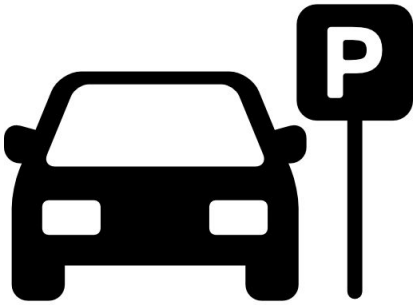

Documentación de la API Parking

Ricardo Cápiro Colomar

API en GitHub

Contents

1 Introducción

La API de gestión de parking es una solución desarrollada en Node.js, utilizando el framework NestJS que permite gestionar de manera eficiente las operaciones de un estacionamiento. Diseñada con una arquitectura RESTful, la API admite varias funcionalidades clave, tales como la reserva de plazas de aparcamiento, la consulta de ocupación actual, la actualización de detalles de usuarios y el acceso a logs de actividad. La API también incorpora sistemas de autenticación y autorización mediante tokens JWT y roles (administrador, empleado y cliente), garantizando que solo usuarios autorizados puedan realizar ciertas acciones.

Para el almacenamiento de datos, se utilizan dos bases de datos: MongoDB para los registros de actividad (logs) y PostgreSQL para la persistencia de las entidades de negocio. Además, la API incluye pruebas end-to-end automatizadas para asegurar el correcto funcionamiento en los casos de uso principales. La implementación también se acompaña de una colección de Thunder Client para facilitar el acceso y la prueba de los endpoints de la aplicación.

2 Bases de Datos

Como se mencionó anteriormente, se utilizaron dos bases de datos: MongoDB para los registros de actividad (logs) y PostgreSQL para la persistencia de las entidades de negocio. Para gestionar la base de datos en PostgreSQL, se utilizó Prisma como ORM, lo que facilita la interacción con la base de datos y permite realizar migraciones, consultas y actualizaciones de manera eficiente y segura.

2.1 MongoDB

MongoDB se emplea para registrar y gestionar los logs de actividad, lo que permite un seguimiento detallado de los eventos críticos, como reservas, cancelaciones y entradas y salidas de vehículos. Para implementar este sistema de logs, se definió un esquema `Log` utilizando la integración de NestJS con Mongoose.

El esquema `Log` contiene tres campos principales:

- **action**: Describe la acción realizada ("Reservación", "Cancelación", "Entrada", "Salida").
- **userId**: Almacena el identificador único del usuario que ejecutó la acción.
- **details**: Proporciona información detallada sobre el evento.

Además, el esquema usa la opción `timestamps`, que agrega automáticamente marcas de tiempo (`createdAt` y `updatedAt`) a cada registro, ayudando a rastrear cuándo ocurrieron las acciones.

2.2 PostgreSQL

PostgreSQL se utiliza como base de datos principal para almacenar y gestionar las entidades clave del sistema. La gestión de esta base de datos se realiza mediante Prisma, un ORM que facilita la creación, migración y consulta de datos en PostgreSQL.

La base de datos está compuesta por los siguientes modelos:

- **User**: Almacena la información de cada usuario, incluyendo campos como `id`, `name`, `email`, `password`, `phone` y `role`, que define el rol del usuario (`CLIENT`, `EMPLOYEE` o `ADMIN`). El campo `email` es único, por lo que no pueden existir dos usuarios con el mismo valor de `email`.
- **Vehicle**: Contiene los detalles de cada vehículo asociado a un usuario, incluyendo `license_vehicle`, `model_vehicle`, y `color_vehicle`. Cada vehículo puede estar asociado a uno o más clientes. En el modelo no pueden existir dos pares de cliente-vehículo iguales.
- **Parking_Space**: Representa las plazas de estacionamiento en el sistema. Cada plaza tiene un `init_dateTime` y `end_dateTime` para registrar la ocupación temporal. También tiene relaciones con el cliente (usuario) y el vehículo que ocupan la plaza.
- **Reservations**: Este modelo permite gestionar las reservas, almacenando la fecha y hora de inicio y fin (`init_dateTime`, `end_dateTime`) de cada reserva. Cada reserva está vinculada a un cliente, un vehículo, y una plaza de aparcamiento específica.

