

Instituto Tecnológico de Costa Rica**Administración de Tecnologías de Información****TI-3600 Bases de datos**

Grupo 02

Proyecto 2 – Documentación completa

Docente

Ing Jacqueline Solís

Estudiante

Nelson Chavarría Aragón

I Semestre

Marzo, 2025

Contenido

Descripción del proyecto:.....	3
Alcance del proyecto:	3
Arquitectura y Requerimientos Técnicos	3
Roles del equipo.....	4
Requerimientos:	4
Tabla de terminología de la matriz de requerimientos:	4
Tabla de requerimientos de sistema.....	5
Propuesta de selección de tecnologías	11
Comparativa de arquitecturas.....	14
Estimación de costos totales de proyectos.....	16
Costos anuales esperados en el proyecto.....	18
Costos totales en el primer año.....	20
Análisis de costos CAPEX/OPEX	20
Diagrama ER.....	21
Cronograma con diagrama de Gantt.....	22
Implementación de la base de datos	22
Capturas de la creación de las tablas.....	22
Capturas de las funciones, triggers y procedimientos	27
Mecanismos de control de concurrencia	37
Índices creados	41
Corridas de ejemplo de la base de datos.....	42
API REST.....	45
Entorno de desarrollo	45
Diseño de software para la implementación del api	45
Instrucciones para el uso del API	46
Resultados de uso del api	49
Demostración de funcionamiento del API REST.....	50
Pruebas de carga sobre el API	52
Recuperación de datos ante problemas.....	55
Anexos.....	56

Descripción del proyecto:

El proyecto se basará en diseñar e implementar un sistema que provea escalabilidad en la gestión de reservas de hoteles para una cadena internacional, esta misma debe de ofrecer la posibilidad de realizar diferentes operaciones, la misma debe de garantizar una alta disponibilidad, seguridad, y capacidad de crecimiento proyectado a cinco años.

Alcance del proyecto:

Este proyecto busca diseñar e implementar una plataforma centralizada de reservas para una cadena hotelera con presencia internacional, capaz de manejar múltiples propiedades de forma unificada. La solución priorizará la escalabilidad, la seguridad y la integración con sistemas existentes, garantizando operaciones fluidas incluso ante un crecimiento estimado del 10% anual. Entre los objetivos críticos destacan la tolerancia a fallos, con un RPO (pérdida máxima de datos) de 5 minutos y un RTO (tiempo de recuperación) de 30 minutos, asegurando una disponibilidad del 99.98%.

El sistema cubrirá todas las etapas del ciclo de reservas: gestión de clientes (incluyendo preferencias e historial), control de inventario de habitaciones en tiempo real, procesamiento de reservas individuales y grupales, así como integración con pasarelas de pago. Para agilizar procesos, se incorporarán automatizaciones mediante *triggers* y procedimientos almacenados, enfocados en confirmaciones, cancelaciones y políticas personalizadas.

Arquitectura y Requerimientos Técnicos

Se adoptará un diseño modular multiplataforma (Linux, Windows, macOS) con capacidad para soportar al menos 200 usuarios concurrentes. Se evaluarán dos modelos de alta disponibilidad (activo-activo o activo-pasivo), incluyendo redundancia en servidores, almacenamiento SAN/NAS y *backups* en cintas LTO. La seguridad será transversal, con:

- Cifrado de datos sensibles.
- Control de accesos por roles (RBAC).
- Auditoría detallada de operaciones críticas.

Entregables y Exclusiones

Se proporcionará documentación técnica completa, scripts SQL para despliegue y pruebas, una API funcional para demostración, y un análisis costo-beneficio de las arquitecturas propuestas. Quedan fuera del alcance:

- Interfaces de usuario final (solo se entregarán APIs para integración).
- Adquisición física de hardware (aunque se definirán especificaciones técnicas).

Este enfoque garantiza una solución robusta, adaptable y alineada con las necesidades operativas y normativas de la industria hotelera.

Roles del equipo:

Básicamente trabajaré de forma individual en este proyecto y por ende se dejarán de lado ciertos aspectos como lo son los roles de equipo.

Requerimientos:

Tabla de terminología de la matriz de requerimientos:

Columna	Descripción
Categoría	La agrupación general del requerimiento (ej. Seguridad, API, Disponibilidad).
Requerimiento	Nombre breve y claro del requerimiento.
Tipo	Clasificación del requerimiento: F Funcional / NF No funcional / T Técnica / N Negocio.
Descripción Detallada	Explicación específica del requerimiento.
Prioridad	Nivel de importancia: Crítica / Alta / Media / Baja.
Justificación	Razón por la cual el requerimiento es necesario.
Fuente	Origen del requerimiento: normativas, usuarios, equipo interno, etc.
Criterio de Aceptación	Condición para considerar que el requerimiento está cumplido.

Tabla de requerimientos de sistema

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
Accessibilidad	Cumplir WCAG 2.1 AA	NF	El sistema debe de establecer un contraste adecuado, lectores de pantalla, etc. De forma predeterminada	Baja	Accesibilidad para huéspedes discapacitados.	Testado con herramientas como axe.
API	Endpoints RESTful para CRUD de reservas	F	El sistema debe permitir crear, consultar, modificar y cancelar reservaciones vía API RESTful.	Alta	Facilita integración con canales de venta.	API documentada con Swagger y 100% cobertura Postman.
API	Soportar versionado de API (v1, v2)	T	El sistema debe de mantener compatibilidad con versiones anteriores del mismo durante 6 meses.	Media	Permite actualizaciones sin romper integraciones.	Requests a /v1/reservations funcionan post-implementación.
API	Limitar rate limiting	T	En el sistema se debe implementar throttling por API key.	Media	Previene DDoS y abuse.	Requests sobre límite reciben HTTP 429.
Auditoría	Registrar cambios en reservas	F	El sistema debe de guardar información sobre loggear usuario, timestamp y cambios en tabla de auditoría.	Media	Cumple con SOX para trazabilidad.	Bitácora contiene 100% de modificaciones.
Auditoría	Registro de accesos, consultas y modificaciones	F	El sistema debe registrar toda actividad sobre datos sensibles en la base de datos.	Crítica	Permite rastrear incidentes y auditoría.	Log de auditoría activo con revisiones periódicas
Backup	Ejecutar respaldos incrementales cada 4h	T	El sistema de forma automática debe generar backups diferenciales entre respaldos completos diarios.	Alta	Cumple RPO de 5 minutos.	Logs muestran backups exitosos cada 4h ±2 min.
Base de Datos	Implementar replicación sincrónica	T	Configurar réplica en caliente con retraso máximo de 5 segundos.	Alta	Cumple RPO de 5 minutos en fallos.	Test de failover valida sincronización.
Base de Datos	Normalizar modelo a 3FN	F	El sistema debe de implementar una forma solvente de normalización de datos.	Alta	Garantiza integridad referencial.	Diseño ER correcto.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
BI	Exportar datos a Power BI	T	El sistema debe de mostrar un feed diario a data warehouse para análisis.	Baja	Habilita reporting avanzado.	ETL ejecutado diariamente sin errores.
Cache	Implementar Redis para consultas	T	El sistema debe cachear resultados de búsqueda de habitaciones por 5 min.	Media	Reduce carga en BD.	Hit rate >80% en producción.
Clientes	Registrar preferencias históricas	F	El sistema debe almacenar tipo de habitación favorita, alergias y solicitudes especiales.	Media	Personaliza servicio.	Dashboard muestra datos históricos por cliente.
Clientes	Integrar con programa de fidelización	F	El sistema debe de sincronizar puntos y beneficios con sistema de membresía existente.	Alta	Incentiva recompensas.	API consume/actualiza puntos correctamente.
Cloud	Desplegar en multi-AZ	T	El sistema puede distribuir carga en 2 zonas de disponibilidad.	Alta	Aumenta resiliencia ante fallos.	Simulacro de caída de AZ no afecta servicio.
Compliance	Auditoría trimestral de seguridad	NF	Debe de existir un Pen testing interno + informe de vulnerabilidades.	Media	Cumple con ISO 27001.	Reporte con findings corregidos en <7 días.
Conurrencia	Bloquear habitaciones durante checkout	F	El sistema tiene que implementar SELECT FOR UPDATE en transacciones.	Crítica	Evita doble asignación.	Test de estrés con 200 usuarios simultáneos.
Costos	Consultoría externa en seguridad	N	El sistema debe de establecer presupuesto para expertos externos si no se cuenta con personal calificado interno.	Alta	Asegura diseño adecuado de políticas de seguridad.	Contrato de consultoría aprobado e incluido en plan de trabajo
CRM	Marcar clientes frecuentes	F	Etiquetar usuarios con +5 estancias en últimos 12 meses.	Baja	Facilita ofertas personalizadas .	Segmentación visible en dashboard.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
DevOps	Implementar CI/CD	T	Se debe de contar con Pipeline automatizado con tests unitarios/integración.	Media	Agiliza despliegues seguros.	Deploys en <15 min con rollback automático.
Disponibilidad	Garantizar 99.98% uptime anual	NF	El sistema debe de ser escalable al uso de clustering activo-activo con balanceo de carga.	Crítica	Asegura continuidad operativa en horario comercial.	Monitoreo muestra <1.75h de downtime/año.
Documentación	Generar Swagger/Open API	T	Sí debe de documentar endpoints con ejemplos request/response.	Media	Facilita integración por terceros.	95% de endpoints documentados.
Facturación	Exportar datos contables	F	El sistema debe de generar un archivo SAF-T automático para contabilidad mensual.	Baja	Obligatorio en jurisdicciones europeas.	Archivo pasa validación oficial.
Habitaciones	Gestionar estados (limpieza/mantenimiento)	F	El sistema deba actualizar estados mediante interfaz housekeeping.	Alta	Optimiza asignación de habitaciones.	Reporte diario de estados coincide con físico.
Habitaciones	Aplicar tarifas dinámicas	F	El sistema debe de calcular precios según temporada, ocupación y demanda. Probablemente signifique implementar algunos trigger.	Media	Maximiza ingresos.	Reglas de pricing validadas con históricos.
Hardware	Módulos HSM opcionales	T	El sistema de ofrecer la posibilidad de uso de Hardware Security Modules para gestión de claves.	Baja	Mejora seguridad en infraestructuras críticas.	Evaluación de viabilidad completada

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
I18n	Soportar múltiples monedas	F	Se deben mostrar precios en USD, EUR, GBP según ubicación.	Media	Mejora experiencia internacional.	Conversión actualizada diariamente via API.
Infraestructura	Soportar 10K reservas/mes por hotel	T	Se debe escalar horizontalmente con auto-balancing.	Alta	Prepara para crecimiento.	Load testing con JMeter valida capacidad.
Integración	Conectar con CRM corporativo	T	Debe de implementarse el API REST para sincronizar datos de clientes con Salesforce.	Media	Unifica visión del cliente.	Test end-to-end valida flujo bidireccional.
Licencias	Licencias de software de cifrado	T	Se debe contar con licenciamiento de herramientas adicionales si no están incluidas en el SGBD.	Media	Garantiza cumplimiento sin depender del motor de BD.	Licencias activas y legalmente válidas
Licencias	Licencias de herramientas SIEM y auditoría	N	Se debe contar con licenciamiento adecuado para herramientas de monitoreo y auditoría.	Alta	Evita sanciones por uso no autorizado y garantiza soporte.	Herramientas licenciadas correctamente con soporte vigente
Localización	Soportar multi-idioma (EN/ES/FR)	NF	Se puede traducir interfaz y correos automáticos.	Baja	Necesidad de hoteles internacionales.	UI muestra idioma según preferencia usuario.
Logística	Alertar housekeeping al check-out	F	Notificar a sistema de limpieza al liberar habitación.	Media	Reduce tiempo entre ocupaciones.	Notificaciones push recibidas en tablets.
Logs	Centralizar registros en ELK	T	El sistema debe enviar logs a stack Elasticsearch para análisis.	Baja	Facilita troubleshooting.	Logs disponibles por 90 días.
Mobile	Notificar confirmación vía SMS	F	El sistema puede enviar SMS con código de reserva al completar booking.	Media	Reduce no-shows.	95% de SMS entregados en <2 min.
Monitoreo	Alertar SLA breaches	NF	El sistema puede notificar en Slack/Email cuando uptime <99.9%.	Alta	Permite acción correctiva inmediata.	Alertas probadas en simulacros.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
Monitoreo	Integración con SIEM	T	Debe integrarse con herramientas SIEM para alertas y detección de amenazas.	Alta	Mejora la detección temprana de incidentes.	Alertas de prueba generadas y recibidas correctamente
Onboarding	Migrar datos históricos	F	Se debe poder transferir 3 años de reservas activas desde legacy system.	Alta	Garantiza continuidad operativa.	99.9% de registros migrados sin errores.
Onboarding	Validar KYC de clientes	F	Se debe de poder escanear documento de identidad al registrar nuevo cliente.	Alta	Cumple regulaciones antifraude.	OCR detecta 99% de documentos válidos.
Pagos	Procesar transacciones PCI-compliant	F	El sistema debe integrar pasarelas de pago tokenizadas sin almacenar PAN.	Crítica	Cumple estándares de seguridad.	Certificación QSA obtenida.
Pagos	Generar facturas electrónicas	F	El sistema debe emitir CFDI 4.0 con folios autorizados por SAT.	Alta	Requerimiento legal en Costa Rica.	Facturas pasan validación del PAC.
Reportes	Generar ocupación por segmento	F	El sistema debe de poder exportar CSV con ocupación por tipo de cliente (corporativo/turista).	Media	Optimiza estrategias comerciales.	Reporte generado diariamente a las 6:00 AM.
Reservaciones	Validar disponibilidad en tiempo real	F	El sistema debe verificar inventario antes de confirmar reservas.	Crítica	Evita overbooking.	Test simultáneos muestran consistencia.
Reservaciones	Soportar reservas multi-hotel	F	Se deben permitir reservas consecutivas en distintos hoteles de la cadena.	Alta	Mejora experiencia cliente.	Flujo completo documentado en diagramas UML.
Seguridad	Implementar autenticación multifactor (MFA)	NF	El sistema debe requerir MFA para accesos administrativos y operaciones críticas.	Crítica	Mitiga riesgos de acceso no autorizado.	MFA activado para todos los usuarios con permisos elevados.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
Seguridad	Cifrar datos sensibles en tránsito y reposo	T	El sistema debe de emplear AES-256 para datos en reposo y TLS 1.3+ en tránsito.	Crítica	Cumple con GDPR y protege información de clientes.	Auditoría confirma cifrado activo en todas las capas.
Seguridad	Roles de usuario con mínimo privilegio	F	El sistema debe permitir definir y asignar roles con el menor acceso necesario.	Crítica	Minimiza el riesgo de accesos no autorizados.	Accesos están limitados por rol y verificados en pruebas de seguridad
Seguridad	Políticas de contraseñas y MFA	NF	El sistema debe de validar que las contraseñas sean fuertes, con expiración periódica y soporte para autenticación multifactor.	Alta	Fortalece el control de acceso y reduce riesgos de intrusión.	Contraseñas cumplen política y MFA está activa y funcional
Seguridad	Segmentación de red y firewall	T	Se debe de poder tener acceso interno segmentado, uso de DMZ para servicios expuestos, firewall activo.	Crítica	Limita superficie de ataque y controla flujos de red.	Segmentación implementada y validada en pruebas de red
Seguridad	Cifrado en reposo	T	El sistema debe de establecer un cifrado AES-256 de base de datos, respaldos y logs.	Crítica	Protege datos sensibles ante acceso físico o pérdida.	Auditoría muestra cifrado activo en almacenamiento
Seguridad	Cifrado en tránsito	T	El sistema debe de hacer uso de SSL/TLS en todas las comunicaciones con la base de datos.	Crítica	Previene ataques de interceptación de datos.	Certificados válidos y cifrado activo verificado en conexiones
Servicios	Vender paquetes (hospedaje+experiencias)	F	El sistema debe de agrupar habitación, spa y tours en un solo SKU.	Media	Aumenta venta cruzada.	Checkout muestra paquetes con descuento.
Soporte	Ofrecer helpdesk 24/7	NF	El sistema debe de ofrecer soporte telefónico y chat en 3 idiomas.	Media	Mejora experiencia cliente.	Tickets resueltos en <15 min (urgentes).
Training	Capacitar personal en sitio	F	Se debe de ofrecer la posibilidad de 8 horas de training para recepcionistas y administradores.	Alta	Asegura adopción efectiva.	Encuesta post-training con ≥4/5 satisfacción.
UI	Responsive para móviles	F	La interfaz debe de ser adaptable a smartphones (iOS/Android).	Alta	30% de reservas provienen de móviles.	Pruebas en BrowserStack sin defects.

