Universidad Tecnológica de Querétaro.

Gestión del proceso de software

1ra evaluación

Grupo: IDSG 11

Alumnos:

Salinas Jiménez María Dolores

Torres Jiménez Ricardo

Miralrío Espinoza Hugo Alberto

Serrano Cruz Hernán

Maestro: Brando Efrén Venegas Olvera

14/10/2025

Caso de Estudio: Plataforma de Gestión de Servicios Turísticos

El proyecto consiste en desarrollar una plataforma web y móvil que conecte a turistas con oferentes locales (anfitriones) que brindan servicios como alojamiento, gastronomía, recorridos, transporte o experiencias culturales. El objetivo es digitalizar la oferta turística local, mejorar la visibilidad de los servicios y facilitar la reserva, pago y evaluación de experiencias en tiempo real.

1. Parámetros de Configuración de las Herramientas Utilizadas

Herramienta	Versión / Configuración	Descripción
Frontend: React + Vite	React 18.x, Vite 5.x	Creación de la interfaz de usuario rápida y modular.
Backend: Node.js (Express.js)	Node 20.x	API REST para manejo de usuarios, reservas y pagos.
Base de datos: MySQL	v8.x	Almacenamiento de datos de usuarios, servicios y transacciones.
Control de versiones: Git + GitHub	Flujo GitFlow	Control de versiones y colaboración en equipo.
Pruebas: Postman + Jest	Configuración base	Validación de endpoints y funciones críticas.
Despliegue: Render (backend) + Vercel (frontend)	CI/CD integrado	Entornos de prueba y producción automatizados.
Diseño: Figma	Prototipado UI/UX	Diseño previo de la interfaz y estructura visual.

Editor: Visual Studio Code	ESLint + Prettier	Editor principal con reglas de estilo y calidad de código.

Configuraciones clave:
- Archivo .env para variables sensibles (puertos, API keys, DB credentials).
- Scripts en package.json para desarrollo, pruebas y despliegue (npm run dev, npm test, npm run deploy).
- Uso de ESLint para detectar errores de sintaxis y mantener buenas prácticas.

2. Plan de Pruebas

El plan de pruebas busca asegurar que la plataforma funcione correctamente en todos sus módulos antes de ser liberada al público.

Objetivos:

- Garantizar la correcta autenticación y registro de usuarios.
- Asegurar que los servicios turísticos se puedan publicar, reservar y calificar.
- Confirmar la estabilidad y comunicación entre frontend, backend y base de datos.

Tipos de Pruebas:

1. Unitarias: Validan funciones individuales.
2. Integración: Comprueban interacción entre componentes.
3. Funcionales: Simulan la experiencia del usuario.
4. Rendimiento: Evalúan tiempos de respuesta.
5. Aceptación: Verifican cumplimiento de requerimientos.

Criterios de aceptación:

- Los usuarios pueden registrarse e iniciar sesión sin errores.
- Los anfitriones pueden publicar servicios correctamente.
- Los turistas pueden realizar reservas y dejar comentarios.
- No existen fallos críticos en las rutas principales.

3. Casos de Pruebas

Caso de Prueba	Descripción	Entrada / Acción	Resultado Esperado

CP-01: Publicación de servicio turístico	Verificar que un anfitrión pueda registrar un nuevo servicio.	Llenar formulario con nombre, descripción, precio y fotos.	El servicio se guarda y aparece listado en la categoría correspondiente.
CP-02: Reserva de experiencia	Comprobar que un turista pueda reservar una experiencia disponible.	Seleccionar servicio, fecha y confirmar reserva.	Se genera una reserva y se envía confirmación al usuario y anfitrión.
CP-03: Calificación y comentario	Validar que los usuarios puedan evaluar servicios.	Escribir reseña y asignar puntuación tras finalizar la experiencia.	La calificación se guarda y actualiza la valoración promedio del servicio.
CP-04: Filtro y búsqueda de alojamientos	Verificar que el usuario pueda buscar hospedajes según ubicación, precio o tipo.	Ingresar criterios de búsqueda (ciudad, rango de precios, tipo de alojamiento).	El sistema muestra una lista de alojamientos que coinciden con los filtros aplicados.
CP-05: Cancelación de reserva	Validar que un usuario pueda cancelar una reserva activa dentro del plazo permitido.	Acceder al historial de reservas y seleccionar “Cancelar”.	La reserva cambia su estado a “Cancelada” y se notifica al anfitrión.

