL TEMPLATE:**

json

{

"package\_info": {

"version": "1.0.0",

"generated\_at": "TIMESTAMP",

"company\_id": "COMPANY\_ID",

"package\_type": "standard"

},

"wifi\_config": {

"ssid": "COMPANY\_WIFI\_NAME",

"password": "WIFI\_PASSWORD",

"security": "WPA2",

"hidden": false

},

"portal\_config": {

"company\_name": "COMPANY\_NAME",

"service\_name": "SERVICE\_NAME",

"primary\_color": "PRIMARY\_COLOR",

"secondary\_color": "SECONDARY\_COLOR",

"logo\_path": "LOGO\_PATH",

"welcome\_message": "WELCOME\_MESSAGE"

},

"content\_config": {

"movies\_enabled": true,

"music\_enabled": true,

"games\_enabled": true,

"max\_concurrent\_users": 50

},

"system\_config": {

"auto\_sync": true,

"sync\_interval": 3600,

"log\_level": "INFO",

"analytics\_enabled": true

}

}

**🎯 CRITERIOS DE ÉXITO:**

* Template es válido JSON
* Todos los placeholders están definidos
* Estructura es completa
* Se personaliza correctamente

**🚀 FASE 3: SISTEMA QR Y CONFIGURACIÓN WIFI**

**📋 DIVISIÓN EN MÓDULOS:**

**FASE 3 se divide en 5 MÓDULOS específicos:**

**📚 MÓDULO 3.1: admin/views/qr-system/index.php**

**Propósito:** Gestión central de códigos QR por empresa y bus

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de gestión con tabla y generación
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** QRController::index()
* **Modelo:** QRCode::findAll(), Company::findAll()
* **Layout:** layouts/base.php
* **APIs:** api/qr/generate-qr.php

**📊 DATOS A MOSTRAR:**

* Lista de QR codes generados
* Empresa asociada
* Número de bus
* SSID del WiFi
* URL del portal
* Fecha de generación
* Estado (activo/inactivo)
* Número de escaneos

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Tabla con filtros por empresa
* Generar QR individual o masivo
* Preview del QR code
* Descargar QR para impresión
* Estadísticas de uso por QR
* Activar/desactivar QR codes

**📈 ESTADÍSTICAS MOSTRADAS:**

* Total de QR codes generados
* QR codes activos por empresa
* Escaneos totales hoy/semana/mes
* QR codes más utilizados

**🎯 CRITERIOS DE ÉXITO:**

* Lista se muestra correctamente
* Generación funciona sin errores
* Preview se muestra bien
* Descarga funciona
* Estadísticas son precisas

**📚 MÓDULO 3.2: admin/views/qr-system/generate.php**

**Propósito:** Formulario para generar QR codes personalizados

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Formulario de generación con preview
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET (mostrar) / POST (generar)
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** QRController::generate()
* **Modelo:** Company::findAll(), QRCode::create()
* **API:** api/qr/generate-qr.php
* **Librería:** QR code generator (PHP)

**📝 CAMPOS DEL FORMULARIO:**

* Selección de empresa
* Número de bus
* SSID personalizado (opcional)
* Contraseña WiFi personalizada
* URL del portal (auto-generada)
* Tamaño del QR code
* Nivel de corrección de errores

**🎛️ FUNCIONALIDADES REQUERIDAS:**

* Preview en tiempo real del QR
* Validación de SSID y contraseña
* Generación masiva para flota completa
* Diferentes formatos de descarga (PNG, SVG, PDF)
* Configuración avanzada de WiFi

**📋 TIPOS DE QR GENERADOS:**

* **QR WiFi:** Conexión automática al WiFi
* **QR Portal:** Directo al portal web
* **QR Combinado:** WiFi + Portal en secuencia

**🎯 CRITERIOS DE ÉXITO:**

* Preview se actualiza en tiempo real
* Validaciones funcionan correctamente
* Generación masiva funciona
* Diferentes formatos se descargan bien

**📚 MÓDULO 3.3: admin/views/qr-system/print.php**

**Propósito:** Página optimizada para impresión de QR codes

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de impresión sin layout
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 25 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** QRController::print()
* **Modelo:** QRCode::findById(), Company::findById()
* **CSS:** Estilos específicos para impresión

**🖨️ LAYOUT DE IMPRESIÓN:**

* QR code grande y centrado
* Información de la empresa
* Instrucciones para el pasajero
* Número de bus si aplica
* Información de contacto

**📐 CONFIGURACIONES DE IMPRESIÓN:**

* Tamaño estándar (A4, Carta)
* Múltiples QR por página
* Versión en blanco y negro
* Versión con colores de empresa

**🎯 CRITERIOS DE ÉXITO:**

* Página se imprime correctamente
* QR code es escaneable en impresión
* Layout es profesional
* Instrucciones son claras

**📚 MÓDULO 3.4: api/qr/generate-qr.php**

**Propósito:** API para generar códigos QR con configuración WiFi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** POST únicamente
* **Tiempo estimado:** 50 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** QRController::generateQR()
* **Modelo:** QRCode::create(), Company::findById()
* **Librería:** PHP QR Code generator
* **Destino:** companies/[id]/qr-codes/

**⚙️ PROCESO DE GENERACIÓN:**

1. **Validar datos** de entrada (empresa, configuración WiFi)
2. **Generar string** WiFi: WIFI:T:WPA;S:SSID;P:password;H:false;;
3. **Agregar URL** del portal al string
4. **Generar QR code** con librería
5. **Guardar imagen** en directorio de empresa
6. **Registrar en BD** con metadatos
7. **Retornar información** del QR generado

**📱 FORMATO DEL QR WIFI:**

WIFI:T:WPA;S:[SSID];P:[PASSWORD];H:false;;

URL:[PORTAL\_URL]

**🎨 PERSONALIZACIÓN:**

* Colores de empresa en QR
* Logo de empresa en centro (opcional)
* Diferentes niveles de corrección
* Tamaños personalizados

**🎯 CRITERIOS DE ÉXITO:**

* QR se genera correctamente
* WiFi se conecta automáticamente
* URL del portal se abre automáticamente
* Archivo se guarda en ubicación correcta
* Base de datos se actualiza

**📚 MÓDULO 3.5: api/qr/wifi-config.php**

**Propósito:** API para generar configuraciones WiFi personalizadas

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** Requerida (token/sesión)
* **Método HTTP:** GET/POST
* **Tiempo estimado:** 30 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** QRController::wifiConfig()
* **Modelo:** Company::getWifiConfig()
* **Templates:** WiFi configuration templates

**📝 CONFIGURACIONES GENERADAS:**

* Configuración para Android
* Configuración para iOS
* Configuración para Windows
* Archivo de configuración para routers

**⚙️ TIPOS DE CONFIGURACIÓN:**

1. **Automática:** SSID basado en nombre de empresa
2. **Personalizada:** SSID y contraseña específicos
3. **Segura:** Contraseñas generadas aleatoriamente
4. **Temporal:** Configuraciones con expiración

**🔒 SEGURIDAD:**

* Contraseñas seguras generadas automáticamente
* Rotación periódica de contraseñas
* Configuración de seguridad WPA2/WPA3

**🎯 CRITERIOS DE ÉXITO:**

* Configuraciones se generan correctamente
* Son compatibles con todos los dispositivos
* Seguridad es apropiada
* Se integran bien con QR codes

**🚀 FASE 4: PORTAL WEB PARA PASAJEROS (TIPO NETFLIX)**

**📋 DIVISIÓN EN MÓDULOS:**

**FASE 4 se divide en 12 MÓDULOS específicos:**

**📚 MÓDULO 4.1: passenger-portal/index.php**

**Propósito:** Página principal estilo Netflix para pasajeros

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página principal SPA (Single Page Application)
* **Autenticación:** No requerida (acceso público local)
* **Método HTTP:** GET
* **Tiempo estimado:** 1 hora 30 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PortalController::index()
* **Modelo:** Content::getActiveContent()
* **APIs:** api/get-content.php, api/company-branding.php
* **Assets:** netflix-style.css, portal-main.js

**🎨 DISEÑO ESTILO NETFLIX:**

* **Hero Banner:** Contenido destacado rotativo
* **Carruseles:** Películas populares, música trending, juegos nuevos
* **Navegación:** Menú superior con categorías
* **Responsive:** Optimizado para móviles y tablets
* **Branding:** Colores y logo personalizado por empresa

**📱 SECCIONES PRINCIPALES:**

1. **Header:** Logo empresa, menú navegación, búsqueda
2. **Hero Banner:** Contenido destacado con trailer/preview
3. **Películas:** Carrusel horizontal con posters
4. **Música:** Grid con carátulas y artistas
5. **Juegos:** Grid con screenshots atractivos
6. **Footer:** Información de la empresa y soporte

**🎛️ FUNCIONALIDADES INTERACTIVAS:**

* Carruseles deslizables tactiles
* Hover effects en contenido
* Search bar con resultados en vivo
* Filtros rápidos por categoría
* Favoritos (localStorage)
* Contenido recientemente visto

**📊 DATOS DINÁMICOS:**

* Contenido personalizado por empresa
* Estadísticas de popularidad
* Contenido agregado recientemente
* Recomendaciones basadas en uso

**🎯 CRITERIOS DE ÉXITO:**

* Carga rápida (< 3 segundos)
* Navegación fluida en táctil
* Responsive en todos los dispositivos
* Branding se aplica correctamente
* Contenido se carga dinámicamente

