

Parcial 3 - Ingeniería Web 2024/25

ReViews - Aplicación Web de Reseñas de Establecimientos

Mapas, OAuth 2.0, Imágenes y Geocodificación

Nombre del Alumno

Universidad de Málaga

Escuela Técnica Superior de Ingeniería Informática

Asignatura: Ingeniería Web

Fecha: 12 de diciembre de 2025

Índice

1. URL de Despliegue en la Nube	4
1.1. URLs de Acceso	4
1.2. Consideraciones sobre Render (Plan Gratuito)	4
2. Email de Pruebas	4
3. Tecnologías Utilizadas	5
3.1. Stack Tecnológico Completo	5
3.1.1. Frontend	5
3.1.2. Backend	5
3.1.3. Base de Datos	5
3.1.4. Servicios Externos	6
3.1.5. Despliegue	6
3.2. Arquitectura del Sistema	6
3.2.1. Frontend - Clean Architecture	6
3.2.2. Backend - Monolito Modular	7
4. Instrucciones de Instalación y Despliegue	7
4.1. Requisitos Previos	7
4.2. Obtención de Credenciales	8
4.2.1. MongoDB Atlas	8
4.2.2. Cloudinary	8
4.2.3. Google OAuth	8
4.3. Configuración de Variables de Entorno	8
4.3.1. Backend (app/backend/.env)	8
4.3.2. Frontend (app/frontend/.env)	9
4.4. Ejecución en Local (con Docker)	9
4.4.1. Acceso a la Aplicación Local	10
4.5. Despliegue en la Nube	10
4.5.1. Backend en Render	10
4.5.2. Frontend en Vercel	10
5. Configuración del Proyecto	11
5.1. Variables de Entorno del Backend	11
5.2. Variables de Entorno del Frontend	11
5.3. Resumen de Credenciales	12
6. Credenciales de Acceso a la Base de Datos	12
6.1. Estructura de Documentos	12
7. Funcionalidad Implementada	13
7.1. Resumen de Requisitos	13
7.2. Identificación OAuth (2 puntos)	13
7.3. Mapas y Geocoding (2 puntos)	14
7.4. Imágenes (2 puntos)	14

7.5. Listado y Detalle de Reseñas (2 puntos)	15
7.6. Endpoints de la API	16
7.7. Características Técnicas Adicionales	16
8. Limitaciones y Problemas Encontrados	16
8.1. Limitaciones Conocidas	16
8.2. Problemas Encontrados y Soluciones	17
9. Problemas Técnicos Detallados	17
9.1. Problema SSL con MongoDB Atlas en Entorno Local (Docker)	17
9.1.1. Descripción del Problema	17
9.1.2. Causa del Problema	18
9.1.3. Soluciones Implementadas	18
9.2. Problema con el Servicio de Geocodificación Nominatim	18
9.2.1. Descripción del Problema	18
9.2.2. Causa del Problema	19
9.2.3. Soluciones Implementadas	19
9.2.4. Fallback de Coordenadas por Defecto	19
9.2.5. Proveedores de Geocodificación Utilizados	20
9.3. Recomendaciones para Producción	20
10. Conclusiones	21

1. URL de Despliegue en la Nube

La aplicación **ReViews** ha sido desplegada en servicios cloud públicos, separando frontend y backend:

1.1. URLs de Acceso

Servicio	URL
Frontend (Vercel)	<code>https://tu-app.vercel.app</code>
Backend API (Render)	<code>https://tu-backend.onrender.com</code>
Documentación Swagger	<code>https://tu-backend.onrender.com/docs</code>
Documentación ReDoc	<code>https://tu-backend.onrender.com/redoc</code>
OpenAPI JSON	<code>https://tu-backend.onrender.com/openapi.json</code>

Cuadro 1: URLs de acceso a la aplicación desplegada

Nota: Sustituir las URLs por las reales del despliegue.

1.2. Consideraciones sobre Render (Plan Gratuito)

Importante: Render, en su plan gratuito, suspende automáticamente el servicio tras 15 minutos de inactividad. Esto significa que:

- La primera petición tras el periodo de inactividad puede tardar entre 30-50 segundos en responder.
- Se recomienda acceder primero a la URL del backend para “despertar” el servicio antes de usar la aplicación.
- La aplicación incluye un banner de notificación que avisa al usuario cuando el servidor está iniciándose.

