**Seguridad, Pruebas y Despliegue**

**VI. Plan de Seguridad**

La seguridad es un aspecto crítico para ZenzSpa, especialmente considerando el manejo de datos personales, de salud y transacciones financieras. El plan de seguridad abarca múltiples capas.

**A. Autenticación y Autorización**

* **A. Autenticación y Autorización (Reforzado):**
  + **Permisos a Nivel de API:** Se crearán clases de permisos personalizadas en Django REST Framework (ej. IsAdminUser, IsStaffUser) para proteger endpoints sensibles.
  + **Enmascaramiento de Datos:** Los serializadores se adaptarán para ocultar información sensible según el rol. El STAFF no podrá ver el teléfono o email de los clientes a través de la API.
* **Roles y Permisos (RF03):** Django REST Framework se utilizará para definir permisos granulares basados en los roles de usuario (CLIENT, VIP, STAFF, ADMIN), asegurando que cada usuario solo pueda acceder a los recursos y realizar las acciones para las que está autorizado.1

**B. Protección contra Ataques Comunes**

* **OWASP Top 10:** El desarrollo se guiará por las recomendaciones del OWASP Top 10 para prevenir las vulnerabilidades web más comunes.1
* **Content Security Policy (CSP):** Se implementará una política de seguridad de contenido estricta utilizando django-csp para mitigar ataques de XSS y de inyección de datos, restringiendo los orígenes desde los cuales se puede cargar contenido.1
* **Rate Limiting:**
  + Para el endpoint de login (/api/v1/auth/login), se aplicará un límite de 5 intentos por minuto por IP.1
  + Para la verificación de SMS (/api/v1/auth/verify), el límite será de 3 intentos cada 10 minutos por IP.1 Estas medidas se implementarán con django-axes o django-ratelimit.
  + Se aplicará throttling general a otros endpoints sensibles de la API utilizando las capacidades de DRF para prevenir abusos y ataques de fuerza bruta.1
* **Prevención de Cross-Site Scripting (XSS):** Svelte, por defecto, escapa las variables en sus plantillas, lo que ayuda a prevenir XSS. De manera similar, las plantillas de Django también realizan auto-escapado. Se prestará especial atención a cualquier instancia donde se renderice HTML directamente para asegurar que no se introduzcan vulnerabilidades.
* **Prevención de SQL Injection:** El uso del ORM de Django protege inherentemente contra ataques de inyección SQL, ya que las consultas se construyen de forma parametrizada.

Aunque Svelte ofrece protecciones contra XSS por defecto, es crucial recordar que la seguridad integral de la aplicación depende fundamentalmente de validaciones robustas en el backend y un diseño de API seguro. El cambio de frontend no disminuye la importancia de las medidas de seguridad del lado del servidor; las validaciones críticas (permisos, integridad de datos) deben realizarse siempre en el backend (Django), ya que el código del lado del cliente puede ser manipulado o eludido.

**C. Seguridad de Datos**

* **SSL/TLS Forzado:** Todo el tráfico entre el cliente y el servidor se cifrará obligatoriamente mediante HTTPS. Se utilizará Let’s Encrypt para la gestión automática y renovación de certificados SSL.1
* **Cumplimiento Legal (Colombia):**
  + **Ley 1581 de 2012 (Habeas Data):** Se implementará un registro detallado de los consentimientos otorgados por los usuarios para el tratamiento de sus datos personales. La política de tratamiento de datos será visible y accesible. Se solicitará consentimiento explícito para la recolección de datos personales y, especialmente, de salud.1
  + **Banderas de Salud:** El acceso a la información sensible de salud de los clientes estará estrictamente restringido al personal autorizado (terapeutas y administradores) y solo para los fines pertinentes al servicio.

**D. Verificación y Protección de Formularios**

* **Verificación Telefónica (Twilio Verify - RF01):** Este proceso no solo verifica la autenticidad del número de teléfono del usuario, sino que también añade una capa de seguridad al registro, dificultando la creación de cuentas falsas o abusivas.1
* **reCAPTCHA:** Se integrará reCAPTCHA (de Google u otro proveedor similar) en los formularios de registro y login para proteger contra el abuso por parte de bots.1
* **D. Verificación y Protección de Formularios:**
  + **Mecanismo "Cliente No Grato":** La funcionalidad de bloquear usuarios y prevenir su re-registro por número de teléfono

**E. Gestión de Dependencias**

Se establecerá un proceso para escanear regularmente las dependencias del proyecto (tanto de frontend como de backend) en busca de vulnerabilidades conocidas, utilizando herramientas como Snyk 1 o pip-audit / npm audit. Las dependencias desactualizadas o vulnerables serán actualizadas de manera oportuna.