**📚 MÓDULO 4.2: passenger-portal/movies.php**

**Propósito:** Catálogo completo de películas estilo Netflix

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de catálogo con grid responsive
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 50 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PortalController::movies()
* **Modelo:** Content::getMovies()
* **APIs:** api/get-content.php?type=movies
* **Player:** player/video-player.php

**🎬 LAYOUT DE PELÍCULAS:**

* **Filtros superiores:** Género, año, duración
* **Grid responsive:** 2-6 columnas según dispositivo
* **Cada película:** Poster, título, duración, rating
* **Hover/Tap:** Preview con descripción y botón play
* **Paginación:** Infinite scroll o páginas

**🎭 INFORMACIÓN POR PELÍCULA:**

* Poster de alta calidad
* Título y subtítulo
* Duración y año
* Género y clasificación
* Descripción corta
* Rating/estrellas

**🔍 FUNCIONALIDADES:**

* Búsqueda en tiempo real
* Filtros múltiples combinables
* Ordenamiento (popular, reciente, A-Z)
* Vista en lista o grid
* Favoritos personales

**🎯 CRITERIOS DE ÉXITO:**

* Grid es responsive y rápido
* Filtros funcionan sin recargar
* Búsqueda es instantánea
* Player se abre correctamente
* Navegación es intuitiva

**📚 MÓDULO 4.3: passenger-portal/music.php**

**Propósito:** Reproductor de música estilo YouTube/Spotify

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de reproductor de música
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 1 hora 15 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PortalController::music()
* **Modelo:** Content::getMusic()
* **Player:** player/music-player.php
* **APIs:** api/get-content.php?type=music

**🎵 LAYOUT DE MÚSICA:**

* **Player principal:** En parte superior fija
* **Lista de canciones:** Tabla con artista, álbum, duración
* **Sidebar:** Playlists, favoritos, recientes
* **Visualización:** Carátula grande del álbum actual
* **Controles:** Play/pause, siguiente, anterior, volumen

**🎼 FUNCIONALIDADES DEL PLAYER:**

* Reproducción continua
* Lista de reproducción dinámica
* Shuffle y repeat
* Control de volumen
* Barra de progreso interactiva
* Información del track actual

**📱 CONTROLES TÁCTILES:**

* Botones grandes para móviles
* Gestures para cambiar canción
* Control de volumen por gestos
* Lock screen controls

**🎯 CRITERIOS DE ÉXITO:**

* Player funciona sin interrupciones
* Controles son responsive táctiles
* Lista de reproducción funciona
* Audio se reproduce sin cortes
* Interfaz es intuitiva

**📚 MÓDULO 4.4: passenger-portal/games.php**

**Propósito:** Catálogo de juegos HTML5 con launcher

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Galería de juegos con launcher integrado
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PortalController::games()
* **Modelo:** Content::getGames()
* **Launcher:** games/launcher.php
* **APIs:** api/games/load-game.php

**🎮 LAYOUT DE JUEGOS:**

* **Grid de juegos:** Screenshots atractivos
* **Categorías:** Puzzle, Arcade, Estrategia, Cartas
* **Cada juego:** Screenshot, título, descripción, rating
* **Preview:** Modal con información e instrucciones
* **Launcher:** Iframe o ventana nueva

**🕹️ INFORMACIÓN POR JUEGO:**

* Screenshot o GIF animado
* Título del juego
* Categoría y dificultad
* Instrucciones básicas
* Controles necesarios
* Tiempo estimado de juego

**🎪 FUNCIONALIDADES:**

* Filtros por categoría
* Búsqueda de juegos
* Favoritos personales
* Leaderboard si aplica
* Fullscreen para juegos

**🎯 CRITERIOS DE ÉXITO:**

* Grid se carga rápidamente
* Juegos se lanzan sin errores
* Fullscreen funciona correctamente
* Controles son claros
* Navegación entre juegos es fluida

**📚 MÓDULO 4.5: passenger-portal/player/video-player.php**

**Propósito:** Reproductor de video optimizado para móviles

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Player de video HTML5 personalizado
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 1 hora

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PlayerController::video()
* **Modelo:** Content::getMovie()
* **APIs:** api/track-usage.php
* **Assets:** video-controls.js, player.css

**📺 CARACTERÍSTICAS DEL PLAYER:**

* **HTML5 Video:** Compatibilidad universal
* **Controles personalizados:** Grandes para táctil
* **Fullscreen:** Orientación automática
* **Calidad adaptativa:** Según ancho de banda
* **Subtítulos:** Si están disponibles

**🎛️ CONTROLES INCLUIDOS:**

* Play/Pause (botón grande central)
* Barra de progreso (draggable)
* Control de volumen
* Botón fullscreen
* Botón configuración
* Tiempo actual/total

**📱 OPTIMIZACIONES MÓVILES:**

* Controles auto-hide
* Tap para mostrar/ocultar controles
* Pinch to zoom en fullscreen
* Orientación landscape automática
* Prevención de sleep del dispositivo

**📊 TRACKING:**

* Tiempo de visualización
* Pausas y abandono
* Calidad de reproducción
* Errores de buffering

**🎯 CRITERIOS DE ÉXITO:**

* Video se reproduce sin buffering
* Controles son táctiles y responsivos
* Fullscreen funciona correctamente
* Calidad se adapta automáticamente
* Tracking registra correctamente

**📚 MÓDULO 4.6: passenger-portal/player/music-player.php**

**Propósito:** Reproductor de audio con controles avanzados

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Player de audio HTML5 con visualizaciones
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 50 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PlayerController::music()
* **Modelo:** Content::getMusic()
* **Assets:** music-controls.js, audio-visualizer.js

**🎵 CARACTERÍSTICAS DEL PLAYER:**

* **Audio HTML5:** Alta calidad de reproducción
* **Visualizador:** Barras de frecuencia animadas
* **Metadata:** Artista, álbum, duración
* **Artwork:** Carátula del álbum grande
* **Playlist:** Cola de reproducción visible

**🎚️ CONTROLES AVANZADOS:**

* Play/Pause/Stop
* Anterior/Siguiente
* Shuffle/Repeat
* Control de volumen con mute
* Ecualizador básico (opcional)
* Speed control

**📱 DISEÑO RESPONSIVO:**

* Layout vertical para móviles
* Controles grandes táctiles
* Gestures para navegación
* Background play (si es posible)

**🎯 CRITERIOS DE ÉXITO:**

* Audio se reproduce con calidad
* Visualizador funciona fluido
* Controles responden inmediatamente
* Playlist navega correctamente
* Diseño es atractivo

**📚 MÓDULO 4.7: passenger-portal/games/launcher.php**

**Propósito:** Launcher universal para juegos HTML5

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Iframe sandbox para juegos
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** GamesController::launch()
* **Modelo:** Content::getGame()
* **APIs:** api/games/load-game.php, api/games/save-score.php
* **Assets:** games-manager.js

**🎮 CARACTERÍSTICAS DEL LAUNCHER:**

* **Iframe seguro:** Sandbox con permisos limitados
* **Fullscreen:** Modo pantalla completa
* **Comunicación:** PostMessage con portal principal
* **Error handling:** Manejo de juegos que no cargan
* **Exit button:** Volver al portal fácilmente

**🛡️ SEGURIDAD:**

* Sandbox attributes apropiados
* Dominio aislado para juegos
* Validación de contenido HTML5
* Prevención de scripts maliciosos

**📊 INTEGRACIÓN:**

* Tracking de tiempo de juego
* Guardado de puntuaciones
* Comunicación de eventos
* Manejo de errores de carga

**🎯 CRITERIOS DE ÉXITO:**

* Juegos cargan correctamente en iframe
* Fullscreen funciona sin errores
* Comunicación portal-juego funciona
* Exit button siempre accesible
* No hay vulnerabilidades de seguridad

**📚 MÓDULO 4.8: passenger-portal/api/get-content.php**

**Propósito:** API para obtener contenido dinámico del portal

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API REST endpoint
* **Autenticación:** No requerida (local)
* **Método HTTP:** GET
* **Tiempo estimado:** 35 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PortalController::getContent()
* **Modelo:** Content::getActiveContent()
* **Cache:** Sistema de cache local

**📋 PARÁMETROS SOPORTADOS:**

* type: movies/music/games/all
* category: Categoría específica
* limit: Número de resultados
* offset: Para paginación
* search: Término de búsqueda
* featured: Solo contenido destacado

**📊 RESPUESTA JSON:**

json

{

"success": true,

"data": [

{

"id": "movie\_123",

"title": "Título",

"type": "movie",

"thumbnail": "url\_thumbnail",

"duration": 7200,

"metadata": {...}

}

],

"pagination": {

"total": 150,

"page": 1,

"per\_page": 20

}

}

**⚡ OPTIMIZACIONES:**

* Cache de respuestas frecuentes
* Compresión JSON
* Lazy loading de thumbnails
* Minificación de metadata

**🎯 CRITERIOS DE ÉXITO:**

* Respuesta rápida (< 200ms)
* JSON bien formateado
* Filtros funcionan correctamente
* Paginación funciona
* Cache mejora rendimiento