4. Flujo de Trabajo para el Control de Versiones

El flujo de trabajo implementa GitFlow, ideal para equipos distribuidos y proyectos con múltiples etapas.

1. Rama main: versión estable en producción.
2. Rama develop: contiene las últimas funcionalidades.

3. Ramas feature/nombre-funcion: nuevas características.
4. Ramas test/nombre: validación antes del release.
5. Ramas fix/nombre-error: correcciones en producción.
6. Ramas release/x.x.x: preparan versiones finales.

Buenas prácticas:
- Commits descriptivos (ejemplo: feat: add booking module).
- Pull Requests revisados antes de integrar.
- Etiquetas (tags) para versiones estables.

5. Estrategia de Despliegue

Se adopta un enfoque de despliegue continuo (CI/CD) que permite liberar versiones estables de forma rápida y segura.

Fases del Despliegue:
1. Entorno de desarrollo: pruebas locales con Docker y MySQL.
2. Entorno de pruebas (staging): despliegue automático en Render y Vercel.
3. Entorno de producción: versión estable desplegada con GitHub Actions y respaldos automáticos.

Monitoreo y rollback:
- Logs y métricas (PM2, LogRocket) para detectar fallos.
- Rollback con git checkout a la última versión estable.

6. Artifacts que se implementaran en el proyecto

- Aplicación Móvil (Frontend - React Native): Proporcionar a los usuarios una interfaz
amigable para consultar opciones de hospedaje y comida en una ubicación específica.
Incluye módulos para búsqueda, filtrado, visualización de detalles, calificaciones y
mapas.
- API Backend (Node.js con Express): Gestionar la lógica de negocio, servir datos a la
aplicación móvil, autenticar usuarios y procesar solicitudes relacionadas con hospedaje
y comida.
- Base de Datos (MySQL o PostgreSQL): Almacenar información sobre hospedajes,
restaurantes, usuarios y reservas.

- Panel de Administración (Web - React o Next.js): Permitir a administradores registrar
nuevos hospedajes, restaurantes, actualizar información y gestionar usuarios.
- Documentación Técnica: Facilitar la comprensión y mantenimiento del sistema,
incluyendo manual de usuario, manual de instalación y API docs (Swagger).

7. Plataformas y herramientas de versionamiento a utilizar
- GitHub: Plataforma principal para alojar el repositorio del proyecto.
- Git: Sistema de control de versiones distribuido.
- GitHub Actions: Automatización de pruebas y despliegue continuo (CI/CD).
- Docker: Contenerización de los servicios para asegurar portabilidad.

8. Estrategia de despliegue
El proyecto se desplegará bajo un enfoque ágil y de bajo costo, utilizando servicios Serverless y contenedores ligeros para minimizar los costos operativos y simplificar el mantenimiento.
Dado que se cuenta con un tiempo limitado de 8 semanas para cubrir la planeación, desarrollo y despliegue, se priorizará la automatización, la escalabilidad inmediata y la reducción de tareas de configuración manual.

El entorno principal será Google Cloud Platform (GCP) por su facilidad de integración con servicios gratuitos y su modelo de facturación flexible. No obstante, la arquitectura es portable y puede adaptarse a AWS o Azure.

Componentes y Flujo General

1. Frontend (Aplicación Web y Móvil):
   - El frontend estará desarrollado con React Native.
   - La versión web será desplegada en Firebase Hosting por su integración gratuita, facilidad de despliegue con un solo comando y soporte para HTTPS automático.
   - La aplicación móvil será compilada y distribuida en Google Play.

2. Backend (API REST Serverless):
   - El backend estará implementado con Node.js y Express, empaquetado como funciones Serverless utilizando Cloud Functions de GCP (o AWS Lambda si se usa AWS).

- Estas funciones se ejecutan bajo demanda, eliminando la necesidad de mantener servidores activos, y se escalan automáticamente según la carga.
- Cada microservicio (usuarios, reservas, pagos, notificaciones, etc.) se implementará como una función independiente, garantizando modularidad y aislamiento.

3. Base de Datos:
   - Se utilizará PostgreSQL y Firebase dependiendo de la necesidad de relaciones.
   - Ambas opciones ofrecen planes gratuitos, backups automáticos y alta disponibilidad sin configuración manual.

