Taller API

Versión 2.0.0

Antonio Martin Sosa

05 de junio de 2025

Contenido:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CAPÍTULO 1

Introducción

Bienvenido a la documentación técnica de la API de gestión de vehículos e informes de diagnóstico.

Este sistema permite registrar usuarios, añadir vehículos, capturar errores OBD-II (DTC), generar informes personalizados y compartirlos mediante enlaces públicos enviados por correo electrónico.

Tecnologías utilizadas:

- **FastAPI**: para la creación de endpoints RESTful modernos.
- **SQLAlchemy**: ORM para gestión de base de datos relacional.
- **Pydantic**: validación de datos.
- **JWT (JSON Web Tokens)**: autenticación y autorización.
- **FastAPI-Mail**: envío de correos con informes.
- **MySQL**: base de datos principal.
- **Sphinx**: generación de esta documentación.

Esta documentación se divide en las siguientes secciones:

- Descripción de los endpoints disponibles.
- Modelos de datos (ORM y Pydantic).
- Configuración del sistema (DB, correo, JWT, etc.).

CAPÍTULO 2

Endpoints de la API

A continuación se detallan todos los endpoints disponibles en la API, clasificados por funcionalidad.

class main.**Base**(***kwargs: Any*)

Bases: `object`

Configuración del sistema de envío de correos (FastAPI Mail):

- Las credenciales y parámetros se cargan desde variables de entorno.
- *FastMail* se instancia con esta configuración para ser usado en envíos.

metadata: MetaData = MetaData()

registry: registry = <sqlalchemy.orm.decl_api.registry object>

class main.**ErrorVehiculo**(***kwargs*)

Bases: *Base*

Modelo ORM que almacena los errores OBD-II (códigos DTC) de un vehículo.

Atributos:
id (int): ID del error. vehiculo_id (int): ID del vehículo asociado. codigo_dtc (str): Código de diagnóstico (ej. P0301).

Relaciones:
vehiculo (Vehiculo): Vehículo asociado.

codigo_dtc

id

vehiculo

vehiculo_id

class main.**ErrorVehiculoRegistro**(** (Keyword-only parameters separator (PEP 3102))*, *codigo_dtc: list[str]*, *vehiculo_id: int*)

Bases: BaseModel

Modelo de solicitud para registrar errores OBD-II de un vehículo.

Atributos:
codigo_dtc (list[str]): Lista de códigos DTC (códigos de diagnóstico). vehiculo_id (int): ID del vehículo al que se le asocian los errores.

codigo_dtc: list[str]

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

vehiculo_id: int

class main.**InformeCompartido**(***kwargs*)

Bases: *Base*

Modelo ORM que representa un informe compartido con un cliente por email.

Atributos:
id (int): ID del informe. token (str): Token único para acceder al informe. vehiculo_id (int): ID del vehículo relacionado. email_cliente (str): Email al que se envía el informe. creado_en (str): Fecha y hora de creación del informe (ISO format).

Relaciones:
vehiculo (Vehiculo): Vehículo asociado.

creado_en

email_cliente

id

token

vehiculo

vehiculo_id

class main.**InformeRequest**(*, *email: str*)

Bases: BaseModel

Modelo de solicitud para generar y enviar un informe por correo.

Atributos:
email (str): Dirección de email del cliente destinatario.

email: str

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class main.**Usuario**(***kwargs*)

Bases: *Base*

Modelo ORM que representa a los usuarios del sistema.

Atributos:
id (int): ID autoincremental (clave primaria). username (str): Nombre de usuario, único. password_hash (str): Contraseña hasheada con bcrypt.

Relaciones:
vehiculos (List[Vehiculo]): Lista de vehículos registrados por el usuario.

id

password_hash

username

vehiculos

class main.**UsuarioLogin**(*, *username: str*, *password: str*)

Bases: BaseModel

Modelo de solicitud para iniciar sesión de usuario.

Atributos:
username (str): Nombre de usuario. password (str): Contraseña en texto plano.

model_config: ClassVar[ConfigDict] = {}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

password: str

username: str

class main.**UsuarioRegistro**(*, *username: str*, *password: str*)

Bases: BaseModel

Modelo de solicitud para registrar un nuevo usuario.