**📚 MÓDULO 4.9: passenger-portal/api/track-usage.php**

**Propósito:** API para tracking de uso de contenido

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint para analytics
* **Autenticación:** No requerida
* **Método HTTP:** POST
* **Tiempo estimado:** 25 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** AnalyticsController::track()
* **Modelo:** UsageLog::create()

**📊 DATOS TRACKED:**

* Contenido reproducido/jugado
* Tiempo de uso/reproducción
* Tipo de dispositivo
* Momento del día
* Abandonos prematuros
* Errores de reproducción

**📝 EVENTOS SOPORTADOS:**

* content\_view: Ver contenido
* content\_play: Reproducir contenido
* content\_pause: Pausar reproducción
* content\_complete: Completar contenido
* content\_error: Error en reproducción
* search\_query: Búsqueda realizada

**⚡ PROCESAMIENTO:**

* Guardado asíncrono
* Batch processing para múltiples eventos
* Limpieza automática de datos antiguos

**🎯 CRITERIOS DE ÉXITO:**

* Tracking no afecta rendimiento
* Datos se guardan correctamente
* No hay pérdida de eventos
* Procesamiento es eficiente

**📚 MÓDULO 4.10: passenger-portal/api/company-branding.php**

**Propósito:** API para obtener branding personalizado por empresa

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 20 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** BrandingController::get()
* **Modelo:** Company::getBranding()

**🎨 DATOS DE BRANDING:**

* Colores primario y secundario
* Logo de la empresa
* Nombre del servicio
* Mensaje de bienvenida
* Configuraciones de tema
* Assets personalizados

**📱 RESPUESTA JSON:**

json

{

"success": true,

"branding": {

"company\_name": "Transportes ABC",

"service\_name": "EntretenimientoABC",

"primary\_color": "#FF6B35",

"secondary\_color": "#F7931E",

"logo\_url": "url\_logo",

"welcome\_message": "Bienvenido a bordo",

"theme": "light"

}

}

**💨 CACHE:**

* Cache agresivo (24 horas)
* CDN para assets si disponible
* Compresión de respuestas

**🎯 CRITERIOS DE ÉXITO:**

* Branding se aplica correctamente
* Colores se actualizan dinámicamente
* Logo se carga rápidamente
* Cache funciona eficientemente

**📚 MÓDULO 4.11: passenger-portal/assets/css/netflix-style.css**

**Propósito:** Estilos principales estilo Netflix para el portal

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Archivo CSS principal
* **Responsivo:** Mobile-first design
* **Tiempo estimado:** 1 hora 30 minutos

**🎨 COMPONENTES PRINCIPALES:**

* **Header:** Navegación fija superior
* **Hero Banner:** Sección destacada
* **Carousels:** Deslizables horizontales
* **Grid:** Responsive para catálogos
* **Modals:** Para detalles de contenido
* **Player:** Controles de reproducción

**📱 BREAKPOINTS:**

* Mobile: 320px - 767px
* Tablet: 768px - 1023px
* Desktop: 1024px+

**🎭 ANIMACIONES:**

* Hover effects suaves
* Transiciones de página
* Loading animations
* Carousel slide effects

**🌈 VARIABLES CSS:**

css

:root {

--primary-color: */\* Dinámico por empresa \*/*;

--secondary-color: */\* Dinámico por empresa \*/*;

--background-dark: #141414;

--text-light: #ffffff;

--accent-red: #e50914;

}

**🎯 CRITERIOS DE ÉXITO:**

* Responsive en todos los dispositivos
* Animaciones fluidas (60fps)
* Colores se personalizan correctamente
* Carga rápida de estilos
* Compatibilidad cross-browser

**📚 MÓDULO 4.12: passenger-portal/assets/js/portal-main.js**

**Propósito:** JavaScript principal del portal con todas las funcionalidades

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Archivo JavaScript principal
* **Dependencias:** Vanilla JS (sin frameworks)
* **Tiempo estimado:** 2 horas

**🔗 FUNCIONALIDADES PRINCIPALES:**

* Inicialización del portal
* Carga dinámica de contenido
* Navegación SPA
* Gestión de reproducción
* Touch gestures
* Search functionality

**📱 FEATURES MÓVILES:**

* Touch/swipe gestures
* Orientación responsive
* Fullscreen management
* Device-specific optimizations

**⚡ OPTIMIZACIONES:**

* Lazy loading de contenido
* Image optimization
* Debounced search
* Efficient DOM manipulation

**🎮 INTEGRACIÓN CON OTROS MÓDULOS:**

* Video player controls
* Music player management
* Games launcher
* Analytics tracking

**🎯 CRITERIOS DE ÉXITO:**

* Navegación fluida sin recargas
* Gestos táctiles funcionan perfectamente
* Búsqueda es instantánea
* No hay memory leaks
* Performance es óptimo en móviles

**🚀 FASE 5: SISTEMA DE JUEGOS HTML5**

**📋 DIVISIÓN EN MÓDULOS:**

**FASE 5 se divide en 6 MÓDULOS específicos:**

**📚 MÓDULO 5.1: passenger-portal/games/iframe-sandbox.php**

**Propósito:** Sandbox seguro para ejecutar juegos HTML5

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Contenedor iframe con sandboxing
* **Seguridad:** Máxima restricción con permisos específicos
* **Método HTTP:** GET
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** GameSandboxController::load()
* **Modelo:** Content::getGame()
* **APIs:** api/games/load-game.php
* **Security:** Sandbox policies

**🛡️ CARACTERÍSTICAS DE SEGURIDAD:**

* **Sandbox attributes:** allow-scripts allow-same-origin allow-forms
* **CSP Headers:** Content Security Policy estricta
* **Domain isolation:** Subdomain para juegos
* **Resource limits:** CPU y memoria limitados
* **No external access:** Sin acceso a APIs externas

**🎮 FUNCIONALIDADES:**

* Carga dinámica de juegos
* Comunicación vía postMessage
* Manejo de errores de carga
* Timeout para juegos que no responden
* Exit/Back button siempre visible

**📱 RESPONSIVE:**

* Fullscreen automático en móviles
* Orientación landscape para juegos que la requieren
* Touch optimization
* Performance monitoring

**🎯 CRITERIOS DE ÉXITO:**

* Juegos cargan sin problemas de seguridad
* Sandbox previene accesos no autorizados
* Comunicación bidireccional funciona
* Performance es adecuado
* No hay vulnerabilidades

**📚 MÓDULO 5.2: passenger-portal/games/leaderboard.php**

**Propósito:** Sistema de puntuaciones y rankings

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de leaderboard con rankings
* **Autenticación:** No requerida (anonymous scores)
* **Método HTTP:** GET
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** LeaderboardController::index()
* **Modelo:** GameScore::getTopScores()
* **APIs:** api/games/get-leaderboard.php

**🏆 CARACTERÍSTICAS:**

* **Rankings globales:** Top scores de todos los juegos
* **Rankings por juego:** Mejores puntuaciones por juego específico
* **Rankings temporales:** Hoy, semana, mes, histórico
* **Nombres anónimos:** Sin identificación personal
* **Animaciones:** Efectos visuales para nuevos récords

**📊 DATOS MOSTRADOS:**

* Posición en ranking
* Nombre/inicial del jugador
* Puntuación obtenida
* Juego específico
* Fecha del récord
* Badge especial para top 3

**🎮 FUNCIONALIDADES:**

* Filtros por juego
* Filtros por período de tiempo
* Búsqueda por nombre
* Destacar récords recientes
* Comparación con puntuación personal

**🎯 CRITERIOS DE ÉXITO:**

* Rankings se actualizan en tiempo real
* Filtros funcionan correctamente
* Animaciones son atractivas
* Performance es bueno
* Datos son precisos

**📚 MÓDULO 5.3: api/games/load-game.php**

**Propósito:** API para cargar juegos HTML5 específicos

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 35 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** GamesController::loadGame()
* **Modelo:** Content::getGame()
* **File system:** content/games/extracted/[game-id]/

**⚙️ PROCESO DE CARGA:**

1. **Validar game ID** existe y está activo
2. **Verificar archivos** del juego están completos
3. **Generar URL base** para recursos del juego
4. **Preparar configuración** del juego
5. **Retornar metadata** y URLs necesarias

**📋 RESPUESTA JSON:**

json

{

"success": true,

"game": {

"id": "game\_123",

"title": "Tetris Clone",

"base\_url": "/content/games/extracted/game\_123/",

"entry\_point": "index.html",

"config": {

"controls": ["keyboard", "touch"],

"orientation": "portrait",

"fullscreen": true

}

}

}

**🎮 VALIDACIONES:**

* Verificar que index.html existe
* Validar estructura de directorios
* Comprobar que no hay archivos maliciosos
* Verificar tamaño total del juego

**🎯 CRITERIOS DE ÉXITO:**

* Juegos cargan rápidamente
* Validaciones previenen errores
* URLs generadas son correctas
* Metadata es completa y precisa

**📚 MÓDULO 5.4: api/games/save-score.php**

**Propósito:** API para guardar puntuaciones de juegos

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** No requerida (anonymous)
* **Método HTTP:** POST
* **Tiempo estimado:** 30 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** GamesController::saveScore()
* **Modelo:** GameScore::create()
* **Validation:** Score validation rules

**📝 DATOS REQUERIDOS:**

* Game ID
* Score (número)
* Player name/initials (opcional)
* Game duration (tiempo jugado)
* Level reached (si aplica)
* Timestamp

