DOCUMENTACIÓN DE DESARROLLADOR

Gestor de Reservas Multinegocio

David Tortosa Sánchez

Este documento forma parte de la documentación oficial del proyecto. Está orientado a desarrolladores que deseen saber como funciona el proyecto, sus características y cosas a tener en cuenta en caso de querer modificarlo.

Puedes usar el índice para acceder a cada sección. Esta guía incluye desde los requisitos iniciales hasta el despliegue.

Índice

1. Introducción

1.1 Breve descripción del sistema

Actualmente, muchos pequeños negocios no cuentan con herramientas digitales para organizar sus reservas, confiando en métodos tradicionales como agendas físicas o mensajes de texto. Esto puede provocar errores, citas duplicadas o confusión en la gestión diaria. Una plataforma centralizada permitiría automatizar estos procesos, mejorando la organización y la experiencia del cliente, y brindando a cada negocio la posibilidad de digitalizar su gestión sin necesidad de desarrollar una solución propia.

La plataforma permite que múltiples negocios gestionen sus horarios, servicios y reservas en un entorno centralizado. Los clientes pueden registrarse, buscar servicios y agendar citas.

1.2 Objetivos

Objetivo General

Desarrollar una plataforma web multinegocio para la gestión de reservas, facilitando la administración de horarios y servicios entre negocios y clientes.

Objetivos Específicos

- Permitir a los negocios registrar y gestionar su perfil, horarios y servicios.
- Facilitar a los clientes la búsqueda de negocios y la reserva de servicios.
- Implementar un sistema de notificaciones automáticas para confirmar y solicitar citas.
- Ofrecer un panel de control para que cada negocio gestione sus reservas y visualice su agenda.

1.3 Tecnologías utilizadas

Frontend

- **React con Next.js** → Para la creación de la interfaz y el manejo de páginas dinámicas.
- **Tailwind CSS** → Para el diseño de una interfaz moderna.

Backend

- **Next.js (API Routes)** → Para la lógica del servidor y la gestión de reservas.
- **Node.js** → Para la creación del backend.
- **NextAuth.js** → Para la gestión de la autenticación de usuarios.

Base de Datos

- **PostgreSQL** → Para la gestión relacional de los datos.
- **Prisma ORM** → Para facilitar las consultas y manipulación de la base de datos.

Notificaciones

- **Nodemailer** → Para el envío de correos electrónicos automáticos.

Despliegue

- **Vercel** → Para el despliegue de la aplicación, aprovechando su integración nativa con Next.js.

2. Requisitos del entorno

2.1 Software necesario

- Editor de código como visual code
- Node.js ≥ 18.0
- Git
- npm ≥ 9.0

2.2 Dependencias externas

Será necesario tener instalados y actualizados algunos modulos de la aplicación. Esto se puede hacer mediante el comando '**npm install "xxxx**" '.

Los principales son:

- Prisma y @prisma/client
- Next-auth
- Tailwindcss
- Nodemailer

También serán necesarios instalar:

- bcryptjs
- postcss
- autoprefixer
- @react-google-maps/api
- react-hot-toast
- @tailwindcss/forms
- @tailwindcss/typography

- lucide-react
- @tailwindcss/aspect-ratio
- react-datepicker

Ademas algunas instalaciones tendrán opciones para desarrollador asi que sera necesario añadirle al comando el flag '**--save-dev**' seguido del nombre:

- Prisma
- @types/next-auth
- @types/nodemailer

3. Instalación del entorno de desarrollo

3.1 Clonar el repositorio

Para trabajar con el código del proyecto, primero se deberá clonar el repositorio de github con el comando:

git clone https://github.com/DavidTortosaaa/gestor-reservas.git

3.2 Instalar dependencias

Mediante el comando '**npm install**' se instalará y actualizarán todas las dependencias.