2. Email de Pruebas

Para las pruebas y validación de la aplicación se ha utilizado la siguiente cuenta de Google:

tu-email-de-pruebas@gmail.com

Nota: Sustituir por el email real utilizado para las pruebas.

Este email se utiliza para:

- Autenticación mediante Google OAuth 2.0
- Creación de reseñas de establecimientos
- Asociación de reseñas con el autor
- Verificación de permisos (solo el autor puede eliminar sus reseñas)

3. Tecnologías Utilizadas

3.1. Stack Tecnológico Completo

3.1.1. Frontend

Tecnología	Versión/Detalle
Lenguaje	TypeScript 5.0+
Framework	React 18+
Build Tool	Vite 5.x
Estilos	TailwindCSS 3.x (Glassmorphism)
Iconos	Font Awesome 6.x
Mapas	React-Leaflet 4.x + Leaflet 1.9.x
HTTP Client	Axios
OAuth	@react-oauth/google
Arquitectura	Clean Architecture

Cuadro 2: Tecnologías del Frontend

3.1.2. Backend

Tecnología	Versión/Detalle
Lenguaje	Python 3.11+
Framework Web	FastAPI 0.109+
Driver BD	Motor (MongoDB async)
Validación	Pydantic V2
Servidor ASGI	Uvicorn
OAuth	google-auth
Imágenes	Cloudinary Python SDK
Geocoding	Nominatim (OpenStreetMap)
JWT	python-jose
Arquitectura	Monolito Modular

Cuadro 3: Tecnologías del Backend

3.1.3. Base de Datos

- **Tipo:** NoSQL (Orientada a documentos)
- **Motor:** MongoDB 7.x
- **Proveedor:** MongoDB Atlas (Cloud)
- **Plan:** M0 (Free Tier - 512 MB)
- **Colección principal:** reviews

3.1.4. Servicios Externos

Servicio	Proveedor	Uso
Mapas	OpenStreetMap	Visualización de mapas
Geocoding	Nominatim + Photon + Geocode.maps.co	Conversión dirección → coordenadas
Imágenes	Cloudinary	Almacenamiento cloud de fotos
OAuth 2.0	Google Identity	Autenticación de usuarios

Cuadro 4: Servicios externos utilizados

3.1.5. Despliegue

Componente	Proveedor
Frontend	Vercel (Plan Free)
Backend	Render (Plan Free)
Base de datos	MongoDB Atlas (Plan M0 Free)
Contenedores	Docker + Docker Compose

Cuadro 5: Proveedores de despliegue

3.2. Arquitectura del Sistema

3.2.1. Frontend - Clean Architecture

El frontend sigue estrictamente los principios de Clean Architecture:

```
1 src/
2 +-- domain/           # Capa de Dominio (sin dependencias externas)
3 |   +-- models/       # Interfaces TypeScript puras
4 |   |   +-- Review.ts  # Review_Model, Review_Create_Data
5 |   +-- repositories/ # Contratos de repositorio
6 |       +-- ReviewRepository.ts
7 +-- infrastructure/   # Capa de Infraestructura
8 |   +-- api/
9 |   |   +-- axios_client.ts # Cliente HTTP configurado
10 |   +-- repositories/
11 |       +-- HttpReviewRepository.ts # Implementacion HTTP
12 +-- presentation/     # Capa de Presentacion
13 |   +-- components/   # Componentes reutilizables
14 |   |   +-- ReviewCard.tsx
15 |   |   +-- ReviewDetail.tsx
16 |   |   +-- ReviewForm.tsx
17 |   |   +-- StarRating.tsx
18 |   |   +-- MapComponent.tsx
19 |   |   +-- LoginButton.tsx
20 |   +-- pages/        # Paginas completas
21 |       +-- LoginPage.tsx
22 |       +-- ReviewsPage.tsx
23 +-- hooks/            # Custom Hooks (Use Cases)
24 |   +-- use_reviews.ts
```

```

25 | +-- context/           # Estado global
26 | | +-- AuthContext.tsx
27 | +-- router/
28 | +-- AppRouter.tsx

```

1: Estructura del Frontend

Principios aplicados:

- La capa de **Dominio** no tiene dependencias externas (no importa Axios ni React).
- Los **Componentes UI** no hacen llamadas API directamente, usan hooks.
- Los **Hooks** (Application Layer) encapsulan la lógica de negocio.
- La **Infraestructura** implementa los contratos definidos en el dominio.