Atributos:
username (str): Nombre de usuario. password (str): Contraseña en texto plano.

model_config: ClassVar[ConfigDict] = {}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

password: str

username: str

class main.**Vehiculo**(***kwargs*)

Bases: *Base*

Modelo ORM que representa un vehículo registrado.

Atributos:
id (int): ID del vehículo. marca (str): Marca del vehículo. modelo (str): Modelo del vehículo. year (int): Año de fabricación. rpm (int): Revoluciones por minuto. velocidad (int): Velocidad actual. vin (str): Número VIN único del vehículo. revision (str): Información de revisión técnica. usuario_id (int): ID del usuario al que pertenece el vehículo.

Relaciones:
usuario (Usuario): Usuario propietario. errores (List[ErrorVehiculo]): Lista de errores asociados. informes_compartidos (List[InformeCompartido]): Informes generados con token público.

errores

id

informes_compartidos

marca

modelo

revision

rpm

usuario

usuario_id

velocidad

vin

year

class main.**VehiculoEdicion**(*, *marca: str*, *modelo: str*, *year: int*, *rpm: int*, *velocidad: int*, *vin: str*)

Bases: BaseModel

Modelo de solicitud para editar un vehículo existente.

Atributos:

marca (str): Marca del vehículo. modelo (str): Modelo del vehículo. year (int): Año de fabricación. rpm (int): Revoluciones por minuto. velocidad (int): Velocidad actual. vin (str): Número VIN del vehículo.

marca: str

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

modelo: str

rpm: int

velocidad: int

vin: str

year: int

class main.**VehiculoRegistro**(*, *marca: str*, *modelo: str*, *year: int*, *rpm: int*, *velocidad: int*, *vin: str*, *revision: dict*)

Bases: BaseModel

Modelo de solicitud para registrar un nuevo vehículo.

Atributos:

marca (str): Marca del vehículo. modelo (str): Modelo del vehículo. year (int): Año del vehículo. rpm (int): RPM del motor. velocidad (int): Velocidad del vehículo. vin (str): Número VIN único del vehículo. revision (dict): Detalles de la revisión técnica (estructura flexible).

marca: str

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

modelo: str

revision: dict

rpm: int

velocidad: int

vin: str

year: int

async main.**crear_informe**(*vehiculo_id: int*, *request:* InformeRequest, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Crea un informe de errores del vehículo y lo envía al email del cliente.

Este endpoint genera un enlace único que da acceso a una vista del informe de diagnóstico del vehículo. Se envía un correo al cliente con dicho enlace.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo del que se desea generar el informe.
- **request** (InformeRequest) – Objeto que contiene el email del cliente.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve

Mensaje de éxito, token generado y enlace de acceso.

Tipo del valor devuelto

dict

Muestra

- **HTTPException 400** – Si el email no es válido.
- **HTTPException 404** – Si el vehículo no pertenece al usuario.
- **HTTPException 500** – Si ocurre un error al guardar el informe o enviar el correo.

main.**crear_token**(*data: dict*, *expira_en: int = 300*)

Genera un token JWT con los datos proporcionados y un tiempo de expiración opcional.

Parámetros

- **datos** (*dict*) – Datos a incluir en el payload del token.
- **tiempo_expiracion** (*Optional[timedelta]*) – Tiempo personalizado de expiración. Si no se especifica, se usarán 30 minutos por defecto.

Devuelve

Token JWT firmado.

Tipo del valor devuelto

str

Muestra
Exception – Si hay un error al codificar el token.

main.**editar_vehiculo**(*vehiculo_id: int*, *datos:* VehiculoEdicion, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Actualiza los datos de un vehículo existente del usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo a modificar.
- **datos_actualizados** (*VehiculoBase*) – Nuevos datos del vehículo.
- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado.

Devuelve
Mensaje de éxito.

Tipo del valor devuelto
dict

Muestra
HTTPException 404 – Si el vehículo no existe o no pertenece al usuario.

main.**eliminar_vehiculo**(*vehiculo_id: int*, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Elimina un vehículo registrado por el usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo a eliminar.
- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.

Devuelve
Mensaje de éxito.