**VII. Estrategia de Pruebas**

Una estrategia de pruebas exhaustiva es esencial para garantizar la calidad, fiabilidad y correcto funcionamiento de ZenzSpa, especialmente tras la introducción de nuevas tecnologías y optimizaciones.

**A. Niveles de Prueba y Herramientas**

La estrategia de pruebas se adaptará para incluir herramientas específicas para el nuevo stack de frontend basado en Svelte/SvelteKit.

* **1. Pruebas Unitarias (Backend):**
  + **Herramientas:** pytest junto con factory\_boy para la generación eficiente de datos de prueba.1
  + **Cobertura Meta:** Se buscará una cobertura de código del 75% para la lógica de negocio, modelos y servicios en Django.1
  + **Foco:** Validar el comportamiento de unidades aisladas de código en el backend (funciones, métodos de modelos, lógica de servicios).
* **2. Pruebas de API (Integración Backend):**
  + **Herramientas:** Se utilizará APITestCase de Django REST Framework.1
  + **Cobertura Meta:** Se priorizará la cobertura de los endpoints críticos, especialmente aquellos involucrados en los flujos de reserva, pago, autenticación y gestión de usuarios.1
  + **Foco:** Verificar los ciclos completos de solicitud/respuesta de la API, incluyendo la correcta serialización/deserialización de datos, códigos de estado HTTP, autenticación y autorización.
* **3. Pruebas Unitarias (Frontend - Svelte):**
  + **Herramientas:** Se utilizará **Vitest**, un framework de pruebas unitarias rápido y moderno que se integra de forma nativa con Vite (el motor de compilación de SvelteKit), junto con **Svelte Testing Library**. Estas herramientas reemplazan a React Testing Library, mencionada en el stack anterior.1
  + **Cobertura Meta:** Se mantendrá el objetivo de una cobertura del 70% para los componentes de Svelte.1
  + **Foco:** Probar componentes Svelte individuales, verificando su renderizado, props, eventos, slots, y comportamiento reactivo de manera aislada. Svelte Testing Library fomenta escribir pruebas que interactúan con los componentes de la misma manera que lo haría un usuario.
* **4. Pruebas End-to-End (E2E):**
  + **Herramientas:** Se continuará utilizando Cypress, ejecutando pruebas en navegadores Chrome y Firefox.1
  + **Cobertura Meta:** Se cubrirán los flujos de usuario más críticos y completos, como el proceso de reserva de una cita de principio a fin (incluyendo pago simulado), registro de nuevos usuarios, inicio de sesión, y la configuración del "Momento Zen".1
  + **Foco:** Simular interacciones reales de los usuarios a través de toda la aplicación, desde el frontend SvelteKit hasta el backend Django y la base de datos, asegurando que todos los componentes integrados funcionen correctamente juntos.

La adopción de Svelte/SvelteKit requiere un ajuste en las herramientas de prueba del frontend. Vitest es una elección natural por su compatibilidad y velocidad con proyectos basados en Vite. Svelte Testing Library ofrece utilidades específicas para interactuar con componentes Svelte, siguiendo principios similares a los de React Testing Library pero adaptados al ecosistema Svelte, lo que facilitará la transición para equipos con experiencia previa en pruebas de componentes.

**B. Entorno de Pruebas**

* Se configurará una base de datos PostgreSQL dedicada para la ejecución de pruebas, asegurando el aislamiento de los datos de desarrollo y producción.
* Los servicios externos como Wompi y Twilio serán mockeados o se utilizarán entornos de sandbox durante las pruebas automatizadas. Esto garantiza pruebas deterministas, reduce costos y evita la dependencia de la disponibilidad de servicios de terceros. Herramientas como MSW (Mock Service Worker) pueden ser útiles para mockear APIs en el frontend durante las pruebas de componentes o E2E.1

**C. Integración Continua (CI)**

Todas las pruebas (unitarias de backend y frontend, de API y E2E) se ejecutarán automáticamente como parte del pipeline de Integración Continua en GitHub Actions. Esto ocurrirá en cada push a una rama o al crear un pull request, proporcionando feedback rápido sobre la calidad del código y ayudando a prevenir regresiones antes de que el código se integre a la rama principal.1

**VIII. Plan de DevOps y Despliegue**

El plan de DevOps y despliegue está diseñado para asegurar entregas eficientes, consistentes y fiables de la aplicación ZenzSpa, incorporando las particularidades del stack tecnológico actualizado.