3.3 Configurar variables de entorno

En el archivo **.env** se encuentran las variables de entorno:

- **DATABASE_URL="postgresql://direccion_de_la_base_de_datos"**
- **NEXTAUTH_URL="direccion o dominio de la aplicación"**
- **NEXTAUTH_SECRET="Clave Secreta"**
- **NEXT_PUBLIC_GOOGLE_MAPS_API_KEY="clave generada por la api de google maps en tu cuenta"**
- **Email_User="email de la aplicación"**
- **Email_Pass="contraseña de la aplicación generada por google con el email de la aplicacion"**

3.4 Inicializar base de datos

Para iniciar la base de datos con prisma, primero deberás definir el modelo de la base de datos en el archivo **/prisma/schema.prisma**. Si no encuentra este archivo, puede crearlo con el comando '**npx prisma init**'.

Una vez establecido el modelo, puede usar el comando '**npx prisma generate**' para generar el cliente de prisma y de esta forma poder realizar consultas en la aplicación.

En esta aplicación ya está creado, así que solo sería necesario usar el comando '**npx prisma migrate**' para generar el modelo de la base de datos en la base de datos que esté usando.

3.4 Ejecución

Para ejecutarlo localmente será necesario usar el comando **'npm run dev'**.

4. Estructura del proyecto

El proyecto sigue una estructura modular basada en rutas usando el sistema de app/ introducido en **Next.js 13+**.

Al mismo nivel que app/ encontramos:

- Archivos de configuración
- **types/:** Se encuentra el archivo de configuración de autenticación next-auth.d.ts
- **prisma/:** Se encuentra el modelo de la base de datos
- **lib/:** Se encuentran archivos varios. Estos incluyen configuracion de autenticación, archivos toast, archivos para configurar prisma y el envio de emails
- **components/:** Se encuentran fragmentos de código que se usan en las páginas, como formularios, tarjetas donde se muestran los negocios… Existe la carpeta ui/ que contiene componentes reutilizables como botones, inputs….

Dentro de app/ se encuentra el código principal, separado en carpetas por funcionalidades:

- **api/:** Aquí se encuentran todos los endpoints organizados por funciones.
- **login/**: Página de login
- **register/:** Página de registro
- **negocios/**: Negocios y sus funcionalidades
- **perfil/**: Página del perfil
- **reservas/:** Reservas y sus funcionalidades

Como detalles adicionales, es importante mencionar algunos puntos de la nomenclatura de las carpetas y archivos, procedente de la version 13+ de next.js

- **Rutas dinamicas [id]**:
  Las carpetas con nombres entre corchetes como `[id]` en Next.js indican rutas dinámicas. Esto permite capturar valores variables desde la URL, como identificadores únicos. Por ejemplo, una ruta definida como `/negocios/[id]/editar` responderá a `/negocios/123/editar`, donde `123` será accesible como un parámetro dentro del código.

- **Archivos route.ts**:
  Con el nuevo sistema App Router de Next.js (a partir de la versión 13), los archivos `route.ts` representan controladores para las rutas de API. Cada `route.ts` puede definir funciones para los métodos HTTP estándar como `GET`, `POST`, `PUT`, o `DELETE`. Por ejemplo, `/api/reservas/route.ts` gestionará las peticiones a `/api/reservas`, y `/api/reservas/[id]/route.ts` permitirá manejar reservas individuales por ID.

- **Archivos page.tsx**:
  Cada carpeta que contiene un archivo `page.tsx` representa una ruta de página en la interfaz del usuario. El archivo `page.tsx` actúa como el componente React principal que se renderiza cuando se accede a esa URL. Por ejemplo, `/perfil/page.tsx` mostrará el perfil del usuario al acceder a `/perfil`, y `/negocios/[id]/page.tsx` renderiza la vista de un negocio específico.

5. Base de datos

5.1 Sistema

El sistema de base de datos usado en la aplicación es **PostgresSQL**. Además, en este caso, la base de datos se encuentra alojada en https://railway.com/ .