3.2.2. Backend - Monolito Modular

El backend sigue una arquitectura en capas:

```

1 app/backend/
2 +-- api/v1/
3 | +-- router.py           # Router principal
4 | +-- endpoints/
5 | | +-- auth.py          # Endpoints de autenticacion
6 | | +-- reviews.py      # CRUD de resenas
7 +-- core/
8 | +-- config.py          # Configuracion (Settings)
9 | +-- database.py        # Conexion MongoDB
10 +-- models/
11 | +-- review.py         # Modelo MongoDB (ReviewModel)
12 +-- schemas/
13 | +-- auth.py           # Schemas de autenticacion
14 | +-- review.py         # Schemas de resenas (Pydantic V2)
15 | +-- common.py         # Schemas comunes
16 +-- repositories/
17 | +-- review_repository.py # Acceso a datos
18 +-- services/
19 | +-- auth_service.py    # Logica de OAuth/JWT
20 | +-- image_service.py   # Subida a Cloudinary
21 | +-- map_service.py     # Geocodificacion
22 +-- main.py              # Punto de entrada FastAPI

```

2: Estructura del Backend

4. Instrucciones de Instalación y Despliegue

4.1. Requisitos Previos

- Docker y Docker Compose instalados
- Node.js 18+ (opcional, solo si se ejecuta sin Docker)

- Python 3.11+ (opcional, solo si se ejecuta sin Docker)
- Cuenta de MongoDB Atlas configurada
- Cuenta de Cloudinary con credenciales activas
- Proyecto de Google Cloud con OAuth 2.0 configurado

4.2. Obtención de Credenciales

4.2.1. MongoDB Atlas

1. Ir a <https://www.mongodb.com/cloud/atlas>
2. Crear cuenta y clúster gratuito (M0)
3. En **Database Access**, crear usuario y contraseña
4. En **Network Access**, permitir acceso desde cualquier IP (0.0.0.0/0)
5. Copiar la Connection String desde **Database** → **Connect**

4.2.2. Cloudinary

1. Registrarse en <https://cloudinary.com>
2. En el Dashboard, copiar: Cloud Name, API Key, API Secret

4.2.3. Google OAuth

1. Ir a <https://console.cloud.google.com>
2. Crear nuevo proyecto
3. **APIs & Services** → **OAuth consent screen** → External
4. **Credentials** → **Create OAuth client ID** → Web application
5. Añadir orígenes autorizados: <http://localhost:5173>
6. Copiar Client ID y Client Secret

4.3. Configuración de Variables de Entorno

4.3.1. Backend (app/backend/.env)

```
1 # MongoDB Atlas
2 MONGO_URI=mongodb+srv://usuario:password@cluster.mongodb.net/
3 DATABASE_NAME=reviews_db
4
5 # Cloudinary
6 CLOUDINARY_CLOUD_NAME=tu_cloud_name
7 CLOUDINARY_API_KEY=tu_api_key
```



```
8 CLOUDINARY_API_SECRET=tu_api_secret
9
10 # Google OAuth
11 GOOGLE_CLIENT_ID=tu_client_id.apps.googleusercontent.com
12 GOOGLE_CLIENT_SECRET=tu_client_secret
13
14 # JWT Secret (generar uno aleatorio y seguro)
15 SECRET_KEY=
    una_clave_secreta_muy_larga_y_segura_de_al_menos_32_caracteres
```

3: Variables de entorno del Backend

4.3.2. Frontend (app/frontend/.env)

```
1 # URL del Backend (para desarrollo local)
2 VITE_API_URL=http://localhost:8000/api/v1
3
4 # Google OAuth Client ID (el mismo que en backend)
5 VITE_GOOGLE_CLIENT_ID=tu_client_id.apps.googleusercontent.com
```

