MEMORIA TÉCNICA

Arquitectura, Seguridad y Despliegue

Proyecto Final de Grado - Desarrollo de Aplicaciones Web

Autor: Sergio Peral López

1. Resumen Ejecutivo del Proyecto

Casafinder representa la culminación de un proceso de ingeniería de software enfocado en la robustez, escalabilidad y experiencia de usuario. Es una plataforma B2C (Business to Consumer) que facilita la intermediación inmobiliaria mediante una arquitectura moderna, segura y eficiente. El sistema ha sido diseñado siguiendo los principios SOLID y patrones de diseño empresariales.

2. Pila Tecnológica (Tech Stack)

La elección de tecnologías responde a la necesidad de un ecosistema JavaScript unificado (Full Stack JS) que permita alta concurrencia y facilidad de mantenimiento.

2.1. Backend: Node.js & Express Ecosystem

- Runtime: Node.js v18+ por su modelo de I/O no bloqueante, ideal para aplicaciones con alta carga de lectura (I/O bound).
- Framework: Express.js v4.18 por su minimalismo y sistema de middleware robusto.
- Motor de Plantillas: Pug (anteriormente Jade) para renderizado SSR en vistas específicas.
- Utilidades: bcrypt (hashing), jsonwebtoken (auth), pdfkit (generación de documentos), speakeasy (criptografía 2FA).

2.2. Frontend: Vanilla JavaScript Moderno

Se ha optado por NO utilizar frameworks reactivos (React/Vue) para demostrar un dominio profundo del lenguaje y reducir el bundle size. El frontend utiliza ES6 Modules, Fetch API para comunicación asíncrona y manipulación directa del DOM optimizada.

2.3. Persistencia: MySQL & Sequelize ORM

Base de datos relacional MySQL 8.0 para garantizar integridad referencial (ACID). Sequelize se utiliza como capa de abstracción para prevenir inyecciones SQL y facilitar la migración de esquemas.

3. Arquitectura del Software

El proyecto implementa el patrón MVC (Modelo-Vista-Controlador) de forma estricta para separar responsabilidades.

3.1. Controladores (Controllers)

Encapsulan la lógica de negocio. Ejemplos clave:

- controladorAuth.js: Orquesta el flujo de autenticación, incluyendo la máquina de estados del login (verificación pass -> verificación 2FA -> emisión token).
- controladorPropiedades.js: Gestiona el ciclo de vida de los inmuebles (CRUD) y la lógica de búsqueda con operadores SQL (LIKE).
- controladorMensajes.js: Implementa la lógica de comunicación, validando permisos (un usuario no puede auto-escribirse).

3.2. Modelos de Datos (Models)

• Usuario: Define atributos críticos como "secret_2fa" (Clave TOTP) y "token_confirmacion". Incluye Hooks (beforeCreate) para el hashing automático de contraseñas.

• Propiedad: Relación 1:N con Usuarios. Almacena metadatos y geolocalización.

• Mensaje/Consulta: Relación compleja que vincula Usuario (Emisor), Propiedad (Contexto) y Usuario (Receptor).

4. Seguridad y Criptografía

La seguridad ha sido una prioridad desde la fase de diseño (Security by Design).

4.1. Autenticación Stateless (JWT)

El sistema no mantiene sesiones en memoria. Cada petición es autenticada independientemente mediante un JSON Web Token firmado con algoritmo HMAC SHA256. Esto permite escalar horizontalmente el backend sin problemas de afinidad de sesión.

4.2. Implementación de 2FA (RFC 6238)

```
Algoritmo: TOTP (Time-based One-Time Password)
Ventana de Tiempo: 30 segundos
Longitud: 6 dígitos
Codificación Secreto: Base32
```

El flujo 2FA verifica dos factores: "Algo que sabes" (Contraseña) y "Algo que tienes" (Dispositivo móvil con semilla criptográfica).

4.3. Protección de Datos

• Contraseñas: Hasheadas con bcrypt (Salt rounds = 10).

• CSRF/XSS: Sanitización de inputs mediante express-validator para prevenir inyecciones.

• Cabeceras HTTP: Configuración de seguridad básica.

5. API RESTful y Endpoints

La API expone recursos mediante verbos HTTP semánticos:

Auth

```
POST /auth/login
POST /auth/registro
POST /auth/verify-2fa
```

Propiedades

```
GET /propiedades (Listado público)
POST /propiedades/crear (Protegido)
GET /propiedades/pdf/:id (Generación binaria)
```

Mensajería

```
GET /mensajes (Inbox)
```

```
POST /mensajes/enviar
```

6. Infraestructura y Despliegue

6.1. Contenerización (Docker)

El proyecto incluye un docker-compose.yml que orquesta los servicios necesarios (App Node.js + Base de datos MySQL), garantizando paridad entre los entornos de desarrollo y producción.

6.2. Servidor Web (Nginx)

Configurado como Reverse Proxy para gestionar las conexiones entrantes, servir archivos estáticos (caching) y redirigir el tráfico de API al servicio de Node.js.

6.3. Cloud (AWS)

Desplegado en instancia EC2. Estrategia de "Always-on" mediante gestores de procesos (PM2) para garantizar disponibilidad 99.9%.