5.2 Modelo E/R

El modelo de la base de datos y como está definido en el archivo **schema.prisma** es el siguiente:

5.2.1 Usuario

Representa a los usuarios del sistema, que pueden ser clientes o propietarios de negocios.

- **Atributos**:
  - Id (String): Identificador único del usuario (UUID).
  - Nombre (String): Nombre del usuario.
  - Email (String): Correo electrónico único del usuario.
  - Password (String): Contraseña del usuario.

- •Telefono (String, opcional): Teléfono de contacto.
- •Latitud (Float, opcional): Latitud de la ubicación del usuario.
- •Longitud (Float, opcional): Longitud de la ubicación del usuario.
- •Direccion (String, opcional): Dirección del usuario.

•**Relaciones**:

- •Negocios (Negocio[]): Lista de negocios asociados al usuario como propietario.
- •Reservas (Reserva[]): Lista de reservas realizadas por el usuario.

5.2.2 Negocio

Representa un negocio que ofrece servicios reservables.

•**Atributos**:

- •Id (String): Identificador único del negocio (UUID).
- •Nombre (String): Nombre del negocio.
- •Email (String): Correo electrónico único del negocio.
- •Telefono (String, opcional): Teléfono de contacto del negocio.
- •Latitud (Float, opcional): Latitud de la ubicación del negocio.
- •Longitud (Float, opcional): Longitud de la ubicación del negocio.
- •Direccion (String, opcional): Dirección del negocio.
- •Horario_apertura (String): Horario de apertura del negocio.
- •Horario_cierre (String): Horario de cierre del negocio.

•**Relaciones**:

- •Propietario (Usuario): Usuario propietario del negocio.
- •Servicios (Servicio[]): Lista de servicios ofrecidos por el negocio.

5.2.3 Servicio

Representa un servicio ofrecido por un negocio.

•**Atributos**:

- •Id (String): Identificador único del servicio (UUID).
- •Nombre (String): Nombre del servicio.
- •Descripcion (String, opcional): Descripción del servicio.
- •Duracion (Int): Duración del servicio en minutos.
- •Precio (Float): Precio del servicio.

•**Relaciones**:

- •Negocio (Negocio): Negocio al que pertenece el servicio.
- •Reservas (Reserva[]): Lista de reservas asociadas al servicio.

5.2.4 Reserva

Representa una reserva realizada por un cliente para un servicio.

•**Atributos**:

- •Id (String): Identificador único de la reserva (UUID).
- •FechaHora (DateTime): Fecha y hora de la reserva.
- •Estado (String): Estado de la reserva (e.g., "pendiente", "confirmada", "cancelada").

•**Relaciones**:

•Cliente (Usuario): Usuario que realizó la reserva.
•Servicio (Servicio): Servicio reservado.

5.2.5 Relaciones entre Entidades

1.**Usuario - Negocio**:
•Un usuario puede ser propietario de múltiples negocios.
•Cada negocio tiene un único propietario.
2.**Negocio - Servicio**:
•Un negocio puede ofrecer múltiples servicios.
•Cada servicio pertenece a un único negocio.
3.**Servicio - Reserva**:
•Un servicio puede tener múltiples reservas asociadas.
•Cada reserva está asociada a un único servicio.
4.**Usuario - Reserva**:
•Un usuario puede realizar múltiples reservas.
•Cada reserva está asociada a un único usuario.