Propuesta de selección de tecnologías

A) Diagrama de arquitectura

En esta fase, se utiliza el modelo Ansi/Sparc de 3 capas.

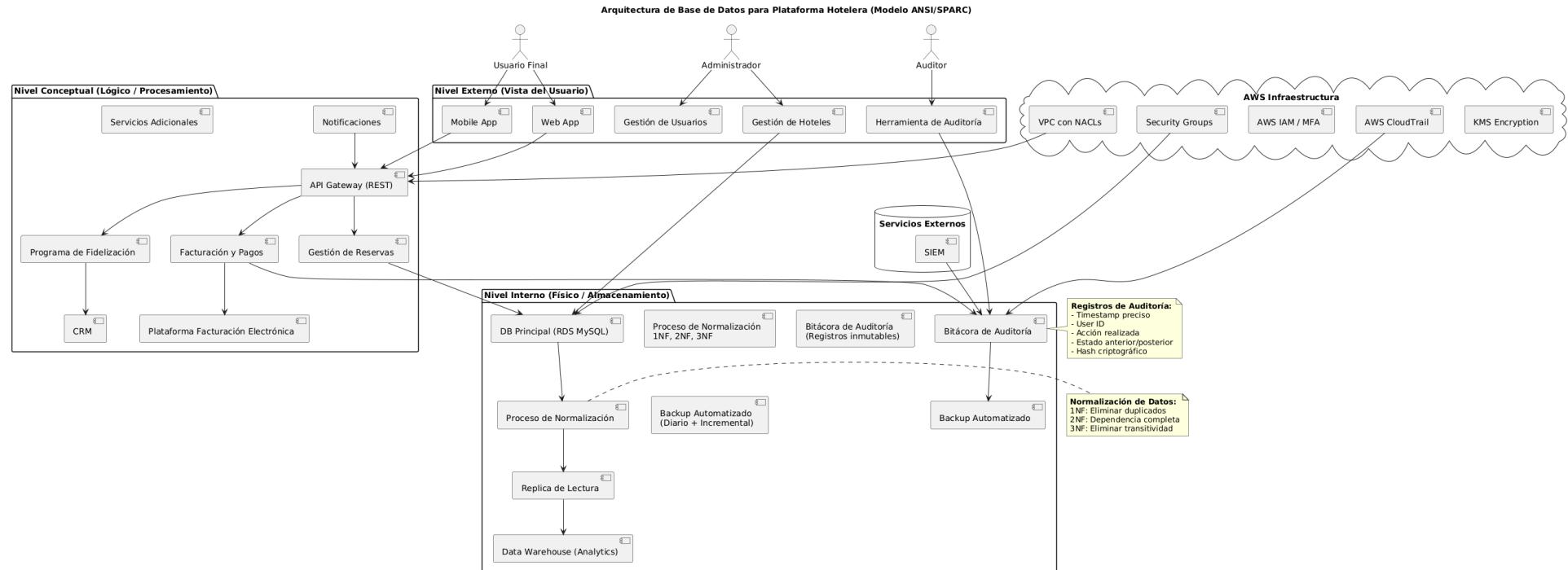


Figura de autoría propia, creada con la herramienta plantUML

B) Arquitecturas propuestas

Para la propuesta de estas arquitecturas, en aws me basaré en experiencia propia sumado a documentación oficial de *Amazon web service*.

- AWS RDS Multi-AZ

Características:

1. Replicación síncrona entre Availability Zones (AZs).
2. Failover automático (30-60 segundos) sin cambios en la conexión de la app.
3. Sin gestión de servidores: AWS maneja parches, backups y escalado.
4. Ideal para: Aplicaciones empresariales que requieren HA sin complejidad.

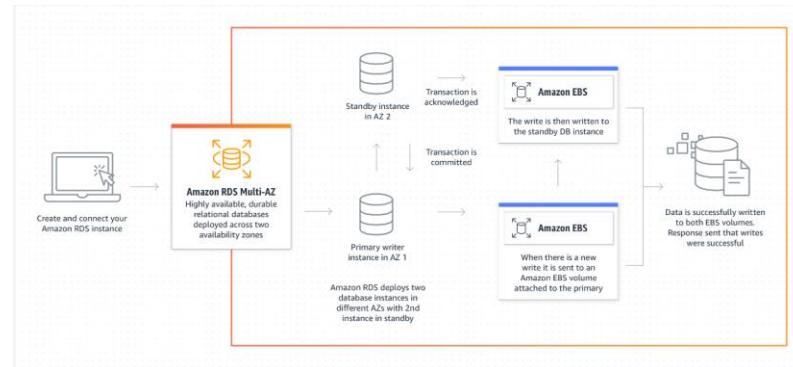
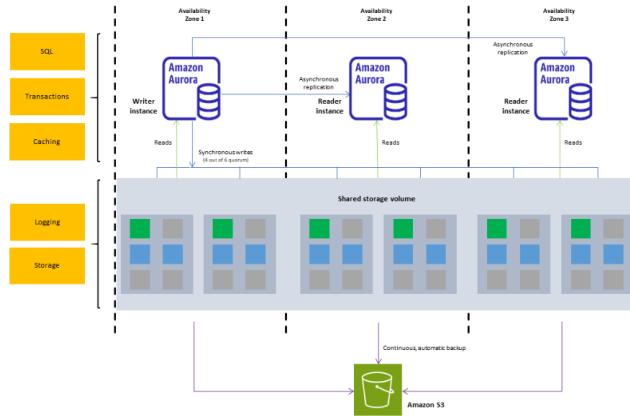


Imagen obtenida de (AWS, 2025) documentación oficial

- AWS Aurora

Características:

1. Hasta 15 réplicas de lectura con consistencia en milisegundos.
2. Almacenamiento autoescalable hasta 128 TB.
3. Failover más rápido que RDS estándar.
4. Ideal para: Cargas de trabajo intensivas en lecturas.



Documentación oficial (AWS, 2025)

C) Estimación de costes de implementación

Para la primera propuesta en AWS, RDS multi A-Z, la estimación de costes se basará en las siguientes suposiciones, se tomará la región de OHIO como base, se tomará en cuenta que la base de datos será de normal-alto consumo y que se partirá de una instancia en large dado a que el proyecto es medio-grande, añadido a lo anterior, el coste mensual es de 372 dólares por mes.

The screenshot shows the AWS RDS Pricing calculator interface. At the top, there are three tabs: 'Despliegue single-AZ' (selected), 'Despliegue multi-AZ (una espera)', and 'Despliegue multi-AZ (dos esperas legibles)'. Below the tabs, a note says: 'Para obtener más información sobre multi-AZ, consulte la sección de [opciones de despliegue de alta disponibilidad](#)'. A dropdown menu for 'Región:' is set to 'Este de EE. UU. (Ohio)'. The main table lists 'Instancias estándar – Generación actual' with their corresponding 'Precio por hora':

Instancia	Precio por hora
db.t4g.micro	0,032 USD
db.t4g.small	0,065 USD
db.t4g.medium	0,129 USD
db.t4g.large	0,258 USD
db.t4g.xlarge	0,517 USD

Imagen obtenida de la documentación oficial (AWS, 2025)

Para la segunda propuesta, me basaré en AWS Aurora, utilizaré la calculadora de costes que ofrece AWS de la siguiente forma:

The screenshot shows the 'Precios de Instancias reservadas de Amazon Aurora' calculator. It has two dropdown menus: 'Tipo de ubicación' (set to 'Región de AWS') and 'Región' (set to 'Este de EE. UU. (Ohio)'). Below these are two dropdown menus: 'Duración del término' (set to '1 año') and 'Opciones de pago' (set to 'Sin gastos iniciales').

Partiendo de que se buscan los costes de un negocio promedio, el coste mensual es de 352 dólares. Se utilizó el db.x2g.xlarge.

The screenshot shows a table comparing the cost of reserved instances for various Aurora instance types. The columns are: Nombre de la instancia, Cuota inicial de IR, Cuotas mensuales de IR*, Cuota por hora efectiva de IR**, comparación con modalidad bajo demanda, and Tarifa bajo demanda.

Nombre de la instancia	Cuota inicial de IR	Cuotas mensuales de IR*	Cuota por hora efectiva de IR**	comparación con modalidad bajo demanda	Tarifa bajo demanda
db.t4g.medium	0 USD	42,12 USD	0,058 USD	21 %	0,0730 USD
db.t4g.large	0 USD	84,17 USD	0,115 USD	21 %	0,1460 USD
db.t3.small	0 USD	24,82 USD	0,034 USD	17 %	0,0410 USD
db.t3.medium	0 USD	47,45 USD	0,065 USD	21 %	0,0820 USD
db.t3.large	0 USD	99,28 USD	0,136 USD	17 %	0,1640 USD
db.x2g.large	0 USD	178,78 USD	0,245 USD	35 %	0,3770 USD
db.x2g.xlarge	0 USD	357,55 USD	0,490 USD	35 %	0,7540 USD

Imagen obtenida de la documentación oficial (AWS, 2025)

Comparativa de arquitecturas

1. Propuesta 1: AWS RDS (PostgreSQL/MySQL) - Instancia db.xlarge (372 USD/mes)

Arquitectura Propuesta:

- Capa de Presentación: Aplicación web/móvil hospedada en AWS Elastic Beanstalk o EC2 Auto Scaling.
- Capa de Lógica de Negocio: Microservicios en AWS Lambda o ECS (Docker) con API Gateway.
- Capa de Base de Datos: AWS RDS (PostgreSQL/MySQL) en configuración Multi-AZ (alta disponibilidad).
- Redes: Conexiones seguras mediante AWS Site-to-Site VPN o AWS Direct Connect.
- Almacenamiento: EBS gp3 con snapshots automatizados.
- Backup: AWS Backup + Amazon S3 Glacier para retención a largo plazo.
- Monitoreo: Amazon CloudWatch + AWS Config para auditoría.

Fortalezas:

Alta Disponibilidad Automatizada:

- RDS Multi-AZ garantiza failover automático (< 2 min de downtime).
- Réplica sincrónica en otra AZ.

Escalabilidad Horizontal:

- Lecturas escalables con réplicas de lectura (hasta 5).
- Escalado vertical sin downtime (cambio de instancia).

Seguridad Integrada:

- Cifrado en tránsito (SSL/TLS) y en reposo (KMS).
- IAM para gestión de accesos.

Sopporte Multiplataforma:

- Compatibilidad con aplicaciones en Linux, Windows y macOS.
- Cumplimiento Normativo:
- Certificaciones SOC, ISO, PCI DSS, GDPR.

Debilidades:

Costo Operativo Elevado:

- El precio aumenta con réplicas y almacenamiento adicional.

Limitaciones en Customización:

- No se puede modificar el kernel de la base de datos.

Dependencia de AWS:

- Lock-in tecnológico.

2. Propuesta 2: Amazon Aurora PostgreSQL/Mysql - Instancia db.x2g.xlarge (352 USD/mes)

Arquitectura Propuesta:

- Capa de Presentación: AWS AppSync (GraphQL) + Amplify para frontend.
- Capa de Lógica de Negocio: AWS Fargate (ECS sin servidor) con balanceo de carga.
- Capa de Base de Datos: Aurora PostgreSQL con almacenamiento distribuido (6 copias en 3 AZs).
- Redes: AWS PrivateLink para conexiones internas seguras.
- Almacenamiento: Aurora Storage Auto-Scaling (hasta 128 TB).
- Backup: Puntos de recuperación continuos (Continuous Backup) + Cross-Region Replication.
- Monitoreo: Amazon RDS Performance Insights + AWS GuardDuty.

Fortalezas:

- Alta Disponibilidad Superior, Aurora replica datos en 3 AZs ($RPO \approx 0$, $RTD < 1$ min). Posee un clúster Aurora con hasta 15 réplicas de lectura.
- Rendimiento Optimizado, perfectamente hasta 5x mejor rendimiento que RDS PostgreSQL. Con un escalado automático de almacenamiento.
- Respaldo Continuo: Puntos de recuperación con granularidad de segundos.
- Reducción de Costos: Menor costo por IOPS (no se paga por operaciones de disco).
- Compatibilidad con PostgreSQL/MySQL: Migración sencilla desde bases existentes.

Debilidades:

- Costo Inicial de Migración, esto parte de que se requieren ajustes en *queries* para aprovechar Aurora.
- Limitaciones en Customización, pueden haber problemas de escalabilidad dado a que no soporta todas las extensiones de PostgreSQL.
- Dependencia de AWS, menos portabilidad que soluciones on-premise. Aunque una total cohesión en el proyecto.

A continuación, se muestra una tabla comparando el desempeño a alto nivel ofrecido por las tecnologías elegidas, he aquí una tabla de terminología:

Tabla de terminología de la tabla comparativa de desempeño de arquitecturas	
Columnas	Justificación
Requisitos/Requerimientos	Define el requerimiento o la información a lo que apela la fila.
AWS RDS	Define como el requisito se aplica en base a la tecnología de AWS RDS.
Amazon Aurora	Define como el requisito se aplica en base a la tecnología de Amazon Aurora

Tabla comparativa de desempeño de arquitectura		
Resquisitos	AWS RDS	Amazon Aurora
Alta Disponibilidad (RTO ≤ 30 min, RPO ≤ 5 min)	(Multi-AZ)	(Clúster Multi-AZ)
Soporte 200+ usuarios concurrentes	(Depende de instancia)	(Mejor rendimiento)
Escalado Horizontal	(Réplicas de lectura)	(Hasta 15 réplicas)
Redundancia de Hardware	(Multi-AZ)	(6 copias en 3 AZs)
Respaldo Automatizado	(Snapshots)	(Backup continuo)
Soporte Multiplataforma		
Conexiones Seguras (VPN/VPC)		
Recuperación ante borrado accidental	(PITR)	(PITR con segundos)

Estimación de costos totales de proyectos

Para realizar una correcta estimación de los costes mensuales de tecnología de base de datos en el primer año en las diferentes arquitecturas, se plantea una tabla comparativa de las arquitecturas, he aquí de la tabla de terminología de estas.

Tabla de terminología de la tabla de comparativa de costes por arquitectura utilizada	
Termino	Explicación
Componente	Define los servicios que poseen un coste asociado en el sistema.
AWS RDS (PostgreSQL/MySQL)	Detalla el coste mensual del servicio especificado al mes en AWS RDS.
Amazon Aurora (Postgresql/MySQL)	Detalla el coste mensual del servicio especificado al mes en Amazon Aurora.

Tabla comparativa de costes por arquitectura utilizada			
Componente	AWS RDS (PostgreSQL/MySQL)	Amazon Aurora PostgreSQL	Notas
Costo Base (Instancia Principal)	\$372/mes (db.xlarge)	\$352/mes (db.x2g.xlarge)	Precio por nodo primario (Multi-AZ incluido).
Réplicas de Lectura	\$372/mes cada una (hasta 5)	\$352/mes cada una (hasta 15)	Aurora permite más réplicas sin penalización de IOPS.
Almacenamiento (GB/mes)	\$0.115/GB (gp3)	\$0.10/GB (Aurora Auto-Scaling)	Aurora optimiza almacenamiento sin costos adicionales por IOPS.
Backup Automatizado	\$0.095/GB-mes (Snapshots)	\$0.021/GB-mes (Backup Continuo)	Aurora incluye backups granulares sin costo adicional.
Conexión VPN (Site-to-Site)	0.05/hora+0.05/hora+0.02/GB	0.05/hora+0.05/hora+0.02/GB	Igual en ambas opciones.
Monitoreo (CloudWatch)	\$30/mes (métricas básicas)	\$30/mes (métricas básicas)	Costo similar.
Licencias de HA	Incluido en RDS	Incluido en Aurora	No requiere licencias adicionales.
Costo Estimado (1 Año)	6,500–6,500–10,000	6,000–6,000–9,00	Depende de réplicas y almacenamiento.

