Conexión Mongo : mongodb+srv://admin:admin @frontend-cluster.imsr6.mongodb.net/?retryWrites=true&w=majority&appName=frontend-cluster

Usuario y contraseña: admin

Para cada cambio en vercel

Frontend: Npm run build

Luego en root: vercel –prod

Memoria del examen de Frontend de Ingeniería Web

Mario Merino Zapata.

1. Diseño de la base de datos

He optado por un diseño de base de datos sencillo, basado en gran parte en la estructura proporcionada en el enunciado del ejercicio. La base de datos está compuesta por una entidad principal: **Entidad**. Por lo que así queda el diseño de la base de datos:

- **Entidad** (Entidad)
  - email (String)
  - nombre (String)
  - foto (String)
  - latitud (String)
  - longitud (String)
  - zoom (String)

2. Acceso a la base de datos

La base de datos utilizada en el proyecto es MongoDB Atlas, una solución en la nube que ofrece alta disponibilidad y seguridad. La conexión a la base de datos se realiza mediante el siguiente connection string, que puede ser configurado en el entorno del proyecto:

[enlace al proyecto].

Este connection string permite gestionar la base de datos de forma remota y segura, manteniendo los datos centralizados en un entorno escalable.

3. Tecnologías usadas en el proyecto

- **Lenguaje de programación: JavaScript**
    - Proyecto desarrollado en JavaScript para manejar la lógica de servidor de manera rápida y eficiente.
- **Entorno de ejecución (runtime) de JavaScript: Node.js**
    - Node.js permite ejecutar JavaScript en el servidor, ofreciendo herramientas para trabajar con archivos, red, y gestionar peticiones HTTP.
- **Framework backend: Express.js**
    - Express.js es un framework para construir APIs RESTful y aplicaciones web, facilitando el manejo de rutas, peticiones y middlewares.
- **Base de datos: MongoDB y**
    - MongoDB es una base de datos NoSQL orientada a documentos en formato JSON, y Mongoose facilita el modelado de datos, validaciones y esquemas. Además, **MongoDB Atlas** ofrece una solución de base de datos en la nube totalmente administrada, permitiendo un despliegue, escalado y monitoreo eficiente de las bases de datos en entornos distribuidos.
- **Contenerización / Gestión de Entorno: Docker**
    - Docker y Docker Compose aseguran un entorno de desarrollo consistente y gestionan servicios en contenedores interconectados.
- **Herramientas de Desarrollo: Nodemon y dotenv**
    - Nodemon reinicia automáticamente el servidor en cada cambio, y dotenv permite manejar variables de entorno de forma segura.
- **Framework frontend: Vue.js**
    - Vue.js es un framework progresivo para construir interfaces de usuario dinámicas y aplicaciones web de una forma reactiva y modular.
- **Mapas: Open Street Maps**
    - OpenStreetMap es una plataforma de mapas de código abierto que proporciona datos geográficos accesibles para integrar mapas y servicios de localización.
- **Fotos: Cloudinary**
    - Cloudinary es una solución para gestionar y optimizar imágenes y videos, permitiendo su almacenamiento, transformación y entrega eficiente.
- **Despliegue: Vercel**

    - Vercel es una plataforma que permite desplegar aplicaciones web de forma rápida y sencilla, integrando flujos de trabajo continuos y ofreciendo un rendimiento óptimo.

- **Estilos: TailwindCSS**
    - TailwindCSS es un framework de utilidades para CSS que permite construir diseños modernos y personalizados directamente en el HTML, acelerando el desarrollo frontend.
- **Peticiones HTTP: Axios**
    - Axios es una librería de JavaScript que facilita el manejo de peticiones HTTP, permitiendo interactuar con APIs de forma eficiente y estructurada.
- **Seguridad: CORS**
    - CORS (Cross-Origin Resource Sharing) es una política de seguridad implementada en el servidor que controla cómo los recursos son compartidos entre dominios diferentes, asegurando el acceso adecuado a las APIs.