4: Variables de entorno del Frontend

4.4. Ejecución en Local (con Docker)

```
1 # Clonar el repositorio
2 git clone <url-del-repositorio>
3 cd iweb-p2
4
5 # Crear archivos .env en app/backend/ y app/frontend/
6 # (copiar las plantillas y rellenar con credenciales)
7
8 # Construir y ejecutar con Docker Compose
9 docker-compose up --build
10
11 # O ejecutar en segundo plano
12 docker-compose up -d --build
13
14 # Ver logs en tiempo real
15 docker-compose logs -f
16
17 # Detener los servicios
18 docker-compose down
```

5: Comandos para ejecución local

4.4.1. Acceso a la Aplicación Local

Servicio	URL
Frontend	<code>http://localhost:5173</code>
Backend API	<code>http://localhost:8000</code>
Swagger UI	<code>http://localhost:8000/docs</code>
ReDoc	<code>http://localhost:8000/redoc</code>

Cuadro 6: URLs de acceso local

4.5. Despliegue en la Nube

4.5.1. Backend en Render

1. Crear cuenta en `https://render.com`
2. Crear nuevo **Web Service** desde repositorio Git
3. Configurar:
 - **Root Directory:** `app/backend`
 - **Runtime:** Docker
4. Añadir **Environment Variables**:
 - `MONGO_URI`
 - `DATABASE_NAME`
 - `CLOUDINARY_CLOUD_NAME`
 - `CLOUDINARY_API_KEY`
 - `CLOUDINARY_API_SECRET`
 - `GOOGLE_CLIENT_ID`
 - `GOOGLE_CLIENT_SECRET`
 - `SECRET_KEY`

5. Deploy automático al hacer push a main

4.5.2. Frontend en Vercel

1. Crear cuenta en `https://vercel.com`
2. Importar proyecto desde repositorio Git
3. Configurar:
 - **Root Directory:** `app/frontend`
 - **Framework Preset:** Vite

4. Añadir **Environment Variables**:

- **VITE_API_URL**: URL del backend en Render (ej: `https://tu-app.onrender.com/api/v1`)
- **VITE_GOOGLE_CLIENT_ID**: Client ID de Google

5. **Importante**: Actualizar Google Cloud Console añadiendo el dominio de Vercel a:

- Authorized JavaScript origins
- Authorized redirect URIs

5. Configuración del Proyecto

A continuación se detallan las variables de entorno utilizadas en el proyecto para su correcto funcionamiento.

5.1. Variables de Entorno del Backend

El archivo `app/backend/.env` contiene la siguiente configuración:

```
1 # Configuración del Servidor
2 MONGO_URI=mongodb+srv://eduardo:eduardo@cluster0.n63md1g.mongodb.net/?
  appName=Cluster0
3 DATABASE_NAME=plantilla
4 API_NAME=IWEB Exam API
5
6 # Cloudinary (Imágenes)
7 CLOUDINARY_CLOUD_NAME=dhvhzdfwh
8 CLOUDINARY_API_KEY=682229474789636
9 CLOUDINARY_API_SECRET=4XDfe2qeDc63g-w6sh870Y4v96U
10
11 # Google OAuth (Backend verifica el token)
12 GOOGLE_CLIENT_ID=1024296056000-rk7006r9ch3fjn85ou47tebu0fla9f8q.apps.
  googleusercontent.com
13 GOOGLE_CLIENT_SECRET=GOCS PX-RUaPKAGgH5ElipTDc9LMiwNkG8Q5
14
15 # JWT
16 SECRET_KEY=secret_key_placeholder_change_me
17
18 # Desarrollo local
19 ALLOWED_ORIGINS=http://localhost:5173
```

6: Configuración del Backend (.env)

5.2. Variables de Entorno del Frontend

El archivo `app/frontend/.env` contiene la siguiente configuración:

```
1 # URL del Backend (Para Axios)
2 VITE_API_URL=http://localhost:8000/api/v1
3
4 # Google OAuth (Para el botón de login en React)
```

```
5 VITE_GOOGLE_CLIENT_ID=1024296056000-rk7006r9ch3fjn85ou47tebu0fla9f8q.  
apps.googleusercontent.com
```

7: Configuración del Frontend (.env)