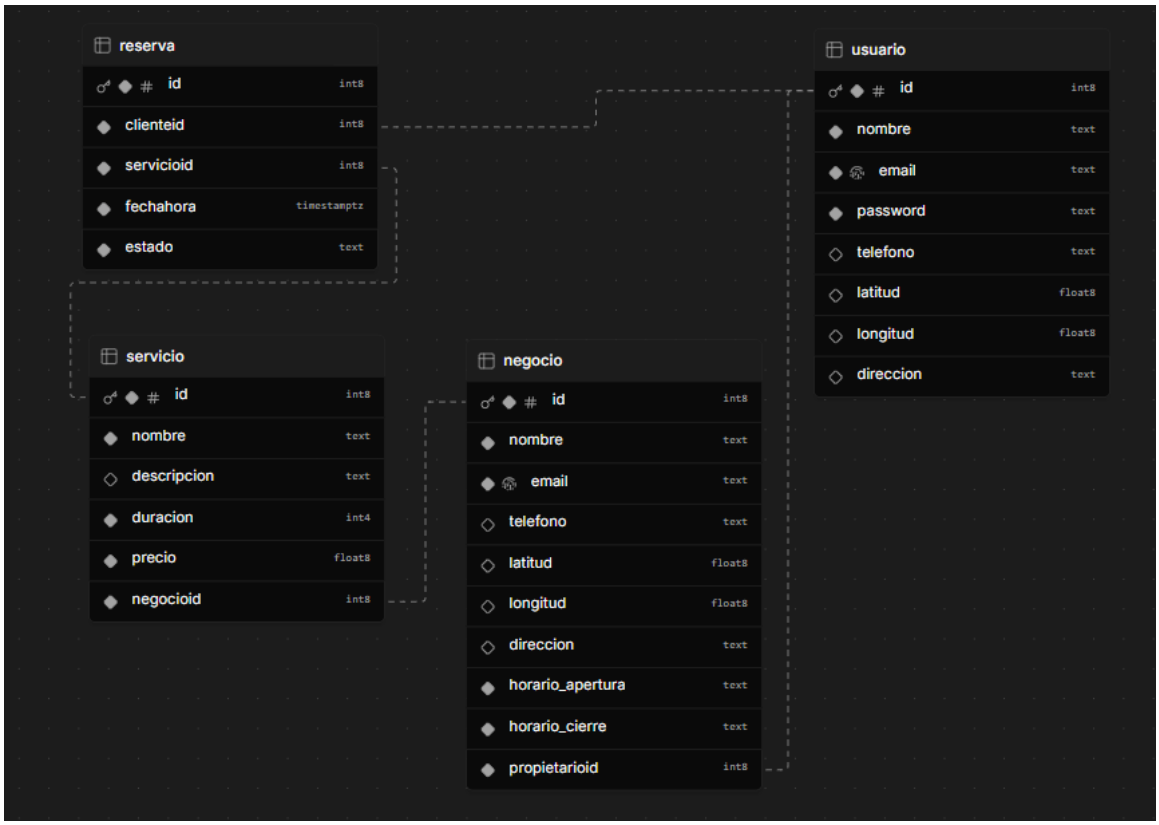
5.2.6 Reglas de Integridad

•**Eliminación en Cascada**:

•Si un negocio es eliminado, todos sus servicios y reservas asociadas también se eliminan.

•Si un servicio es eliminado, todas sus reservas asociadas también se eliminan.

•Si un usuario es eliminado, todas sus reservas asociadas y negocios también se eliminan.



5.3 Scripts de migración

Como se ha mencionado anteriormente, para migrar el esquema de la base de datos a la base de datos utilizada, es necesario aplicar el comando **'npx prisma migrate'.**

6. API

6.1 Autenticación

Para la autenticación se usa nextAuth.js con el email y contraseña como credenciales.

6.2 Endpoints principales

Los endpoints de la aplicación se encuentran en la carpeta /app/api/. Por la forma en que está construida la aplicación y el la autenticación, no es posible realizar peticiones desde postman o similares.

6.2.1 Autenticación de usuario

- **Método:** POST
- **Ruta:** /api/auth/[...nextauth]
- **Descripción:**

Este endpoint gestiona la autenticación del usuario mediante el proveedor de credenciales configurado en NextAuth. Verifica email y contraseña, y mantiene la sesión mediante JWT. Se utiliza principalmente desde el frontend a través de signIn() de NextAuth.