**🛡️ VALIDACIONES:**

* Score debe ser numérico positivo
* Game ID debe existir
* Rate limiting (no spam)
* Validación de score realista
* Filtro de nombres inapropiados

**⚡ PROCESAMIENTO:**

* Guardado inmediato
* Verificar si es récord
* Actualizar rankings
* Limpiar scores antiguos
* Notificar nuevo récord si aplica

**🎯 CRITERIOS DE ÉXITO:**

* Scores se guardan correctamente
* Validaciones previenen fraude
* Rankings se actualizan inmediatamente
* No hay duplicados
* Performance es bueno

**📚 MÓDULO 5.5: api/games/get-leaderboard.php**

**Propósito:** API para obtener rankings y leaderboards

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint
* **Autenticación:** No requerida
* **Método HTTP:** GET
* **Tiempo estimado:** 25 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** LeaderboardController::get()
* **Modelo:** GameScore::getLeaderboard()
* **Cache:** Rankings cacheados

**📋 PARÁMETROS:**

* game\_id: Específico juego o 'all'
* period: 'today', 'week', 'month', 'all'
* limit: Número de resultados (default 10)
* offset: Para paginación

**📊 RESPUESTA JSON:**

json

{

"success": true,

"leaderboard": [

{

"rank": 1,

"player\_name": "AAA",

"score": 15000,

"game\_title": "Tetris",

"date": "2024-08-12",

"is\_recent": true

}

],

"meta": {

"total\_players": 150,

"period": "week",

"updated\_at": "2024-08-12 10:30:00"

}

}

**⚡ OPTIMIZACIONES:**

* Cache de 5 minutos para rankings
* Query optimization para grandes datasets
* Pagination eficiente
* Índices de base de datos optimizados

**🎯 CRITERIOS DE ÉXITO:**

* Rankings son precisos y actualizados
* Response time < 200ms
* Paginación funciona correctamente
* Cache mejora performance

**📚 MÓDULO 5.6: content/games/extracted/[game-id]/game-config.json**

**Propósito:** Archivo de configuración estándar para juegos HTML5

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Archivo de configuración JSON
* **Propósito:** Metadata y configuración de cada juego
* **Tiempo estimado:** 20 minutos

**📝 ESTRUCTURA ESTÁNDAR:**

json

{

"game\_info": {

"title": "Nombre del Juego",

"version": "1.0.0",

"author": "Desarrollador",

"description": "Descripción del juego",

"category": "puzzle|arcade|strategy|cards",

"difficulty": "easy|medium|hard",

"estimated\_play\_time": 300

},

"technical": {

"entry\_point": "index.html",

"required\_files": ["index.html", "game.js", "assets/"],

"supports\_touch": true,

"supports\_keyboard": true,

"preferred\_orientation": "portrait|landscape|any",

"fullscreen\_recommended": true,

"minimum\_screen\_size": "320x480"

},

"gameplay": {

"has\_scores": true,

"has\_levels": true,

"multiplayer": false,

"save\_progress": false,

"controls\_info": "Tap to jump, swipe to move"

},

"assets": {

"thumbnail": "assets/thumbnail.png",

"icon": "assets/icon.png",

"screenshots": ["assets/screen1.png", "assets/screen2.png"]

}

}

**🎮 USOS DEL CONFIG:**

* Validación de estructura del juego
* Configuración del launcher
* Información para el catálogo
* Optimizaciones específicas
* Control de características

**🎯 CRITERIOS DE ÉXITO:**

* JSON es válido y completo
* Información es precisa
* Se usa en todo el pipeline
* Facilita integración de juegos

**🚀 FASE 6: SISTEMA RASPBERRY PI**

**📋 DIVISIÓN EN MÓDULOS:**

**FASE 6 se divide en 8 MÓDULOS específicos:**

**📚 MÓDULO 6.1: pi-system/install/setup.sh**

**Propósito:** Script maestro de instalación completa del sistema Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Bash script de instalación
* **Plataforma:** Raspberry Pi OS (Debian)
* **Privilegios:** Requiere sudo
* **Tiempo estimado:** 1 hora 30 minutos

**🔗 DEPENDENCIAS:**

* **Scripts llamados:** wifi-ap.sh, web-server.sh, auto-start.sh
* **Archivos config:** Todos los archivos en pi-system/config/
* **Paquetes:** Apache, PHP, hostapd, dnsmasq

**⚙️ PROCESO DE INSTALACIÓN:**

1. **Verificar sistema** (Raspberry Pi, Debian, conectividad)
2. **Actualizar paquetes** del sistema
3. **Instalar dependencias** necesarias
4. **Configurar WiFi AP** (llamar wifi-ap.sh)
5. **Instalar servidor web** (llamar web-server.sh)
6. **Copiar archivos** del portal
7. **Configurar permisos** de archivos
8. **Configurar auto-inicio** (llamar auto-start.sh)
9. **Configurar firewall** básico
10. **Verificar instalación** completa
11. **Crear script** de desinstalación
12. **Generar reporte** de instalación

**🛡️ VALIDACIONES:**

* Verificar que es Raspberry Pi
* Comprobar espacio en disco suficiente
* Validar conectividad a internet
* Verificar permisos de sudo
* Comprobar que puertos están libres

**📝 LOGS:**

* Log detallado en /var/log/playmi-install.log
* Reporte de errores específicos
* Timestamp de cada paso
* Información de rollback si falla

**🎯 CRITERIOS DE ÉXITO:**

* Instalación completa sin errores
* WiFi AP funciona inmediatamente
* Portal web es accesible
* Auto-inicio configurado
* Logs son claros y útiles

**📚 MÓDULO 6.2: pi-system/install/wifi-ap.sh**

**Propósito:** Configurar Raspberry Pi como WiFi Access Point

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Bash script de configuración WiFi
* **Servicios:** hostapd, dnsmasq
* **Tiempo estimado:** 45 minutos

**🔗 ARCHIVOS DE CONFIGURACIÓN:**

* **hostapd.conf:** Configuración del hotspot WiFi
* **dnsmasq.conf:** Configuración DHCP y DNS
* **dhcpcd.conf:** Configuración de interfaces de red

**📡 CONFIGURACIÓN WIFI:**

* **SSID:** Personalizable por empresa
* **Password:** WPA2 personalizable
* **Canal:** Auto-selección o manual
* **Rango IP:** 192.168.4.1/24 (configurable)
* **DHCP:** Rango 192.168.4.2-192.168.4.20
* **DNS:** Redirigir todo tráfico al portal

**⚙️ PROCESO DE CONFIGURACIÓN:**

1. **Instalar paquetes** hostapd y dnsmasq
2. **Configurar interfaz** wlan0 como estática
3. **Configurar hostapd** con SSID y password
4. **Configurar dnsmasq** para DHCP y DNS
5. **Configurar iptables** para NAT (opcional)
6. **Habilitar servicios** en boot
7. **Reiniciar servicios** de red
8. **Verificar funcionamiento**

**🔧 TROUBLESHOOTING:**

* Backup de configuraciones originales
* Rollback automático si falla
* Verificación de hardware WiFi
* Test de conectividad

**🎯 CRITERIOS DE ÉXITO:**

* WiFi AP visible desde dispositivos
* Conexión automática funciona
* DHCP asigna IPs correctamente
* DNS redirige al portal

**📚 MÓDULO 6.3: pi-system/install/web-server.sh**

**Propósito:** Instalar y configurar servidor web Apache con PHP

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Bash script de instalación web
* **Stack:** Apache 2.4 + PHP 7.4+
* **Tiempo estimado:** 30 minutos

**🔗 COMPONENTES INSTALADOS:**

* **Apache2:** Servidor web principal
* **PHP:** Con módulos necesarios
* **Módulos Apache:** rewrite, ssl, headers
* **Utilidades:** curl, wget, unzip

**⚙️ CONFIGURACIÓN APACHE:**

* **DocumentRoot:** /var/www/playmi/
* **Virtual Host:** Portal de pasajeros
* **URL Rewriting:** Pretty URLs habilitado
* **Security headers:** Configuraciones básicas
* **Error pages:** Páginas personalizadas
* **Compression:** Gzip habilitado

**🔧 OPTIMIZACIONES:**

* **Memory limit:** Ajustado para Pi
* **Max execution time:** Optimizado
* **File uploads:** Configurado para contenido
* **Cache headers:** Para assets estáticos
* **KeepAlive:** Configurado apropiadamente

**📁 ESTRUCTURA DE DIRECTORIOS:**

/var/www/playmi/

├── portal/ # Portal de pasajeros

├── content/ # Contenido multimedia

├── logs/ # Logs del sistema

└── temp/ # Archivos temporales

**🎯 CRITERIOS DE ÉXITO:**

* Apache funciona y responde
* PHP ejecuta correctamente
* Portal es accesible vía HTTP
* Permisos de archivos son correctos
* Performance es adecuado para Pi

**📚 MÓDULO 6.4: pi-system/install/auto-start.sh**

**Propósito:** Configurar inicio automático de todos los servicios

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Bash script de configuración de servicios
* **Sistema:** systemd (Raspberry Pi OS)
* **Tiempo estimado:** 20 minutos

**🔗 SERVICIOS CONFIGURADOS:**

* **WiFi AP:** hostapd, dnsmasq
* **Web server:** apache2
* **Sync script:** Sincronización automática
* **Health check:** Monitoreo del sistema
* **Log rotation:** Gestión de logs