4. Instrucciones de instalación y despliegue

El proyecto está desplegado en Vercel, lo que asegura un despliegue rápido y confiable con integración continua. Puedes acceder al proyecto directamente a través del siguiente enlace:

[enlace al proyecto]

Esto permite visualizar y probar la aplicación en un entorno en línea, sin necesidad de instalar dependencias o configurar el entorno local.

No es necesario para su consulta, basta con abrir el enlace que aparece arriba, pero para realizar el despliegue en **Vercel** desde el entorno local, sigue estos pasos:

1. Instala la herramienta de línea de comandos de Vercel (CLI).

npm i -g vercel

2. Inicia sesión en tu cuenta de Vercel mediante el comando correspondiente.

vercel login

3. Entra en el directorio de **frontend** y construye el proyecto ejecutando el comando de build.

npm run build

4. Desde el directorio raíz del proyecto, realiza el despliegue en producción con el comando adecuado.

vercel --prod

Con estos pasos, la aplicación estará desplegada en producción y accesible a través del enlace proporcionado por Vercel.

5. Diseño del frontend

Para el diseño del frontend he optado por una solución usando vue y por tanto javascript mi programa será sencillo emtiendo marcadores pero funcional.

6. Limitaciones y problemas de la solución

El principal problema que he encontrado al resolver el ejercicio ha sido el tiempo. He tenido que ir lo más rápido posible escribiendo el código, completando la memoria, pensando en como realizar los endpoints más complejos... Me ha resultado un gran reto manejar todos los aspectos del proyecto simultáneamente. Los principales problemas que he encontrado a causa del tiempo son:

- Para el despliegue he encontrado muchos problemas a la hora de implementarlo con vue. Al final lo he solucionado, pero hay que buildear el frontend manualmente.
- No me ha dado tiempo a aprender a usar OAuth ya que no hemos tenido tiempo con las entregas.
- Otro problema que he tenido es que conseguí que el proyecto hiciera los marcadores con imágenes usando cloudinary pero por algún motivo no se guardaban correctamente en la base de datos, igualmente presentaré el código hecho de HomeView.vue que hace lo que digo :

  <script setup>

  import { ref, onMounted } from 'vue';

  import L from 'leaflet'; // Importar Leaflet

  import 'leaflet/dist/leaflet.css';

  import { getUserMarkers, addMarker } from '@/services/uploadService';

  import { uploadFileToCloudinary } from '@/services/uploadService';

  const email = 'user@example.com'; // Email del usuario autenticado

  const markers = ref([]);

  const map = ref(null);

  const selectedImage = ref(null); // Imagen seleccionada

  const imageSrc = ref('');