**A. Integración Continua y Entrega Continua (CI/CD)**

* **Plataforma:** Se utilizará **GitHub Actions** para la orquestación de los flujos de CI/CD.1
* **Flujo de CI/CD:**
  1. **Activación:** El flujo se activa con un push a cualquier rama o la creación/actualización de un pull request en GitHub.
  2. **Linting y Análisis Estático:** Ejecución de linters (e.g., Flake8 para Python, ESLint/Prettier para Svelte/JavaScript, Stylelint para CSS) para asegurar la calidad y consistencia del código.
  3. **Pruebas Automatizadas:** Ejecución de todas las suites de pruebas: unitarias de backend (Pytest), unitarias de frontend (Vitest), pruebas de API (DRF APITestCase) y pruebas E2E (Cypress).
  4. **Construcción (Build):**
     + **Backend (Django):** Construcción de una imagen Docker optimizada para producción, utilizando un Dockerfile multistage para reducir el tamaño final de la imagen.1
     + **Frontend (SvelteKit):** Compilación de la aplicación SvelteKit. El resultado dependerá del adaptador SvelteKit elegido (e.g., adapter-node genera un servidor Node.js, adapter-static genera archivos estáticos).
  5. **Push a Registro:** La imagen Docker del backend se subirá a un registro de contenedores (e.g., Docker Hub, GitHub Container Registry). Los artefactos de compilación del frontend (si son estáticos o parte de una imagen Node.js) también se almacenarán o empaquetarán según corresponda.
  6. **Despliegue (Deploy):** Despliegue automatizado o manual (según la estrategia para cada entorno) al servidor VPS en DigitalOcean, utilizando docker-compose para orquestar los contenedores del backend. El despliegue del frontend SvelteKit se coordinará con esto.
  7. Se considera una buena práctica la implementación de "review apps" (entornos temporales por rama para revisión) si la infraestructura y el presupuesto lo permiten, como se sugiere en el material de referencia.1

**B. Infraestructura**

* **Hosting:** Servidores Privados Virtuales (VPS Droplets) en **DigitalOcean**.1
* **Contenedorización:** **Docker** y docker-compose se utilizarán para gestionar la aplicación Django, la base de datos PostgreSQL, el servidor Redis y el servidor web Nginx. Esto asegura entornos consistentes desde el desarrollo hasta la producción.1
  + Se utilizará un Dockerfile multistage para la aplicación Django, separando el entorno de compilación del de producción para optimizar el tamaño de la imagen final.1
* **Servidor Web y Proxy Inverso:** **Nginx** actuará como servidor web principal. Sus funciones incluirán:
  + Servir los archivos estáticos generados por SvelteKit (especialmente si se usa adapter-static o para los assets de una aplicación SSR).
  + Actuar como proxy inverso para el servidor de aplicaciones Python (Gunicorn sirviendo Django).
  + Si SvelteKit se despliega como un servidor Node.js (usando adapter-node para SSR), Nginx también actuará como proxy inverso para este servidor Node.js.
* **Base de Datos:** **PostgreSQL 15**. Podrá ser un servicio gestionado ofrecido por DigitalOcean para simplificar la administración y backups, o un contenedor Docker gestionado por docker-compose.1
* **Caché y Message Broker:** **Redis**. Similarmente a PostgreSQL, puede ser un servicio gestionado o un contenedor Docker.

El despliegue de una aplicación SvelteKit introduce matices dependiendo de su modo de renderizado. Si se opta mayoritariamente por Server-Side Rendering (SSR) con adapter-node, la aplicación SvelteKit se ejecutará como un servidor Node.js independiente. Nginx deberá configurarse para dirigir las peticiones de las rutas del frontend a este servidor Node.js, además de las peticiones de API al backend Django (Gunicorn). Si, por otro lado, se utiliza Static Site Generation (SSG) con adapter-static, SvelteKit generará archivos HTML, CSS y JS estáticos que Nginx podrá servir directamente, simplificando esta parte del despliegue. El plan de DevOps debe detallar estos pasos específicos para la construcción y el despliegue del frontend SvelteKit.