Por otro lado, también se detallan los diferentes costes adicionales no tan importantes pero que si pueden dar un beneficio relativamente importante al proyecto. Se detalla el nombre del servicio, el coste, a que arquitectura se aplica y comentarios breves sobre la importancia en el proyecto.

Tabla de posibles costes adicionales			
Servicio Adicional	Costo (USD/mes)	Aplicable a	Notas
AWS Direct Connect	300–300–1,500+	Ambas	Conexión dedicada (reduce latencia).
AWS Backup (Políticas Avanzadas)	5–5–50	Ambas	Gestión centralizada.
Amazon GuardDuty (Seguridad)	10–10–500	Ambas	Detección de amenazas.
Personal Especializado (DBA/Arquitecto)	3,000–3,000–8,000	Ambas	Salarios externos (si se contrata).

Costos anuales esperados en el proyecto

En esta sección se establecerá una cantidad de costes de la infraestructura utilizada y los costes de personal del proyecto a lo largo del primer año de desarrollo del proyecto. A continuación, se detallan los costes asociados al personal del proyecto, se detallará en una tabla que se basará en la siguiente tabla de terminología.

Término	Explicación
Rol	Rango del personal que participará en la elaboración del proyecto.
Salario Mensual	Se calcula a partir de los salarios promedios de profesionales TI.
Salario Anual	Se multiplica el mensual por 12.
Notas	Importancia en el proyecto, se omite el detallado completo de responsabilidades del personal.

Costo personal primer año			
Rol	Salario Mensual (USD)	Salario Anual (USD)	Notas
Arquitecto de BD	7,000–9,000	84,000–108,000	Diseño HA, replicación, optimización.
Administrador de BD (DBA)	5,000–7,000	60,000–84,000	Mantenimiento, tuning, respaldos.
Ingeniero de Infraestructura	6,500–8,500	78,000–102,000	AWS, redes, redundancia.
Especialista en Backup	4,500–6,000	54,000–72,000	Veeam, cintas LTO, recuperación.
Especialista en Seguridad	6,000–8,000	72,000–96,000	CISSP, cifrado, IAM.
Coordinador de Proyecto	5,500–7,500	66,000–90,000	PMP, Scrum, gestión de equipo.
Auditor de TI (Opcional)	6,000–8,000	72,000–96,000	GDPR, PCI DSS, normativas.
Total Personal (Anual)	40,500-54,000	486,000–648,000	

En la siguiente parte se detallarán específicamente los costes relacionados con la infraestructura tecnológica que se utilizará en el presente proyecto durante un año, a continuación la tabla de terminología.

Tabla de terminología de la tabla de costo Anual de infraestructura tecnológica planteada	
Termino	Explicación
Componente	Define los servicios que poseen un coste asociado en el sistema.
AWS RDS (PostgreSQL/MySQL)	Detalla el coste anual del servicio especificado al mes en AWS RDS.
Amazon Aurora (Postgresql/MySQL)	Detalla el coste anual del servicio especificado al mes en Amazon Aurora.
Notas	Se da información relacionada a cada componente.

Costo Anual de infraestructura tecnológica planteada			
Componente	AWS RDS (PostgreSQL/MySQL)	Amazon Aurora PostgreSQL	Notas
Instancia Principal + Multi-AZ	\$4,464/año (db.xlarge)	\$4,224/año (db.x2g.xlarge)	Alta disponibilidad incluida.
Réplicas de Lectura (2)	\$8,928/año (2 x db.xlarge)	\$8,448/año (2 x db.x2g.xlarge)	Escalabilidad para lecturas.
Almacenamiento (500 GB)	\$690/año (gp3)	\$600/año (Aurora)	Aurora optimiza costos.
Backup Automatizado	\$114/año (100 GB)	Incluido	Aurora tiene backup continuo sin costo extra.
VPN (Site-to-Site)	\$438/año	\$438/año	Conexión segura entre sedes.
Monitoreo (CloudWatch)	\$360/año	\$360/año	Alertas y métricas.
Total, Infraestructura (Anual)	\$14,994/año	\$14,070/año	Aurora es \$924 más económico.

A continuación, se detallan algunos costes adicionales a la infraestructura tecnológica y que se podría incurrir en ellos si es del gusto de los gestores de costo, de detallan nombre, precio e importancia de cada uno.

Costes adicionales de infraestructura tecnológica		
Concepto	Costo (USD)	Notas
Unidad de Cinta LTO-8	3,000–5,000	Drive de respaldo físico.
Cintas LTO-8 (10 unidades)	1,000–2,000	Rotación y almacenamiento offsite.
Software de Backup (Veeam)	\$1,500/año	Licencia empresarial.
Servidores Locales (Opcional)	10,000–20,000	Si se requiere hibridación.
UPS/Generador (Opcional)	5,000–15,000	Para infraestructura on-premise.
Total Adicional (Inicial)	10,500–42,000	Depende de necesidades.

Costos totales en el primer año

A continuación, se detallan los costes anuales del proyecto, en dependencia de la tecnología utilizada.

Costos totales del primer año		
Arquitectura	AWS RDS	Amazon Aurora
Personal (Promedio)	\$567,000	\$567,000
Infraestructura (AWS)	\$14,994	\$14,070
Hardware/Software Extra	\$20,000*	\$20,000*
Total Estimado (1er Año)	\$601,994	\$601,070

Análisis de costos CAPEX/OPEX

A continuación, se presenta un Análisis de Costos Realista a 5 Años (CAPEX/OPEX) para el proyecto, considerando las dos arquitecturas propuestas: AWS RDS y Amazon Aurora. Este análisis incluye los costos operativos (OPEX) y los de capital (CAPEX), mostrando una proyección para cinco años con crecimiento estimado, tomando como base el primer año ya calculado.

Amazon RDS

Concepto	Año 1	Año 2	Año 3	Año 4	Año 5	Total 5 años
Personal	\$567,000	\$567,000	\$567,000	\$567,000	\$567,000	\$2,835,000
Infraestructura AWS	\$14,070	\$16,192	\$18,634	\$21,429	\$24,643	\$94,968
Hardware/Software adicional	\$20,000	\$0	\$0	\$0	\$0	\$20,000
Total anual	\$601,070	\$583,192	\$585,634	\$588,429	\$591,643	\$2,949,968

Amazon Aurora

Concepto	Año 1	Año 2	Año 3	Año 4	Año 5	Total 5 años
Personal	\$567,000	\$567,000	\$567,000	\$567,000	\$567,000	\$2,835,000
Infraestructura AWS	\$14,994	\$17,243	\$19,830	\$22,804	\$26,224	\$101,095
Hardware/Software adicional	\$20,000	\$0	\$0	\$0	\$0	\$20,000
Total anual	\$601,994	\$584,243	\$586,830	\$589,804	\$593,224	\$2,956,095

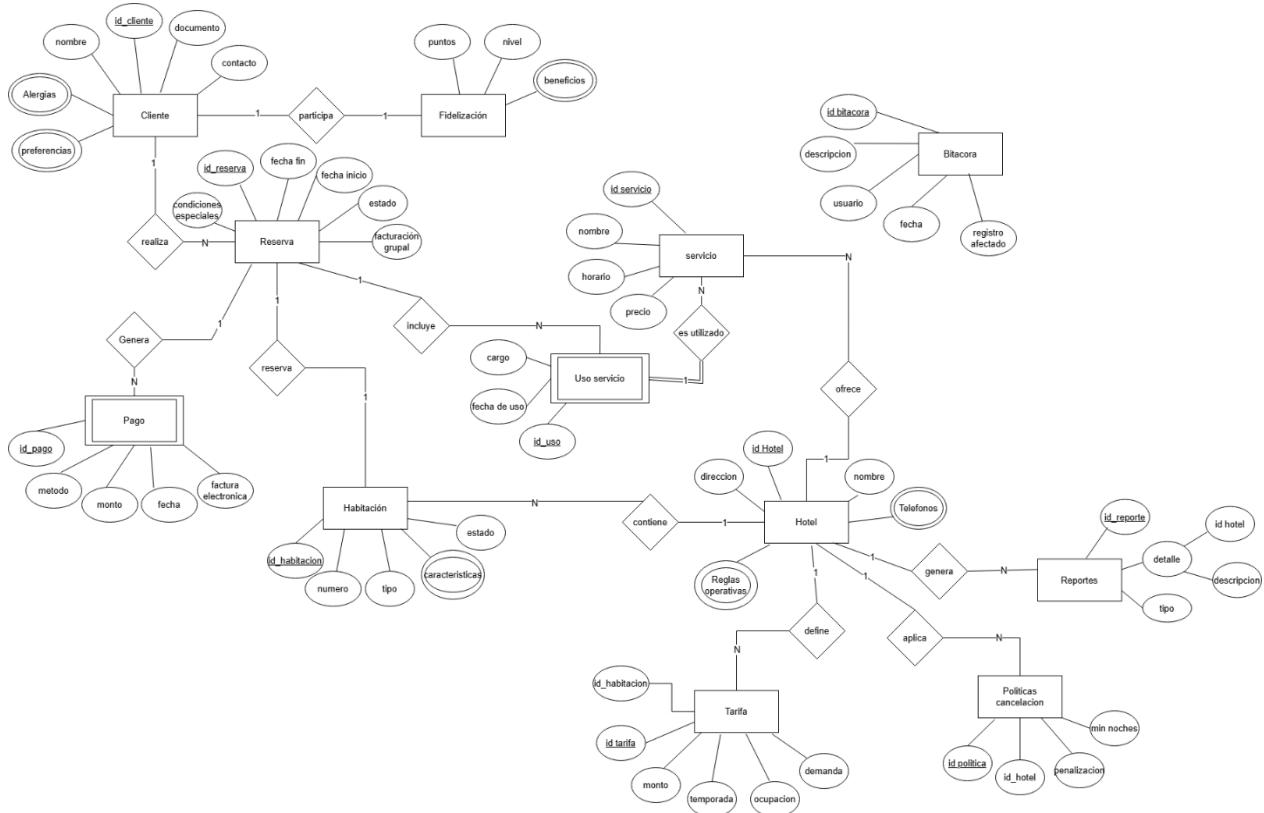
Resumen de ambas

Categoría	AWS RDS	Amazon Aurora
CAPEX	\$20,000	\$20,000
Total		
OPEX Total	\$2,936,095	\$2,929,968
Total 5 años	\$2,956,095	\$2,949,968

Podríamos concluir en la estimación de costos que

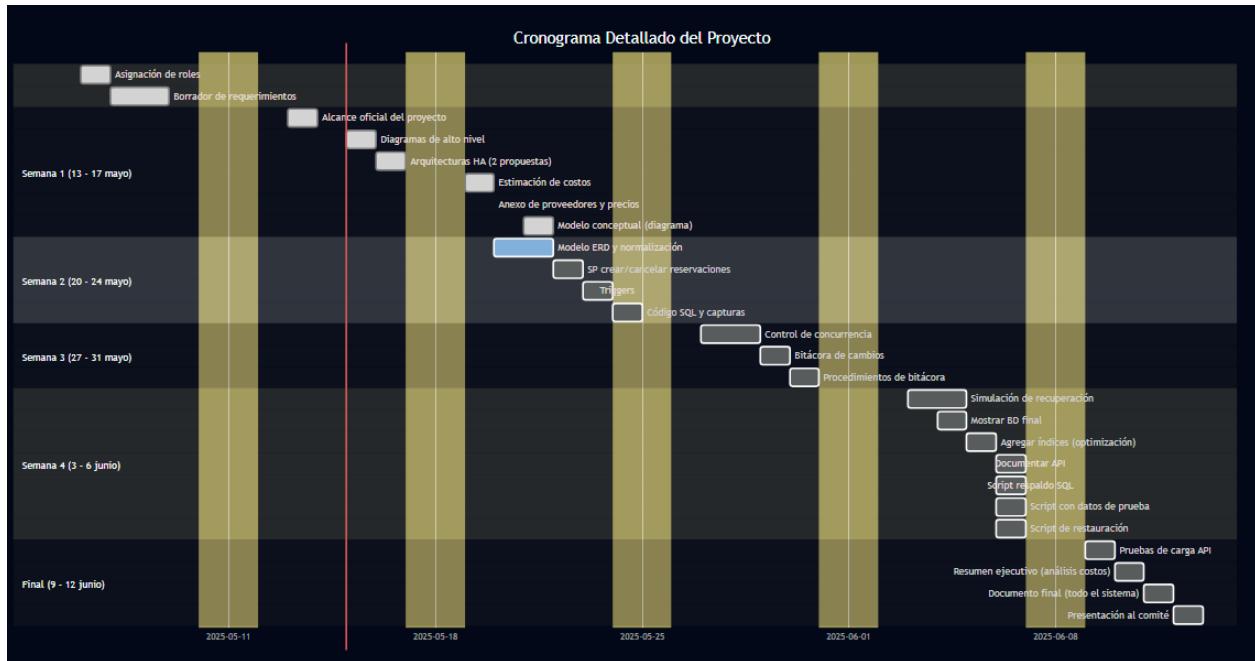
- Amazon Aurora PostgreSQL es ligeramente más económico a largo plazo ($\approx \$6,000$ en ahorro en 5 años), gracias a su eficiencia en almacenamiento y backups.
- Ambas arquitecturas escalan bien y cumplen con los requerimientos técnicos del proyecto.
- La mayor inversión está en el personal especializado (95% del costo total), por lo tanto, optimizaciones en infraestructura no impactan drásticamente en el total.
- Aurora ofrece mejor rendimiento y escalabilidad horizontal, lo cual puede justificar su elección incluso sin el ahorro directo.

Diagrama ER



Cronograma con diagrama de Gantt

He de aclarar que al yo realizar el proyecto de manera individual, pues todo lo efectuaré yo.



