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

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

1.1. URLs de Acceso

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

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 API	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:

```
src/
          domain/              # Capa de Dominio (sin dependencias externas)
                models/          # Interfaces TypeScript puras
                      Review.ts    # Review_Model, Review_Create_Data
                repositories/    # Contratos de repositorio
                    ReviewRepository.ts
          infrastructure/      # Capa de Infraestructura
                api/
                      axios_client.ts  # Cliente HTTP configurado
                repositories/
                    HttpReviewRepository.ts  # Implementacion HTTP
          presentation/        # Capa de Presentacion
                components/      # Componentes reutilizables
                      ReviewCard.tsx
                      ReviewDetail.tsx
                      ReviewForm.tsx
                      StarRating.tsx
                      MapComponent.tsx
                      LoginButton.tsx
                pages/           # Paginas completas
                      LoginPage.tsx
                      ReviewsPage.tsx
                hooks/           # Custom Hooks (Use Cases)
```

```
                    use_reviews.ts
                context/          # Estado global
                    AuthContext.tsx
                router/
                    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:

```
app/backend/
        api/v1/
            router.py            # Router principal
            endpoints/
                auth.py          # Endpoints de autenticacion
                reviews.py       # CRUD de resenas
        core/
            config.py            # Configuracion (Settings)
            database.py          # Conexion MongoDB
        models/
            review.py            # Modelo MongoDB (ReviewModel)
        schemas/
            auth.py              # Schemas de autenticacion
            review.py            # Schemas de resenas (Pydantic V2)
            common.py            # Schemas comunes
        repositories/
            review_repository.py # Acceso a datos
        services/
            auth_service.py      # Logica de OAuth/JWT
            image_service.py     # Subida a Cloudinary
            map_service.py       # Geocodificacion
        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`)

```
# MongoDB Atlas
MONGO_URI=mongodb+srv://usuario:password@cluster.mongodb.net/
DATABASE_NAME=reviews_db

# Cloudinary
CLOUDINARY_CLOUD_NAME=tu_cloud_name
CLOUDINARY_API_KEY=tu_api_key
```

```
CLOUDINARY_API_SECRET=tu_api_secret

# Google OAuth
GOOGLE_CLIENT_ID=tu_client_id.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=tu_client_secret

# JWT Secret (generar uno aleatorio y seguro)
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)

```
# URL del Backend (para desarrollo local)
VITE_API_URL=http://localhost:8000/api/v1

# Google OAuth Client ID (el mismo que en backend)
VITE_GOOGLE_CLIENT_ID=tu_client_id.apps.googleusercontent.com
```

4: Variables de entorno del Frontend

4.4. Ejecución en Local (con Docker)

```
# Clonar el repositorio
git clone <url-del-repositorio>
cd iweb-p2

# Crear archivos .env en app/backend/ y app/frontend/
# (copiar las plantillas y rellenar con credenciales)

# Construir y ejecutar con Docker Compose
docker-compose up --build

# O ejecutar en segundo plano
docker-compose up -d --build

# Ver logs en tiempo real
docker-compose logs -f

# Detener los servicios
docker-compose down
```

5: Comandos para ejecución local

4.4.1. Acceso a la Aplicación Local

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

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. 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 7: Credenciales de MongoDB Atlas

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

5.1. Estructura de Documentos

```
{
    "_id": ObjectId("..."),
    "establishment_name": "Casa Lola",
    "address": "Calle Granada 46, Malaga, Espana",
    "latitude": 36.7213028,
    "longitude": -4.4214089,
    "rating": 4,
    "image_urls": [
        "https://res.cloudinary.com/xxx/image/upload/v123/abc.jpg",
        "https://res.cloudinary.com/xxx/image/upload/v123/def.jpg"
    ],
    "author_email": "usuario@gmail.com",
    "author_name": "Juan Garcia",
    "auth_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "created_at": ISODate("2024-12-12T10:30:00.000Z"),
    "expires_at": ISODate("2024-12-13T10:30:00.000Z")
}
```

6: Estructura de documento Review en MongoDB

6. Funcionalidad Implementada

6.1. 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 8: Estado de implementación de requisitos

6.2. 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`

6.3. 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`)

6.4. 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`

6.5. 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`

6.6. 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 9: Endpoints de la API REST

6.7. 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.

7. Limitaciones y Problemas Encontrados

7.1. 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.

7.2. 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 CORSMiddleware.
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.o`
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).

8. 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.