  const address = ref('');

```
const markerInstance = ref(null);


// Inicializa el mapa y carga los marcadores del usuario
onMounted(async () => {
 map.value = L.map('map').setView([34, -43], 5);
 L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: '© OpenStreetMap contributors',
 }).addTo(map.value);


 const userMarkers = await getUserMarkers(email);
 userMarkers.forEach((marker) => {
  addMapMarker(marker.location.lat, marker.location.lon, marker.location.city ||
marker.location.country, marker.imageUrl);
 });
});


// Añadir un marcador al mapa
const addMapMarker = (lat, lon, popupText, imageUrl) => {
 const popupContent = imageUrl
  ? `<p>${popupText}</p><img src="${imageUrl}" alt="Location Image"
style="width: 100px; height: auto;" />`
  : `<p>${popupText}</p>`;


 L.marker([lat, lon]).addTo(map.value).bindPopup(popupContent).openPopup();
};


// Manejo de subida de archivos
const handleFileUpload = async (event) => {
 const file = event.target.files[0];
 if (file) {
  try {
   const imageUrl = await uploadFileToCloudinary(file);
```

```
    selectedImage.value = imageUrl;
    imageSrc.value = imageUrl;
    console.log('Archivo subido exitosamente:', imageUrl);
   } catch (error) {
    console.error('Error al subir la imagen:', error);
    alert('Error uploading the image. Please try again.');
   }
  }
};

// Añadir un marcador nuevo con imagen y ubicación
const addNewMarker = async () => {
 if (!address.value.trim()) {
  alert('Please enter an address.');
  return;
 }

 try {
  // Fetch geocoding data from Nominatim API
  const response = await fetch(

`https://nominatim.openstreetmap.org/search?format=json&q=${encodeURIComponent(address.value)}`
  );
  const data = await response.json();

  if (data.length === 0) {
   alert('No results found for the given address.');
   return;
  }

  const { lat, lon, display_name } = data[0];
```

```
    // Añadir marcador en el mapa
    addMapMarker(lat, lon, display_name, selectedImage.value);

    // Guardar el marcador en la base de datos
    const markerData = {
      email,
      country: display_name,
      city: '', // Ajustar según necesidades
      location: { lat, lon },
      imageUrl: selectedImage.value,
    };

    console.log('Enviando marcador:', markerData);
    await addMarker(markerData);

    markers.value.push(markerData);
    console.log('Marcador añadido con éxito.');
  } catch (error) {
    console.error('Error al añadir el marcador:', error);
    alert('Failed to add marker. Please try again.');
  }
};
</script>

<template>
  <main>
    <!-- Input para dirección y subida de imagen -->
    <div class="mb-4">
      <input
        type="text"
```

```
      v-model="address"
      placeholder="Enter an address"
      class="p-2 border rounded w-full mb-2"
    />
    <div class="flex items-center space-x-4">
      <input type="file" @change="handleFileUpload" class="hidden" ref="fileInput"
/>
      <button
        @click="$refs.fileInput.click()"
        class="px-6 py-2 bg-primary text-background rounded-lg shadow-md"
      >
        Upload Image
      </button>
      <input
        v-model="imageSrc"
        type="text"
        readonly
        placeholder="Image URL"
        class="flex-1 border-2 border-gray-300 rounded-lg p-3"
      />
    </div>
    <button @click="addNewMarker" class="p-2 bg-blue-500 text-white rounded
mt-2">Add Marker</button>
  </div>

  <!-- Mapa -->
  <div id="map" style="height: 500px; width: 100%;"></div>

  <!-- Galería de imágenes -->
  <div class="mt-4 grid grid-cols-3 gap-4">
    <div v-for="marker in markers" :key="marker._id" class="border rounded">
```

```
      <img :src="marker.imageUrl" alt="Marker Image" class="w-full h-32 object-cover" />
      <p class="text-center p-2">{{ marker.location.country }}</p>
    </div>
  </div>
 </main>
</template>
```

- Como no he implementado oauth el visitas lo hago con una simulación pero funciona correctamente
- Problema
- Problema

Memoria Examen - Backend

Diseño de la base de datos

Colecciones

Para la base de datos no relacional he escogido MongoDB, la cual almacena la información en las siguientes colecciones:

- **Tasks:** las tareas en las que se puede colaborar.
    - **owner:** responsable de la tarea (campo de email)
    - **description**: descripción de la tarea.
    - **skillsRequired**: habilidades necesarias para desempeñarla
    - **segments**: duración, representada por múltiplos de 1 hora.
    - **collaborators**: lista de los emails de los que participan en la tarea
- **Collaborators**: Usuarios de la aplicación:
    - **email**: email del colaborador, lo utilizo como identificador
    - **name**: nombre
    - **skills**: array de sus habilidades

[Por defecto, he incluido los campos de createdAt y updatedAt de mongoDB a ambas colecciones.]

URI de la Base de Datos

Este servicio se ha desarrollado para su despliegue de forma local, utilizando docker para contenerizar, de forma que no se necesita configurar el acceso a una base de datos remota en MongoDB Atlas o similar.