Implementación de la base de datos

Capturas de la creación de las tablas

Tabla de clientes

```

proyecto_2.sql bases_datos_p2.sql X
D: > Personal > tec > universidad > 2025_Semestre_1 > Base de datos > Proyectos > proyecto 2 > bases_datos_p2.sql

1 -- Tabla de Hoteles
2 CREATE TABLE hoteles (
3     hotel_id SERIAL PRIMARY KEY,
4     nombre VARCHAR(100) NOT NULL,
5     direccion TEXT NOT NULL,
6     ciudad VARCHAR(50) NOT NULL,
7     pais VARCHAR(50) NOT NULL,
8     telefono VARCHAR(20) NOT NULL,
9     email VARCHAR(100) NOT NULL,
10    estrellas INTEGER CHECK (estrellas BETWEEN 1 AND 5),
11    activo BOOLEAN DEFAULT TRUE,
12    fecha_apertura DATE,
13    descripcion TEXT
14 );
15

```

Tabla de clientes

```
17 CREATE TABLE clientes (
18     cliente_id SERIAL PRIMARY KEY,
19     nombre VARCHAR(100) NOT NULL,
20     documento_identidad VARCHAR(20) NOT NULL,
21     tipo_documento VARCHAR(20) NOT NULL,
22     nacionalidad VARCHAR(50) NOT NULL,
23     telefono VARCHAR(20) NOT NULL,
24     email VARCHAR(100) NOT NULL,
25     fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
26     activo BOOLEAN DEFAULT TRUE,
27     preferencias TEXT,
28     alergias TEXT,
29     CONSTRAINT unique_document UNIQUE (tipo_documento, documento_identidad),
30     CONSTRAINT unique_email UNIQUE (email)
31 );
32
```

Tabla de fidelización de clientes

```
34 CREATE TABLE fidelizacion_clientes (
35     fidelizacion_id SERIAL PRIMARY KEY,
36     cliente_id INTEGER REFERENCES clientes(cliente_id),
37     hotel_id INTEGER REFERENCES hoteles(hotel_id),
38     puntos_acumulados INTEGER DEFAULT 0,
39     nivel_membresia VARCHAR(20),
40     beneficios TEXT,
41     fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
42     CONSTRAINT unique_cliente_hotel UNIQUE (cliente_id, hotel_id)
43 );
44
```

Tabla de tipos de habitación

```
46 CREATE TABLE tipos_habitacion (
47     tipo_id SERIAL PRIMARY KEY,
48     hotel_id INTEGER REFERENCES hoteles(hotel_id),
49     nombre VARCHAR(50) NOT NULL,
50     descripcion TEXT,
51     capacidad INTEGER NOT NULL,
52     tamano INTEGER, -- en m2
53     comodidades TEXT,
54     precio_base DECIMAL(10, 2) NOT NULL,
55     CONSTRAINT unique_tipo_hotel UNIQUE (hotel_id, nombre)
56 );
57
```

Tabla de las habitaciones

```
CREATE TABLE habitaciones (
    habitacion_id SERIAL PRIMARY KEY,
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    numero VARCHAR(10) NOT NULL,
    tipo_id INTEGER REFERENCES tipos_habitacion(tipo_id),
    piso INTEGER NOT NULL,
    caracteristicas_especiales TEXT,
    estado VARCHAR(20) NOT NULL CHECK (estado IN ('disponible', 'ocupada', 'mantenimiento', 'limpieza')),
    notas TEXT,
    CONSTRAINT unique_numero_hotel UNIQUE (hotel_id, numero)
);
```

Tablas de las políticas y tarifas por temporada

```
71 -- Tabla de Políticas de Temporada (por hotel)
72 CREATE TABLE politicas_temporada (
73     politica_id SERIAL PRIMARY KEY,
74     hotel_id INTEGER REFERENCES hoteles(hotel_id),
75     nombre VARCHAR(100) NOT NULL,
76     fecha_inicio DATE NOT NULL,
77     fecha_fin DATE NOT NULL,
78     descripcion TEXT,
79     reglas TEXT,
80     CONSTRAINT fechas_validas CHECK (fecha_fin > fecha_inicio),
81     CONSTRAINT unique_nombre_hotel UNIQUE (hotel_id, nombre)
82 );
83
84 -- Tabla de Tarifas por Temporada
85 CREATE TABLE tarifas_temporada (
86     tarifa_id SERIAL PRIMARY KEY,
87     politica_id INTEGER REFERENCES politicas_temporada(politica_id),
88     tipo_id INTEGER REFERENCES tipos_habitacion(tipo_id),
89     precio DECIMAL(10, 2) NOT NULL,
90     descripcion TEXT,
91     CONSTRAINT unique_politica_tipo UNIQUE (politica_id, tipo_id)
92 );
93
```

Tabla de reservaciones

```
94 -- Tabla de Reservaciones
95 CREATE TABLE reservaciones (
96     reservacion_id SERIAL PRIMARY KEY,
97     hotel_id INTEGER REFERENCES hoteles(hotel_id),
98     cliente_id INTEGER REFERENCES clientes(cliente_id),
99     fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
100    fecha_entrada DATE NOT NULL,
101    fecha_salida DATE NOT NULL,
102    adultos INTEGER NOT NULL,
103    ninos INTEGER DEFAULT 0,
104    estado VARCHAR(20) NOT NULL CHECK (estado IN ('pendiente', 'confirmada', 'cancelada', 'completada')),
105    tipo_reserva VARCHAR(20) NOT NULL CHECK (tipo_reserva IN ('individual', 'grupo', 'corporativa')),
106    solicitudes_especiales TEXT,
107    codigo_reserva VARCHAR(20) UNIQUE,
108    CONSTRAINT fechas_validas CHECK (fecha_salida > fecha_entrada)
109 );
```

Tabla de reservación de habitaciones, tabla de detalle

```

111 -- Tabla de Detalles de Reservación (Habitaciones asignadas)
112 CREATE TABLE reservacion_habitaciones (
113     detalle_id SERIAL PRIMARY KEY,
114     reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
115     habitacion_id INTEGER REFERENCES habitaciones(habitacion_id),
116     tarifa_aplicada DECIMAL(10, 2) NOT NULL,
117     notas TEXT,
118     CONSTRAINT unique_reservacion_habitacion UNIQUE (reservacion_id, habitacion_id)
119 );
120

```

Tablas de servicios contratados

```

-- Tabla de Detalles de Servicios en Reserva
CREATE TABLE reservacion_servicios (
    detalle_servicio_id SERIAL PRIMARY KEY,
    reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
    servicio_id INTEGER, -- Referencia flexible (puede ser a servicios_hotel o servicios_externos)
    tipo_servicio VARCHAR(50) NOT NULL, -- 'hotel' o 'externo'
    descripcion VARCHAR(100) NOT NULL,
    fecha_servicio TIMESTAMP NOT NULL,
    cantidad INTEGER DEFAULT 1,
    precio_unitario DECIMAL(10, 2) NOT NULL,
    notas TEXT,
    estado VARCHAR(20) DEFAULT 'pendiente' CHECK (estado IN ('pendiente', 'completado', 'cancelado'))
);

```

Tabla de servicios brindados de hotel o externos

```

5 -- Tabla de Servicios del Hotel
6 CREATE TABLE servicios_hotel (
7     servicio_id SERIAL PRIMARY KEY,
8     hotel_id INTEGER REFERENCES hoteles(hotel_id),
9     nombre VARCHAR(100) NOT NULL,
0     descripcion TEXT,
1     precio_base DECIMAL(10, 2) NOT NULL,
2     categoria VARCHAR(50) NOT NULL,
3     horario_disponibilidad TEXT,
4     activo BOOLEAN DEFAULT TRUE,
5     CONSTRAINT unique_servicio_hotel UNIQUE (hotel_id, nombre)
6 );
7

```

Tabla de pagos

```

148 -- Tabla de Pagos
149 CREATE TABLE pagos (
150     pago_id SERIAL PRIMARY KEY,
151     reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
152     monto DECIMAL(10, 2) NOT NULL,
153     metodo_pago VARCHAR(50) NOT NULL,
154     fecha_pago TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
155     estado VARCHAR(20) NOT NULL CHECK (estado IN ('pendiente', 'completado', 'reembolsado', 'fallido')),
156     referencia VARCHAR(100),
157     descripcion TEXT
158 );

```

Tabla de facturas

```

160  -- Tabla de Facturas
161  CREATE TABLE facturas (
162      factura_id SERIAL PRIMARY KEY,
163      pago_id INTEGER REFERENCES pagos(pago_id),
164      hotel_id INTEGER REFERENCES hoteles(hotel_id),
165      numero_factura VARCHAR(50) NOT NULL UNIQUE,
166      fecha_emision DATE NOT NULL,
167      subtotal DECIMAL(10, 2) NOT NULL,
168      impuestos DECIMAL(10, 2) NOT NULL,
169      total DECIMAL(10, 2) NOT NULL,
170      datos_cliente TEXT NOT NULL,
171      detalles TEXT NOT NULL
172 );

```

Tabla de registro de estadías

```

174  -- Tabla de Histórico de Estadías
175  CREATE TABLE historico_estadias (
176      historico_id SERIAL PRIMARY KEY,
177      hotel_id INTEGER REFERENCES hoteles(hotel_id),
178      cliente_id INTEGER REFERENCES clientes(cliente_id),
179      reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
180      fecha_entrada DATE NOT NULL,
181      fecha_salida DATE NOT NULL,
182      habitacion_id INTEGER REFERENCES habitaciones(habitacion_id),
183      comentarios TEXT,
184      calificacion INTEGER CHECK (calificacion BETWEEN 1 AND 5),
185      preferencias_registradas TEXT
186 );
187

```

Tabla de bitácoras de reservaciones

```

188
189  CREATE TABLE bitacora_reservaciones (
190      bitacora_id SERIAL PRIMARY KEY,
191      fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
192      usuario VARCHAR(50),
193      accion VARCHAR(20) NOT NULL,
194      tabla_afectada VARCHAR(30) NOT NULL,
195      registro_id INTEGER NOT NULL,
196      detalles TEXT,
197      ip_origen VARCHAR(50)
198 );
199

```

Capturas de las funciones, triggers y procedimientos

Función para registrar en bitácora

```
-- Función para registrar en bitácora
CREATE OR REPLACE FUNCTION registrar_bitacora(
    p_usuario VARCHAR,
    p_accion VARCHAR,
    p_tabla VARCHAR,
    p_registro_id INTEGER,
    p_detalles TEXT,
    p_ip VARCHAR
) RETURNS VOID AS $$
BEGIN
    INSERT INTO bitacora_reservaciones (usuario, accion, tabla_afectada, registro_id, detalles, ip_origen)
    VALUES (p_usuario, p_accion, p_tabla, p_registro_id, p_detalles, p_ip);
END;
$$ LANGUAGE plpgsql;
```

Trigger para cancelar reservación

```
219 CREATE OR REPLACE FUNCTION cancelacion_reservacion_trigger()
220 RETURNS TRIGGER AS $$
221 DECLARE
222     v_dias_restantes INTEGER;
223     v_hotel_nombre VARCHAR(100);
224     v_cliente_nombre VARCHAR(100);
225     v_habitaciones TEXT;
226 BEGIN
227     -- Solo actuar cuando el estado cambia a 'cancelada'
228     IF NEW.estado = 'cancelada' AND OLD.estado ≠ 'cancelada' THEN
229         -- Calcular días restantes hasta la fecha de entrada
230         v_dias_restantes := NEW.fecha_entrada - CURRENT_DATE;
231
232         -- Registrar en bitácora
233         PERFORM registrar_bitacora(
234             CURRENT_USER,
235             'CANCELACION',
236             'reservaciones',
237             NEW.reservacion_id,
238             'Reserva cancelada. Días restantes: ' || v_dias_restantes,
239             inet_client_addr()::TEXT
240         );
241
242         -- Enviar alerta si es cancelación de última hora (menos de 3 días)
243         IF v_dias_restantes < 3 THEN
244             -- Obtener información para la alerta
245             SELECT nombre INTO v_hotel_nombre FROM hoteles WHERE hotel_id = NEW.hotel_id;
246             SELECT nombre INTO v_cliente_nombre FROM clientes WHERE cliente_id = NEW.cliente_id;
247
248             -- Registrar alerta en bitácora
249             PERFORM registrar_bitacora(
250                 CURRENT_USER,
251                 'ALERTA',
252                 'reservaciones',
253                 NEW.reservacion_id,
254                 'ALERTA: Cancelación de última hora. Cliente: ' || v_cliente_nombre || ', Hotel: ' || v_hotel_nombre
255                 inet_client_addr()::TEXT
256             );
257
258             -- Aquí podrías agregar lógica para enviar email/notificación
259             RAISE NOTICE 'ALERTA: Cancelación de última hora. Reservación ID: %, Cliente: %, Hotel: %',
260                         | | | | NEW.reservacion_id, v_cliente_nombre, v_hotel_nombre;
261         END IF;

```

```

-- Liberar habitaciones asociadas
UPDATE habitaciones h
SET estado = 'disponible'
FROM reservacion_habitaciones rh
WHERE rh.habitacion_id = h.habitacion_id
AND rh.reservacion_id = NEW.reservacion_id;

-- Registrar liberación en bitácora
PERFORM registrar_bitacora(
    CURRENT_USER,
    'ACTUALIZACION',
    'habitaciones',
    NEW.reservacion_id,
    'Habitaciones liberadas por cancelación de reserva',
    inet_client_addr()::TEXT
);
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

286 CREATE TRIGGER tr_cancelacion_reservacion
287 AFTER UPDATE ON reservaciones
288 FOR EACH ROW
289 EXECUTE FUNCTION cancelacion_reservacion_trigger();
290

```

Trigger para registrar pago

```

291    2. Trigger para actualizar estado de reservación cuando se registra pago
292 CREATE OR REPLACE FUNCTION pago registrado_trigger()
293 RETURNS TRIGGER AS $$
294 BEGIN
295     -- Solo actuar cuando el pago se marca como completado
296     IF NEW.estado = 'completado' AND OLD.estado ≠ 'completado' THEN
297         -- Actualizar estado de la reservación a 'confirmada'
298         UPDATE reservaciones
299         SET estado = 'confirmada'
300         WHERE reservacion_id = NEW.reservacion_id;
301
302         -- Registrar en bitácora
303         PERFORM registrar_bitacora(
304             CURRENT_USER,
305             'CONFIRMACION',
306             'reservaciones',
307             NEW.reservacion_id,
308             'Reserva confirmada por pago completado. Pago ID: ' || NEW.pago_id,
309             inet_client_addr()::TEXT
310         );
311     END IF;
312
313     RETURN NEW;
314 END;
315 $$ LANGUAGE plpgsql;
316

```

```
317 -- Asociar trigger a la tabla pagos
318 CREATE TRIGGER tr_pagoRegistrado
319 AFTER UPDATE ON pagos
320 FOR EACH ROW
321 EXECUTE FUNCTION pagoRegistradoTrigger();
322
```

Procedimiento para verificar disponibilidad de habitaciones

```
324 CREATE OR REPLACE FUNCTION verificar_disponibilidad(
325     p_hotel_id INTEGER,
326     p_tipo_id INTEGER,
327     p_fecha_entrada DATE,
328     p_fecha_salida DATE
329 ) RETURNS TABLE (
330     habitacion_id INTEGER,
331     numero VARCHAR,
332     piso INTEGER,
333     precio_recomendado DECIMAL(10, 2)
334 ) AS $$
335 BEGIN
336     RETURN QUERY
337     SELECT h.habitacion_id, h.numero, h.piso,
338             COALESCE(
339                 (SELECT t.precio
340                  FROM tarifas_temporada t
341                  JOIN politicas_temporada p ON t.politica_id = p.politica_id
342                  WHERE t.tipo_id = h.tipo_id
343                  AND p.fecha_inicio <= p_fecha_entrada
344                  AND p.fecha_fin >= p_fecha_salida
345                  LIMIT 1),
346                 th.precio_base
347             ) AS precio_recomendado
348     FROM habitaciones h
349     JOIN tipos_habitacion th ON h.tipo_id = th.tipo_id
350     WHERE h.hotel_id = p_hotel_id
```

```

    JOIN tipos_habitacion th ON h.tipo_id = th.tipo_id
    WHERE h.hotel_id = p_hotel_id
    AND h.tipo_id = p_tipo_id
    AND h.estado = 'disponible'
    AND NOT EXISTS (
        SELECT 1 FROM reservacion_habitaciones rh
        JOIN reservaciones r ON rh.reservacion_id = r.reservacion_id
        WHERE rh.habitacion_id = h.habitacion_id
        AND r.estado NOT IN ('cancelada', 'no-show')
        AND (
            (r.fecha_entrada <= p_fecha_entrada AND r.fecha_salida > p_fecha_entrada) OR
            (r.fecha_entrada < p_fecha_salida AND r.fecha_salida >= p_fecha_salida) OR
            (r.fecha_entrada >= p_fecha_entrada AND r.fecha_salida <= p_fecha_salida)
        )
    );
END;
$$ LANGUAGE plpgsql;

```

Procedimiento para crear reservación

```

369 ~ CREATE OR REPLACE FUNCTION crear_reservacion(
370     p_hotel_id INTEGER,
371     p_cliente_id INTEGER,
372     p_fecha_entrada DATE,
373     p_fecha_salida DATE,
374     p_adultos INTEGER,
375     p_ninos INTEGER DEFAULT 0,
376     p_tipo_reserva VARCHAR(20) DEFAULT 'individual',
377     p_solicitudes_especiales TEXT DEFAULT NULL,
378     p_usuario VARCHAR(50) DEFAULT CURRENT_USER,
379     p_ip_origen VARCHAR(50) DEFAULT inet_client_addr()::TEXT,
380     p_tipos_habitaciones JSONB DEFAULT '[{"tipo_id": 1, "cantidad": 1}]'::JSONB
381 ) RETURNS INTEGER AS $$
382 ~ DECLARE
383     v_reservacion_id INTEGER;
384     v_codigo_reserva VARCHAR(20);
385     v_item JSONB;
386     v_habitacion RECORD;
387     v_disponibles INTEGER;
388     v_tarifa DECIMAL(10, 2);
389 ~ BEGIN
390     -- Validaciones iniciales
391     IF p_fecha_salida <= p_fecha_entrada THEN
392         RAISE EXCEPTION 'La fecha de salida debe ser posterior a la fecha de entrada';
393     END IF;

