Propuesta de Arquitectura y Plan de Implementación

Sistema Integral para Proveedores de Internet (ISP)

1. Arquitectura Propuesta

Considerando tus necesidades específicas (equipo pequeño, facilidad de mantenimiento, servidor Ubuntu existente con Jellyfin) y el plazo de 20 días, recomendamos una arquitectura moderna pero accesible:

1.1 Stack Tecnológico Recomendado

- **Frontend**: Vue.js 3
  - Framework progresivo, fácil de aprender y mantener
  - Excelente documentación y comunidad activa
  - Vuetify como biblioteca de componentes para UI profesional
  - Soporte para PWA (Progressive Web App)
- **Backend**: Node.js con Express
  - JavaScript en ambos frontend y backend (facilita el mantenimiento)
  - Alta disponibilidad de librerías para todas las integraciones requeridas
  - Excelente rendimiento para operaciones I/O (interacciones con APIs)
  - Fácil de escalar horizontalmente
- **Base de Datos**:
  - PostgreSQL como base de datos principal
  - InfluxDB para métricas y monitoreo (series temporales)
  - Redis para caché y sesiones
- **Infraestructura**:
  - Docker para contenerización
  - Nginx como proxy inverso
  - Certificados SSL mediante Let's Encrypt/Certbot

1.2 Justificación de Tecnologías

- **¿Por qué Vue.js?**
  - Más fácil de aprender que Angular
  - Mejor documentación y curva de aprendizaje que React
  - Gran cantidad de recursos educativos gratuitos
  - Comunidad activa en español

- **¿Por qué Node.js?**
  - Mismo lenguaje que el frontend (JavaScript/TypeScript)
  - Excelente para operaciones asíncronas (ideal para sistemas con múltiples APIs)
  - Gran disponibilidad de librerías para todas las integraciones requeridas
  - NPM como gestor de paquetes centralizado
- **¿Por qué PostgreSQL?**
  - Base de datos relacional robusta y gratuita
  - Excelente para datos estructurados complejos
  - Soporte para JSON para datos semi-estructurados
  - Amplia documentación en español
- **¿Por qué Docker?**
  - Facilita la gestión de dependencias
  - Permite actualizaciones sin tiempo de inactividad
  - Hace posible una arquitectura de microservicios ligera
  - Facilita el escalado futuro

2. Plan de Implementación (20 días)

Fase 1: Configuración de Infraestructura (Días 1-3)

- **Día 1**:
  - Configuración del entorno de desarrollo
  - Instalación de Docker y Docker Compose
  - Creación de repositorio Git
- **Día 2**:
  - Configuración de contenedores Docker
  - Configuración de Nginx como proxy inverso
  - Configuración de SSL/TLS con Let's Encrypt
- **Día 3**:
  - Configuración de bases de datos (PostgreSQL, InfluxDB, Redis)
  - Configuración de respaldos automáticos
  - Pruebas de infraestructura

Fase 2: Desarrollo del Core del Sistema (Días 4-10)

- **Día 4**:
  - Desarrollo del sistema de autenticación y autorización

    - Implementación del sistema de roles y permisos
    - Creación de API para gestión de usuarios
- **Día 5-6**:
    - Desarrollo del módulo de gestión de clientes
    - Implementación de búsqueda avanzada
    - Estructura jerárquica de clientes por nodos/sectores
- **Día 7-8**:
    - Integración con Mikrotik RouterOS API
    - Integración con Ubiquiti UNMS/UISP API
    - Desarrollo de adaptadores para APIs
- **Día 9-10**:
    - Desarrollo del sistema de monitoreo
    - Implementación de métricas y alertas
    - Dashboard principal de estado de red

Fase 3: Módulos Complementarios (Días 11-17)

- **Día 11**:
    - Desarrollo del sistema de tickets
    - Integración con comunicaciones (Email)
- **Día 12-13**:
    - Desarrollo del módulo de inventario
    - Implementación de control de stock y asignación
- **Día 14**:
    - Integración con Jellyfin y JFA-GO
    - Automatización de invitaciones y provisioning
- **Día 15-16**:
    - Desarrollo del módulo financiero básico
    - Integración con pasarelas de pago
    - Sistema de facturación simple
- **Día 17**:
    - Desarrollo del sistema de comunicaciones multicanal
    - Integración con WhatsApp/Telegram
    - Plantillas de mensajes personalizadas

Fase 4: Finalización e Integración (Días 18-20)

- **Día 18**:
  - Implementación de búsqueda global
  - Optimizaciones de rendimiento
  - Pruebas de integración completa
- **Día 19**:
  - Documentación del sistema
  - Creación de manuales de usuario
  - Configuración de respaldos automáticos
- **Día 20**:
  - Despliegue en producción
  - Migración de datos existentes
  - Capacitación inicial a usuarios

3. Priorización de Módulos

Considerando el corto plazo (20 días) y equipo pequeño (3 personas), recomendamos priorizar los siguientes módulos:

3.1 Módulos Prioritarios (Fase 1)

1. **Sistema de Autenticación y Permisos**
   - Base esencial para todo el sistema
   - Seguridad y control de acceso
2. **Gestión de Clientes**
   - Core del negocio ISP
   - Organización jerárquica por nodos/sectores
3. **Integración con Equipos de Red**
   - Control de routers Mikrotik y equipos Ubiquiti
   - Gestión de ancho de banda y cuentas PPPoE
4. **Monitoreo Básico**
   - Dashboard de estado de red
   - Alertas esenciales