Además, la base de datos cuenta con una enumeración especial **Role** que define los roles en el sistema (`CLIENT`, `EMPLOYEE`, `ADMIN`), controlando el acceso y los permisos de los usuarios.

3 Módulos

Para facilitar la organización y mantenimiento de la API, la lógica de negocio se ha dividido en varios módulos, cada uno con responsabilidades específicas:

- **Auth**: Gestiona la autenticación y autorización de usuarios en la API. Este módulo se encarga de la creación y validación de tokens JWT, así como del control de acceso basado en roles.
- **Logs**: Maneja los registros de actividad (logs) en MongoDB, permitiendo llevar un seguimiento de eventos importantes, como reservaciones, cancelaciones, y entradas y salidas de vehículos.
- **Parking_Space**: Controla las operaciones relacionadas con las plazas de estacionamiento, incluyendo la consulta de disponibilidad, asignación y liberación de plazas.
- **Reservations**: Gestiona el sistema de reservas de plazas de aparcamiento, permitiendo a los usuarios crear, consultar, modificar y cancelar reservas.
- **Roles**: Define y gestiona los roles de los usuarios (CLIENT, EMPLOYEE, ADMIN), estableciendo los permisos y accesos adecuados para cada tipo de usuario.
- **User**: Administra la información y operaciones de los usuarios, tales como la creación, actualización y eliminación de perfiles.
- **Vehicle**: Gestiona los datos de los vehículos de los usuarios, incluyendo su registro y asociación con plazas de aparcamiento y reservas.

3.1 Auth

A continuación se detallan los endpoints relacionados con la autenticación:

3.1.1 POST /auth/login

Descripción: Autentica al usuario y genera un token JWT.
Parámetros del cuerpo:

- `email` (string): Correo del usuario.
- `password` (string): Contraseña del usuario.

Respuesta:

- `200 OK`: Retorna el usuario y el token de acceso.
- `401 Unauthorized`: Credenciales incorrectas.
- `400 Bad Request`: Error en los parámetros del cuerpo.

3.1.2 POST /auth/register

Descripción: Registra los datos del usuario.
Parámetros del cuerpo:

- `role` (Role): Role del usuario.
- `name` (string): Nombre del usuario.
- `phone` (string): Número de teléfono del usuario.
- `email` (string): Correo del usuario.
- `password` (string): Contraseña del usuario.

Respuesta:

- `201 Created`: Retorna los datos del usuario registrado.
- `400 Bad Request`: Error en los parámetros del cuerpo.

3.2 Logs

A continuación se detallan los endpoints relacionados con el registro de actividad (logs):

3.2.1 GET /logs

Descripción: Registra los datos del usuario.
Acceso a: ADMIN
Respuesta:

- `200 OK`: Retorna los registros de actividad.
- `404 Not Found`: No existen registros de actividad.

3.3 Parking Space

A continuación se detallan los endpoints relacionados con las plazas de estacionamiento:

3.3.1 `POST /parking-space/:amount`

Descripción: Crea plazas de estacionamiento.
Acceso a: ADMIN
Parámetros: El parámetro `amount` de la url, identifica la cantidad de plazas a crear.
Respuesta:

- `201 Created`: Éxito en las plazas creadas.

3.3.2 `GET /parking-space`

Descripción: Obtener todas las plazas de estacionamiento.
Acceso a: ADMIN, CLIENT, EMPLOYEE
Respuesta:

- `200 OK`: Retorna la cantidad de plazas vacías y una lista con la información de cada plaza ocupada.