- **Autenticación requerida:** No.
- **Cuerpo de la solicitud:**
  - email (string) — Correo del usuario
  - password (string) — Contraseña del usuario
    (se envía como formulario, no JSON)
- **Códigos de error:**
  - 401 Unauthorized – Credenciales inválidas
  - 500 Internal Server Error – Error en la autenticación

6.2.2 Registro de Usuario

- **Método:** POST
- **Ruta:** /api/auth/register
- **Descripción:**
  Este endpoint permite registrar nuevos usuarios en el sistema. Antes de crear el usuario, verifica si el correo ya está registrado y encripta la contraseña usando bcryptjs.
- **Autenticación requerida:** No.
- **Cuerpo de la solicitud (JSON):**

```
{
  "nombre": "Juan Pérez",
  "email": "juan@example.com",
  "password": "123456",
  "telefono": "123456789",
  "latitud": 40.7128,
  "longitud": -74.0060,
  "direccion": "Calle Falsa 123"
}
```

- **Codigos de error:**
    - 400 Bad Request – El usuario ya existe
    - 500 Internal Server Error – Fallo inesperado en el servidor

6.2.3 Crear Negocio

- **Método:** POST
- **Ruta:** /api/negocios
- **Descripción:**
  Permite a un usuario autenticado registrar un nuevo negocio. El negocio quedará vinculado al usuario autenticado como propietario.
- **Autenticación requerida:** Sí.
- **Cuerpo de la solicitud (JSON):**

```
{
  "nombre": "Centro Estética",
  "email": "estetica@example.com",
  "telefono": "654321987",
  "direccion": "Av. Belleza 456",
  "latitud": 40.4168,
  "longitud": -3.7038,
  "horario_apertura": "09:00",
  "horario_cierre": "18:00"
}
```

- **Códigos de error:**
    - 401 Unauthorized – Usuario no autenticado
    - 404 Not Found – Usuario autenticado no encontrado en la base de datos
    - 500 Internal Server Error – Error al crear el negocio

6.2.4 Obtener Negocios Cercanos

- **Método:** GET
- **Ruta:** /api/negocios/cercanos
- **Descripción:**

Este endpoint permite obtener una lista de negocios cercanos (a menos de 50 km) al usuario autenticado. Utiliza la fórmula de Haversine para calcular la distancia entre la ubicación del usuario y los negocios registrados. Devuelve los negocios ordenados por cercanía.

- **Autenticación requerida:** Sí.
- **Cuerpo de la solicitud:** No requiere. La ubicación del usuario se toma desde su perfil en base de datos.
- **Códigos de error:**
    - 401 Unauthorized – El usuario no ha iniciado sesión.
    - 400 Bad Request – El usuario no tiene configurada su ubicación (latitud/longitud).
    - 200 OK – Lista de negocios cercanos con sus distancias en kilómetros.

6.2.5 Actualizar Negocio

- **Método:** PUT
- **Ruta:** /api/negocios/:id
- **Descripción:**
  Permite a un usuario autenticado actualizar la información de un negocio del que es propietario. Valida que el usuario coincida con el propietario registrado del negocio antes de permitir la modificación.
- **Autenticación requerida:** Sí.
- **Cuerpo de la solicitud (JSON):**

```
{
  "nombre": "Peluquería Renova",
  "email": "renova@negocio.com",
  "telefono": "600112233",
  "direccion": "Calle Pelos 25",
  "latitud": 40.4168,
  "longitud": -3.7038,
  "horario_apertura": "09:00",
  "horario_cierre": "18:00"
}
```

- **Códigos de error:**
    - 401 Unauthorized – El usuario no está autenticado
    - 400 Bad Request – El ID del negocio no se proporcionó correctamente en la URL
    - 403 Forbidden – El negocio no pertenece al usuario autenticado
    - 500 Internal Server Error – Error al actualizar el negocio

