**Arquitectura del Software, DB y Esquema de Endpoints a APIS**

Este documento proporciona una descripción exhaustiva y detallada de la arquitectura de software para la plataforma digital de ZenzSpa. El objetivo es definir claramente cada componente, justificar las decisiones tecnológicas y establecer una base sólida y escalable para el desarrollo presente y futuro del sistema.

**A. Patrón Arquitectónico: Monolito Híbrido con Extensiones de Servicio**

La arquitectura de ZenzSpa se define como un **"Monolito Híbrido"**. Este enfoque combina la robustez y simplicidad de gestión de un núcleo monolítico con la flexibilidad y escalabilidad de los microservicios para funcionalidades específicas y desacopladas.

* **El Núcleo Monolítico (Django):**
  + **Responsabilidad:** Contiene la lógica de negocio central, crítica y altamente acoplada. Esto incluye la gestión de usuarios y autenticación, el procesamiento de citas, la gestión de productos y órdenes, y las reglas de negocio financieras.
  + **Justificación:** Centralizar estas funcionalidades reduce la latencia de comunicación entre componentes críticos (no hay llamadas de red entre la creación de una orden y la validación de un usuario), simplifica las transacciones de base de datos (garantizando consistencia ACID) y facilita el desarrollo y despliegue inicial. Mantener la lógica de negocio cohesionada en un solo lugar previene la complejidad distribuida de un sistema de microservicios puro en esta etapa.
* **Las Extensiones de Servicio (Serverless Functions):**
  + **Responsabilidad:** Manejan tareas específicas, independientes y que pueden requerir escalabilidad o tecnologías distintas. El principal ejemplo es el **backend del Chatbot**.
  + **Justificación:**
    1. **Desacoplamiento Tecnológico:** El chatbot necesita interactuar con la API de IA de Google y realizar búsquedas vectoriales. Aislar esta lógica en una función serverless (Node.js o Python) permite usar las librerías y el entorno más adecuados para esa tarea sin sobrecargar el monolito de Django.
    2. **Escalabilidad Independiente:** El tráfico del chatbot puede ser impredecible y con picos. Una función serverless puede escalar de cero a miles de instancias automáticamente, manejando cualquier carga sin impactar el rendimiento del núcleo de la aplicación (agendamiento de citas, pagos).
    3. **Seguridad:** El servicio del chatbot se comunica con el monolito a través de una API REST segura, utilizando un token de API dedicado. Esto limita la superficie de ataque; una vulnerabilidad en la extensión de servicio no compromete directamente la base de datos principal.

**B. Frontend: Svelte/SvelteKit - Rendimiento y Experiencia de Usuario**

La elección de Svelte/SvelteKit no es solo una preferencia, sino una decisión estratégica para lograr una interfaz de usuario extremadamente rápida y reactiva.

* **Modelo de Compilación:** A diferencia de React o Vue que utilizan un DOM Virtual en el navegador, Svelte es un **compilador**. Convierte los componentes .svelte en código JavaScript imperativo, altamente optimizado y sin sobrecargas en tiempo de ejecución. Esto resulta en:
  + **Bundles más pequeños:** Menos código para descargar, lo que significa tiempos de carga inicial más rápidos.
  + **Mayor rendimiento:** Las actualizaciones del DOM son quirúrgicas y directas, sin la necesidad de "diffing" de un DOM virtual.
* **Renderizado del Lado del Servidor (SSR):** SvelteKit ofrece SSR de forma nativa. Esto es crucial para ZenzSpa por dos razones:
  + **SEO (Search Engine Optimization):** Las páginas de servicios y productos se renderizan en el servidor y se envían como HTML completo, permitiendo a los motores de búsqueda indexar el contenido eficazmente.
  + **Time to First Contentful Paint (FCP):** El usuario ve el contenido de la página casi instantáneamente, mejorando drásticamente la percepción de velocidad, incluso antes de que el JavaScript interactivo se cargue por completo.
* **Arquitectura Integrada:** SvelteKit proporciona un enrutador basado en el sistema de archivos, endpoints de API (para interactuar con el backend de Django) y una gestión de estado simple, creando un marco de trabajo cohesivo que simplifica el desarrollo del frontend.