```

```
-- Verificar disponibilidad para cada tipo de habitación solicitada
FOR v_item IN SELECT * FROM jsonb_array_elements(p_tipos_habitaciones) LOOP
    -- Verificar que existe el tipo de habitación en el hotel
    PERFORM 1 FROM tipos_habitacion
    WHERE tipo_id = (v_item->>'tipo_id')::INTEGER AND hotel_id = p_hotel_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'El tipo de habitación % no existe en este hotel', (v_item->>'tipo_id');
    END IF;

    -- Contar habitaciones disponibles
    SELECT COUNT(*) INTO v_disponibles
    FROM verificar_disponibilidad(
        p_hotel_id,
        (v_item->>'tipo_id')::INTEGER,
        p_fecha_entrada,
        p_fecha_salida
    );

    IF v_disponibles < (v_item->>'cantidad')::INTEGER THEN
        RAISE EXCEPTION 'No hay suficientes habitaciones disponibles del tipo %', (v_item->>'tipo_id');
    END IF;
END LOOP;

-- Generar código de reserva único
v_codigo_reserva := 'RES-' || p_hotel_id || '-' ||
    EXTRACT(YEAR FROM CURRENT_DATE) || '-' ||
    LPAD(FLOOR(RANDOM() * 10000)::TEXT, 4, '0');

-- Crear la reservación principal
INSERT INTO reservaciones (
    hotel_id,
    cliente_id,
    fecha_entrada,
    fecha_salida,
    adultos,
    ninos,
    estado,
    tipo_reserva,
    solicitudes_especiales,
    codigo_reserva
) VALUES (
    p_hotel_id,
    p_cliente_id,
    p_fecha_entrada,
    p_fecha_salida,
    p_adultos,
    p_ninos,
    'confirmada', -- Estado inicial
    .
```

```

    p_tipo_reserva,
    p_solicitudes_especiales,
    v_codigo_reserva
) RETURNING reservacion_id INTO v_reservacion_id;

-- Asignar habitaciones específicas
FOR v_item IN SELECT * FROM jsonb_array_elements(p_tipos_habitaciones) LOOP
    -- Obtener tarifa aplicable
    SELECT COALESCE(
        (SELECT t.precio
         FROM tarifas_temporada t
         JOIN politicas_temporada p ON t.politica_id = p.politica_id
         WHERE t.tipo_id = (v_item->>'tipo_id')::INTEGER
         AND p.fecha_inicio <= p.fecha_entrada
         AND p.fecha_fin >= p.fecha_salida
         LIMIT 1),
        (SELECT precio_base FROM tipos_habitacion WHERE tipo_id = (v_item->>'tipo_id')::INTEGER)
    ) INTO v_tarifa;

    -- Asignar habitaciones disponibles
    FOR v_habitacion IN
        SELECT * FROM verificar_disponibilidad(
            p_hotel_id,
            (v_item->>'tipo_id')::INTEGER,
            p_fecha_entrada,
            p_fecha_salida
        ) LIMIT (v_item->>'cantidad')::INTEGER
    LOOP
        -- Asignar habitación a la reserva
        INSERT INTO reservacion_habitaciones (
            reservacion_id,
            habitacion_id,
            tarifa_aplicada,
            notas
        ) VALUES (
            v_reservacion_id,
            v_habitacion.habitacion_id,
            v_tarifa,
            'Asignación inicial'
        );

        -- Marcar habitación como ocupada
        UPDATE habitaciones
        SET estado = 'ocupada'
        WHERE habitacion_id = v_habitacion.habitacion_id;
    END LOOP;
END LOOP;

-- Registrar en bitácora
PERFORM registrar_bitacora(
    p_usuario,
    'CREACION',
    'reservaciones',
    v_reservacion_id,
    'Reserva creada con código ' || v_codigo_reserva,
    p_ip_origen
);

RETURN v_reservacion_id;
END;
$$ LANGUAGE plpgsql;

```

Procedimiento para cancelar reservación

```
CREATE OR REPLACE FUNCTION cancelar_reservacion(
    p_reservacion_id INTEGER,
    p_usuario VARCHAR,
    p_razon TEXT DEFAULT NULL
) RETURNS VOID AS $$
DECLARE
    v_estado_actual VARCHAR;
BEGIN
    -- Obtener estado actual
    SELECT estado INTO v_estado_actual FROM reservaciones WHERE reservacion_id = p_reservacion_id;

    IF v_estado_actual IS NULL THEN
        RAISE EXCEPTION 'Reservación no encontrada';
    END IF;

    IF v_estado_actual = 'cancelada' THEN
        RAISE NOTICE 'La reservación ya está cancelada';
        RETURN;
    END IF;

    -- Actualizar estado a cancelada
    UPDATE reservaciones
    SET estado = 'cancelada'
    WHERE reservacion_id = p_reservacion_id;
```

```
-- Registrar en bitácora (el trigger manejará la liberación de habitaciones)
PERFORM registrar_bitacora(
    p_usuario,
    'CANCELACION',
    'reservaciones',
    p_reservacion_id,
    'Reserva cancelada. Razón: ' || COALESCE(p_razon, 'No especificada'),
    inet_client_addr()::TEXT
);
END;
$$ LANGUAGE plpgsql;
```

Procedimiento para cambiar el estado de pago

```
549  CREATE OR REPLACE FUNCTION actualizar_estado_pago(
550    |   p_pago_id INTEGER,
551    |   p_estado VARCHAR,
552    |   p_usuario VARCHAR
553 ) RETURNS VOID AS $$
554 BEGIN
555     -- Validar estado
556     IF p_estado NOT IN ('pendiente', 'completado', 'reembolsado', 'fallido') THEN
557         |   RAISE EXCEPTION 'Estado de pago no válido';
558     END IF;
559
560     -- Actualizar estado del pago (el trigger manejará la actualización de la reserva)
561     UPDATE pagos
562     SET estado = p_estado
563     WHERE pago_id = p_pago_id;
564
565     -- Registrar en bitácora
566     PERFORM registrar_bitacora(
567         |   p_usuario,
568         |   'ACTUALIZACION',
569         |   'pagos',
570         |   p_pago_id,
571         |   'Estado de pago actualizado a: ' || p_estado,
572         |   inet_client_addr()::TEXT
573     );
574 END;
575 $$ LANGUAGE plpgsql;
576
```

Trigger de registro automático de bitácora sobre cambios de reservaciones:

```

CREATE OR REPLACE FUNCTION bitacora_reservaciones_trigger()
RETURNS TRIGGER AS $$

DECLARE
    v_accion VARCHAR;
    v_detalles TEXT;
BEGIN
    IF TG_OP = 'INSERT' THEN
        v_accion := 'CREACION';
        v_detalles := 'Nueva reserva creada. Estado: ' || NEW.estado;
    ELSIF TG_OP = 'UPDATE' THEN
        v_accion := 'ACTUALIZACION';
        v_detalles := 'Estado cambiado de ' || OLD.estado || ' a ' || NEW.estado;

        -- Detalles adicionales para cambios específicos
        IF OLD.fecha_entrada <> NEW.fecha_entrada OR OLD.fecha_salida <> NEW.fecha_salida THEN
            v_detalles := v_detalles || '. Fechas modificadas.';
        END IF;
    ELSIF TG_OP = 'DELETE' THEN
        v_accion := 'ELIMINACION';
        v_detalles := 'Reserva eliminada';
    END IF;

    PERFORM registrar_bitacora(
        CURRENT_USER,
        v_accion,
        'reservaciones',
        COALESCE(NEW.reservacion_id, OLD.reservacion_id),
        v_detalles,
        inet_client_addr()::TEXT
    );
END;

```

```

0   PERFORM registrar_bitacora(
1       CURRENT_USER,
2       v_accion,
3       'reservaciones',
4       COALESCE(NEW.reservacion_id, OLD.reservacion_id),
5       v_detalles,
6       inet_client_addr()::TEXT
7   );

8   IF TG_OP = 'DELETE' THEN
9       RETURN OLD;
0   ELSE
1       RETURN NEW;
2   END IF;
3
4 END;
5 $$ LANGUAGE plpgsql;
6
7 -- Asociar trigger a la tabla reservaciones
8 CREATE TRIGGER tr_bitacora_reservaciones
9 AFTER INSERT OR UPDATE OR DELETE ON reservaciones
0 FOR EACH ROW
1 EXECUTE FUNCTION bitacora_reservaciones_trigger();
2

```

Trigger para guardar registros de los pagos realizados:

```

624 CREATE OR REPLACE FUNCTION bitacora_pagos_trigger()
625 RETURNS TRIGGER AS $$ 
626 BEGIN
627     IF TG_OP = 'INSERT' THEN
628         PERFORM registrar_bitacora(
629             CURRENT_USER,
630             'CREACION',
631             'pagos',
632             NEW.pago_id,
633             'Nuevo pago registrado. Monto: ' || NEW.monto || ', Método: ' || NEW.metodo_pago,
634             inet_client_addr()::TEXT
635         );
636     ELSIF TG_OP = 'UPDATE' THEN
637         IF OLD.estado ≠ NEW.estado THEN
638             PERFORM registrar_bitacora(
639                 CURRENT_USER,
640                 'ACTUALIZACION',
641                 'pagos',
642                 NEW.pago_id,
643                 'Estado de pago cambiado de ' || OLD.estado || ' a ' || NEW.estado,
644                 inet_client_addr()::TEXT
645             );
646         END IF;
647     ELSIF TG_OP = 'DELETE' THEN
648         PERFORM registrar_bitacora(

```

```

647     ELSIF TG_OP = 'DELETE' THEN
648         PERFORM registrar_bitacora(
649             CURRENT_USER,
650             'ELIMINACION',
651             'pagos',
652             OLD.pago_id,
653             'Pago eliminado',
654             inet_client_addr()::TEXT
655         );
656     END IF;
657
658     IF TG_OP = 'DELETE' THEN
659         RETURN OLD;
660     ELSE
661         RETURN NEW;
662     END IF;
663 END;
664 $$ LANGUAGE plpgsql;
665
666 -- Asociar trigger a la tabla pagos
667 CREATE TRIGGER tr_bitacora_pagos
668 AFTER INSERT OR UPDATE OR DELETE ON pagos
669 FOR EACH ROW
670 EXECUTE FUNCTION bitacora_pagos_trigger();

```

Mecanismos de control de concurrencia

Se parte de la idea de que el sistema debe de soportar un alto nivel uso, es decir, escalabilidad y para atenderlo, lo ideal es un sistema de colas para manejar la alta concurrencia y en definitiva, crear una tabla y funciones para enrutar los pagos, es lo ideal.

```
-- Tabla de cola de pagos
CREATE TABLE cola_pagos (
    pago_queue_id SERIAL PRIMARY KEY,
    reservacion_id INTEGER NOT NULL REFERENCES reservaciones(reservacion_id),
    monto DECIMAL(10, 2) NOT NULL,
    metodo_pago VARCHAR(50) NOT NULL,
    usuario VARCHAR(50) NOT NULL,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado VARCHAR(20) DEFAULT 'pendiente' CHECK (estado IN ('pendiente', 'procesando',
    'completado', 'fallido')),
    intentos INTEGER DEFAULT 0,
    ultimo_intento TIMESTAMP,
    mensaje_error TEXT
);
```

Procedimiento para encolar pagos y procedimiento para procesar pagos, es muy importante destacar que utilizando skip locked se bloquean los pagos para manejar la concurrencia.

```
CREATE OR REPLACE FUNCTION encolar_pago(
    p_reservacion_id INTEGER,
    p_monto DECIMAL(10, 2),
    p_metodo_pago VARCHAR,
    p_usuario VARCHAR
) RETURNS INTEGER AS $$

DECLARE
    v_pago_queue_id INTEGER;
BEGIN
    INSERT INTO cola_pagos (
        reservacion_id, monto, metodo_pago, usuario
    ) VALUES (
        p_reservacion_id, p_monto, p_metodo_pago, p_usuario
    ) RETURNING pago_queue_id INTO v_pago_queue_id;

    RETURN v_pago_queue_id;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION procesar_cola_pagos()
RETURNS VOID AS $$
DECLARE
    v_pago RECORD;
BEGIN
    -- Se bloquea el procesamiento usando SKIP LOCKED para manejar concurrencia
    SELECT * INTO v_pago FROM cola_pagos
    WHERE estado = 'pendiente'
    ORDER BY fecha_creacion
    FOR UPDATE SKIP LOCKED
    LIMIT 1;

    IF FOUND THEN
        UPDATE cola_pagos
        SET estado = 'procesando',
            intentos = intentos + 1,
            ultimo_intento = CURRENT_TIMESTAMP
        WHERE pago_queue_id = v_pago.pago_queue_id;

        BEGIN
            PERFORM procesar_pago_concurrente(
                v_pago.reservacion_id,
                v_pago.monto,
                v_pago.metodo_pago,
                v_pago.usuario
            );

            UPDATE cola_pagos
            SET estado = 'completado'
            WHERE pago_queue_id = v_pago.pago_queue_id;
        EXCEPTION
            WHEN OTHERS THEN
                UPDATE cola_pagos
                SET estado = CASE WHEN intentos >= 3 THEN 'fallido' ELSE 'pendiente' END,
                    mensaje_error = SQLERRM
                WHERE pago_queue_id = v_pago.pago_queue_id;
        END;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

```
BEGIN
    PERFORM procesar_pago_concurrente(
        v_pago.reservacion_id,
        v_pago.monto,
        v_pago.metodo_pago,
        v_pago.usuario
    );

    UPDATE cola_pagos
    SET estado = 'completado'
    WHERE pago_queue_id = v_pago.pago_queue_id;
EXCEPTION
    WHEN OTHERS THEN
        UPDATE cola_pagos
        SET estado = CASE WHEN intentos >= 3 THEN 'fallido' ELSE 'pendiente' END,
            mensaje_error = SQLERRM
        WHERE pago_queue_id = v_pago.pago_queue_id;
END;
$$ LANGUAGE plpgsql;
```

Tablas para la auditoria de conflicto de concurrencia

```
CREATE TABLE conflictos_concurrencia (
    conflicto_id SERIAL PRIMARY KEY,
    fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    tipo_conflicto VARCHAR(50) NOT NULL,
    tabla_afectada VARCHAR(50) NOT NULL,
    id_registro INTEGER,
    usuario VARCHAR(50),
    detalles TEXT,
    resolucion VARCHAR(50)
);
```

Trigger para registrar automáticamente problemas de concurrencia

```
CREATE OR REPLACE FUNCTION registrar_conflicto()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO conflictos_concurrencia (
        tipo_conflicto, tabla_afectada, id_registro, usuario, detalles
    ) VALUES (
        TG_OP, TG_TABLE_NAME, COALESCE(NEW.id, OLD.id), CURRENT_USER,
        'Intento de modificación concurrente'
    );
    RETURN NULL; -- La idea es que falle para indicar que no pasó nada
END;
$$ LANGUAGE plpgsql;
```

Procedimientos de inserción

```
CREATE OR REPLACE PROCEDURE sp_crear_reservacion(
    p_id_cliente INTEGER,
    p_id_habitacion INTEGER,
    p_fecha_entrada DATE,
    p_fecha_salida DATE,
    p_numero_huespedes INTEGER,
    p_solicitudes_especiales TEXT,
    p_tarifa_aplicada NUMERIC(10,2),
    OUT p_reservacion_id INTEGER,
    OUT p_resultado TEXT
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_disponible BOOLEAN;
    v_habitacion_bloqueada BOOLEAN := FALSE;
BEGIN
    — Verificar disponibilidad con bloqueo de fila
    BEGIN
        SELECT NOT EXISTS (
            SELECT 1 FROM reservacion_habitaciones rh
            JOIN reservaciones r ON rh.reservacion_id = r.reservacion_id
            WHERE rh.habitacion_id = p_id_habitacion
            AND r.estado NOT IN ('cancelada', 'finalizada')
            AND (r.fecha_entrada, r.fecha_salida) OVERLAPS (p_fecha_entrada, p_fecha_salida)
            FOR UPDATE OF rh NOWAIT
        ) INTO v_disponible;
        EXCEPTION WHEN lock_not_available THEN
            v_habitacion_bloqueada := TRUE;
    END;

    IF v_habitacion_bloqueada THEN
        p_resultado := 'La habitación está siendo procesada por otro usuario. Intente nuevamente.';
        RETURN;
    END IF;