**C. Gestión de Activos Estáticos y CDN**

* **Frontend (SvelteKit):** SvelteKit compila el código fuente en archivos estáticos (JavaScript, CSS, imágenes optimizadas). Estos serán servidos eficientemente por Nginx, configurado con cabeceras de cacheo apropiadas.
* **Media y Contenido Pesado (Videos 4K, Imágenes HD):** Para contenido como videos promocionales o imágenes de alta resolución del blog VIP (RF12), se utilizará un servicio de CDN como **DigitalOcean Spaces** (o un servicio compatible con S3 como AWS S3). Esto descarga al servidor principal de la tarea de servir estos archivos pesados, mejorando los tiempos de carga y reduciendo el consumo de ancho de banda del servidor de la aplicación.1

**D. Observabilidad**

* **Monitorización de Errores:** **Sentry** se integrará tanto en el frontend SvelteKit como en el backend Django para la captura y alerta temprana de errores en tiempo real.1
* **Logging:** Se utilizará **Grafana Loki** para la agregación centralizada, búsqueda y visualización de logs generados por todos los componentes de la aplicación (Django, Nginx, SvelteKit si es SSR).1
* **Métricas:** **Grafana** (potencialmente con Prometheus si se requiere un monitoreo de métricas más detallado) se usará para visualizar métricas de rendimiento del sistema y de la aplicación.

**E. Gestión de Secretos**

Las credenciales sensibles (claves de API, contraseñas de base de datos, SECRET\_KEY de Django) se gestionarán de forma segura:

* Durante el CI/CD, se utilizarán **GitHub Secrets**.1
* En el servidor de producción, se inyectarán como variables de entorno a los contenedores Docker, gestionadas a través de docker-compose.prod.yml o herramientas específicas de gestión de secretos del proveedor de cloud si se escala la infraestructura.

**F. Mantenimiento**

* **Renovación de SSL:** Se configurará un cron job para la renovación automática de los certificados SSL de Let’s Encrypt, asegurando que la comunicación HTTPS no se interrumpa.1
* **Backups de Base de Datos:** Se implementará una estrategia de backups regulares y automáticos para la base de datos PostgreSQL. Se explorará la opción de Point-in-Time Recovery (PITR) si es ofrecida por el servicio de base de datos gestionada o se configurará adecuadamente si es un contenedor.1

**IX. Requisitos No Funcionales Clave**

Los requisitos no funcionales (NFRs) definen los atributos de calidad del sistema ZenzSpa. Son cruciales para asegurar una buena experiencia de usuario y la viabilidad técnica del proyecto.

**A. Rendimiento**

Este es un NFR primordial, impulsado por el objetivo de lograr una "sensación instantánea".

* **First Contentful Paint (FCP):** El contenido principal visible de la página debe cargarse en **menos de 1.5 segundos** en una conexión 3G simulada. La elección de SvelteKit, con sus bundles pequeños y compilación eficiente, contribuye directamente a este objetivo.1
* **Time to Interactive (TTI):** La página debe ser completamente interactiva en **menos de 3 segundos** en una conexión 3G simulada.1
* **Rendimiento del Backend:** El servidor debe ser capaz de manejar **500 usuarios concurrentes** con un tiempo de respuesta mediano (p50) inferior a **250 milisegundos** para las solicitudes a la API.1 La optimización de consultas y el cacheo con Redis son claves para este NFR.
* **"Sensación Instantánea":** Este es el objetivo cualitativo general que guía las decisiones de optimización tanto en el frontend como en el backend. Se materializa a través del cumplimiento de métricas cuantitativas como FCP y TTI. La arquitectura con SvelteKit y las optimizaciones de backend (cacheo, resolución de N+1) están diseñadas específicamente para alcanzar estos umbrales y, por ende, la percepción de instantaneidad por parte del usuario.2

**B. Seguridad**

* Cumplimiento de las directrices **OWASP Top 10** como línea base para la prevención de vulnerabilidades web.1
* **SSL/TLS forzado** en toda la comunicación de la aplicación para garantizar la encriptación de los datos en tránsito.1
* Adherencia a la **Ley 1581 de 2012 de Colombia** para la protección y tratamiento de datos personales.1