Tipo del valor devuelto
dict

Muestra
HTTPException 404 – Si el vehículo no existe o no pertenece al usuario.

main.**fm = <fastapi_mail.fastmail.FastMail object>**

Configuración de seguridad:

- *SECRET_KEY*, *ALGORITHM* y tiempo de expiración definen la seguridad del JWT.
- *pwd_context* se usa para hashear contraseñas con bcrypt.
- *oauth2_scheme* se usa como dependencia para extraer el token del header Authorization.

main.**get_car_image**(*searchTerm: str*)

Obtiene una URL de imagen representativa de un vehículo usando el término de búsqueda proporcionado.

Este endpoint consulta la API externa de carimagery.com para devolver la URL de una imagen que coincida con el término (por ejemplo, «Toyota Corolla 2020»).

Parámetros
searchTerm (*str*) – Término de búsqueda del vehículo (marca, modelo, año, etc.).

Devuelve
URL de la imagen del vehículo.

Tipo del valor devuelto
str

Muestra
HTTPException 500 – Si hay un error al consultar la API externa.

main.**get_db**()

Dependencia de FastAPI para obtener una sesión de base de datos.

Se utiliza con *Depends(get_db)* para abrir una sesión, cederla al endpoint y cerrarla automáticamente.

main.**guardar_errores**(*datos:* ErrorVehiculoRegistro, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Guarda una lista de códigos de error OBD-II (DTC) asociados a un vehículo del usuario autenticado.

Este endpoint es utilizado por el cliente Python que recibe errores del escáner OBD-II y los envía al backend para su almacenamiento.

Parámetros

- **datos** (ErrorVehiculoRegistro) – Objeto que contiene el ID del vehículo y una lista de códigos DTC.
- **usuario** (Usuario) – Usuario autenticado, obtenido desde el token JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Mensaje de confirmación si los errores fueron guardados correctamente.

Tipo del valor devuelto
dict

Muestra

- **HTTPException 400** –
  - Si el ID del vehículo no es válido (no entero o negativo). - Si la lista de códigos está vacía o contiene valores vacíos. - Si hay códigos DTC duplicados.
- **HTTPException 404** – Si el vehículo no pertenece al usuario autenticado.
- **HTTPException 500** – Si ocurre un error inesperado al guardar en la base de datos.

main.**guardar_vehiculo**(*datos:* VehiculoRegistro, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Guarda un nuevo vehículo en la base de datos asociado al usuario autenticado.

Parámetros

- **vehiculo** (*VehiculoBase*) – Datos del vehículo (marca, modelo, año, color, etc.).
- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado, extraído desde el token JWT.

Devuelve
Mensaje de confirmación.

Tipo del valor devuelto
dict

Muestra
HTTPException 401 – Si no se proporciona un token válido.

main.**login**(*datos:* UsuarioLogin, *db: Session = Depends(get_db)*)

Autentica al usuario y devuelve un token JWT válido.

Parámetros

- **datos** (UsuarioLogin) – Credenciales de usuario.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Token JWT si la autenticación fue exitosa.

Tipo del valor devuelto
dict

Muestra

- **HTTPException 400** – Datos inválidos.
- **HTTPException 401** – Usuario no encontrado o contraseña incorrecta.
- **HTTPException 500** – Error al generar el token.

main.**obtener_errores**(*vehiculo_id: int*, *usuario:* Usuario *= Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Devuelve todos los errores DTC (códigos OBD-II) asociados a un vehículo del usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo para el que se desean consultar los errores.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Lista de errores registrados.

Tipo del valor devuelto
List[*ErrorVehiculo*]

Muestra
HTTPException 404 – Si no existen errores para ese vehículo.

main.**obtener_usuario_desde_token**(*token: str = Depends(OAuth2PasswordBearer)*, *db: Session = Depends(get_db)*)

Extrae y valida el usuario actual a partir del token JWT proporcionado.

Parámetros

- **token** (*str*) – Token JWT incluido en el encabezado de autorización.
- **db** (*Session*) – Sesión de base de datos.

Devuelve
Instancia del usuario autenticado.

Tipo del valor devuelto
Usuario

Muestra
HTTPException 401 – Si el token es inválido o ha expirado.

main.**obtener_vehiculo**(*vehiculo_id: int*, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Recupera la información de un vehículo específico registrado por el usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo a consultar.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Objeto del vehículo solicitado.