6.2.6 Eliminar Negocio

- **Método:** DELETE
- **Ruta:** /api/negocios/:id
- **Descripción:**
  Permite a un usuario autenticado eliminar un negocio que le pertenece. Verifica que el negocio exista y que el usuario actual sea su propietario antes de realizar la eliminación.
- **Autenticación requerida:** Sí.
- **Cuerpo de la solicitud:** No requiere cuerpo. El ID del negocio debe estar en la URL.
- **Códigos de error:**
    - 401 Unauthorized – El usuario no está autenticado
    - 400 Bad Request – El ID del negocio no se proporcionó correctamente en la URL
    - 403 Forbidden – El usuario no tiene permiso para eliminar ese negocio
    - 500 Internal Server Error – Error al eliminar el negocio

6.2.7 Obtener Perfil de Usuario

- **Método:** GET
- **Ruta:** /api/usuario/perfil
- **Descripción:**
  Este endpoint devuelve la información del perfil del usuario autenticado. Incluye datos personales y de localización.
- **Autenticación requerida:** Sí.
- **Códigos de error:**
    - 401 Unauthorized – El usuario no ha iniciado sesión

- 404 Not Found – Usuario no encontrado en la base de datos

6.2.8 Actualizar Perfil de Usuario

- **Método:** PATCH
- **Ruta:** /api/usuario/perfil
- **Descripción:**
  Permite al usuario autenticado modificar su información personal, incluyendo nombre, teléfono, dirección, ubicación (latitud y longitud), y contraseña (que se encripta si se proporciona).
- **Autenticación requerida:** Sí.
- **Cuerpo de la solicitud (JSON):**

```
{
  "nombre": "Juan Renovado",
  "telefono": "612345678",
  "password": "nuevaclave123",
  "latitud": 40.1234,
  "longitud": -3.5678,
  "direccion": "Calle Nueva 123"
}
```

- **Códigos de error:**
  - 401 Unauthorized – Usuario no autenticado
  - 400 Bad Request – Nombre inválido o contraseña demasiado corta

6.2.9 Crear Servicio

- **Método:** POST
- **Ruta:** /api/servicios
- **Descripción:**
  Este endpoint permite a un usuario autenticado crear un nuevo servicio dentro de uno de sus negocios. Antes de crear el servicio, valida que el negocio pertenezca al usuario autenticado.

- **Autenticación requerida:** Sí.
- **Cuerpo de la solicitud (JSON):**

```
{
  "nombre": "Corte de Pelo",
  "descripcion": "Corte moderno y lavado",
  "duracion": 30,
  "precio": 20.0,
  "negocioId": "uuid-del-negocio"
}
```

- **Códigos de error:**
  - 401 Unauthorized – Usuario no autenticado
  - 400 Bad Request – Faltan datos obligatorios
  - 403 Forbidden – El negocio no pertenece al usuario
  - 500 Internal Server Error – Error inesperado al crear el servicio

6.2.10 Actualizar Servicio

- **Método:** PUT
- **Ruta:** /api/servicios/:id
- **Descripción:**
  Este endpoint permite al propietario del negocio modificar un servicio ya existente. Solo el propietario del negocio al que pertenece el servicio puede actualizarlo.
- **Autenticación requerida:** Sí.
- **Parámetro en la URL:**
  - id – UUID del servicio a actualizar
- **Cuerpo de la solicitud (JSON):**

```
{
  "nombre": "Servicio actualizado",
  "descripcion": "Descripción opcional",
  "duracion": 45,
  "precio": 35.0
```

}

- **Códigos de error:**
    - 401 Unauthorized – Usuario no autenticado
    - 400 Bad Request – ID ausente o inválido en la URL
    - 403 Forbidden – El usuario no es propietario del servicio
    - 500 Internal Server Error – Error al actualizar el servicio

