Diseño de Arquitectura del Sistema - TaskManager

**Fecha:**

**Autor:**

**Versión:** 1.0

# 1. Introducción

Este documento describe de manera detallada la arquitectura del sistema para la aplicación web **Panorama**, cuyo propósito es optimizar la gestión de turistas, mejorar la comunicación entre la empresa y los visitantes, y contribuir al incremento de la satisfacción y fidelización de los clientes nacionales e internacionales.

Se define la estructura general del sistema, los componentes que lo conforman, sus interacciones y las tecnologías empleadas.

# 2. Visón General de la Arquitectura

## 2.1. Estilo Arquitectónico

El sistema adopta una **arquitectura modular híbrida (monolito modular en 3 capas)**, fundamentada en el patrón **cliente-servidor**. Esta organización busca un equilibrio entre simplicidad de implementación y escalabilidad futura, permitiendo que el sistema pueda evolucionar sin perder orden ni mantenibilidad.

Las **tres capas principales** que conforman la arquitectura son:

* **Capa de Presentación (Frontend):** Implementada como una aplicación web responsiva desarrollada en **React** . Es el punto de interacción directo con los usuarios (turistas y personal administrativo). Proporciona interfaces intuitivas para el acceso a itinerarios, reportes y paneles de administración.
* **Capa de Lógica de Negocio (Backend):** Desarrollada en **Node.js con Express**, organizada en **módulos independientes** (auth, turistas, itinerarios, reportes, notificaciones, administración). Esta capa gestiona las reglas de negocio, la validación de datos, la seguridad, el control de sesiones y la comunicación entre la interfaz de usuario y la base de datos.
* **Capa de Persistencia de Datos (Base de Datos):** Basada en **MySQL** como sistema gestor relacional. Se encarga del almacenamiento estructurado, recuperación eficiente de información y mantenimiento de integridad entre entidades como turistas, itinerarios, hoteles, visitas y usuarios.

## 2.2. Diagrama General de Arquitectura:

Diagrama

El contenido generado por IA puede ser incorrecto.

# 3. Descripción de Componentes Principales

## 3.1. Frontend (Presentación)

**Framework:** React.js

**Estilo:** Tailwind CSS, CSS3

**Responsabilidades:**

* Renderizar interfaces web responsivas e intuitivas para turistas y personal administrativo.
* Capturar entradas y acciones de los usuarios.
* Mostrar mensajes, errores y validaciones de manera clara.
* Comunicarse con la capa de lógica de negocio (backend) mediante fetch/axios para obtener o enviar datos.
* Visualizar itinerarios, reportes y paneles de administración.

## 3.2. Backend (Lógica de negocio)

**Lenguaje y Framework:** Node.js con Express

**Responsabilidades:**

* Exponer API RESTful para que el frontend pueda interactuar con los datos.
* Validar datos recibidos y gestionar errores de manera centralizada.
* Aplicar reglas de negocio, como manejo de tareas vencidas, listas y notificaciones.
* Controlar el acceso mediante JWT (JSON Web Tokens) para autenticación y autorización.
* Coordinar la comunicación entre el frontend y la base de datos.
* Generar reportes y procesamiento de información de itinerarios y usuarios.

### 3.2.1. Módulos del Backend

El backend está organizado en módulos específicos para mantener la modularidad, escalabilidad y facilidad de mantenimiento. Cada módulo tiene responsabilidades concretas dentro del sistema:

* **authController.js**
  + Manejo de login y registro de usuarios.
  + Generación y validación de tokens JWT para autenticación y control de acceso.
  + Gestión de permisos según roles (turista, administrativo, personal).
* **taskController.js**
  + Operaciones CRUD sobre tareas (crear, leer, actualizar, eliminar).
  + Aplicación de reglas de negocio, como tareas vencidas o pendientes.
  + Validación de datos recibidos del frontend y gestión de errores.
* **listController.js**
  + Operaciones CRUD sobre listas de tareas.
  + Asociación de tareas a listas específicas según el usuario.
* **middleware/auth.js**
  + Validación de tokens JWT en solicitudes protegidas.
  + Control de acceso basado en roles y permisos definidos en el sistema.
* **services/emailService.js**
  + Envío de notificaciones internas o por correo electrónico.
  + Gestión de plantillas y contenido de los mensajes según eventos del sistema.