3.2 Módulos Secundarios (Fase 2)

5. **Sistema de Tickets**
   - Gestión básica de soporte técnico

6. **Inventario Simple**
    - Control básico de equipamiento
7. **Integración con Jellyfin**
    - Aprovechando la instalación existente
8. **Facturación Básica**
    - Sistema simple de pagos y control

3.3 Módulos para Futuras Iteraciones

- Sistema avanzado de rutas para técnicos
- Marketing y campañas
- Análisis avanzado y reportes
- Aplicación móvil para técnicos y clientes
- Integraciones con sistemas fiscales

4. Consideraciones para la Implementación

4.1 Enfoque de Desarrollo

- **Desarrollo Iterativo**: Entregar módulos funcionales frecuentemente
- **Priorizar UX**: Interfaces intuitivas que requieran mínima capacitación
- **Documentación Incorporada**: Ayuda contextual dentro del sistema
- **Automatización**: Configurar CI/CD desde el inicio

4.2 Arquitectura Técnica

- **API First**: Desarrollar APIs RESTful bien documentadas
- **Microservicios Ligeros**: Modularidad sin excesiva complejidad
- **Stateless**: Diseño sin estado para facilitar escalabilidad
- **Caché Inteligente**: Reducir consultas a APIs externas

4.3 Seguridad

- **HTTPS Everywhere**: Toda comunicación cifrada
- **JWT para Autenticación**: Tokens seguros con expiración adecuada
- **Almacenamiento Seguro**: Credenciales de APIs cifradas en base de datos
- **Validación Estricta**: Prevención de inyecciones y XSS
- **Auditoría Completa**: Registro de todas las acciones administrativas

5. Arquitectura de Datos

5.1 Estructura de Base de Datos Principal (PostgreSQL)

- **Usuarios y Permisos**
  - usuarios, roles, permisos, acciones_usuario
- **Clientes**
  - clientes, documentos_cliente, historial_pagos, servicios_contratados, notas_cliente
- **Estructura de Red**
  - nodos, repetidores, sectores, enlaces, ip_pools
- **Equipamiento**
  - dispositivos_red, configuraciones, credenciales_apis, firmware
- **Soporte**
  - tickets, comentarios_ticket, categorias_ticket, sla, base_conocimiento
- **Inventario**
  - items_inventario, movimientos, ubicaciones, proveedores
- **Facturación**
  - facturas, pagos, planes_servicio, impuestos
- **Comunicaciones**
  - plantillas_mensaje, historial_comunicaciones, canales, programacion_mensajes

5.2 Métricas (InfluxDB)

- **Rendimiento de Red**
  - trafico_por_cliente, estado_enlaces, latencia, paquetes_perdidos
- **Utilización**
  - cpu_dispositivos, memoria, temperatura, capacidad_discos
- **QoS**
  - calidad_enlaces, interferencias, señal_clientes

5.3 Caché (Redis)

- **Sesiones**
  - tokens_jwt, datos_sesion
- **Datos Frecuentes**
  - estado_dispositivos, configuraciones_activas, clientes_activos
- **Colas**
  - notificaciones_pendientes, tareas_programadas

6. Integración con Jellyfin y JFA-GO

Dado que ya tienes Jellyfin y JFA-GO instalados en tu servidor Ubuntu, la integración consistirá en:

6.1 Enfoque de Integración

- **Automatización JFA-GO**:
  - Desarrollar scripts para la creación automática de invitaciones
  - Integrar con el proceso de alta de clientes
- **Monitoreo de Jellyfin**:
  - Utilizar la API de Jellyfin para obtener estadísticas de uso
  - Implementar límites de ancho de banda específicos para streaming

6.2 Proceso de Automatización

1. Al dar de alta un cliente en el sistema ISP:
   - Generar automáticamente una invitación en JFA-GO
   - Configurar parámetros según el plan contratado
   - Enviar invitación por correo electrónico al cliente
2. Sincronización de estado:
   - Suspender acceso a Jellyfin cuando se suspenda el servicio de internet
   - Monitorear y reportar uso de streaming en el perfil del cliente

7. Requerimientos de Hardware

Para un sistema de esta escala inicial (soporte para un ISP pequeño), los requerimientos mínimos serían:

- **Servidor Principal**:
  - CPU: 4-8 cores (Intel Xeon o AMD Ryzen)
  - RAM: 16-32 GB
  - Almacenamiento: 500 GB SSD (sistema + bases de datos)
  - Conexión: 1 Gbps mínimo
- **Respaldos**:
  - Disco adicional para respaldos locales
  - Configuración de respaldos remotos (opcional)

8. Consideraciones Finales

8.1 Escalabilidad

El diseño propuesto permite escalar horizontalmente a medida que crezca el ISP:

- Microservicios dockerizados facilitan despliegue en múltiples servidores

- Bases de datos pueden migrarse a clusters
- Caché distribuido para optimizar rendimiento

8.2 Mantenimiento

- Logs centralizados para diagnóstico rápido
- Monitoreo de salud del sistema
- Actualizaciones sin tiempo de inactividad vía Docker

8.3 Siguientes Pasos

Tras el despliegue inicial, recomendamos:

1. Implementar monitoreo avanzado del sistema
2. Desarrollar integraciones adicionales
3. Mejorar UX basado en feedback inicial
4. Optimizar rendimiento tras uso real
5. Implementar análisis avanzado de datos

Este plan está diseñado para ser realista dentro del plazo de 20 días y con un equipo pequeño. Prioriza la funcionalidad esencial para un ISP mientras establece las bases para un crecimiento futuro.