3.3.3 `GET /parking-space/:id`

Descripción: Obtener una plaza específica del estacionamiento.
Acceso a: ADMIN, EMPLOYEE
Parámetros: El parámetro `id` de la url, identifica la plaza a obtener.
Respuesta:

- `200 OK`: Retorna los datos de la plaza obtenida.
- `404 Not Found`: No existe una plaza con ese `id`.

3.3.4 `PUT /parking-space/entry?id=<number>&is_entry=<boolean>`

Descripción: Actualiza los datos de una plaza específica de estacionamiento. Si `is_entry` es `true` se produce, además, una entrada de un vehículo al estacionamiento; en caso contrario simplemente son actualizados los datos de la plaza.
Acceso a: ADMIN, EMPLOYEE
Parámetros del cuerpo:

- `init_dateTime` (Date): Fecha y hora de la entrada del vehículo.
- `end_dateTime` (Date): Fecha y hora a la que debería salir el vehículo.
- `clientId` (number): Id del cliente asociado al vehículo.
- `vehicleId` (number): Id del vehículo.

Parámetros de la url:

- `id` (number): Id de la plaza a actualizar.
- `is_entry` (boolean): Valor booleano que indica si se produce una entrada de un vehículo o no.

Respuesta:

- `201 Created`: Retorna los datos actualizados de la plaza.
- `400 Bad Request`: Error en los parámetros del cuerpo o url.
- `404 Not Found`: No existe una plaza con ese `id`.

3.3.5 `PUT /parking-space/exit?id=<number>&date_exit=<Date>`

Descripción: Vacía los datos de una plaza específica de estacionamiento y se produce una salida del vehículo en la plaza especificada.
Acceso a: ADMIN, EMPLOYEE
Parámetros de la url:

- `id` (number): Id de la plaza a actualizar.
- `date_exit` (boolean): Fecha y hora que se produjo la salida del vehículo.

Respuesta:

- `200 OK`: Retorna los datos actualizados de la plaza.
- `400 Bad Request`: Error en los parámetros de la url.
- `404 Not Found`: No existe una plaza con ese `id`.

3.3.6 `DELETE /parking-space/:id`

Descripción: Elimina una plaza específica de estacionamiento.
Acceso a: ADMIN
Parámetros: El parámetro `id` de la url, identifica la plaza a eliminar.
Respuesta:

- `200 OK`: Retorna los datos de la plaza eliminada.
- `404 Not Found`: No existe una plaza con ese `id`.

3.4 Reservations

A continuación se detallan los endpoints relacionados con las reservaciones:

3.4.1 POST /reservations

Descripción: Crea una reservación.
Acceso a: ADMIN, CLIENT
Parámetros del cuerpo:

- `init_dateTime` (Date): Fecha y hora de la entrada del vehículo.
- `end_dateTime` (Date): Fecha y hora a la que debería salir el vehículo.
- `clientId` (number): Id del cliente asociado al vehículo.
- `vehicleId` (number): Id del vehículo.
- `placeId` (number): Id de la plaza a ocupar.

Respuesta:

- `201 Created`: Retorna los datos de la reservación creada.
- `400 Bad Request`: Error en los parámetros del cuerpo.
- `500 Internal Server Error`: Error inesperado en el servidor.

3.4.2 GET /reservations

Descripción: Obtiene todas las reservaciones.
Acceso a: ADMIN, EMPLOYEE
Respuesta:

- `200 OK`: Retorna una lista con los datos de las reservaciones.

3.4.3 GET /reservations/:id

Descripción: Obtener una reservación específica.
Acceso a: ADMIN, EMPLOYEE
Parámetros: El parámetro `id` de la url, identifica la reservación a obtener.
Respuesta:

- `200 OK`: Retorna los datos de la reservación obtenida.
- `404 Not Found`: No existe una reservación con ese `id`.