**C. Backend (Django): Profundización en Optimizaciones y Lógica**

* **Optimización de Consultas a la Base de Datos:**
  + select\_related**:** Se usará para relaciones uno a uno (O2O) y muchos a uno (FK). Ejemplo: al obtener una lista de Appointment, se usará Appointment.objects.select\_related('user', 'service', 'staff\_member') para traer toda la información relacionada en una única y eficiente consulta SQL con JOINs, en lugar de N+1 consultas.
  + prefetch\_related**:** Se usará para relaciones muchos a muchos (M2M) y relaciones inversas. Ejemplo: al obtener el perfil de un cliente (UserProfile), se usará UserProfile.objects.prefetch\_related('localizedpain\_set') para obtener todos sus registros de dolor en una segunda consulta, uniendo los datos en Python de forma eficiente.
* **Estrategia de Caching Detallada con Redis:**
  + **Caché de Entidades:** Listas de Service, ServiceCategory y Package se cachearán con un TTL (Time-To-Live) largo (ej. 24 horas), ya que su contenido cambia con poca frecuencia.
  + **Invalidación de Caché:** La caché se invalidará automáticamente al realizar una operación de escritura (Crear, Actualizar, Eliminar) sobre estos modelos, utilizando señales de Django (post\_save, post\_delete) para purgar las claves correspondientes.
  + **Caché de Cálculos:** Los resultados de consultas complejas para reportes se podrán cachear por períodos cortos (ej. 1 hora) para evitar recalcularlos en cada visita al panel de analíticas.
* **Tareas Asíncronas Detalladas (Celery):**
  + recordatorio\_cita **(Tarea Periódica - Cron):** Se ejecuta cada 15 minutos. Busca citas confirmadas cuyo start\_time sea en las próximas 24-25 horas y que no hayan recibido un recordatorio. Envía el email correspondiente.
  + cancelacion\_por\_inactividad **(Tarea Periódica - Cron):** Se ejecuta cada 10 minutos. Busca citas PENDING\_PAYMENT cuyo start\_time sea en menos de 24 horas y cambia su estado a CANCELLED\_UNPAID.
  + otorgar\_beneficio\_lealtad **(Tarea Periódica - Cron):** Se ejecuta una vez al día (ej. a medianoche). Busca usuarios VIP cuya suscripción continua cumpla 3 meses y les asigna un voucher de servicio gratuito.
  + enviar\_email\_transaccional **(Tarea Eventual):** Se dispara inmediatamente después de un evento (ej. pago confirmado, registro de usuario). Recibe el tipo de email y los datos del destinatario y se encarga del envío en segundo plano para no demorar la respuesta de la API.