**⚙️ CONFIGURACIÓN SYSTEMD:**

* **Servicios habilitados** en boot
* **Dependencias** entre servicios configuradas
* **Restart policies** para recuperación automática
* **Timeouts** apropiados
* **User permissions** correctos

**📝 SCRIPTS DE INICIO:**

* Script de health check cada 5 minutos
* Script de sincronización cada hora
* Script de limpieza de logs semanal
* Script de backup de configuración

**🔧 MONITOREO:**

* Verificación de servicios críticos
* Alertas por email si está configurado
* Logs centralizados
* Métricas básicas de sistema

**🎯 CRITERIOS DE ÉXITO:**

* Todos los servicios inician automáticamente
* Servicios se recuperan de errores
* Monitoreo funciona correctamente
* Sistema es estable después de reboot

**📚 MÓDULO 6.5: pi-system/scripts/sync-content.sh** **Propósito:** Sincronizar contenido desde servidor central al Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Bash script de sincronización
* **Protocolo:** HTTPS/rsync/API calls
* **Frecuencia:** Cada 1-6 horas (configurable)
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES:**

* **Servidor central:** APIs de sincronización
* **Local storage:** Directorio de contenido local
* **Config files:** Configuración de empresa
* **Logs:** Sistema de logging detallado

**⚙️ PROCESO DE SINCRONIZACIÓN:**

1. **Verificar conectividad** al servidor central
2. **Autenticar** con servidor usando API key
3. **Comparar versiones** de paquetes
4. **Descargar actualizaciones** si están disponibles
5. **Verificar integridad** de archivos descargados
6. **Backup** del contenido actual
7. **Instalar** nuevo contenido
8. **Actualizar** configuraciones
9. **Reiniciar servicios** si es necesario
10. **Reportar estado** al servidor central

**🔄 TIPOS DE SINCRONIZACIÓN:**

* **Full sync:** Descarga completa inicial
* **Delta sync:** Solo cambios incrementales
* **Emergency sync:** Sincronización forzada
* **Rollback sync:** Restaurar versión anterior

**🛡️ SEGURIDAD Y VALIDACIÓN:**

* Verificación de checksums MD5/SHA256
* Validación de firma digital de paquetes
* Backup automático antes de actualizar
* Rollback automático si falla validación

**📊 REPORTING:**

* Estado de sincronización al servidor
* Métricas de ancho de banda usado
* Errores y reintentos
* Tiempo de última sincronización exitosa

**🎯 CRITERIOS DE ÉXITO:**

* Sincronización funciona automáticamente
* Maneja errores de conectividad graciosamente
* Verifica integridad de todos los archivos
* Reporta estado correctamente al servidor
* No interrumpe el servicio durante sync

**📚 MÓDULO 6.6: pi-system/scripts/update-system.sh**

**Propósito:** Actualizar sistema operativo y software del Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Bash script de mantenimiento
* **Frecuencia:** Semanal o bajo demanda
* **Tiempo estimado:** 30 minutos

**🔄 PROCESO DE ACTUALIZACIÓN:**

1. **Verificar conectividad** a internet
2. **Crear backup** del sistema crítico
3. **Actualizar lista** de paquetes
4. **Verificar actualizaciones** disponibles
5. **Instalar actualizaciones** de seguridad
6. **Actualizar software** de PLAYMI si hay versión nueva
7. **Verificar funcionamiento** post-actualización
8. **Limpiar archivos** temporales
9. **Reportar estado** de actualización

**🛡️ SEGURIDAD:**

* Solo actualizaciones de seguridad críticas
* Validación de fuentes de paquetes
* Backup antes de cualquier cambio mayor
* Rollback automático si sistema no funciona

**⚙️ MANTENIMIENTO INCLUIDO:**

* Limpieza de logs antiguos
* Limpieza de cache de paquetes
* Verificación de espacio en disco
* Optimización de base de datos local

**📝 LOGGING:**

* Log detallado de todas las operaciones
* Notificación de actualizaciones críticas
* Reporte de estado al servidor central

**🎯 CRITERIOS DE ÉXITO:**

* Sistema se mantiene actualizado y seguro
* Actualizaciones no rompen funcionalidad
* Backups funcionan correctamente
* Reportes son precisos y útiles

**📚 MÓDULO 6.7: pi-system/scripts/health-check.sh**

**Propósito:** Monitoreo continuo de salud del sistema Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Bash script de monitoreo
* **Frecuencia:** Cada 5-10 minutos
* **Tiempo estimado:** 35 minutos

**🔍 COMPONENTES MONITOREADOS:**

* **WiFi AP:** Estado de hostapd y dnsmasq
* **Servidor web:** Apache y PHP funcionando
* **Conectividad:** Internet y servidor central
* **Recursos:** CPU, memoria, disco, temperatura
* **Servicios:** Todos los servicios críticos
* **Logs:** Detección de errores críticos

**⚙️ VERIFICACIONES ESPECÍFICAS:**

1. **WiFi AP activo** y aceptando conexiones
2. **Portal web** responde correctamente
3. **Contenido** es accesible
4. **APIs locales** funcionan
5. **Temperatura CPU** dentro de límites
6. **Espacio en disco** suficiente
7. **Memoria** no saturada
8. **Conectividad** al servidor central

**🚨 ALERTAS Y ACCIONES:**

* **WiFi down:** Reintentar servicio automáticamente
* **Web server down:** Reiniciar Apache
* **Alta temperatura:** Throttling y alerta
* **Poco espacio:** Limpiar archivos temporales
* **Sin conectividad:** Modo offline y reintento

**📊 MÉTRICAS RECOLECTADAS:**

* Uptime del sistema
* Número de usuarios conectados
* Uso de ancho de banda
* Errores por minuto/hora
* Performance de los servicios

**📝 REPORTING:**

* Estado cada hora al servidor central
* Alertas inmediatas para problemas críticos
* Log local con historia de 30 días
* Dashboard básico vía web (opcional)

**🎯 CRITERIOS DE ÉXITO:**

* Detecta problemas antes que afecten usuarios
* Recupera automáticamente de errores comunes
* Reporta métricas precisas
* Mantiene historial útil para debugging
* No consume recursos excesivos

**📚 MÓDULO 6.8: pi-system/config/company-config.json**

**Propósito:** Configuración específica de empresa para el Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Archivo de configuración JSON
* **Generado por:** Sistema de paquetes (Fase 2.3)
* **Tiempo estimado:** 15 minutos

**📝 ESTRUCTURA DE CONFIGURACIÓN:**

json

{

"company\_info": {

"id": "company\_123",

"name": "Transportes ABC",

"service\_name": "EntretenimientoABC",

"contact\_email": "soporte@transportesabc.com"

},

"wifi\_settings": {

"ssid": "TransportesABC\_WiFi",

"password": "ABC123456",

"channel": 6,

"hidden": false

},

"portal\_settings": {

"company\_name": "Transportes ABC",

"primary\_color": "#FF6B35",

"secondary\_color": "#F7931E",

"logo\_path": "/assets/company-logo.png",

"welcome\_message": "Bienvenido a bordo de Transportes ABC"

},

"content\_settings": {

"movies\_enabled": true,

"music\_enabled": true,

"games\_enabled": true,

"adult\_content": false,

"max\_concurrent\_streams": 10

},

"system\_settings": {

"sync\_interval": 3600,

"auto\_update": true,

"analytics\_enabled": true,

"debug\_mode": false

},

"qr\_settings": {

"qr\_codes": [

{

"bus\_number": "001",

"qr\_data": "WIFI:T:WPA;S:TransportesABC\_WiFi;P:ABC123456;H:false;;",

"portal\_url": "http://192.168.4.1"

}

]

}

}

**🔗 USOS DE LA CONFIGURACIÓN:**

* **WiFi AP:** Configuración de SSID y password
* **Portal web:** Branding y personalización
* **Sincronización:** Intervalos y configuraciones
* **Contenido:** Filtros y restricciones
* **QR codes:** Datos para generación

**🔄 ACTUALIZACIÓN:**

* Actualizado automáticamente durante sync
* Validación de estructura JSON
* Backup de configuración anterior
* Aplicación inmediata de cambios

**🎯 CRITERIOS DE ÉXITO:**

* Configuración es válida JSON
* Se aplica correctamente a todos los servicios
* Cambios se reflejan inmediatamente
* Backup y rollback funcionan

**🚀 FASE 7: SISTEMA DE SINCRONIZACIÓN**

**📋 DIVISIÓN EN MÓDULOS:**

**FASE 7 se divide en 8 MÓDULOS específicos:**

**📚 MÓDULO 7.1: api/sync/pi-register.php**

**Propósito:** Registro inicial de nuevos Raspberry Pi en el sistema

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint de registro
* **Autenticación:** API key de empresa
* **Método HTTP:** POST
* **Tiempo estimado:** 40 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** SyncController::registerPi()
* **Modelo:** RaspberryPi::create(), Company::findById()
* **Validación:** API key validation, hardware validation

**📝 DATOS DE REGISTRO:**

* **Company ID:** Empresa propietaria del Pi
* **Pi Serial:** Número de serie único del hardware
* **Pi Model:** Modelo de Raspberry Pi
* **MAC Address:** Dirección MAC WiFi
* **Software Version:** Versión del software PLAYMI instalado
* **Location:** Ubicación/bus específico (opcional)
* **Initial Config:** Configuración inicial del Pi