6.2.11 Eliminar Servicio

- **Método:** DELETE
- **Ruta:** /api/servicios/:id
- **Descripción:**
  Permite eliminar un servicio asociado a un negocio, siempre que el usuario autenticado sea el propietario del negocio al que pertenece dicho servicio.
- **Autenticación requerida:** Sí.
- **Parámetro en la URL:**
    - id – UUID del servicio a eliminar
- **Códigos de error:**
    - 401 Unauthorized – Usuario no autenticado
    - 400 Bad Request – ID inválido
    - 403 Forbidden – No autorizado para eliminar este servicio
    - 500 Internal Server Error – Error al eliminar el servicio

6.2.12 Crear Reserva

- **Método:** POST
- **Ruta:** /api/reservas
- **Descripción:**
  Este endpoint permite a un usuario autenticado crear una reserva para un servicio. Se valida la disponibilidad del servicio en la fecha/hora solicitada, que no sea fin de semana, y que no esté en el pasado. También se notifica al propietario del negocio por correo electrónico.
- **Autenticación requerida:** Sí.

- **Cuerpo de la solicitud (JSON):**

```
{
  "servicioId": "uuid-del-servicio",
  "fechaHora": "2025-06-15T10:00:00",
  "estado": "pendiente"
}
```

- **Códigos de error:**
  - 401 Unauthorized – Usuario no autenticado
  - 404 Not Found – Usuario o servicio no encontrado
  - 400 Bad Request – Faltan datos, fecha inválida o intento de reservar en fin de semana
  - 409 Conflict – Ya existe una reserva en ese horario
  - 500 Internal Server Error – Error en el servidor

6.2.13 Obtener Reservas del Usuario

- **Método:** GET
- **Ruta:** /api/reservas
- **Descripción:**
  Devuelve todas las reservas activas (pasadas o futuras) del usuario autenticado. Antes de devolver los resultados, elimina automáticamente las reservas canceladas o pendientes que ya hayan pasado.
- **Autenticación requerida:** Sí.
- **Códigos de error:**
  - 401 Unauthorized – Usuario no autenticado
  - 404 Not Found – Usuario no encontrado
  - 500 Internal Server Error – Error al recuperar las reservas

6.2.14 Verificar Disponibilidad de un Servicio

- **Método:** POST
- **Ruta:** /api/reservas/disponibilidad

- **Descripción:**
  Este endpoint permite consultar qué horas están ocupadas para un servicio en una fecha concreta. Las horas ocupadas se calculan teniendo en cuenta la duración del servicio y todas las reservas no canceladas que existen para ese día. El resultado devuelve intervalos de tiempo ocupados en pasos de 5 minutos. Este endpoint actualmente funciona pero en el front no se encuentra bien implementado.
- **Autenticación requerida:** No
- **Cuerpo de la solicitud (JSON):**

```
{
  "servicioId": "uuid-del-servicio",
  "fecha": "2025-06-18"
}
```

- **Respuesta (JSON):**

```
{
  "horasOcupadas": ["10:00", "10:05", "10:10", ..., "11:30"]
}
```

- **Códigos de error:**
  - 400 Bad Request – Faltan datos obligatorios
  - 404 Not Found – Servicio no encontrado
  - 500 Internal Server Error – Error durante el procesamiento de la disponibilidad

6.2.15 Confirmar o Cancelar Reserva (por el propietario)

- **Método:** POST
- **Ruta:** /api/reservas/:id
- **Descripción:**
  Permite al propietario del negocio confirmar o cancelar una reserva. Se envía una notificación por correo al cliente tras el cambio.
- **Autenticación requerida:** Sí (como propietario del negocio al que pertenece el servicio).
- **Cuerpo de la solicitud (form-data):**

  accion: confirmar | cancelar

- **Códigos de error:**
    - 400 Bad Request – ID inválido o acción no válida
    - 401 Unauthorized – No autenticado
    - 403 Forbidden – El usuario no es el propietario de la reserva
    -

6.2.16 Cancelar Reserva (por el cliente)

- **Método:** PATCH
- **Ruta:** /api/reservas/:id
- **Descripción:**
  Permite al cliente cancelar su propia reserva si aún no ha pasado.
- **Autenticación requerida:** Sí (como cliente que hizo la reserva).
- **Cuerpo de la solicitud (JSON):**

```
{
  "nuevoEstado": "cancelada"
}
```