Tipo del valor devuelto
Vehiculo

Muestra
HTTPException 404 – Si el vehículo no pertenece al usuario o no existe.

main.**obtener_vehiculos**(*usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Obtiene todos los vehículos registrados por el usuario autenticado.

Parámetros

- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.

Devuelve
Lista de vehículos asociados al usuario.

Tipo del valor devuelto
List[VehiculoBase]

main.**register**(*datos:* UsuarioRegistro, *db: Session = Depends(get_db)*)

Registra un nuevo usuario en la base de datos.

Parámetros

- **datos** (UsuarioRegistro) – Objeto que contiene el nombre de usuario y la contraseña.
- **db** (*Session*) – Sesión activa de la base de datos, proporcionada por FastAPI.

Devuelve
Un mensaje indicando si el usuario fue registrado exitosamente.

Tipo del valor devuelto
dict

Muestra
HTTPException 400 – Si los campos son inválidos o el nombre de usuario ya existe.

async main.**saludo**()

Devuelve un mensaje simple para verificar que la API está activa.

Este endpoint puede utilizarse para pruebas de conectividad o para confirmar que el backend está desplegado correctamente.

Devuelve
Mensaje de saludo indicando que la API funciona.

Tipo del valor devuelto
dict

main.**ver_informe**(*token: str*, *db: Session = Depends(get_db)*)

Devuelve los datos del informe generado a partir de un token único.

Este endpoint permite el acceso público a un informe de diagnóstico de vehículo mediante un enlace con token generado previamente. No requiere autenticación, pero valida que el token sea legítimo.

Parámetros
token (*str*) – Token único del informe generado.

Devuelve
Información del vehículo (marca, modelo, año, etc.) y lista de errores DTC.

Tipo del valor devuelto
dict

Muestra

- **HTTPException 400** – Si el token no es válido o demasiado corto.
- **HTTPException 404** – Si no se encuentra el informe, el vehículo o los errores asociados.
- **HTTPException 500** – Si ocurre un error inesperado al procesar la solicitud.

main.**verificar_password**(*plain_password*, *hashed_password*)

Verifica si una contraseña en texto plano coincide con su hash almacenado.

Parámetros

- **password_plano** (*str*) – Contraseña proporcionada por el usuario.
- **password_hash** (*str*) – Hash almacenado en la base de datos.

Devuelve
True si coinciden, False si no.

Tipo del valor devuelto
bool

CAPÍTULO 3

Modelos de Datos

3.1 Modelos ORM (SQLAlchemy)

Los modelos ORM representan las tablas en la base de datos. Están definidos con SQLAlchemy e incluyen relaciones entre entidades.

- *Usuario*: representa un usuario registrado.
- *Vehiculo*: vehículo asociado a un usuario.
- *ErrorVehiculo*: errores OBD-II asociados a un vehículo.
- *InformeCompartido*: informes generados para ser enviados por correo.

3.2 Modelos de validación (Pydantic)

Los modelos Pydantic se utilizan para validar entradas y salidas en la API:

- *UsuarioRegistro*, *UsuarioLogin*: manejo de usuarios.
- *VehiculoRegistro*, *VehiculoEdicion*: datos de los vehículos.
- *ErrorVehiculoRegistro*: errores recibidos desde el cliente OBD.
- *InformeRequest*: datos para enviar el informe por email.

class main.**Base**(***kwargs: Any*)

Bases: object

Configuración del sistema de envío de correos (FastAPI Mail):

- Las credenciales y parámetros se cargan desde variables de entorno.
- *FastMail* se instancia con esta configuración para ser usado en envíos.

class main.**ErrorVehiculo**(***kwargs*)

Bases: *Base*

Modelo ORM que almacena los errores OBD-II (códigos DTC) de un vehículo.

Atributos:
id (int): ID del error. vehiculo_id (int): ID del vehículo asociado. codigo_dtc (str): Código de diagnóstico (ej. P0301).

Relaciones:
vehiculo (Vehiculo): Vehículo asociado.

class main.**ErrorVehiculoRegistro**(***, *codigo_dtc: list[str]*, *vehiculo_id: int*)

Bases: BaseModel

Modelo de solicitud para registrar errores OBD-II de un vehículo.