**⚙️ PROCESO DE REGISTRO:**

1. **Validar API key** de la empresa
2. **Verificar serial** único del Pi
3. **Generar ID único** para el Pi en el sistema
4. **Crear entrada** en base de datos
5. **Generar configuración** inicial específica
6. **Crear API key** único para el Pi
7. **Preparar paquete** inicial de contenido
8. **Retornar credenciales** y configuración inicial

**🔒 SEGURIDAD:**

* API key válido de empresa requerido
* Rate limiting para prevenir spam
* Validación de hardware legítimo
* Generación de credenciales únicas por Pi

**📊 RESPUESTA DE REGISTRO:**

json

{

"success": true,

"pi\_id": "pi\_abc123def456",

"api\_key": "generated\_api\_key\_for\_pi",

"sync\_endpoints": {

"check\_updates": "/api/sync/check-updates.php",

"download\_package": "/api/sync/download-package.php",

"report\_status": "/api/sync/report-status.php"

},

"initial\_config": { */\* configuración inicial \*/* }

}

**🎯 CRITERIOS DE ÉXITO:**

* Pi se registra correctamente una sola vez
* Credenciales generadas son únicas y seguras
* Configuración inicial es correcta
* Base de datos se actualiza apropiadamente

**📚 MÓDULO 7.2: api/sync/check-updates.php**

**Propósito:** Verificar si hay actualizaciones disponibles para el Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint de verificación
* **Autenticación:** API key del Pi
* **Método HTTP:** GET/POST
* **Tiempo estimado:** 30 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** SyncController::checkUpdates()
* **Modelo:** RaspberryPi::findByApiKey(), Package::getLatestForCompany()

**📋 PARÁMETROS:**

* **Pi API Key:** Identificación del Pi
* **Current Version:** Versión actual del paquete instalado
* **System Info:** Información básica del sistema
* **Last Sync:** Timestamp de última sincronización

**⚙️ PROCESO DE VERIFICACIÓN:**

1. **Autenticar Pi** con API key
2. **Obtener información** de empresa asociada
3. **Comparar versiones** actual vs disponible
4. **Verificar espacio** disponible en Pi
5. **Determinar tipo** de actualización (full/delta)
6. **Calcular prioridad** de actualización
7. **Retornar información** de actualizaciones disponibles

**📊 RESPUESTA:**

json

{

"success": true,

"updates\_available": true,

"current\_version": "1.2.3",

"latest\_version": "1.3.0",

"update\_type": "delta",

"update\_size": 52428800,

"update\_priority": "normal",

"download\_url": "/api/sync/download-package.php?token=abc123",

"changelog": ["Nueva funcionalidad X", "Fix bug Y"]

}

**🔄 TIPOS DE ACTUALIZACIÓN:**

* **Full:** Paquete completo nuevo
* **Delta:** Solo archivos modificados
* **Critical:** Actualización de seguridad crítica
* **Optional:** Mejoras opcionales

**⏰ SCHEDULING:**

* Cache de respuesta por 30 minutos
* Prioridad alta para actualizaciones críticas
* Throttling para Pi con poco ancho de banda

**🎯 CRITERIOS DE ÉXITO:**

* Detecta actualizaciones correctamente
* Calcula tamaños de descarga precisos
* Respeta prioridades de actualización
* Cache mejora performance

**📚 MÓDULO 7.3: api/sync/download-package.php**

**Propósito:** Descargar paquetes de actualización para Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint de descarga
* **Autenticación:** Token temporal de descarga
* **Método HTTP:** GET
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** SyncController::downloadPackage()
* **Modelo:** Package::findById(), DownloadToken::validate()
* **File System:** Streaming de archivos grandes

**⚙️ PROCESO DE DESCARGA:**

1. **Validar token** de descarga temporal
2. **Verificar permisos** del Pi para el paquete
3. **Preparar headers** de descarga
4. **Iniciar streaming** del archivo
5. **Monitorear progreso** de descarga
6. **Registrar estadísticas** de descarga
7. **Limpiar recursos** después de completar

**🔒 SEGURIDAD:**

* Tokens temporales con expiración
* Validación de Pi autorizado
* Rate limiting por Pi
* Verificación de integridad de archivos

**📊 FEATURES DE DESCARGA:**

* **Resume support:** Descargas interrumpidas
* **Compression:** Compresión gzip en vuelo
* **Chunked transfer:** Para archivos grandes
* **Progress tracking:** Monitoreo de progreso
* **Bandwidth limiting:** Control de ancho de banda

**📝 LOGGING:**

* Registro de todas las descargas
* Estadísticas de ancho de banda
* Errores y reintentos
* Tiempo de descarga por Pi

**🎯 CRITERIOS DE ÉXITO:**

* Descargas son confiables y resumibles
* No saturan el ancho de banda del servidor
* Manejan interrupciones de red graciosamente
* Logging es completo y útil

**📚 MÓDULO 7.4: api/sync/report-status.php**

**Propósito:** Recibir reportes de estado desde los Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint de reporte
* **Autenticación:** API key del Pi
* **Método HTTP:** POST
* **Tiempo estimado:** 35 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** SyncController::reportStatus()
* **Modelo:** RaspberryPi::updateStatus(), StatusLog::create()

**📊 DATOS REPORTADOS:**

* **System health:** CPU, memoria, temperatura, disco
* **Service status:** Estado de todos los servicios críticos
* **User activity:** Número de usuarios conectados
* **Content usage:** Estadísticas de reproducción
* **Error logs:** Errores recientes del sistema
* **Network info:** Calidad de conexión, ancho de banda

**📝 ESTRUCTURA DEL REPORTE:**

json

{

"pi\_id": "pi\_abc123def456",

"timestamp": "2024-08-12T10:30:00Z",

"system\_health": {

"cpu\_usage": 15.5,

"memory\_usage": 67.2,

"disk\_usage": 45.8,

"temperature": 45.2,

"uptime": 86400

},

"services": {

"wifi\_ap": "active",

"web\_server": "active",

"portal": "active"

},

"activity": {

"connected\_users": 12,

"content\_requests": 156,

"errors\_last\_hour": 2

},

"content\_stats": {

"movies\_played": 45,

"music\_played": 78,

"games\_played": 23

}

}

**⚙️ PROCESAMIENTO:**

1. **Validar API key** del Pi
2. **Parsear datos** del reporte
3. **Actualizar estado** en base de datos
4. **Detectar anomalías** o problemas
5. **Triggerar alertas** si es necesario
6. **Actualizar métricas** agregadas
7. **Confirmar recepción** del reporte

**🚨 ALERTAS AUTOMÁTICAS:**

* Temperatura alta del CPU
* Servicios críticos caídos
* Errores frecuentes
* Poco espacio en disco
* Sin actividad de usuarios

**🎯 CRITERIOS DE ÉXITO:**

* Reportes se procesan rápidamente
* Alertas se disparan apropiadamente
* Datos se almacenan eficientemente
* Dashboard refleja estado actual

**📚 MÓDULO 7.5: admin/views/pi-management/index.php**

**Propósito:** Panel de gestión de todos los Raspberry Pi registrados

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Dashboard de administración
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 1 hora

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PiManagementController::index()
* **Modelo:** RaspberryPi::findAll(), StatusLog::getRecent()
* **APIs:** api/pi/get-pi-status.php, api/pi/push-update.php
* **Layout:** layouts/base.php

**📊 VISTA GENERAL:**

* **Mapa/Lista:** Ubicación de todos los Pi
* **Estado global:** Resumen de estado de toda la flota
* **Alertas:** Pi con problemas o que requieren atención
* **Estadísticas:** Métricas agregadas de toda la red

**🎛️ FUNCIONALIDADES:**

* **Lista filtrable** de todos los Pi
* **Estado en tiempo real** de cada Pi
* **Comandos remotos** (reiniciar, actualizar, configurar)
* **Histórico de estado** por Pi
* **Alertas y notificaciones**
* **Generación de reportes**

**📈 MÉTRICAS MOSTRADAS:**

* Total de Pi activos/inactivos
* Distribución por empresa
* Problemas activos
* Uso agregado de contenido
* Performance promedio de la red

**🔧 ACCIONES DISPONIBLES:**

* Enviar actualización a Pi específico
* Reiniciar Pi remotamente
* Cambiar configuración
* Ver logs detallados
* Generar reporte de Pi específico

**🎯 CRITERIOS DE ÉXITO:**

* Vista global clara de toda la flota
* Estado en tiempo real es preciso
* Acciones remotas funcionan
* Alertas son útiles y precisas

**📚 MÓDULO 7.6: admin/views/pi-management/monitoring.php**

**Propósito:** Monitoreo en tiempo real de la red de Pi

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Dashboard de monitoreo en tiempo real
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET con WebSockets/AJAX polling
* **Tiempo estimado:** 1 hora 15 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** MonitoringController::index()
* **APIs:** api/pi/get-pi-status.php (real-time)
* **WebSockets:** Para actualizaciones en vivo
* **JavaScript:** Charts y visualizaciones

**📊 DASHBOARDS INCLUIDOS:**

* **Mapa en tiempo real:** Estado de Pi por ubicación
* **Gráficos de performance:** CPU, memoria, temperatura
* **Usuarios activos:** Conexiones por Pi en tiempo real
* **Uso de contenido:** Estadísticas de reproducción live
* **Alertas activas:** Problemas que requieren atención