3.4.4 PUT /reservations/:id

Descripción: Actualiza los datos de una reservación.
Acceso a: ADMIN, EMPLOYEE
Parámetro de la url: El parámetro `id` de la url, identifica la reservación a actualizar.
Parámetros del cuerpo:

- `init_dateTime` (Date): Fecha y hora de la entrada del vehículo (opcional).
- `end_dateTime` (Date): Fecha y hora a la que debería salir el vehículo (opcional).
- `clientId` (number): Id del cliente asociado al vehículo (opcional).
- `vehicleId` (number): Id del vehículo (opcional).
- `placeId` (number): Id de la plaza a ocupar (opcional).

Respuesta:

- `200 OK`: Retorna los datos de la reservación actualizada.
- `400 Bad Request`: Error en los parámetros del cuerpo o url.
- `500 Internal Server Error`: Error inesperado en el servidor.

3.4.5 DELETE /reservations?id=<number>&is_cancellation=<boolean>

Descripción: Elimina una reservación específica. Si `is_cancellation` es `true` se produce, además, una cancelación de una reserva; en caso contrario simplemente es eliminada la reservación especificada.
Acceso a: ADMIN, EMPLOYEE
Parámetros de la url:

- `id` (number): Id de la reservación a eliminar.
- `is_cancellation` (boolean): Valor booleano que indica si se produce una cancelación de una reserva o no.

Respuesta:

- `200 OK`: Retorna los datos de la reservación eliminada.
- `400 Bad Request`: Error en los parámetros de la url.
- `404 Not Found`: No existe una reservación con ese `id`.

3.5 User

A continuación se detallan los endpoints relacionados con los usuarios:

3.5.1 `POST /user`

Descripción: Crea un usuario.
Acceso a: ADMIN
Parámetros del cuerpo:

- `role` (Role): Role del usuario.
- `name` (string): Nombre del usuario.
- `phone` (string): Número de teléfono del usuario.
- `email` (string): Correo del usuario.
- `password` (string): Contraseña del usuario.

Respuesta:

- `201 Created`: Retorna los datos del usuario creada.
- `400 Bad Request`: Error en los parámetros del cuerpo.

3.5.2 `GET /user`

Descripción: Obtiene todos los usuarios.
Acceso a: ADMIN
Respuesta:

- `200 OK`: Retorna una lista con los datos de los usuarios.

3.5.3 `GET /user/client`

Descripción: Obtiene todos los clientes.
Acceso a: ADMIN, EMPLOYEE
Respuesta:

- `200 OK`: Retorna una lista con los datos de los clientes.

3.5.4 `GET /user/employee`

Descripción: Obtiene todos los empleados.
Acceso a: ADMIN
Respuesta:

- `200 OK`: Retorna una lista con los datos de los empleados.

3.5.5 GET /user/admin

Descripción: Obtiene todos los administradores.
Acceso a: ADMIN
Respuesta:

- `200 OK`: Retorna una lista con los datos de los administradores.

3.5.6 GET /user/:id

Descripción: Obtener un usuario específico.
Acceso a: ADMIN
Parámetros: El parámetro `id` de la url, identifica el usuario a obtener.
Respuesta:

- `200 OK`: Retorna los datos del usuario obtenido.
- `404 Not Found`: No existe un usuario con ese `id`.

3.5.7 GET /user/client/:id

Descripción: Obtener un cliente específico.
Acceso a: EMPLOYEE
Parámetros: El parámetro `id` de la url, identifica el cliente a obtener.
Respuesta:

- `200 OK`: Retorna los datos del cliente obtenido.
- `404 Not Found`: No existe un cliente con ese `id`.

3.5.8 PUT /user/:id

Descripción: Actualiza los datos de un usuario.
Acceso a: ADMIN, EMPLOYEE, CLIENT
Parámetro de la url: El parámetro `id` de la url, identifica el usuario a actualizar.
Parámetros del cuerpo:

- `name` (string): Nombre del usuario.
- `phone` (string): Número de teléfono del usuario.
- `email` (string): Correo del usuario.
- `password` (string): Contraseña del usuario.