- **Códigos de error:**
    - 400 Bad Request – ID inválido o intento de cancelar una reserva pasada
    - 401 Unauthorized – No autenticado
    - 403 Forbidden – La reserva no pertenece al usuario
    - 200 OK – Reserva cancelada correctamente

7. Lógica del negocio

7.1 Flujo principal

7.1.1 Cliente

1. El cliente accede a la app y se registra o inicia sesion

2. El cliente puede consultar negocios disponibles

3. El cliente puede ver los servicios de un negocio y reservar para una fecha

4. Se enviará un email al dueño del negocio

5. El cliente puede cancelar su reserva si lo desea

7.1.2 Dueño del negocio

1. El cliente accede a la app y se registra o inicia sesion

2. El cliente puede crear su negocio

3. El cliente puede gestionar sus negocios y añadir servicios

4. El cliente puede gestionar las reservas de su negocio y confirmarlas o cancelarlas

5. Se enviará un email si se confirma la reserva

7.2 Componentes clave

Aquí menciono algunos de los componentes claves de configuración de la aplicación. Para mas información, en el propio código vienen documentado todos los archivos incluido estos.

- **/prisma/schema.prisma** → Define el modelo de la base de datos
- **/lib/prisma.ts** → Almacena la instancia de prisma client para interactuar con la base de datos en la aplicación
- **/types/next-auth-d-ts** → Extiende la interfaz de autenticación de next para acceder al usuario.
- **/lib/auth.ts** → Provee la session del usuario para acceder a ella en la aplicación
- **/lib/auth-options.ts** → Une la autenticación de next con la base de datos prisma y establece las credenciales para iniciar sesión y el jwt.
- **/lib/mailer.ts** → Configuración del envio de correo electrónico
- **/app/tailwind.config.ts** → Configuración de los estilos con tailwind
- **/components/MapaUbicacion.tsx** → Este componente usa la API de google maps para generar un mapa interactivo. Sin las variables de entorno necesarias, este componente no funcionará.

8. Despliegue

8.1 Entornos disponibles

- Para la producción se ha usado la plataforma **Vercel.**

- https://gestor-reservas-xi.vercel.app/

8.2 Pasos de build

Vercel detecta automáticamente Next.js y genera la build tras cada push al repositorio. El repositorio se debe seleccionar desde github

8.3 Deploy

Para desplegarlo, una vez seleccionado el repositorio, se debe pulsar en deploy y configurar correctamente las variables de entorno en Vercel

9. Problemas

El proyecto presenta un par de problemas a tener en cuenta, ya que su funcionalidad se puede ver comprometida si no se abordan correctamente. Estos problemas no se pudieron solucionar por falta de tiempo al encontrarlos de forma tardía. Algunos de los más relevantes son:

- **Gestión de reservas concurrentes:** Si varios usuarios intentan reservar el mismo horario simultáneamente, puede haber conflictos si no se implementan medidas como bloqueos temporales o validación transaccional.
- **Múltiples pulsaciones de los botones:** Al pulsar en un botón como crear reserva, negocio, servicio o editar, la pagina necesita unos pocos segundos para cargar y completar la acción. Si durante estos segundos se vuelve a pulsar el botón previamente pulsado, la acción se puede llegar a ejecutar mas de una vez
- **Limitar visibilidad de horarios:** Existe un endpoint para que solo aparezcan las horas disponibles para reservar un servicio, sin embargo en el front no funciona correctamente. Esto no quiere decir que se puedan hacer dos reservas en una misma hora, el endpoint está protegido frente a eso, pero si aparecerá la hora como disponible. Al seleccionarla, un toast se mostrará informando de que la hora esta ocupadá y no permitirá hacer la reserva. Este error se debe a como se maneja la hora (formato ISO…).