4. Autenticación y Seguridad:
   - Se implementará Firebase Authentication para manejar el inicio de sesión con correo y redes sociales.
   - Las funciones del backend estarán protegidas mediante tokens JWT y reglas de seguridad a nivel de Firestore o API Gateway.

5. Almacenamiento de Archivos:
   - Las imágenes o documentos se alojarán en Firebase Storage o Cloud Storage, con acceso restringido mediante reglas de seguridad y URLs firmadas.

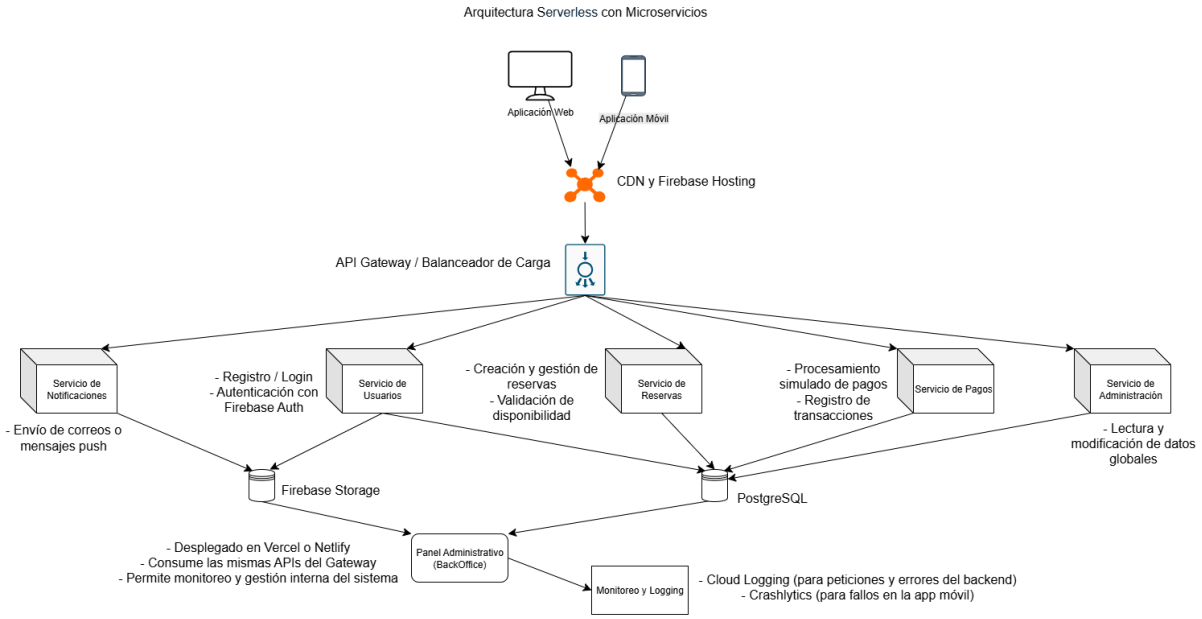
6. Balanceo de Carga:
   - Aunque las funciones serverless escalan automáticamente, se utilizará un API Gateway (por ejemplo, Google API Gateway o AWS API Gateway) para distribuir las solicitudes y definir rutas por microservicio.
   - Esto permite balancear el tráfico y mantener trazabilidad de las peticiones.

7. CDN (Content Delivery Network):
   - Firebase Hosting y Cloud Storage incluyen integración automática con CDN global, lo que optimiza la entrega de contenido estático (imágenes, JS, CSS).

8. Panel de Administración (BackOffice):
   - El panel de administración se desplegará en Vercel o Netlify, conectándose directamente a la API Serverless mediante HTTPS

seguro.

9. Monitoreo y Logs:
   - Se usará Cloud Logging y Firebase Crashlytics para recopilar métricas, errores y tiempos de respuesta, lo que permitirá ajustar el rendimiento sin intervención manual.



10. Repositorio configurado para recibir el código fuente

https://github.com/miralriodev/arroyo-seco-resolve.git

Conclusión

El desarrollo de 'Travel Connect' se apoya en herramientas modernas y metodologías de control que garantizan calidad, eficiencia y escalabilidad. El flujo de trabajo basado en Git Flow, junto con un plan de pruebas estructurado y un despliegue automatizado, asegura entregas continuas y confiables, alineadas con los objetivos de digitalización y mejora de la experiencia turística.