    IF NOT v_disponible THEN
        p_resultado := 'La habitación no está disponible para las fechas seleccionadas.';
        RETURN;
    END IF;
```

```

BEGIN
    INSERT INTO reservaciones (
        cliente_id,
        fecha_entrada,
        fecha_salida,
        huespedes,
        solicitudes_especiales,
        estado,
        fecha_creacion
    ) VALUES (
        p_id_cliente,
        p_fecha_entrada,
        p_fecha_salida,
        p_numero_huespedes,
        p_solicitudes_especiales,
        'pendiente',
        CURRENT_TIMESTAMP
    ) RETURNING reservacion_id INTO p_reservacion_id;

    INSERT INTO reservacion_habitaciones (
        reservacion_id,
        habitacion_id,
        tarifa_aplicada,
        notas
    ) VALUES (
        p_reservacion_id,
        p_id_habitacion,
        p_tarifa_aplicada,
        'Reservación inicial'
    );

    p_resultado := 'Reservación creada exitosamente con ID: ' || p_reservacion_id;

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        p_resultado := 'Error al crear reservación: ' || SQLERRM;
END;
$$;

```

```

CREATE OR REPLACE PROCEDURE sp_actualizar_reservacion(
    p_reservacion_id INTEGER,
    p_fecha_entrada DATE,
    p_fecha_salida DATE,
    p_numero_huespedes INTEGER,
    p_solicitudes_especiales TEXT,
    p_version INTEGER, -- Campo de versión para OCC
    OUT p_resultado TEXT
)
LANGUAGE plpgsql
AS $$

DECLARE
    v_current_version INTEGER;
    v_habitacion_id INTEGER;
    v_disponible BOOLEAN;
BEGIN
    -- Obtener versión actual y habitación asociada
    SELECT version, habitacion_id INTO v_current_version, v_habitacion_id
    FROM reservaciones r
    JOIN reservacion_habitaciones rh ON r.reservacion_id = rh.reservacion_id
    WHERE r.reservacion_id = p_reservacion_id
    FOR UPDATE;

    -- Verificar concurrencia optimista
    IF v_current_version ≠ p_version THEN
        p_resultado := 'La reservación ha sido modificada por otro usuario. Por favor refresque los datos.';
        RETURN;
    END IF;

    -- Verificar disponibilidad si cambian las fechas
    IF (p_fecha_entrada, p_fecha_salida) ≠ (
        SELECT fecha_entrada, fecha_salida FROM reservaciones
        WHERE reservacion_id = p_reservacion_id
    ) THEN
        SELECT NOT EXISTS (
            SELECT 1 FROM reservacion_habitaciones rh
            JOIN reservaciones r ON rh.reservacion_id = r.reservacion_id
            WHERE rh.habitacion_id = v_habitacion_id
            AND r.reservacion_id ≠ p_reservacion_id
            AND r.estado NOT IN ('cancelada', 'finalizada')
            AND (r.fecha_entrada, r.fecha_salida) OVERLAPS (p_fecha_entrada, p_fecha_salida)
        ) INTO v_disponible;
    END IF;

```

```
-- Actualizar reservación
UPDATE reservaciones
SET
    fecha_entrada = p_fecha_entrada,
    fecha_salida = p_fecha_salida,
    numero_huespedes = p_numero_huespedes,
    solicitudes_especiales = p_solicitudes_especiales,
    version = version + 1,
    fecha_actualizacion = CURRENT_TIMESTAMP
WHERE reservacion_id = p_reservacion_id;

p_resultado := 'Reservación actualizada exitosamente';

COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        p_resultado := 'Error al actualizar reservación: ' || SQLERRM;
END;
$$;
```

```
CREATE OR REPLACE PROCEDURE sp_cancelar_reservacion(
    p_reservacion_id INTEGER,
    p_motivo TEXT,
    OUT p_reembolso NUMERIC(10,2),
    OUT p_resultado TEXT
)
LANGUAGE plpgsql
AS $$

DECLARE
    v_estado_actual VARCHAR(20);
    v_fecha_entrada DATE;
    v_pagos_completados NUMERIC(10,2) := 0;
    v_tarifa_total NUMERIC(10,2);
    v_porcentaje_penalidad NUMERIC(5,2) := 0;
BEGIN

    SELECT r.estado, r.fecha_entrada,
           COALESCE(SUM(p.monto), 0)
    INTO v_estado_actual, v_fecha_entrada, v_pagos_completados
    FROM reservaciones r
    LEFT JOIN pagos p ON r.reservacion_id = p.reservacion_id AND p.estado = 'completado'
    WHERE r.reservacion_id = p_reservacion_id
    GROUP BY r.reservacion_id, r.estado, r.fecha_entrada
    FOR UPDATE;

    IF NOT FOUND THEN
        p_resultado := 'Reservación no encontrada';
        RETURN;
    END IF;

    IF v_estado_actual = 'cancelada' THEN
        p_resultado := 'La reservación ya estaba cancelada';
        p_reembolso := 0;
        RETURN;
    END IF;

    SELECT SUM(rh.tarifa_aplicada *
               (r.fecha_salida - r.fecha_entrada))
    INTO v_tarifa_total
    FROM reservaciones r
```

```

10 BEGIN
11     IF p_reembolso < v_pagos_completados THEN
12         INSERT INTO pagos (
13             metodo_pago,
14             estado,
15             descripcion
16         ) VALUES (
17             p_reservacion_id,
18             v_pagos_completados - p_reembolso,
19             'reembolso',
20             'reembolsado',
21             'Penalidad por cancelación: ' || p_motivo
22         );
23     END IF;
24
25     IF p_reembolso > 0 THEN
26         INSERT INTO pagos (
27             reservacion_id,
28             monto,
29             metodo_pago,
30             estado,
31             descripcion
32         ) VALUES (
33             p_reservacion_id,
34             p_reembolso,
35             'reembolso',
36             'reembolsado',
37             'Reembolso por cancelación: ' || p_motivo
38         );
39     END IF;
40
41     p_resultado := 'Reservación cancelada exitosamente. Reembolso: ' || p_reembolso;
42
43     COMMIT;
44
45 EXCEPTION
46     WHEN OTHERS THEN
47         ROLLBACK;
48         p_resultado := 'Error al cancelar reservación: ' || SQLERRM;
49         p_reembolso := 0;
50
51 END;
52 $$;

```

Índices creados

Listado de relaciones					
Esquema	Nombre	Tipo	Dueño		Tabla
public	bitacora_reservas_pkey	Índice	nechavarria		bitacora_reservas
public	clientes_pkey	Índice	nechavarria		clientes
public	facturas_numero_factura_key	Índice	nechavarria		facturas
public	facturas_pkey	Índice	nechavarria		facturas
public	fidelizacion_clientes_pkey	Índice	nechavarria		fidelizacion_clientes
public	habitaciones_pkey	Índice	nechavarria		habitaciones
public	historico_estadias_pkey	Índice	nechavarria		historico_estadias
public	hoteles_pkey	Índice	nechavarria		hoteles
public	unique_tipo_hotel	Índice	nechavarria		tipos_habitacion

(9 filas)

Corridas de ejemplo de la base de datos

Introducción de registros de prueba, también se adjuntan en el .sql

```
sistema_hotelero=# INSERT INTO hoteles (nombre, direccion, ciudad, pais, telefono, email, estrellas, fecha_apertura, descripcion)
sistema_hotelero=# VALUES
sistema_hotelero# ('Hotel Central', 'Av. Principal 123', 'San José', 'Costa Rica', '2222-1111', 'central@hotel.com', 4, '2020-01-01', 'Hotel de lujo en el centro');
INSERT 0 1
sistema_hotelero#
sistema_hotelero# -- Insertar clientes
sistema_hotelero# INSERT INTO clientes (nombre, documento_identidad, tipo_documento, nacionalidad, telefono, email, preferencias, alergias)
sistema_hotelero# VALUES
sistema_hotelero# ('Carlos Pérez', '12345678', 'DNI', 'Costarricense', '8888-9999', 'carlos@example.com', 'Cama King, Piso alto', 'Ninguna');
INSERT 0 1
sistema_hotelero#
sistema_hotelero# -- Fidelización
sistema_hotelero# INSERT INTO fidelizacion_clientes (cliente_id, hotel_id, puntos_acumulados, nivel_membresia, beneficios)
sistema_hotelero# VALUES
sistema_hotelero# (1, 1, 500, 'Oro', 'Check-out tardío, upgrades gratuitos');
INSERT 0 1
sistema_hotelero#
sistema_hotelero# -- Tipos de habitación
sistema_hotelero# INSERT INTO tipos_habitacion (hotel_id, nombre, descripcion, capacidad, tamano, comodidades, precio_base)
sistema_hotelero# VALUES
sistema_hotelero# (1, 'Estándar', 'Cómoda habitación estándar', 2, 20, 'TV, WiFi, A/C', 70.00),
sistema_hotelero# (1, 'Suite', 'Suite con sala', 4, 40, 'TV, WiFi, A/C, Sala', 150.00);
INSERT 0 2
sistema_hotelero#
sistema_hotelero# -- Habitaciones
sistema_hotelero# INSERT INTO habitaciones (hotel_id, numero, tipo_id, piso, caracteristicas_especiales, estado)
sistema_hotelero# VALUES
sistema_hotelero# (1, '101', 1, 1, 'Vista al jardín', 'disponible'),
sistema_hotelero# (1, '102', 1, 1, 'disponible'),
sistema_hotelero# (1, '201', 2, 2, 'Balcón', 'disponible');
INSERT 0 3
sistema_hotelero#
sistema_hotelero# -- Políticas de temporada
sistema_hotelero# INSERT INTO politicas_temporada (hotel_id, nombre, fecha_inicio, fecha_fin, descripcion, reglas)
sistema_hotelero# VALUES
sistema_hotelero# (1, 'Temporada Alta', '2025-12-01', '2026-01-31', 'Altísima demanda', 'Reservas no reembolsables');
INSERT 0 1
sistema_hotelero#
sistema_hotelero# -- Tarifas de temporada
sistema_hotelero# INSERT INTO tarifas_temporada (politica_id, tipo_id, precio, descripcion)
sistema_hotelero# VALUES
sistema_hotelero# (1, 1, 100.00, 'Precio temporada alta - estándar'),
sistema_hotelero# (1, 2, 200.00, 'Precio temporada alta - suite');
INSERT 0 2
```

Función para verificar disponibilidad:

```
sistema_hotelero#
sistema_hotelero# SELECT * FROM verificar_disponibilidad(1, 1, '2025-12-15', '2025-12-20');
habitacion_id | numero | piso | precio_recomendado
-----+-----+-----+
      1 | 101   |    1 |        100.00
      2 | 102   |    1 |        100.00
(2 rows)
```

Crear reservación:

```
sistema_hotelero# SELECT crear_reservacion(
sistema_hotelero#     1, -- hotel_id
sistema_hotelero#     1, -- cliente_id
sistema_hotelero#     '2025-12-15',
sistema_hotelero#     '2025-12-20',
sistema_hotelero#     2, -- adultos
sistema_hotelero#     0, -- niños
sistema_hotelero#     'individual',
sistema_hotelero#     'Necesita cuna',
sistema_hotelero#     '127.0.0.1',
sistema_hotelero#     '[{"tipo_id": 1, "cantidad": 1}]'
sistema_hotelero# );
crear_reservacion
-----
4
(1 row)

sistema_hotelero#
```

Cancelar reserva, se denota como se cancela la reserva, en el SELECT * FROM reserva de arriba se ve el único registro en confirmado y abajo en cancelado:

```
sistema_hotelero=# SELECT * FROM reservaciones;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| reservacion_id | hotel_id | cliente_id | fecha_creacion | fecha_entrada | fecha_salida | adultos | ninos | estado | tipo_reserv
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 4 | 1 | 1 | 2025-05-20 22:23:10.984702 | 2025-12-15 | 2025-12-20 | 2 | 0 | confirmada | individual
| Necesita cuna | RES-1-2025-9739
(1 row)

sistema_hotelero=# SELECT cancelar_reserva(4, 'admin', 'Cambio de planes');
cancelar_reserva
-----
(1 row)

sistema_hotelero=# SELECT * FROM reservaciones;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| reservacion_id | hotel_id | cliente_id | fecha_creacion | fecha_entrada | fecha_salida | adultos | ninos | estado | tipo_reserv
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 4 | 1 | 1 | 2025-05-20 22:23:10.984702 | 2025-12-15 | 2025-12-20 | 2 | 0 | cancelada | individual
| Necesita cuna | RES-1-2025-9739
(1 row)

sistema_hotelero=#
```

Realización de pagos, primero crearé una nueva reserva:

```
sistema_hotelero=# SELECT crear_reserva(
sistema_hotelero(#     1, -- hotel_id
sistema_hotelero(#     1, -- cliente_id
sistema_hotelero(#     '2025-12-13',
sistema_hotelero(#     '2025-12-23',
sistema_hotelero(#     2, -- adultos
sistema_hotelero(#     1, -- niños
sistema_hotelero(#     'grupo',
sistema_hotelero(#     'Necesita cuna',
sistema_hotelero(#     '127.0.3.1',
sistema_hotelero(#     '[{"tipo_id": 1, "cantidad": 1}]' -- JSONB habitaciones
sistema_hotelero(# );
crear_reserva
-----
5
(1 row)
```

Pago y verificación del pago:

```
sistema_hotelero=# INSERT INTO pagos (reserva_id, monto, metodo_pago, estado)
sistema_hotelero# VALUES (5, 100.00, 'Tarjeta', 'pendiente');
INSERT 0 1
sistema_hotelero=# SELECT * FROM pagos;
+-----+-----+-----+-----+-----+-----+-----+
| pago_id | reserva_id | monto | metodo_pago | fecha_pago | estado | referencia | descripcion
+-----+-----+-----+-----+-----+-----+-----+
| 1 | 5 | 100.00 | Tarjeta | 2025-05-20 22:30:34.031821 | pendiente |          |
(1 row)
```

Comprobación de los *triggers* de bitácora:

```
sistema_hotelero=# SELECT * FROM bitacora_reservaciones ORDER BY fecha_hora DESC;
bitacora_id | fecha_hora | usuario | accion | tabla_afectada | registro_id | detalles
| ip_origen
-----+-----+-----+-----+-----+-----+
10 | 2025-05-20 22:33:33.331128 | admin | ACTUALIZACION | pagos | 5 | Estado de pago actualizado a: completado
| ::1/128
9 | 2025-05-20 22:30:34.031821 | postgres | CREACION | pagos | 1 | Nuevo pago registrado. Monto: 100.00, Método: Tarjeta de crédito
| ::1/128
7 | 2025-05-20 22:29:42.72008 | postgres | CREACION | reservaciones | 5 | Nueva reserva creada. Estado: confirmada
| ::1/128
8 | 2025-05-20 22:29:42.72008 | 127.0.3.1 | CREACION | reservaciones | 5 | Reserva creada con código RES-1-2025-7587
| [{"tipo_id": 1, "cantidad": 1}]
6 | 2025-05-20 22:25:49.649673 | admin | CANCELACION | reservaciones | 4 | Reserva cancelada. Razón: Cambio de planeamiento
| ::1/128
3 | 2025-05-20 22:25:49.649673 | postgres | ACTUALIZACION | reservaciones | 4 | Estado cambiado de confirmada a cancelada
| ::1/128
4 | 2025-05-20 22:25:49.649673 | postgres | CANCELACION | reservaciones | 4 | Reserva cancelada. Días restantes: 209
| ::1/128
5 | 2025-05-20 22:25:49.649673 | postgres | ACTUALIZACION | habitaciones | 4 | Habitaciones liberadas por cancelación de reserva
| ::1/128
1 | 2025-05-20 22:23:10.984702 | postgres | CREACION | reservaciones | 4 | Nueva reserva creada. Estado: confirmada
| ::1/128
2 | 2025-05-20 22:23:10.984702 | 127.0.0.1 | CREACION | reservaciones | 4 | Reserva creada con código RES-1-2025-9739
| [{"tipo_id": 1, "cantidad": 1}]
(10 rows)
```

Confirmación de utilización de habitaciones

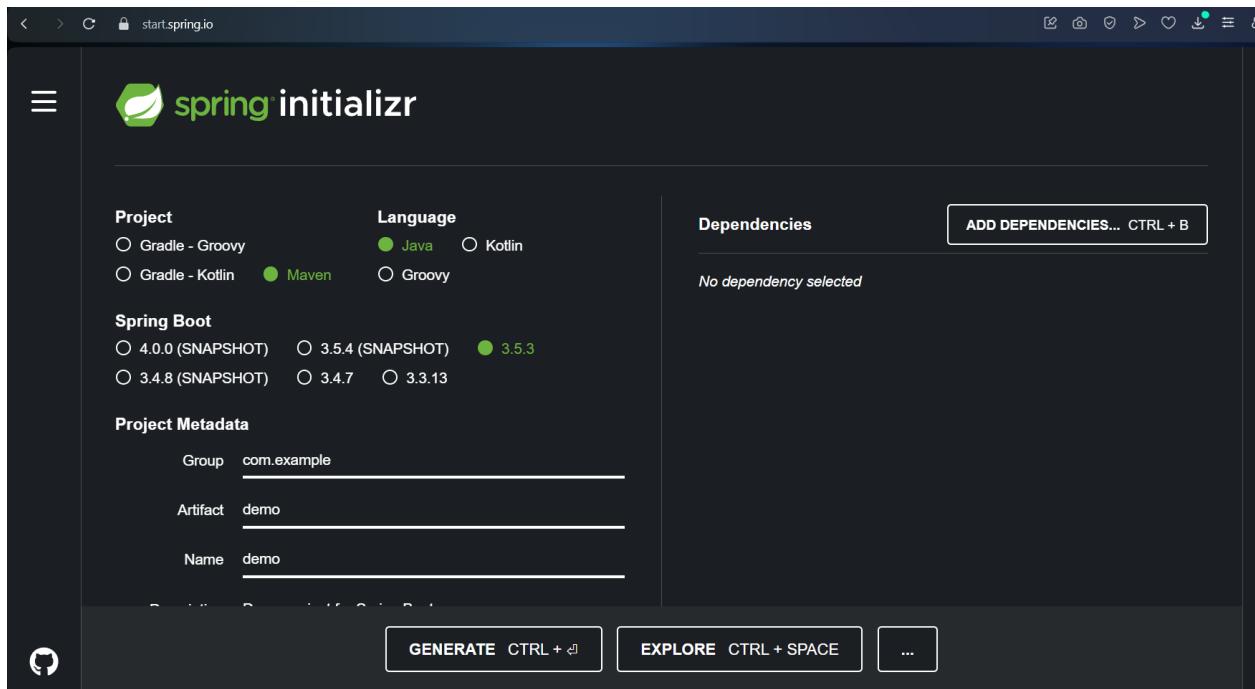
```
sistema_hotelero=# SELECT * FROM habitaciones WHERE estado = 'ocupada';
habitacion_id | hotel_id | numero | tipo_id | piso | caracteristicas_especiales | estado | notas
-----+-----+-----+-----+-----+-----+-----+-----+
1 | 1 | 101 | 1 | 1 | Vista al jardín | ocupada | null
(1 row)
```

API REST

Entorno de desarrollo

El API REST fue desarrollado en SpringBoot en su versión 3.5.3 con Java 24, se utiliza un gestionamiento de dependencias propias de Maven. Este es un ejemplo de como se empieza un proyecto descargando desde la página un proyecto demo.

El IDE de desarrollo utilizado es Intellij IDEA con licencia premium gratuita por ser estudiante.



Diseño de software para la implementación del api

Se trabaja en base a un patrón de diseño arquitectural MVC en donde los controladores funcionan en base al *framework* de código abierto de Jakarta ee, he de aclarar que no funcionan con servlets como un proyecto normal de web en jakarta.

Diseño de la abstracción para el funcionamiento del API

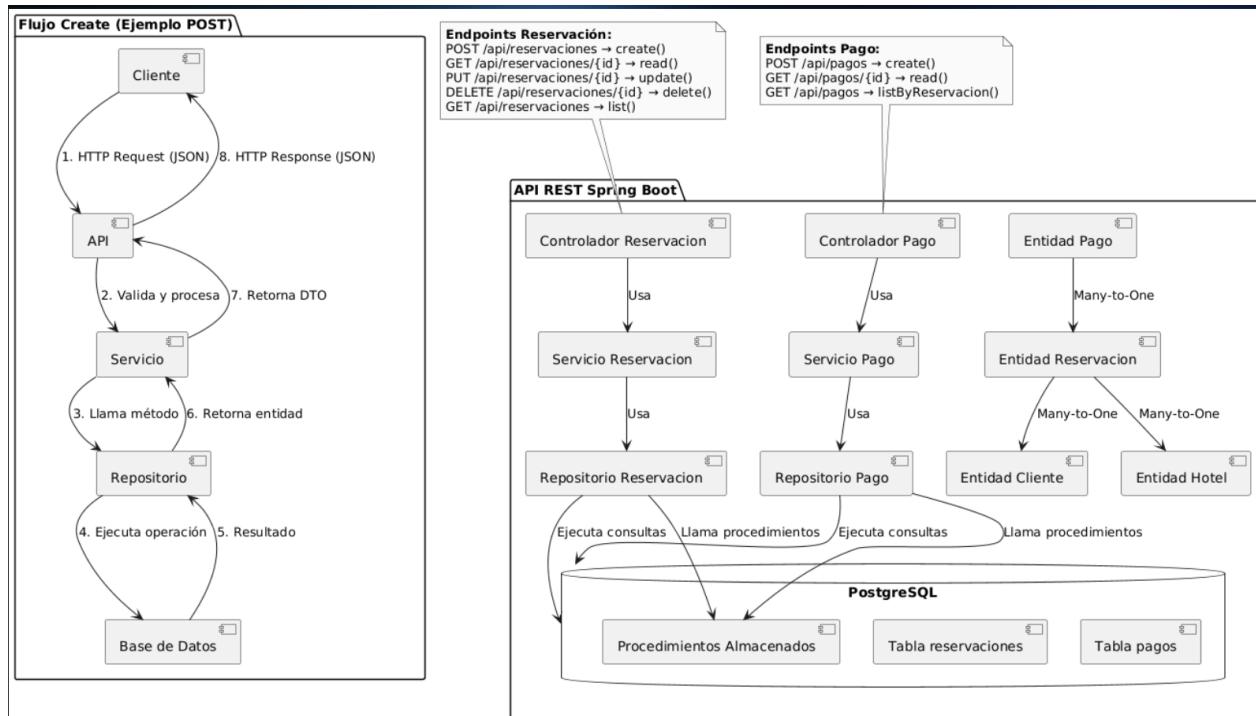
Estructura de Capas:

- Controladores: Manejan las solicitudes HTTP y devuelven respuestas
- Servicios: Contienen la lógica de negocio
- Repositorios: Gestionan el acceso a datos
- Entidades: Representan las tablas de la base de datos

Relaciones CRUD:

- Cada entidad tiene su propio controlador, servicio y repositorio
- Los repositorios interactúan directamente con la base de datos
- Los procedimientos almacenados se llaman desde los repositorios

Ejemplo del comportamiento del API REST, en este caso se ejemplifica el flujo del POST en reservaciones



Instrucciones para el uso del API

Primero que nada, existe un despliegue la base de datos en AWS, AZURE o cualquier servicio, por ejemplo, en esta aplicación se dispone una PaaS, aquí se denota cierta información de la instancia.

Información general

- Subscripción (mover) : Azure for Students
- Id. de suscripción : 394fa0e7-f26e-43cb-869e-add0ac6b9f9a
- Grupo de recursos (mover) : rg-postgres-nechavarria
- Nombre del servidor : postgresnechavarria
- Ubicación : Mexico Central
- Estado : Ready
- Creado el : 2025-06-05 21:24:32.0047847 UTC
- Etiquetas (editar) : Agregar etiquetas

Información esencial

Configuración	Detalles
Punto de conexión	: postgresnechavarria.postgres.database.azure.com
Punto de conexión virtual	: No habilitado
Inicio de sesión del administrador	: nechavarria
Configuración	: Con capacidad de ráfaga, B2s, 2 núcleos virtuales...
Versión de PostgreSQL	: 17.4
Zona de disponibilidad	: 1
Alta disponibilidad	: No habilitado

Propiedades

- Introducción
- Propiedades
- Recomendaciones (1)
- Supervisión
- Tutoriales

Bases de datos

Agree o quite favoritos presionando Ctrl+Shift+F

Proceso y almacenamiento: Cambiar

Redes: Cambiar

Búsqueda: Q Búsqueda

Estado: ESP LAA 23:33 28/6/2025

Para que el API REST se pueda conectar a la base de datos, el ip de origen se tiene que agregar a la configuración de redes de la instancia de la base datos.

The screenshot shows the 'Redes' (Network) section of the PostgreSQL instance settings. A message at the top indicates that the connection security configuration is being updated. On the left, there's a sidebar with various management options like 'Bases de datos', 'Conectar', and 'Replicación'. The main area shows a table for firewall rules:

Nombre de la regla de firewall	Dirección IP inicial	Dirección IP final
ClientIPAddress_2025-6-28_23-42-7	186.151.104.143	186.151.104.143
Nombre de la regla de firewall	Dirección IP inicial	Dirección IP final

Below this, there's a section for 'Puntos de conexión privados' (Private endpoint connections) with a table:

Puntos de conexión privados	Estado de la conexión	Subred o red virtual	Nombre de conexión	Descripción
Sin resultados				

Por otro lado, se deben de agregar los detalles de la conexión al archivo application.properties de la aplicación, por ejemplo, de esta forma, he de aclarar que se debe de incluir el conector de base de datos al archivo pom.xml.

The screenshot shows the `application.properties` file in a code editor. The content includes the database connection details and some JPA configurations:

```

spring.application.name=Api_Rest_nechavarria
spring.datasource.url=jdbc:postgresql://postgresnechavarria.postgres.database.azure.com:5432/sistema-hotelero?user=nechavarria&password=sprigatito#123
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

# JPA
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

# Procedimientos almacenados
spring.jpa.properties.hibernate.proc.param_null_passing=true

```

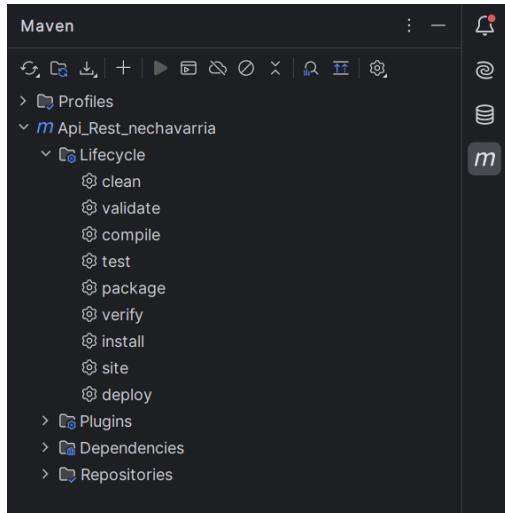
Dependencia para el pom.xml para utilizar la base de datos de postgresql

```

<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>

```

Una vez la aplicación tiene la codificación y las credenciales en funcionamiento, se debe de compilar, por ejemplo, utilizando el *compile* de IntelliJ o por otra parte poniendo el comando *mvn compile package* en el power Shell en la ruta en donde se tenga la raíz del proyecto codificado



Una vez se ejecute/compile el proyecto, se generará en la carpeta de target un archivo .war que básicamente tiene la aplicación, esta misma se debe de ejecutar hacia un puerto web para que así se pueda acceder al API REST en funcionamiento y por ejemplo, hacer pruebas de carga sobre la aplicación, en este caso el .war se llama demo-1.0.1

respaldo > ... API_REST_JAVA > codificacion_java > SpringBoost_demo > API_nechavarria > target >				
	Nombre	Fecha de modificación	Tipo	Tamaño
	classes	28/6/2025 22:55	Carpeta de archivos	
	demo-1.0.1	9/6/2025 02:43	Carpeta de archivos	
	generated-sources	9/6/2025 02:44	Carpeta de archivos	
	generated-test-sources	9/6/2025 02:42	Carpeta de archivos	
	maven-archiver	9/6/2025 02:43	Carpeta de archivos	
	maven-status	9/6/2025 02:39	Carpeta de archivos	
	surefire-reports	9/6/2025 02:43	Carpeta de archivos	
	test-classes	9/6/2025 02:42	Carpeta de archivos	
	demo-1.0.1	9/6/2025 02:43	war	63 622 KB
	demo-1.0.1.war.original	9/6/2025 02:43	Archivo ORIGINAL	53 238 KB

Una vez existiendo este archivo, se tiene que ejecutar, esto se puede hacer con el tomcat o nginx, esto queda a gusto del usuario. De esta forma el archivo será accesible desde el puerto 8080 de la maquina en donde se ejecute.

Para el proceso de CRUD, se debe de crear un archivo para indicar lo que se necesita hacer, por ejemplo, un de script en python, se debe validar un usuario para el uso del API REST esto por temas de seguridad, este es un ejemplo de como se puede hacer un CREATE de usuarios en Python hacia la aplicación

```

create data.py
C:\> Users > Usuario > Desktop > proyecto_2 > API_REST_JAVA > Base de datos y CRUD > CRUD > create data.py > ...
1 import requests
2 import base64
3
4
5 username = "Mimi"
6 password = "4332"
7 auth_string = f"{username}:{password}"
8 auth_bytes = auth_string.encode('ascii')
9 base64_auth = base64.b64encode(auth_bytes).decode('ascii')
10 headers = {
11     "Authorization": f"Basic {base64_auth}",
12     "Content-Type": "application/json"
13 }
14
15 body = {
16     "nombre": "Martin",
17     "apellido": "Pellegrini",
18     "documentoIdentidad": "292929292",
19     "tipoDocumento": "DNI",
20     "nacionalidad": "Costa Rica",
21     "telefono": "88888888",
22     "email": "juan@example.com"
23 }
24
25 url = "http://localhost:8080/clientes"
26 response = requests.post(url, json=body, headers=headers)
27
28 print(f"Status Code: {response.status_code}")
29 print(f"Response: {response.text}")
30
31

```

Resultados de uso del api

Cuando se ejecuta la aplicación se debe de ver así en intelliJ, como he dicho antes, se puede tomar el ejecutable y simplemente ponerlo a correr con herramientas como nginx

The screenshot shows the IntelliJ IDEA interface. In the top navigation bar, the project is named 'API_nechavarria'. Below it, the 'Run' tab is selected, showing a 'DemoApplication' configuration. The main editor area displays a Java file named 'Habitaciones.java' with the following code:

```

public void setHabitacion_id(int habitacion_id) { no usages new 24 18 ^ this.habitacion_id = habitacion_id;

```

Below the editor is a terminal window with the following content:

```

D:\java 24 set\bin\java.exe" ...
.
.
.
:: Spring Boot ::      (v3.5.0)

2025-06-29T00:03:16.278-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] c.nechavarria.proyecto2.DemoApplication : Starting DemoApplication u
2025-06-29T00:03:16.284-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] c.nechavarria.proyecto2.DemoApplication : No active profile set, fal
2025-06-29T00:03:18.145-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repos
2025-06-29T00:03:18.248-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repos
2025-06-29T00:03:19.984-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with po
2025-06-29T00:03:20.008-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-06-29T00:03:20.008-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [
2025-06-29T00:03:20.125-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedd
2025-06-29T00:03:20.126-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext
2025-06-29T00:03:20.626-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing Pers
2025-06-29T00:03:20.744-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] org.hibernate.Version : HHH000412: Hibernate ORM c
2025-06-29T00:03:20.830-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level ca
2025-06-29T00:03:21.448-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: i
2025-06-29T00:03:21.502-06:00 INFO 3336 — [Api_Rest_nechavarria] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting ...

```

En la propia consola se denota cuando se conecta correctamente a la base de datos instanciada

```

[main] c.nechavarria.proyecto2.DemoApplication : Starting DemoApplication using Java 24.0.1 with PID 1/b04 (C:\Users\usuario\Desktop\proyecto_2\API_1)
[main] c.nechavarria.proyecto2.DemoApplication : No active profile set, falling back to 1 default profile: "default"
[main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repository in DEFAULT mode.
[main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 134 ms. Found 3 JPA repository interfaces.
[main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
[main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
[main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.41]
[main] o.a.c.c.Tomcat.[localhost].[] : Initializing Spring embedded WebApplicationContext
[main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4954 ms
[main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
[main] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.6.15.Final
[main] o.h.c.internal.RegionFactoryInitiator : HHH00026: Second-level cache disabled
[main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
[main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting ...
[main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection org.postgresql.jdbc.PgConnection@1a3c4b3e
[main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
[main] org.hibernate.orm.deprecation : HHH90000025: PostgreSQLDialect does not need to be specified explicitly using 'hibernate.dialect' (O
[main] org.hibernate.orm.connections.Pooling : HHH10001005: Database info:

```

Demostración de funcionamiento del API REST

Para exemplificar la carga de datos se adjunta la compilación del un programita en Python que crea, consulta y elimina un cliente de la aplicación para demostrar el funcionamiento del API REST, este archivo se adjunta en la documentación, he aquí capturas de la codificación presente en el.