**D. Stack Tecnológico Consolidado y Detallado**

|  |  |  |  |
| --- | --- | --- | --- |
| **Capa** | **Tecnología** | **Versión/Detalles** | **Justificación Detallada y Notas** |
| **Frontend** | Svelte/SvelteKit | Última estable | Marco reactivo compilado. Proporciona SSR para SEO, bundles de producción mínimos y una experiencia de desarrollo unificada con enrutamiento y endpoints. |
| **Estilos Frontend** | TailwindCSS | Última estable | Framework CSS utility-first. Acelera el desarrollo al permitir construir diseños complejos directamente en el HTML, garantizando consistencia visual sin escribir CSS personalizado. |
| **Backend API (Core)** | Django | 5.x | Framework Python "baterías incluidas". Proporciona ORM, sistema de migraciones, panel de administración y seguridad robusta, reduciendo el tiempo de desarrollo. |
| **Framework API** | Django REST Framework | Última estable | El estándar de oro para construir APIs REST en Django. Proporciona serializadores, vistas genéricas y un sistema de permisos flexible y granular. |
| **Autenticación API** | DRF SimpleJWT | Última estable | Implementación estándar y segura de JSON Web Tokens (Access + Refresh) para APIs sin estado. |
| **Tareas Asíncronas** | Celery | Última estable | Sistema distribuido de colas de tareas. Esencial para ejecutar procesos largos (enviar emails, procesar datos) en segundo plano sin afectar la experiencia del usuario. |
| **Broker Mensajes/Caché** | Redis | Última estable | Almacén de datos en memoria ultra rápido. Se usará con dos bases de datos lógicas: BD 0 como broker para Celery, BD 1 como backend de caché para Django. |
| **BD (Principal)** | PostgreSQL | 15+ (producción) | SGBD relacional de código abierto, altamente fiable y con soporte estricto de transacciones ACID. Ideal para datos financieros y de reservas. |
| **BD (Vectores)** | PostgreSQL + pgvector | Integrado | Extensión para PostgreSQL que permite almacenar y consultar vectores de alta dimensionalidad. Unifica la base de datos, evitando la necesidad de un servicio externo para el chatbot. |
| **Backend (Chatbot)** | Vercel Serverless Functions | Node.js/Python | Entorno serverless que escala automáticamente con la demanda. Perfecto para la lógica del chatbot, que puede tener picos de tráfico independientes del core de la app. |
| **Asistente Virtual (IA)** | Google Gemini 1.5 Flash API | API REST | Modelo de IA de última generación, optimizado para ser rápido y costo-eficiente, ideal para aplicaciones de chat de alto volumen. |
| **Testing** | Pytest con pytest-django | Última estable | Framework de testing para Python. Permite escribir pruebas unitarias y de integración de forma clara y concisa para garantizar la calidad del código del backend. |
| **Formato y Linting** | Black, isort, Flake8 | Últimas estables | Herramientas para formateo de código automático y detección de errores, garantizando un estilo de código consistente y limpio en todo el proyecto. |
| **Pagos** | Wompi | API REST + Webhooks | Pasarela de pagos líder en Colombia, con un flujo de integración claro y seguro basado en webhooks para la confirmación de transacciones. |
| **Email Transaccional** | SendGrid / Brevo | API + SMTP | Servicios especializados en entrega de emails. Garantizan una alta tasa de entrega, superando las limitaciones de los servidores de correo estándar. |
| **SMS** | Twilio Verify | API | Servicio líder en la industria para la verificación de dos factores vía SMS. Fiable y seguro para el proceso de registro. |
| **Notificaciones Real-Time** | Django Channels | Última estable | Extiende Django para manejar protocolos de larga duración como WebSockets, permitiendo la comunicación bidireccional en tiempo real con el frontend. |
| **Infraestructura (Dev)** | Docker, docker-compose | Últimas estables | Contenerización. Garantiza que el entorno de desarrollo sea idéntico al de producción, eliminando problemas de "en mi máquina funciona". |
| **Despliegue** | Vercel | - | Plataforma de despliegue especializada en frontends modernos. Ofrece integración con Git, CI/CD automático, CDN global y funciones serverless, todo en un solo lugar. |

**IV. Arquitectura de la Base de Datos (Detallada)**

* **Motor y Extensión:** Se utilizará PostgreSQL 15, habilitando la extensión pgvector (CREATE EXTENSION vector;). Esto permite crear columnas de tipo vector(N) donde N es la dimensionalidad del modelo de embedding utilizado (ej. 768).
* **Modelo de Dominio y Flujos Clave:**
  + **Reagendamiento:** El flujo para una cita CONFIRMED es unidireccional para el cliente: solo puede pasar a un nuevo estado de CONFIRMED con una fecha distinta (a través del endpoint de reschedule), incrementando el reschedule\_count. La cancelación final con reembolso (CANCELLED\_BY\_ADMIN -> REFUNDED) es una acción exclusiva del administrador.
  + **Transacciones Atómicas:** El endpoint reschedule estará envuelto en transaction.atomic(). Si la actualización de la fecha de la cita tiene éxito pero, por alguna razón, falla el guardado del reschedule\_count, toda la operación se revierte, evitando dejar la base de datos en un estado inconsistente.
* **Tabla Detallada de Modelos Principales:**