5.3. Resumen de Credenciales

Servicio	Parámetro	Valor
MongoDB Atlas	Usuario	eduardo
MongoDB Atlas	Clúster	cluster0.n63md1g.mongodb.net
MongoDB Atlas	Base de datos	plantilla
Cloudinary	Cloud Name	dhvhzdfwh
Cloudinary	API Key	682229474789636
Google OAuth	Client ID	1024296056000-rk7006r9ch3fjn85ou47tebu0fla9f8q

Cuadro 7: Resumen de credenciales de servicios

6. Credenciales de Acceso a la Base de Datos

Para verificar el contenido de la base de datos MongoDB Atlas:

Parámetro	Valor
Proveedor	MongoDB Atlas
Clúster	cluster0
Base de datos	reviews_db
Colección principal	reviews
URI de conexión	mongodb+srv://...
Usuario de solo lectura	readonly_user
Contraseña	*****

Cuadro 8: Credenciales de MongoDB Atlas

Nota: Se recomienda crear un usuario de solo lectura específico para el corrector.

6.1. Estructura de Documentos

```
1 {  
2   "_id": ObjectId("..."),  
3   "establishment_name": "Casa Lola",  
4   "address": "Calle Granada 46, Málaga, España",  
5   "latitude": 36.7213028,  
6   "longitude": -4.4214089,  
7   "rating": 4,  
8   "image_urls": [  
9     "https://res.cloudinary.com/xxx/image/upload/v123/abc.jpg",  
10    "https://res.cloudinary.com/xxx/image/upload/v123/def.jpg"]  
}
```

```

11   ],
12   "author_email": "usuario@gmail.com",
13   "author_name": "Juan Garcia",
14   "auth_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9... ",
15   "created_at": ISODate("2024-12-12T10:30:00.000Z"),
16   "expires_at": ISODate("2024-12-13T10:30:00.000Z")
17 }

```

8: Estructura de documento Review en MongoDB

7. Funcionalidad Implementada

7.1. Resumen de Requisitos

Requisito	Pts	Estado
Identificación OAuth	2	✓ Login/Logout con Google OAuth 2.0
Mapas y Geocoding	2	✓ Mapa con marcadores + búsqueda + geocodificación automática
Imágenes	2	✓ Subida múltiple a Cloudinary (hasta 5 por reseña)
Listado/Detalle	2	✓ Lista de reseñas con toda la información requerida
Despliegue y Entrega	2	✓ Vercel + Render + Memoria técnica
Total	10	Todos los requisitos implementados

Cuadro 9: Estado de implementación de requisitos

7.2. Identificación OAuth (2 puntos)

Funcionalidades implementadas:

- **Login con Google:** Botón de “Sign in with Google” usando `@react-oauth/google`.
- **Verificación de token:** El backend verifica el token de Google y genera un JWT de sesión.
- **Logout:** Limpieza de tokens del `localStorage` y estado de React Context.
- **Persistencia de sesión:** El token JWT se almacena en `localStorage` y se restaura al recargar.
- **Protección de rutas:** Las rutas protegidas redirigen a login si no hay sesión.
- **Header de usuario:** Muestra foto y nombre del usuario autenticado.

Archivos principales:

- `frontend/src/presentation/components/LoginButton.tsx`
- `frontend/src/presentation/context/AuthContext.tsx`
- `backend/services/auth_service.py`
- `backend/api/v1/endpoints/auth.py`

7.3. Mapas y Geocoding (2 puntos)

Funcionalidades implementadas:

- **Mapa interactivo:** Visualización con React-Leaflet y tiles de OpenStreetMap.
- **Marcadores de reseñas:** Cada reseña se muestra como marcador con popup informativo.
- **Popups:** Muestran imagen, nombre, valoración y botón “Ver detalle”.
- **Barra de búsqueda:** Permite buscar direcciones y centrar el mapa en ellas.
- **Geocodificación automática:** Al crear reseña, la dirección se convierte a coordenadas GPS usando Nominatim.
- **Toggle vistas:** Botones para cambiar entre vista lista y vista mapa.