Atributos:
codigo_dtc (list[str]): Lista de códigos DTC (códigos de diagnóstico). vehiculo_id (int): ID del vehículo al que se le asocian los errores.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class main.**InformeCompartido**(***kwargs*)

Bases: *Base*

Modelo ORM que representa un informe compartido con un cliente por email.

Atributos:
id (int): ID del informe. token (str): Token único para acceder al informe. vehiculo_id (int): ID del vehículo relacionado. email_cliente (str): Email al que se envía el informe. creado_en (str): Fecha y hora de creación del informe (ISO format).

Relaciones:
vehiculo (Vehiculo): Vehículo asociado.

class main.**InformeRequest**(***, *email: str*)

Bases: BaseModel

Modelo de solicitud para generar y enviar un informe por correo.

Atributos:
email (str): Dirección de email del cliente destinatario.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class main.**Usuario**(***kwargs*)

Bases: *Base*

Modelo ORM que representa a los usuarios del sistema.

Atributos:
id (int): ID autoincremental (clave primaria). username (str): Nombre de usuario, único. password_hash (str): Contraseña hasheada con bcrypt.

Relaciones:
vehiculos (List[Vehiculo]): Lista de vehículos registrados por el usuario.

class main.**UsuarioLogin**(*, *username: str*, *password: str*)

Bases: BaseModel

Modelo de solicitud para iniciar sesión de usuario.

Atributos:
username (str): Nombre de usuario. password (str): Contraseña en texto plano.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class main.**UsuarioRegistro**(*, *username: str*, *password: str*)

Bases: BaseModel

Modelo de solicitud para registrar un nuevo usuario.

Atributos:
username (str): Nombre de usuario. password (str): Contraseña en texto plano.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class main.**Vehiculo**(**kwargs*)

Bases: *Base*

Modelo ORM que representa un vehículo registrado.

Atributos:
id (int): ID del vehículo. marca (str): Marca del vehículo. modelo (str): Modelo del vehículo. year (int): Año de fabricación. rpm (int): Revoluciones por minuto. velocidad (int): Velocidad actual. vin (str): Número VIN único del vehículo. revision (str): Información de revisión técnica. usuario_id (int): ID del usuario al que pertenece el vehículo.

Relaciones:
usuario (Usuario): Usuario propietario. errores (List[ErrorVehiculo]): Lista de errores asociados. informes_compartidos (List[InformeCompartido]): Informes generados con token público.

class main.**VehiculoEdicion**(*, *marca: str*, *modelo: str*, *year: int*, *rpm: int*, *velocidad: int*, *vin: str*)

Bases: BaseModel

Modelo de solicitud para editar un vehículo existente.

Atributos:
marca (str): Marca del vehículo. modelo (str): Modelo del vehículo. year (int): Año de fabricación. rpm (int): Revoluciones por minuto. velocidad (int): Velocidad actual. vin (str): Número VIN del vehículo.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class main.**VehiculoRegistro**(*, *marca: str*, *modelo: str*, *year: int*, *rpm: int*, *velocidad: int*, *vin: str*, *revision: dict*)

Bases: BaseModel

Modelo de solicitud para registrar un nuevo vehículo.

Atributos:
marca (str): Marca del vehículo. modelo (str): Modelo del vehículo. year (int): Año del vehículo. rpm (int):

RPM del motor. velocidad (int): Velocidad del vehículo. vin (str): Número VIN único del vehículo. revision (dict): Detalles de la revisión técnica (estructura flexible).

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

async main.**crear_informe**(*vehiculo_id: int*, *request:* InformeRequest, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Crea un informe de errores del vehículo y lo envía al email del cliente.

Este endpoint genera un enlace único que da acceso a una vista del informe de diagnóstico del vehículo. Se envía un correo al cliente con dicho enlace.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo del que se desea generar el informe.
- **request** (InformeRequest) – Objeto que contiene el email del cliente.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Mensaje de éxito, token generado y enlace de acceso.