|  |  |  |
| --- | --- | --- |
| **Nombre del Modelo** | **Atributos Clave Detallados** | **Descripción/Propósito en ZenzSpa** |
| CustomUser | phone\_number (unique), email (unique, opcional), password (hash), role (Choices), is\_verified, is\_persona\_non\_grata, internal\_photo\_url, internal\_notes | Modelo central de usuario. El phone\_number es el identificador principal para el login. |
| UserProfile | user (O2O), dosha (Choices), elemento (Choices), diet\_type, sleep\_quality, allergies\_notes, injuries\_history | Perfil clínico holístico del cliente. Vinculado 1-a-1 con CustomUser. |
| ServiceCategory | name (unique), description, is\_low\_supervision (boolean) | Agrupa servicios. El flag is\_low\_supervision habilita la lógica de sobrereserva. |
| Service | category (FK), name, description, duration (integer, en minutos), price, vip\_price (opcional), is\_active | Define un servicio ofrecido. is\_active permite ocultarlo del catálogo sin borrarlo. |
| StaffAvailability | staff\_member (FK a CustomUser con role=STAFF), day\_of\_week (integer 1-7), start\_time, end\_time | Define un bloque de tiempo laboral de un terapeuta. Un terapeuta puede tener múltiples bloques por día. |
| Appointment | user, service, staff\_member (FKs), start\_time, end\_time, status (Choices), price\_at\_purchase, reschedule\_count (integer) | Registro de una cita. status es la clave de su ciclo de vida. price\_at\_purchase congela el precio al momento de la reserva. |
| KnowledgeChunk | source\_document (char), content (text), content\_vector (vector) | **(Nuevo)** Almacena un fragmento de texto de la base de conocimiento del chatbot y su representación vectorial para la búsqueda de similitud. |
|  |  |  |

**V. Esquema de Endpoints API Detallado (v1)**

* **Autenticación y Seguridad (**/api/v1/auth/**,** /api/v1/admin/**)**
  + POST /auth/register/: body: {phone\_number, password}.
  + POST /auth/login/: body: {phone\_number, password} -> response: {access, refresh}.
  + POST /admin/users/{user\_id}/flag-non-grata/: body: {notes, photo\_file}. (Requiere rol ADMIN).
* **Catálogo (**/api/v1/catalog/**)**
  + GET /catalog/services/: params: {category\_id}.
  + GET /catalog/service-categories/.
  + GET /catalog/packages/.
* **Agendamiento y Horarios (**/api/v1/booking/**)**
  + POST /booking/availability-check/: body: {service\_id, date} -> response: [{staff\_id, available\_slots: [...]}].
  + POST /booking/appointments/: body: {service\_id, staff\_id, start\_time}.
  + POST /booking/appointments/{id}/reschedule/: body: {new\_staff\_id, new\_start\_time}. (Requiere rol CLIENT o VIP).
  + POST /admin/appointments/{id}/cancel/: (Requiere rol ADMIN).
* **Chatbot (Endpoint Externo)**
  + POST https://<your-vercel-url>/api/chat/message: body: {message, conversation\_history, user\_id (opcional)}.
* **Gestión de Usuario (**/api/v1/me/**)**
  + GET /me/profile/: Devuelve el perfil del usuario autenticado.
  + GET /me/appointments/: Devuelve el historial de citas del usuario.
  + GET /me/orders/: Devuelve el historial de compras de productos.