Archivos principales:

- `frontend/src/presentation/components/MapComponent.tsx`
- `backend/services/map_service.py`
- `backend/api/v1/endpoints/reviews.py` (endpoint POST `/geocode`)

7.4. Imágenes (2 puntos)

Funcionalidades implementadas:

- **Subida múltiple:** Se pueden subir hasta 5 imágenes por reseña.
- **Almacenamiento cloud:** Las imágenes se suben a Cloudinary, almacenando las URLs en MongoDB.
- **Formatos soportados:** JPEG, PNG, WebP.
- **Previsualización:** Las imágenes seleccionadas se muestran antes de enviar.
- **Eliminación:** Botón para quitar imágenes de la selección.
- **Galería navegable:** En el detalle, galería con botones anterior/siguiente y dots de navegación.

- **Indicador de cantidad:** Badge mostrando número de imágenes adicionales en las tarjetas.

Archivos principales:

- `frontend/src/presentation/components/ReviewForm.tsx`
- `frontend/src/presentation/components/ReviewDetail.tsx`
- `backend/services/image_service.py`

7.5. Listado y Detalle de Reseñas (2 puntos)

Funcionalidades implementadas:

- **Vista de lista:** Grid responsive de tarjetas (ReviewCard) con:
 - Imagen principal del establecimiento
 - Nombre del establecimiento
 - Valoración con estrellas (0-5)
 - Dirección postal
 - Coordenadas GPS (latitud, longitud)
 - Nombre/email del autor
 - Fecha de creación
- **Vista de detalle** (modal ReviewDetail) con información adicional:
 - Galería de todas las imágenes
 - Email completo del autor
 - Nombre del autor
 - Fecha y hora de creación
 - Fecha y hora de caducidad del token
 - Token OAuth utilizado (completo)
 - Botón de eliminación (solo visible para el autor)
- **Creación de reseñas:** Formulario modal con validación.
- **Eliminación:** Solo el autor puede eliminar sus propias reseñas (con confirmación).

Archivos principales:

- `frontend/src/presentation/pages/ReviewsPage.tsx`
- `frontend/src/presentation/components/ReviewCard.tsx`
- `frontend/src/presentation/components/ReviewDetail.tsx`
- `frontend/src/presentation/components/ReviewForm.tsx`
- `frontend/src/presentation/components/StarRating.tsx`

7.6. Endpoints de la API

Método	Endpoint	Descripción
POST	/api/v1/auth/login	Login con Google OAuth
GET	/api/v1/reviews	Listar todas las reseñas
GET	/api/v1/reviews/{id}	Obtener detalle completo de reseña
POST	/api/v1/reviews	Crear nueva reseña (multipart/form-data)
DELETE	/api/v1/reviews/{id}	Eliminar reseña (solo autor)
POST	/api/v1/reviews/geocode	Geocodificar dirección postal

Cuadro 10: Endpoints de la API REST

7.7. Características Técnicas Adicionales

- **Diseño Glassmorphism:** Interfaz moderna con efectos de cristal, blur y sombras coloreadas.
- **Responsive:** Adaptado a móvil, tablet y escritorio usando TailwindCSS.
- **Accesibilidad:** ARIA labels, navegación por teclado, contraste adecuado.
- **Documentación OpenAPI:** Swagger UI y ReDoc generados automáticamente.
- **Convenciones de código:** snake_case para variables/funciones.
- **Documentación:** JSDoc en frontend, docstrings en backend (español).
- **Python moderno:** Sintaxis 3.11+ (`str | None`, `list[str]`).
- **Pydantic V2:** ConfigDict, json_schema_extra para ejemplos.

8. Limitaciones y Problemas Encontrados

8.1. Limitaciones Conocidas

1. **Plan gratuito de Render:** El backend se suspende tras 15 minutos de inactividad, requiriendo 30-50 segundos para “despertar”.
2. **Geocodificación (Nominatim):** API gratuita con límites de tasa. No todas las direcciones se encuentran (depende de la precisión del texto).
3. **MongoDB Atlas M0:** Límite de 512 MB de almacenamiento.
4. **Cloudinary Free:** Límite de 25 créditos/mes. Imágenes muy grandes pueden tardar.
5. **Token JWT:** Expira en 24 horas. No se implementó refresh token automático.
6. **Imágenes:** Máximo 5 por reseña para no saturar Cloudinary.