## 3.3. Base de Datos (Persistencia)

* Sistema Gestor: MYSQL
* ORM: Sequelize
* Esquema de Datos:

| **Tabla** | **Campos** | **Descripción** |
| --- | --- | --- |
| **usuarios** | id\_usuario, nombre\_usuario, email, password (encriptada), rol, created\_at, updated\_at | Almacena los datos de acceso y roles de los usuarios del sistema. |
| **turistas** | id\_turista, id\_usuario, nombre, apellido, dni, pasaporte, nacionalidad, fecha\_nacimiento, genero | Información personal de los turistas registrados. |
| **itinerarios** | id\_itinerario, id\_grupo, fecha\_inicio, fecha\_fin, estado\_presupuesto\_id, created\_at, updated\_at | Contiene los itinerarios turísticos asignados a grupos y turistas. |
| **itinerario\_turistas** | id\_itinerario, id\_turista | Relación entre itinerarios y turistas que participan en ellos. |
| **grupos** | id\_grupo, nombre, descripcion | Información de los grupos turísticos. |
| **estados\_presupuesto** | id\_estado, nombre\_estado | Estado del presupuesto de cada itinerario. |
| **programas** | id\_programa, nombre, descripcion, tipo, duracion, costo | Actividades, tours o visitas incluidas en los itinerarios. |
| **itinerario\_programas** | id\_itinerario\_programa, id\_itinerario, id\_programa, fecha | Relación entre itinerarios y programas asignados. |
| **detalle\_machu\_itinerario** | id\_itinerario\_programa, empresa\_tren, horario\_tren\_ida, horario\_tren\_retor, nombre\_guia | Detalles específicos de visitas a Machu Picchu, horarios y guía asignado. |
| **detalle\_machupicchu** | id\_programa, ruta, tiempo\_visita | Información adicional sobre cada programa en Machu Picchu. |

# 4. Integraciones Externas (Opcionales)

Para cumplir los **RF9: Notificaciones internas**, **RF3: Recuperación de contraseña**, etc.:

### Notificaciones

* **Email:** Nodemailer (desarrollo) + SendGrid (producción).
* **SMS (opcional):** Twilio para alertas a turistas.

### Autenticación Social

* **Google OAuth2** (para inicio de sesión opcional).

### Monitorización

* **Sentry** → Captura errores en backend y frontend.
* **LogRocket** → Opcional si quieres ver la interacción del usuario con el frontend React.

# 5. Seguridad

**Encriptación de contraseñas:**

* Usar **bcrypt** para almacenar contraseñas de usuarios de forma segura.
* Asegurarte de que las contraseñas nunca se almacenen en texto plano.

**Autenticación con JWT:**

* Los tokens permiten mantener sesiones sin necesidad de almacenar estado en el servidor.
* Definir **expiración corta** (ej. 1 hora) y refresco de tokens si es necesario.

**Validación y sanitización de entradas:**

* Evitar ataques de **SQL Injection** y **XSS**.
* Librerías recomendadas: **express-validator**, **validator.js**.

# 6. Escalabilidad y Despliegue

Posibles problemas que pueden surgir durante el desarrollo.

* **Frontend**: Desplegado en Vercel o Netlify
* **Backend**: Deploy en Render, Railway o Heroku
* **Base de Dato**s: PostgreSQL en Supabase o ElephantSQL
* La arquitectura permite escalar horizontalmente el backend y separar la base de datos en instancias dedicadas

# 7. Conclusiones

* La arquitectura y SRS que has definido permiten un **equilibrio entre simplicidad, robustez y escalabilidad**.
* Permite implementar **funcionalidades básicas primero** (turistas, itinerarios, reportes) y añadir **extensiones futuras** (reservas, pagos, app móvil).
* Cumple con **buenas prácticas de desarrollo moderno**, incluyendo seguridad, modularidad y escalabilidad.