Tipo del valor devuelto
dict

Muestra

- **HTTPException 400** – Si el email no es válido.
- **HTTPException 404** – Si el vehículo no pertenece al usuario.
- **HTTPException 500** – Si ocurre un error al guardar el informe o enviar el correo.

main.**crear_token**(*data: dict*, *expira_en: int = 300*)

Genera un token JWT con los datos proporcionados y un tiempo de expiración opcional.

Parámetros

- **datos** (*dict*) – Datos a incluir en el payload del token.
- **tiempo_expiracion** (*Optional[timedelta]*) – Tiempo personalizado de expiración. Si no se especifica, se usarán 30 minutos por defecto.

Devuelve
Token JWT firmado.

Tipo del valor devuelto
str

Muestra
Exception – Si hay un error al codificar el token.

main.**editar_vehiculo**(*vehiculo_id: int*, *datos:* VehiculoEdicion, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Actualiza los datos de un vehículo existente del usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo a modificar.

- **datos_actualizados** (*VehiculoBase*) – Nuevos datos del vehículo.
- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado.

Devuelve
Mensaje de éxito.

Tipo del valor devuelto
dict

Muestra
HTTPException 404 – Si el vehículo no existe o no pertenece al usuario.

main.**eliminar_vehiculo**(*vehiculo_id: int*, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db:* *Session* = *Depends(get_db)*)

Elimina un vehículo registrado por el usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo a eliminar.
- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.

Devuelve
Mensaje de éxito.

Tipo del valor devuelto
dict

Muestra
HTTPException 404 – Si el vehículo no existe o no pertenece al usuario.

main.**fm = <fastapi_mail.fastmail.FastMail object>**

Configuración de seguridad:

- *SECRET_KEY*, *ALGORITHM* y tiempo de expiración definen la seguridad del JWT.
- *pwd_context* se usa para hashear contraseñas con bcrypt.
- *oauth2_scheme* se usa como dependencia para extraer el token del header Authorization.

main.**get_car_image**(*searchTerm: str*)

Obtiene una URL de imagen representativa de un vehículo usando el término de búsqueda proporcionado.

Este endpoint consulta la API externa de carimagery.com para devolver la URL de una imagen que coincida con el término (por ejemplo, «Toyota Corolla 2020»).

Parámetros
searchTerm (*str*) – Término de búsqueda del vehículo (marca, modelo, año, etc.).

Devuelve
URL de la imagen del vehículo.

Tipo del valor devuelto
str

Muestra
HTTPException 500 – Si hay un error al consultar la API externa.

main.**guardar_errores**(*datos:* ErrorVehiculoRegistro, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Guarda una lista de códigos de error OBD-II (DTC) asociados a un vehículo del usuario autenticado.

Este endpoint es utilizado por el cliente Python que recibe errores del escáner OBD-II y los envía al backend para su almacenamiento.

Parámetros

- **datos** (ErrorVehiculoRegistro) – Objeto que contiene el ID del vehículo y una lista de códigos DTC.
- **usuario** (Usuario) – Usuario autenticado, obtenido desde el token JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Mensaje de confirmación si los errores fueron guardados correctamente.

Tipo del valor devuelto
dict

Muestra

- **HTTPException 400** –
  - Si el ID del vehículo no es válido (no entero o negativo). - Si la lista de códigos está vacía o contiene valores vacíos. - Si hay códigos DTC duplicados.
- **HTTPException 404** – Si el vehículo no pertenece al usuario autenticado.
- **HTTPException 500** – Si ocurre un error inesperado al guardar en la base de datos.

main.**guardar_vehiculo**(*datos:* VehiculoRegistro, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Guarda un nuevo vehículo en la base de datos asociado al usuario autenticado.

Parámetros

- **vehiculo** (*VehiculoBase*) – Datos del vehículo (marca, modelo, año, color, etc.).
- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado, extraído desde el token JWT.

Devuelve
Mensaje de confirmación.

Tipo del valor devuelto
dict

Muestra
HTTPException 401 – Si no se proporciona un token válido.

main.**login**(*datos:* UsuarioLogin, *db: Session = Depends(get_db)*)

Autentica al usuario y devuelve un token JWT válido.

Parámetros

- **datos** (UsuarioLogin) – Credenciales de usuario.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Token JWT si la autenticación fue exitosa.