```

# create_data.py
# C:\Users\usuario\Desktop\proyecto_2\API_REST_HAWA\Base de datos y CRUD>CRUD>create_data.py & actualizar_cliente
import base64

# Configuración básica de autenticación
username = "Mimi"
password = "432"
auth_string = f'{username}:{password}'
auth_bytes = auth_string.encode('ascii')
base64_auth = base64.b64encode(auth_bytes).decode('ascii')
headers = {
    "Authorization": f"Basic {base64_auth}",
    "Content-Type": "application/json"
}
base_url = "http://localhost:8080/clientes"

def crear_cliente():
    """
    ***Caso de creación de un nuevo cliente***
    print("\n--- Creando nuevo cliente ---")
    body = {
        "nombre": "Marlon",
        "apellido": "Angelo",
        "documentoIdentidad": "929901192",
        "tipoDocumento": "DNI",
        "nacionalidad": "Costa Rica",
        "telefono": "88888888",
        "email": "juan@example.com"
    }
    response = requests.post(base_url, json=body, headers=headers)
    print(f"Status Code: {response.status_code}")
    print(f"Response: {response.text}")

    # Devolvemos el ID del cliente creado (usando el campo "id" de la respuesta)
    if response.status_code == 200:
        return response.json().get("id") # Cambiado de "cliente" a "id"
    return None

```

```

def obtener_cliente(cliente_id):
    """
    ***Caso de obtención de un cliente con ID***
    print("\n--- Obteniendo cliente ID: {cliente_id} ---")
    url = f'{base_url}/{cliente_id}'
    response = requests.get(url, headers=headers)

    print(f"Status Code: {response.status_code}")
    if response.status_code == 200:
        print("Cliente encontrado:")
        print(response.json())
    else:
        print(f"Response: {response.text}")

def listar_clientes():
    """
    ***Caso de listado de clientes con filtros opcionales***
    print("\n--- Listando clientes ---")
    params = {
        "nombre": "Marlon", # Filtro opcional
        "email": "juan@example.com", # Otros filtros opcionales
        "nacionalidad": "Costa Rica"
    }

    response = requests.get(f'{base_url}/ver', headers=headers, params=params)

    print(f"Status Code: {response.status_code}")
    if response.status_code == 200:
        print("Clientes encontrados:")
        for idx, cliente in enumerate(response.json(), 1):
            print(f"\nCliente #idx:{idx}")
            print(f"ID: {cliente.get('id')}")
            print(f"Nombre: {cliente.get('nombre')}")
            print(f"Email: {cliente.get('email')}")
            print(f"Activo: {cliente.get('activo')}")
    else:
        print(f"Response: {response.text}")

```

```

def actualizar_cliente(cliente_id):
    """Ejemplo de actualización de un cliente"""
    print(f"\n--- Actualizando cliente ID: {cliente_id} ---")
    url = f"{base_url}/{cliente_id}"
    body = [
        "nombre": "Marlon",
        "apellido": "Modificado",
        "documentoIdentidad": "922148201",
        "tipoDocumento": "DNI",
        "nacionalidad": "Costa Rica",
        "telefono": "71235293",
        "email": "martinactualizado@example.com"
    ]
    response = requests.put(url, json=body, headers=headers)
    print(f"Status Code: {response.status_code}")
    print(f"Response: {response.text}")

def eliminar_cliente(cliente_id):
    """Ejemplo de eliminación de un cliente"""
    print(f"\n--- Eliminando cliente ID: {cliente_id} ---")
    url = f"{base_url}/{cliente_id}"
    response = requests.delete(url, headers=headers)
    print(f"Status Code: {response.status_code}")
    print(f"Response: {response.text}")

```

Bloque ejecutado

```

if __name__ == "__main__":
    try:
        # 1. Crear un nuevo cliente
        print("== EJEMPLO COMPLETO CRUD ==")
        cliente_id = crear_cliente()

        if cliente_id:
            # 2. Obtener el cliente recién creado
            obtener_cliente(cliente_id)

            # 3. Listar clientes (con filtro)
            listar_clientes()

            # 4. Actualizar el cliente
            actualizar_cliente(cliente_id)

            # 5. Verificar la actualización
            obtener_cliente(cliente_id)

            # 6. Eliminar el cliente
            eliminar_cliente(cliente_id)

            # 7. Verificar que ya no existe
            listar_clientes()
    except Exception as e:
        print(f"\nOcurrió un error: {str(e)}")

```

Creación de un usuario con el API REST

```

05:55 & C:/Users/Usuario/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Usuario/Desktop/proyecto_2/API_REST_JAVA/Base de datos y CRUD/crea_data.py"
== EJEMPLO COMPLETO CRUD ==
--- Creando nuevo cliente ---
Status Code: 200
Response: {"id":25,"nombre":"Marlon","email":"juan@example.com","activo":true,"preferencias":null,"alergias":null,"passwordHash":null}

--- Obteniendo cliente ID: 25 ---
Status Code: 200
Cliente encontrado:
{"id": 25, "nombre": 'Marlon', 'email': 'juan@example.com', 'activo': True, 'preferencias': None, 'alergias': None, 'passwordHash': None}

```

Consulta de los clientes con el nombre de Marlon, aquí se debería de notar la creación del usuario

```

--- Listando clientes ---
Status Code: 200
Clientes encontrados:

Cliente #1:
ID: 25
Nombre: Marlon
Email: juan@example.com
Activo: True

```

Update sobre el cliente id 25, el recién creado

```

--- Actualizando cliente ID: 25 ---
Status Code: 200
Response: {"id":25,"nombre":"Marlon","email":"martinactualizado@example.com","activo":true,"preferencias":null,"alergias":null,"passwordHash":null}

--- Obteniendo cliente ID: 25 ---
Status Code: 200
Cliente encontrado:
{'id': 25, 'nombre': 'Marlon', 'email': 'martinactualizado@example.com', 'activo': True, 'preferencias': None, 'alergias': None, 'passwordHash': None}

--- Eliminando cliente ID: 25 ---
Status Code: 200
Response: Cliente eliminado

```

Verificación de cliente eliminado correctamente

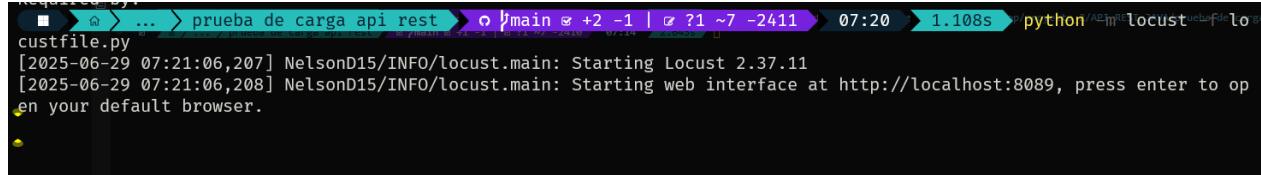
```

--- Listando clientes ---
Status Code: 200
Clientes encontrados:
05:55 4.281s

```

Pruebas de carga sobre el API

Las pruebas se desarrollaron utilizando una herramienta llamada locust, “*Locust es una herramienta de pruebas de carga de código abierto, escrita en Python, que permite simular el comportamiento de miles de usuarios simultáneos para evaluar el rendimiento de aplicaciones web y otros sistemas*” (Vignolo, 2021). Para utilizarla se debe instalar, por ejemplo `pip install locust` y después en la ruta del archivo de configuración de la prueba, se debe realizar el siguiente comando en powerShell



Configuración de la prueba en python

```

from locust import HttpUser, task, between
import base64

class ApiUser(HttpUser):
    wait_time = between(1, 5)

    def on_start(self):
        # Configuración de autenticación
        username = "Mimi"
        password = "4332"
        auth_string = f"{username}:{password}"
        auth_bytes = auth_string.encode('ascii')
        self.base64_auth = base64.b64encode(auth_bytes).decode('ascii')
        self.headers = {
            "Authorization": f"Basic {self.base64_auth}",
            "Content-Type": "application/json"
        }

    @task
    def crear_cliente(self):
        body = {
            "nombre": "Marlon",
            "apellido": "Angelo",
            "documentoIdentidad": "292901192",
            "tipoDocumento": "DNI",
            "nacionalidad": "Costa Rica",
            "telefono": "88888888",
            "email": "juan@example.com"
        }
        self.client.post("/clientes", json=body, headers=self.headers)

    @task(3) # Se ejecutará 3 veces más que 'crear_cliente'
    def obtener_clientes(self):
        self.client.get("/clientes/ver", headers=self.headers)

```

```

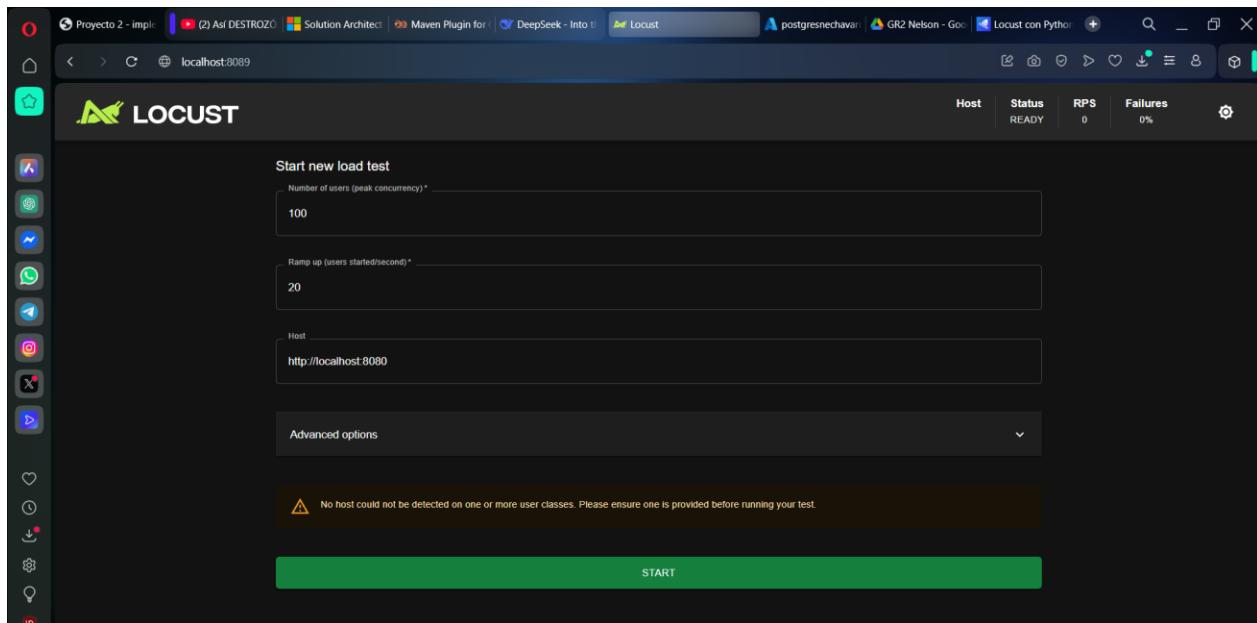
@task
def crear_cliente(self):
    body = {
        "nombre": "Marlon",
        "apellido": "Angelo",
        "documentoIdentidad": "292901192",
        "tipoDocumento": "DNI",
        "nacionalidad": "Costa Rica",
        "telefono": "88888888",
        "email": "juan@example.com"
    }
    self.client.post("/clientes", json=body, headers=self.headers)

@task(3) # Se ejecutará 3 veces más que 'crear_cliente'
def obtener_clientes(self):
    self.client.get("/clientes/ver", headers=self.headers)

@task(2) # Se ejecutará 2 veces más que 'crear_cliente'
def actualizar_cliente(self):
    body = {
        "nombre": "Marlon Actualizado",
        "apellido": "Modificado",
        "documentoIdentidad": "922148201",
        "tipoDocumento": "DNI",
        "nacionalidad": "Costa Rica",
        "telefono": "71235293",
        "email": "martinactualizado@example.com"
    }
    self.client.put("/clientes/1", json=body, headers=self.headers)

```

Una vez hecho lo anterior se debe de configurar la prueba en la pagina web localhost:8089 que se ejecuta al realizar el comando en PowerShell visto antes



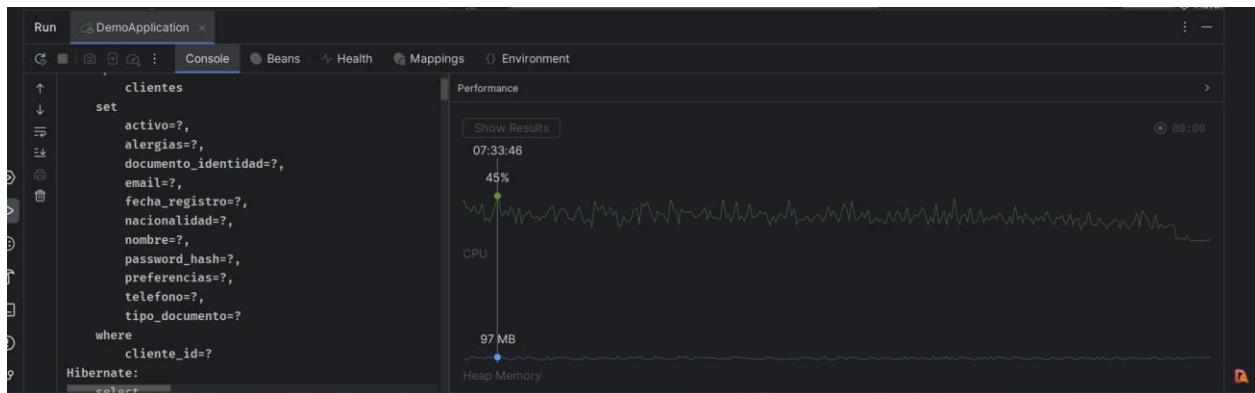
Resultados

Una vez iniciada la prueba los resultados obtenidos son perfectos, no hay errores

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s	
POST	/clientes	176	0	1700	4300	5300	1952.8	863	5980	124.58	3	0
PUT	/clientes/1	335	0	2100	4900	5900	2364.64	711	6566	148	6.8	0
GET	/clientes/ver	557	0	1400	4600	6900	1704.04	325	7181	15085.61	11.8	0
Aggregated		1068	0	1700	4600	6400	1952.24	325	7181	7934.64	21.6	0

Desempeño del API REST cuando se somete a carga

Se denota que no se supera el 45% de uso del CPU y la memoria en 97 megas lo cual es bajo y denota optimización.



Recuperación de datos ante problemas

Para exemplificar la recuperación de dato se adjunta este video

Anexos

Costos de aws

<https://aws.amazon.com/es/ec2/pricing/>

<https://aws.amazon.com/es/rds/aurora/pricing/>

<https://aws.amazon.com/es/rds/pricing/>

Referencias

Vignolo, B. (2021). Locust with Python: Introduction, Correlating Variables, and Basic Assertions. BlazeMeter. <https://www.blazemeter.com/blog/locust-python>

SpringBoot. (2025). Documentation Overview. <https://docs.spring.io/spring-boot/documentation.html>

Oracle (2025). JDK 24 Documentation. <https://docs.oracle.com/en/java/javase/24/>

Postgresql (s.f.). Documentation. <https://www.postgresql.org/docs/current/sql-syntax.html>