**C. Accesibilidad**

* Cumplimiento del nivel **AA de las Web Content Accessibility Guidelines (WCAG) 2.1**. Esto incluye consideraciones como contraste de color adecuado, navegación por teclado, y soporte para tecnologías de asistencia.1
* Se implementarán características como el respeto a la preferencia prefers-reduced-motion y el uso de atributos aria-live para regiones dinámicas, como se sugiere en el material de referencia.1 Svelte también ofrece verificaciones de accesibilidad integradas durante el desarrollo, lo que puede ayudar a identificar y corregir problemas tempranamente.3

**D. Escalabilidad**

* La arquitectura debe estar diseñada para soportar el crecimiento futuro, con una meta inicial de **500 usuarios concurrentes** manteniendo los tiempos de respuesta del servidor (p50 < 250ms).1
* La infraestructura basada en Docker y alojada en DigitalOcean debe permitir el escalado horizontal (añadiendo más instancias de servidor) o vertical (aumentando los recursos de los servidores existentes) según sea necesario.

**E. Mantenibilidad**

* Se promoverá un código limpio, bien documentado y siguiendo las mejores prácticas de cada tecnología.
* En el backend Django, se fomentará la **modularización en apps** cohesivas y con bajo acoplamiento, siguiendo principios como los de Django Clean Architecture si la complejidad lo amerita.1
* La arquitectura basada en componentes de Svelte y su sintaxis concisa pueden contribuir significativamente a la mantenibilidad del código frontend.3

**F. Fiabilidad**

* Implementación de políticas de reintento robustas para las tareas asíncronas gestionadas por Celery, para manejar fallos transitorios.1
* Desarrollo de un mecanismo de fallback o sondeo para los webhooks de Wompi, para asegurar la consistencia de los datos de pago incluso si una notificación de webhook falla.1

**X. Conclusión y Próximos Pasos Recomendados**

La arquitectura y las mejoras tecnológicas propuestas para ZenzSpa representan una evolución significativa orientada a cumplir con las altas expectativas de rendimiento y experiencia de usuario. La adopción de Svelte/SvelteKit para el frontend, combinada con optimizaciones estratégicas en el backend Django y un uso extensivo de Redis para cacheo y tareas asíncronas, sienta las bases para una aplicación moderna, rápida y escalable.

**A. Resumen de Beneficios**

La implementación de estas actualizaciones posiciona a ZenzSpa para:

* **Ofrecer una "sensación instantánea":** Tiempos de carga reducidos y una interfaz de usuario altamente reactiva gracias a SvelteKit y las optimizaciones de backend.
* **Mejorar la escalabilidad y fiabilidad:** Una infraestructura robusta y estrategias de optimización que permiten manejar un mayor número de usuarios y asegurar la consistencia de los datos.
* **Aumentar la mantenibilidad:** Código más limpio y modular tanto en el frontend como en el backend.
* **Fortalecer la seguridad:** Adopción de mejores prácticas y herramientas para proteger los datos de los usuarios y la integridad de la plataforma.
* **Alinear la tecnología con los objetivos de negocio:** Una plataforma de alto rendimiento es fundamental para alcanzar los KPIs de reservas online, retención de usuarios VIP y satisfacción del cliente.1

**C. Consideraciones Adicionales**

* **Capacitación del Equipo (Team Upskilling):** Si el equipo de desarrollo no tiene experiencia previa con Svelte/SvelteKit, se debe planificar un tiempo adecuado para el aprendizaje y la familiarización con esta tecnología y sus mejores prácticas.
* **Validación Iterativa del Rendimiento:** Alcanzar la "sensación instantánea" y cumplir con los NFRs de rendimiento es un proceso continuo. Tras la implementación inicial de SvelteKit y las optimizaciones de backend, es crucial realizar pruebas de rendimiento exhaustivas (utilizando herramientas como Lighthouse, WebPageTest, y perfiles de backend) bajo condiciones realistas. Estos resultados guiarán refinamientos adicionales, como optimizaciones específicas de componentes Svelte, ajustes en las estrategias de cacheo o indexación adicional en la base de datos. Este ciclo de construir, medir, aprender y refinar será fundamental.
* **Revisión de Omisiones Potenciales:** Es importante revisar y planificar el abordaje de las "OMISIONES POTENCIALES A VIGILAR" identificadas en el blueprint original 1, tales como la gestión avanzada de pagos recurrentes fallidos, consideraciones de GDPR si la aplicación se expande geográficamente, la escalabilidad de Redis a largo plazo, y los fallbacks para servicios externos críticos.