8.2. Problemas Encontrados y Soluciones

1. Conexión SSL con MongoDB Atlas:

- *Problema:* Errores intermitentes SSL: UNEXPECTED_EOF_WHILE_READING.
- *Causa:* Configuración de red/firewall, timeout de conexiones idle.
- *Solución:* Configuración correcta de TLS en Motor y timeouts adecuados.

2. CORS en producción:

- *Problema:* El frontend en Vercel no podía comunicarse con el backend en Render.
- *Solución:* Configurar correctamente los orígenes permitidos en FastAPI CORS-Middleware.

3. Iconos de Leaflet con Vite:

- *Problema:* Los marcadores del mapa no mostraban iconos.
- *Solución:* Importar manualmente las imágenes de iconos y configurar `L.Marker.prototype.options.iconUrl`.

4. TypeScript Strict Mode:

- *Problema:* Error TS6196: 'X' is declared but never used.
- *Solución:* Eliminar imports no utilizados para pasar el build de producción.

5. Google OAuth en producción:

- *Problema:* Error de origen no autorizado al desplegar.
- *Solución:* Añadir dominios de Vercel a Google Cloud Console (origins y redirects).

9. Problemas Técnicos Detallados

En esta sección se documentan los problemas técnicos más significativos encontrados durante el desarrollo y despliegue de la aplicación.

9.1. Problema SSL con MongoDB Atlas en Entorno Local (Docker)

9.1.1. Descripción del Problema

Al ejecutar la aplicación en contenedores Docker en entorno local, se producen errores intermitentes de conexión SSL con MongoDB Atlas:

```
1 pymongo.errors.ServerSelectionTimeoutError:
2 SSL handshake failed: ac-nrmtowl-shard-00-01.n63mdl.mongodb.net:27017:
3 [SSL: UNEXPECTED_EOF_WHILE_READING] EOF occurred in violation of
  protocol
4 (_ssl.c:1016) (configured timeouts: socketTimeoutMS: 20000.0ms,
```

```
5 connectTimeoutMS: 20000.0ms)
```

9: Error SSL en MongoDB Atlas

9.1.2. Causa del Problema

Este error se produce por una combinación de factores:

1. **Configuración de red del contenedor:** Docker utiliza una capa de red virtualizada que puede interferir con las conexiones TLS de larga duración.
2. **Firewall/Antivirus:** Algunos antivirus o firewalls (como Windows Defender) inspeccionan el tráfico SSL, causando interrupciones.
3. **Timeout de conexiones idle:** MongoDB Atlas cierra conexiones inactivas, y si el driver no maneja bien la reconexión, se produce el error.
4. **Versión de OpenSSL en el contenedor:** Incompatibilidades entre la versión de OpenSSL del contenedor y el servidor de MongoDB Atlas.

9.1.3. Soluciones Implementadas

1. **Configuración de TLS en la URI de conexión:**

```
1 MONGO_URI=mongodb+srv://user:pass@cluster.mongodb.net/  
2   ?retryWrites=true&w=majority&tls=true  
3   &tlsAllowInvalidCertificates=false  
4
```

2. **Configuración de timeouts en Motor:**

```
1 client = AsyncIOMotorClient(  
2   settings.MONGO_URI,  
3   serverSelectionTimeoutMS=30000,  
4   connectTimeoutMS=20000,  
5   socketTimeoutMS=20000,  
6   maxPoolSize=10,  
7   minPoolSize=1  
8 )  
9
```

3. **Nota importante:** Este problema **no ocurre en producción** (Render), solo en el entorno local con Docker. En Render, la conexión a MongoDB Atlas funciona correctamente sin modificaciones adicionales.

9.2. Problema con el Servicio de Geocodificación Nominatim

9.2.1. Descripción del Problema

Al intentar geocodificar direcciones desde el backend desplegado en Render, el servicio Nominatim (OpenStreetMap) falla constantemente con errores de conexión:

```
1 [Geocoding] Nominatim connection error:  
2 All connection attempts failed
```

10: Error de conexión con Nominatim

9.2.2. Causa del Problema

1. **Bloqueo por IP de datacenter:** Nominatim tiene políticas estrictas de uso que pueden bloquear peticiones desde IPs de servicios cloud conocidos (AWS, Google Cloud, Render, etc.).
2. **Rate limiting:** Nominatim limita a 1 petición por segundo y requiere un User-Agent válido.
3. **Política de uso aceptable:** El servicio gratuito de Nominatim está pensado para uso ocasional, no para aplicaciones en producción con alto tráfico.