La aplicación viene configurada en el '.env' con la cadena de conexión correspondiente para poder acceder al contenedor que contiene la instancia de MongoDB. Las instrucciones para poblarla se encuentran más adelante en este documento, así como en el código fuente (archivo README.md), junto con las *instrucciones de instalación y despliegue*.

Tecnologías utilizadas

Base de datos

- MongoDB, debido a su extensivo uso actualmente, amplia documentación y simplicidad.

Backend

En general, se han escogido tecnologías basadas en JavaScript.

- **Node.js**: runtime de JavaScript.
- **Librerías npm**: se han utilizado varias librerías para facilitar el desarrollo de la funcionalidad de la aplicación, así como el desarrollo local de la misma:
    - **Express.js**: framework no-opinionado de Node.js que facilita la creación de APIs.
    - **Nodemon**: reinicia de forma automática la aplicación al detectar cambios en los archivos, muy útil para el desarrollo en local.
    - **dotenv**: para cargar y utilizar variables de entorno
    - **Jest**: framework de testing para definir 'test suites' de forma sencilla e intuitiva.
    - **Supertest**: módulo/librería basada en Jest, especializada en peticiones http.
    - **axios**: librería para realizar peticiones http (usada para el testeo de los endpoints junto a Jest y Supertest).
    - **mongoose**: ODM de node.js para MongoDB. Nos permite trabajar con schemas (modelo de nuestras entidades) en lugar de queries de mongo para relacionarnos con la base de datos.

Contenedores

Para realizar el despliegue mediante contenedores, se ha utilizado Docker Desktop, que ya es un estándar en la industria a nivel mundial.

Instrucciones de instalación y despliegue

A continuación se describen las dependencias de la aplicación y las instrucciones para ejecutarla correctamente:

Requisitos

Al utilizar Docker Desktop, el despliegue de la aplicación es sencillo, **solo se necesita tener instalado Docker Desktop.** El resto de dependencias del proyecto se encuentran definidas en el docker-compose.yml y el Dockerfile, por lo que no es necesaria su instalación de forma local para ejecutar la aplicación. Sí se necesitará tener Postman para **probar** la aplicación.

Instrucciones

RESUMEN: clonar el repositorio/descomprimir el código fuente > copiar los valores del .env.example a un .env si no existe, ejecutar el archivo docker-compose > poblar la BD con POST requests en Postman.

1. Asegurarse de que Docker Desktop (y Docker engine) se encuentren instalados y en funcionamiento.
2. Clonar el proyecto del repositorio GitHub correspondiente, o bien descomprimir el zip de la entrega con el código fuente.

   https://github.com/antonioortegas/iw-backend.git

3. Hacer cd al root del proyecto, (normalmente al clonarlo, hacer

cd iw-backend/

siendo este el nombre de la carpeta del proyecto. En caso de haber descomprimido el zip, podemos seleccionar la carpeta > click derecho > open in terminal

4. Una vez estamos en la root del proyecto, debemos copiar el contenido de '.env.example' a un archivo al que debemos llamar '.env' si este no existe. En caso de que la terminal usada lo permita, podemos hacer:

   cp .env.example .env

   este .env será el que utilice nuestro docker-compose para las variables de entorno.

5. Una vez tenemos nuestro .env, simplemente ejecutamos nuestros contenedores con

   docker compose up -d

   [para detenerlos, docker compose down]

   esto pulleará la imagen de una instancia de mongoDB oficial desde dockerhub, y construirá el contenedor para nuestra aplicación.

6. La aplicación debería estar en funcionamiento en 'http://localhost:3002/'.

7. Los endpoints desarrollados y las peticiones de prueba para comprobar su correcto funcionamiento se encuentran definidos en una colección Postman adjunta a la entrega.

8. Para poblar la base de datos con algunos datos de prueba, ejecutar primero todas las request POST de postman en collaborators/ y tasks/ (son 8 en total), que contienen datos ‘dummy’ para crear 4 colaboradores y 4 tareas interconectadas.