**📈 VISUALIZACIONES:**

* **Heat map:** Performance de la red
* **Time series:** Métricas históricas
* **Gauge charts:** CPU, memoria, temperatura
* **Activity feed:** Stream de eventos en vivo
* **Network topology:** Conectividad de la red

**⚡ TIEMPO REAL:**

* Updates cada 30 segundos vía WebSocket
* Push notifications para alertas críticas
* Auto-refresh de gráficos
* Streaming de logs en vivo

**🚨 ALERTAS:**

* **Critical:** Pi offline, servicios caídos
* **Warning:** Alta temperatura, poco espacio
* **Info:** Actualizaciones disponibles
* **Success:** Operaciones completadas exitosamente

**🎯 CRITERIOS DE ÉXITO:**

* Dashboard actualiza en tiempo real
* Visualizaciones son claras e informativas
* Alertas llegan inmediatamente
* Performance es fluido con muchos Pi

**📚 MÓDULO 7.7: admin/views/pi-management/logs.php**

**Propósito:** Visualización centralizada de logs de sincronización

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Página de logs con filtros avanzados
* **Autenticación:** Requerida (admin)
* **Método HTTP:** GET
* **Tiempo estimado:** 45 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** LogsController::index()
* **Modelo:** SyncLog::findAll(), StatusLog::findAll()
* **APIs:** Log streaming APIs
* **DataTables:** Para manejo de grandes volúmenes

**📝 TIPOS DE LOGS:**

* **Sync logs:** Sincronizaciones y actualizaciones
* **Status logs:** Reportes de estado de Pi
* **Error logs:** Errores y excepciones
* **Activity logs:** Actividad de usuarios
* **System logs:** Eventos del sistema

**🔍 FILTROS DISPONIBLES:**

* **Por Pi:** Logs de Pi específico
* **Por empresa:** Logs de empresa específica
* **Por tipo:** Tipo de log (sync, error, etc.)
* **Por fecha:** Rango de fechas
* **Por nivel:** Debug, info, warning, error
* **Por keyword:** Búsqueda en contenido

**📊 FUNCIONALIDADES:**

* **Búsqueda en tiempo real** en logs
* **Export** de logs a CSV/JSON
* **Live streaming** de logs nuevos
* **Aggregation** de errores similares
* **Drill-down** a logs detallados

**📈 ANALYTICS:**

* **Error frequency:** Frecuencia de errores por tipo
* **Pi reliability:** Confiabilidad por Pi
* **Sync success rate:** Tasa de éxito de sincronizaciones
* **Performance trends:** Tendencias de performance

**🎯 CRITERIOS DE ÉXITO:**

* Logs se muestran rápidamente
* Filtros funcionan eficientemente
* Búsqueda es rápida y precisa
* Export funciona para grandes volúmenes

**📚 MÓDULO 7.8: api/pi/push-update.php**

**Propósito:** Forzar actualización a Pi específicos

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** API endpoint de comando remoto
* **Autenticación:** Requerida (admin token)
* **Método HTTP:** POST
* **Tiempo estimado:** 30 minutos

**🔗 CONEXIONES DE ARCHIVOS:**

* **Controlador:** PiCommandController::pushUpdate()
* **Modelo:** RaspberryPi::findById(), UpdateQueue::create()
* **Queue system:** Sistema de colas para comandos

**📝 TIPOS DE COMANDOS:**

* **Force update:** Forzar descarga de actualización
* **Restart services:** Reiniciar servicios específicos
* **Restart system:** Reiniciar Pi completo
* **Update config:** Actualizar configuración
* **Clear cache:** Limpiar cache del Pi

**⚙️ PROCESO DE COMANDO:**

1. **Validar permisos** de administrador
2. **Verificar Pi objetivo** existe y está online
3. **Crear comando** en cola de ejecución
4. **Notificar al Pi** sobre comando pendiente
5. **Monitorear ejecución** del comando
6. **Recibir confirmación** del Pi
7. **Actualizar estado** del comando

**📊 QUEUE MANAGEMENT:**

* **Priority queue:** Comandos críticos primero
* **Retry logic:** Reintentos automáticos
* **Timeout handling:** Timeout para comandos
* **Status tracking:** Estado de cada comando

**🔄 RESPUESTA:**

json

{

"success": true,

"command\_id": "cmd\_abc123",

"pi\_id": "pi\_def456",

"command\_type": "force\_update",

"status": "queued",

"estimated\_execution": "2024-08-12T10:35:00Z"

}

**📊 MONITOREO:**

* Track de ejecución de comandos
* Estadísticas de éxito/fallo
* Tiempo de respuesta promedio
* Log de todos los comandos enviados

**🎯 CRITERIOS DE ÉXITO:**

* Comandos se ejecutan reliablemente
* Queue maneja prioridades correctamente
* Timeouts evitan comandos colgados
* Monitoring provee visibilidad completa

**🚀 FASE 8: OPTIMIZACIÓN Y FINALIZACIÓN**

**📋 DIVISIÓN EN MÓDULOS:**

**FASE 8 se divide en 6 MÓDULOS específicos:**

**📚 MÓDULO 8.1: OPTIMIZACIÓN DE PERFORMANCE**

**Propósito:** Optimizar rendimiento de toda la plataforma

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Optimizaciones de código y infraestructura
* **Impacto:** Todo el sistema
* **Tiempo estimado:** 2 horas

**⚡ OPTIMIZACIONES DEL SERVIDOR:**

* **Database optimization:** Índices, queries, connection pooling
* **Caching strategy:** Redis/Memcached para datos frecuentes
* **Asset optimization:** Minificación CSS/JS, compresión imágenes
* **CDN setup:** Content delivery network para assets estáticos
* **Gzip compression:** Compresión de respuestas HTTP

**📱 OPTIMIZACIONES DEL PORTAL:**

* **Image lazy loading:** Carga diferida de imágenes
* **Code splitting:** División de JavaScript por rutas
* **Service workers:** Cache offline y performance
* **Critical CSS:** CSS crítico inline
* **Resource hints:** Preload, prefetch para recursos

**🔧 OPTIMIZACIONES DEL PI:**

* **Memory management:** Optimización de uso de memoria
* **CPU optimization:** Procesos eficientes
* **Storage optimization:** Compresión de contenido
* **Network optimization:** Compression y caching
* **Background tasks:** Optimización de tareas de fondo

**📊 MÉTRICAS DE PERFORMANCE:**

* **Page load time:** < 3 segundos en móviles
* **API response time:** < 200ms promedio
* **Video start time:** < 2 segundos buffering
* **Pi response time:** < 500ms para comandos
* **Sync efficiency:** > 95% success rate

**🎯 CRITERIOS DE ÉXITO:**

* Mejoras medibles en todos los KPIs
* Usuario final nota la mejora
* Menor uso de recursos del servidor
* Pi funcionan más eficientemente

**📚 MÓDULO 8.2: COMPRESIÓN Y CACHE DE CONTENIDO**

**Propósito:** Optimizar almacenamiento y transferencia de contenido

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Sistema de optimización de contenido
* **Herramientas:** FFmpeg, ImageMagick, compresión algoritmos
* **Tiempo estimado:** 1 hora 30 minutos

**🎬 OPTIMIZACIÓN DE VIDEO:**

* **Multiple bitrates:** 480p, 720p, 1080p según capacidad
* **Adaptive streaming:** HLS/DASH para streaming adaptativo
* **Thumbnail optimization:** WebP para thumbnails
* **Video compression:** H.264 optimizado para móviles
* **Chunk optimization:** Segmentos optimizados para streaming

**🎵 OPTIMIZACIÓN DE AUDIO:**

* **Bitrate optimization:** 128kbps, 192kbps, 320kbps
* **Format optimization:** MP3 optimizado, AAC para mejor calidad
* **Metadata stripping:** Remover metadatos innecesarios
* **Normalization:** Normalización de volumen

**🎮 OPTIMIZACIÓN DE JUEGOS:**

* **Asset compression:** Compresión de sprites y sonidos
* **Code minification:** Minificación de JavaScript
* **Resource bundling:** Empaquetado eficiente de recursos
* **Progressive loading:** Carga progresiva de assets

**💾 ESTRATEGIA DE CACHE:**

* **Content-based hashing:** Cache basado en contenido
* **Hierarchical cache:** Cache multinivel (servidor, Pi, dispositivo)
* **Cache invalidation:** Invalidación inteligente
* **Prefetching:** Precarga predictiva de contenido

**🎯 CRITERIOS DE ÉXITO:**

* Reducción significativa en tamaño de paquetes
* Streaming más fluido y rápido
* Menor uso de ancho de banda
* Cache hit rate > 80%

**📚 MÓDULO 8.3: TESTING COMPLETO DEL SISTEMA**

**Propósito:** Testing exhaustivo de toda la plataforma

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Suite de testing completa
* **Herramientas:** PHPUnit, Jest, Selenium, Performance testing
* **Tiempo estimado:** 2 horas 30 minutos

**🧪 TIPOS DE TESTING:**

* **Unit tests:** Todas las funciones críticas
* **Integration tests:** APIs y conexiones entre módulos
* **End-to-end tests:** Flujos completos de usuario
* **Performance tests:** Load testing y stress testing
* **Security tests:** Penetration testing básico
* **Device tests:** Testing en diferentes dispositivos