9.2.3. Soluciones Implementadas

Se implementó un sistema de **geocodificación en cascada** con múltiples proveedores de fallback:

```
1 class GeocodingService:  
2     # Servicios en orden de prioridad  
3     NOMINATIM_URL = "https://nominatim.openstreetmap.org/search"  
4     PHOTON_URL = "https://photon.komoot.io/api/"  
5     GEOCODE_MAPS_URL = "https://geocode.maps.co/search"  
6  
7     async def get_coordinates(self, address: str):  
8         # 1. Intentar Nominatim (OpenStreetMap oficial)  
9         result = await self._try_nominatim(address)  
10        if result: return result  
11  
12        # 2. Intentar Photon (Komoot, basado en OSM)  
13        result = await self._try_photon(address)  
14        if result: return result  
15  
16        # 3. Intentar Geocode.maps.co (alternativa gratuita)  
17        result = await self._try_geocode_maps(address)  
18        if result: return result  
19  
20        # 4. Si todo falla, retornar None  
21        return None
```

11: Sistema de geocodificación con fallbacks

9.2.4. Fallback de Coordenadas por Defecto

Como último recurso, si todos los servicios de geocodificación fallan, se asignan coordenadas por defecto (Málaga) y se notifica al usuario:

```

1 DEFAULT_LATITUDE = 36.7213028 # Malaga, Espana
2 DEFAULT_LONGITUDE = -4.4216366
3 GEOCODING_WARNING = "No se pudo geocodificar la direccion.
4                       Se han asignado coordenadas por defecto (Malaga)."
```

```

5
6 @router.post("/geocode")
7 async def geocode_address(address: str = Form(...)):
8     coords = await geocoding_service.get_coordinates(address)
9
10    if coords:
11        return {"latitude": coords[0], "longitude": coords[1]}
12    else:
13        # Fallback: coordenadas de Malaga con advertencia
14        return {
15            "latitude": DEFAULT_LATITUDE,
16            "longitude": DEFAULT_LONGITUDE,
17            "warning": GEOCODING_WARNING,
18            "is_default": True
19        }
```

12: Coordenadas por defecto como fallback

9.2.5. Proveedores de Geocodificación Utilizados

Prioridad	Servicio	URL Base	Limitaciones
1	Nominatim	nominatim.openstreetmap.org	1 req/s, bloqueo IPs cloud
2	Photon	photon.komoot.io	Sin límite documentado
3	Geocode.maps.co	geocode.maps.co	1 req/s, sin API key
4	Default	–	Málaga (36.72, -4.42)

Cuadro 11: Proveedores de geocodificación en orden de prioridad

9.3. Recomendaciones para Producción

Para un entorno de producción real, se recomienda:

1. **Usar un servicio de geocodificación de pago:** Google Maps Geocoding API, Mapbox, HERE, etc.
2. **Implementar caché de geocodificación:** Almacenar resultados en Redis o MongoDB para evitar peticiones repetidas.
3. **Usar MongoDB Atlas con plan de pago:** Mejor rendimiento y sin las limitaciones del tier gratuito.
4. **Configurar un proxy o VPN:** Para evitar bloqueos de IP en servicios gratuitos.

10. Conclusiones

Se ha desarrollado satisfactoriamente **ReViews**, una aplicación web completa de reseñas de establecimientos que cumple con todos los requisitos especificados en el enunciado del parcial:

- ✓ **Autenticación OAuth 2.0** con Google Identity Platform
- ✓ **Mapas interactivos** con OpenStreetMap y React-Leaflet
- ✓ **Geocodificación automática** de direcciones postales
- ✓ **Subida múltiple de imágenes** a Cloudinary
- ✓ **CRUD completo** de reseñas con validación
- ✓ **Despliegue cloud** en Vercel y Render
- ✓ **Arquitectura limpia** y código documentado
- ✓ **Diseño responsive** y accesible

La aplicación es funcional, estéticamente moderna (estilo Glassmorphism), y sigue las mejores prácticas de desarrollo web contemporáneo, incluyendo Clean Architecture en el frontend y arquitectura modular en el backend.