Respuesta:

- `200 OK`: Retorna los datos del usuario actualizado.
- `400 Bad Request`: Error en los parámetros del cuerpo o url.
- `404 Not Found`: No existe un usuario con ese `id`.

3.5.9 PUT /user/promote/:id

Descripción: Cambia el rol de un empleado a administrador.
Acceso a: ADMIN
Parámetro de la url: El parámetro `id` de la url, identifica el empleado a actualizar.
Respuesta:

- `200 OK`: Retorna los datos del empleado actualizado.
- `404 Not Found`: No existe un empleado con ese `id`.

3.5.10 DELETE /user/:id

Descripción: Elimina un usuario.
Acceso a: ADMIN
Parámetros: El parámetro `id` de la url, identifica el usuario a eliminar.
Respuesta:

- `200 OK`: Retorna los datos del usuario eliminado.
- `404 Not Found`: No existe un usuario con ese `id`.

3.6 Vehicle

A continuación se detallan los endpoints relacionados con los vehículos:

3.6.1 POST /vehicle

Descripción: Crea un vehículo.
Acceso a: ADMIN, EMPLOYEE, CLIENT
Parámetros del cuerpo:

- `clientId` (number): Id del cliente asociado al vehículo.
- `license_vehicle` (string): Licencia del vehículo.
- `model_vehicle` (string): Modelo del vehículo.
- `color_vehicle` (string): Color del vehículo.

Respuesta:

- `201 Created`: Retorna los datos del vehículo creado.
- `400 Bad Request`: Error en los parámetros del cuerpo.
- `500 Internal Server Error`: Error inesperado en el servidor.

3.6.2 GET /vehicle

Descripción: Obtiene todos los usuarios.
Acceso a: ADMIN, EMPLOYEE
Respuesta:

- `200 OK`: Retorna una lista con los datos de los vehículos.

3.6.3 GET /vehicle/client/:id

Descripción: Obtener todos los vehículos de un cliente específico.
Acceso a: CLIENT
Parámetros: El parámetro `id` de la url, identifica el cliente del cual se obtendrán todos sus vehículos.
Respuesta:

- `200 OK`: Retorna una lista con los datos de los vehículos asociados al cliente especificado.
- `404 Not Found`: No existe un cliente con ese `id`.

3.6.4 GET /vehicle/:id

Descripción: Obtener un vehículo específico.
Acceso a: ADMIN, EMPLOYEE
Parámetros: El parámetro `id` de la url, identifica el vehículo a obtener.
Respuesta:

- `200 OK`: Retorna una lista con los datos del vehículo obtenido.
- `404 Not Found`: No existe un vehículo con ese `id`.

3.6.5 PUT /vehicle/:id

Descripción: Actualiza los datos de un vehículo.
Acceso a: ADMIN, CLIENT
Parámetro de la url: El parámetro `id` de la url, identifica el vehículo a actualizar.
Parámetros del cuerpo:

- `clientId` (number): Id del cliente asociado al vehículo (opcional).
- `license_vehicle` (string): Licencia del vehículo (opcional).
- `model_vehicle` (string): Modelo del vehículo (opcional).
- `color_vehicle` (string): Color del vehículo (opcional).

Respuesta:

- `200 OK`: Retorna los datos del vehículo actualizado.
- `400 Bad Request`: Error en los parámetros del cuerpo o url.

- `404 Not Found`: No existe un vehículo con ese `id`.
- `500 Internal Server Error`: Error inesperado en el servidor.

3.6.6 DELETE /vehicle/:id

Descripción: Elimina un vehículo.
Acceso a: ADMIN
Parámetros: El parámetro `id` de la url, identifica el vehículo a eliminar.
Respuesta:

- `200 OK`: Retorna los datos del vehículo eliminado.
- `404 Not Found`: No existe un vehículo con ese `id`.