**📋 ESCENARIOS DE TESTING:**

* **Flujo completo:** Desde admin hasta usuario final
* **Concurrencia:** Múltiples usuarios simultáneos
* **Fallo de red:** Comportamiento durante interrupciones
* **Carga alta:** Performance bajo carga pesada
* **Datos corruptos:** Manejo de datos incorrectos
* **Ataques básicos:** SQL injection, XSS, CSRF

**🎮 TESTING ESPECÍFICO POR MÓDULO:**

* **Admin panel:** CRUD completo, validaciones, permisos
* **Content management:** Upload, processing, delivery
* **Package generation:** Creación y distribución de paquetes
* **QR system:** Generación y funcionalidad de QR codes
* **Portal passenger:** UX, performance, compatibility
* **Games system:** Carga, ejecución, sandbox security
* **Pi system:** Instalación, sync, health monitoring
* **Sync system:** Reliability, error handling, recovery

**📊 MÉTRICAS DE CALIDAD:**

* **Code coverage:** > 80% en funciones críticas
* **Bug density:** < 1 bug por 1000 líneas de código
* **Performance benchmarks:** Todos los KPIs dentro de límites
* **Security score:** Sin vulnerabilidades críticas
* **Compatibility:** 100% en dispositivos objetivo

**🎯 CRITERIOS DE ÉXITO:**

* Todos los tests pasan sin errores críticos
* Performance cumple con KPIs establecidos
* No hay vulnerabilidades de seguridad críticas
* Sistema es estable bajo carga normal

**📚 MÓDULO 8.4: DOCUMENTACIÓN TÉCNICA**

**Propósito:** Documentación completa para desarrolladores y administradores

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Documentación técnica completa
* **Formato:** Markdown, HTML, PDF
* **Tiempo estimado:** 2 horas

**📖 DOCUMENTACIÓN INCLUIDA:**

* **Installation guide:** Instalación paso a paso
* **Configuration guide:** Configuración del sistema
* **API documentation:** Documentación completa de APIs
* **Database schema:** Estructura y relaciones de BD
* **Architecture overview:** Arquitectura del sistema
* **Troubleshooting guide:** Solución de problemas comunes

**👥 AUDIENCIAS:**

* **Desarrolladores:** Documentación técnica detallada
* **Administradores:** Guías de operación y mantenimiento
* **Usuarios finales:** Manual de usuario básico
* **Empresas cliente:** Guía de implementación

**📝 CONTENIDO POR DOCUMENTO:**

**Installation Guide:**

* Requisitos del sistema
* Instalación paso a paso
* Configuración inicial
* Verificación de instalación
* Troubleshooting común

**API Documentation:**

* Endpoints disponibles
* Parámetros y respuestas
* Códigos de error
* Ejemplos de uso
* Rate limiting y autenticación

**Database Schema:**

* Diagrama ER completo
* Descripción de tablas
* Índices y constraints
* Procedimientos almacenados
* Scripts de migración

**Architecture Overview:**

* Diagramas de arquitectura
* Flujo de datos
* Componentes del sistema
* Patrones de diseño utilizados
* Decisiones arquitectónicas

**🎯 CRITERIOS DE ÉXITO:**

* Documentación es completa y precisa
* Ejemplos funcionan correctamente
* Guías permiten instalación exitosa
* Troubleshooting cubre problemas comunes

**📚 MÓDULO 8.5: MANUAL DE USUARIO PARA ADMINISTRADORES**

**Propósito:** Manual completo para operación del sistema

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Manual de usuario ilustrado
* **Formato:** PDF interactivo, HTML
* **Tiempo estimado:** 1 hora 30 minutos

**📖 CONTENIDO DEL MANUAL:**

**Introducción al Sistema:**

* Qué es PLAYMI y cómo funciona
* Roles y permisos
* Primeros pasos
* Navegación básica

**Gestión de Empresas:**

* Agregar nueva empresa
* Configurar branding
* Gestionar información de contacto
* Configurar paquetes

**Gestión de Contenido:**

* Subir películas, música, juegos
* Organizar contenido por categorías
* Configurar metadatos
* Gestionar calidad y formatos

**Generación de Paquetes:**

* Crear paquete personalizado
* Seleccionar contenido específico
* Configurar WiFi y QR codes
* Descargar e instalar en Pi

**Monitoreo y Mantenimiento:**

* Dashboard de monitoreo
* Interpretar métricas
* Solucionar problemas comunes
* Actualizaciones del sistema

**🎨 ELEMENTOS VISUALES:**

* Screenshots de cada pantalla
* Diagramas de flujo de procesos
* Iconografía explicativa
* Videos tutoriales (links)

**📱 FORMATOS:**

* **PDF:** Para impresión y referencia offline
* **HTML:** Para acceso online con búsqueda
* **Video tutorials:** Para procesos complejos
* **Quick reference:** Tarjetas de referencia rápida

**🎯 CRITERIOS DE ÉXITO:**

* Manual permite uso independiente del sistema
* Screenshots son actuales y claros
* Procesos están explicados paso a paso
* Troubleshooting cubre casos reales

**📚 MÓDULO 8.6: CONFIGURACIÓN DE PRODUCCIÓN**

**Propósito:** Preparar sistema para ambiente de producción

**📋 ESPECIFICACIONES TÉCNICAS:**

* **Tipo:** Configuración de producción
* **Ambiente:** Servidor de producción
* **Tiempo estimado:** 1 hora 45 minutos

**🔒 SEGURIDAD DE PRODUCCIÓN:**

* **SSL/TLS:** Certificados para todas las conexiones
* **Firewall:** Configuración restrictiva de puertos
* **User permissions:** Permisos mínimos necesarios
* **Database security:** Usuarios con permisos limitados
* **API rate limiting:** Límites apropiados para producción
* **Input validation:** Validación estricta de todas las entradas

**⚡ OPTIMIZACIONES DE PRODUCCIÓN:**

* **Database tuning:** Optimización para carga de producción
* **Web server config:** Apache/Nginx optimizado
* **PHP configuration:** PHP-FPM optimizado
* **Memory limits:** Límites apropiados para producción
* **Cache configuration:** Redis/Memcached configurado
* **Log rotation:** Rotación automática de logs

**📊 MONITOREO DE PRODUCCIÓN:**

* **Health checks:** Monitoreo automático de servicios
* **Alerting:** Alertas para problemas críticos
* **Metrics collection:** Recolección de métricas de performance
* **Log aggregation:** Centralización de logs
* **Backup strategy:** Backups automáticos de BD y archivos

**🔄 DEPLOYMENT:**

* **CI/CD pipeline:** Pipeline de despliegue automático
* **Blue-green deployment:** Despliegue sin downtime
* **Database migrations:** Migraciones automáticas de BD
* **Asset compilation:** Compilación automática de assets
* **Health verification:** Verificación post-despliegue

**📋 CHECKLIST DE PRODUCCIÓN:**

* SSL certificado instalado y funcionando
* Firewall configurado correctamente
* Database backups automáticos configurados
* Monitoring y alertas funcionando
* Performance bajo límites aceptables
* Security scan sin vulnerabilidades críticas
* Load testing completado exitosamente
* Documentación de operaciones completa

**🎯 CRITERIOS DE ÉXITO:**

* Sistema es seguro y estable en producción
* Monitoring detecta problemas proactivamente
* Backups funcionan y se pueden restaurar
* Performance cumple SLAs establecidos
* Deployment es automático y confiable

**📋 CRITERIOS DE ÉXITO TOTAL DEL PROYECTO**

**✅ AL COMPLETAR TODAS LAS FASES:**

**🎯 FUNCIONALIDADES CORE:**

1. ✅ **Panel administrativo** completo y funcional
2. ✅ **Gestión de empresas** con CRUD completo
3. ✅ **Gestión de contenido** multimedia (películas/música/juegos)
4. ✅ **Generación de paquetes** personalizados automática
5. ✅ **Sistema QR + WiFi** para conexión automática
6. ✅ **Portal tipo Netflix** optimizado para móviles
7. ✅ **Sistema de juegos HTML5** con sandbox seguro
8. ✅ **Instalación automática** en Raspberry Pi
9. ✅ **Sincronización automática** servidor-Pi
10. ✅ **Monitoreo en tiempo real** de toda la red

**📊 KPIS DE ÉXITO:**

* **Performance:** Portal carga en < 3 segundos
* **Uptime:** Sistema disponible > 99.5% del tiempo
* **User experience:** Conexión WiFi automática funciona > 95%
* **Sync reliability:** Sincronización exitosa > 95% de las veces
* **Content delivery:** Video inicia en < 2 segundos
* **System stability:** Pi funcionan sin reiniciar > 7 días

**🔒 SEGURIDAD:**

* Sin vulnerabilidades críticas de seguridad
* Datos de empresas protegidos apropiadamente
* Juegos ejecutan en sandbox seguro
* APIs protegidas contra ataques comunes

**📱 COMPATIBILIDAD:**

* Funciona en 95% de dispositivos móviles modernos
* Compatible con diferentes modelos de Raspberry Pi
* Funciona en diferentes navegadores móviles

**🎯 EXPERIENCIA DE USUARIO:**

* **Admin:** Puede gestionar todo sin conocimiento técnico
* **Pasajero:** Conexión y uso intuitivo en < 30 segundos
* **Empresa:** Instalación y operación sin soporte técnico