Tipo del valor devuelto
dict

Muestra

- **HTTPException 400** – Datos inválidos.
- **HTTPException 401** – Usuario no encontrado o contraseña incorrecta.
- **HTTPException 500** – Error al generar el token.

main.**obtener_errores**(*vehiculo_id: int*, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Devuelve todos los errores DTC (códigos OBD-II) asociados a un vehículo del usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo para el que se desean consultar los errores.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Lista de errores registrados.

Tipo del valor devuelto
List[*ErrorVehiculo*]

Muestra
HTTPException 404 – Si no existen errores para ese vehículo.

main.**obtener_usuario_desde_token**(*token: str = Depends(OAuth2PasswordBearer)*, *db: Session = Depends(get_db)*)

Extrae y valida el usuario actual a partir del token JWT proporcionado.

Parámetros

- **token** (*str*) – Token JWT incluido en el encabezado de autorización.
- **db** (*Session*) – Sesión de base de datos.

Devuelve
Instancia del usuario autenticado.

Tipo del valor devuelto
Usuario

Muestra
HTTPException 401 – Si el token es inválido o ha expirado.

main.**obtener_vehiculo**(*vehiculo_id: int*, *usuario:* Usuario = *Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Recupera la información de un vehículo específico registrado por el usuario autenticado.

Parámetros

- **vehiculo_id** (*int*) – ID del vehículo a consultar.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.
- **db** (*Session*) – Sesión activa de la base de datos.

Devuelve
Objeto del vehículo solicitado.

Tipo del valor devuelto
Vehiculo

Muestra
HTTPException 404 – Si el vehículo no pertenece al usuario o no existe.

main.**obtener_vehiculos**(*usuario: Usuario = Depends(obtener_usuario_desde_token)*, *db: Session = Depends(get_db)*)

Obtiene todos los vehículos registrados por el usuario autenticado.

Parámetros

- **db** (*Session*) – Sesión de base de datos.
- **usuario** (Usuario) – Usuario autenticado mediante JWT.

Devuelve
Lista de vehículos asociados al usuario.

Tipo del valor devuelto
List[VehiculoBase]

main.**register**(*datos: UsuarioRegistro*, *db: Session = Depends(get_db)*)

Registra un nuevo usuario en la base de datos.

Parámetros

- **datos** (UsuarioRegistro) – Objeto que contiene el nombre de usuario y la contraseña.
- **db** (*Session*) – Sesión activa de la base de datos, proporcionada por FastAPI.

Devuelve
Un mensaje indicando si el usuario fue registrado exitosamente.

Tipo del valor devuelto
dict

Muestra
HTTPException 400 – Si los campos son inválidos o el nombre de usuario ya existe.

async main.**saludo**()

Devuelve un mensaje simple para verificar que la API está activa.

Este endpoint puede utilizarse para pruebas de conectividad o para confirmar que el backend está desplegado correctamente.

Devuelve
Mensaje de saludo indicando que la API funciona.

Tipo del valor devuelto
dict

main.**ver_informe**(*token: str*, *db: Session = Depends(get_db)*)

Devuelve los datos del informe generado a partir de un token único.

Este endpoint permite el acceso público a un informe de diagnóstico de vehículo mediante un enlace con token generado previamente. No requiere autenticación, pero valida que el token sea legítimo.

Parámetros
token (*str*) – Token único del informe generado.

Devuelve
Información del vehículo (marca, modelo, año, etc.) y lista de errores DTC.

Tipo del valor devuelto
dict

Muestra

- **HTTPException 400** – Si el token no es válido o demasiado corto.
- **HTTPException 404** – Si no se encuentra el informe, el vehículo o los errores asociados.
- **HTTPException 500** – Si ocurre un error inesperado al procesar la solicitud.

main.**verificar_password**(*plain_password*, *hashed_password*)

Verifica si una contraseña en texto plano coincide con su hash almacenado.

Parámetros

- **password_plano** (*str*) – Contraseña proporcionada por el usuario.
- **password_hash** (*str*) – Hash almacenado en la base de datos.

Devuelve
True si coinciden, False si no.

Tipo del valor devuelto
bool

CAPÍTULO 4

Configuración del Proyecto

Este proyecto cuenta con una configuración flexible basada en variables de entorno y herramientas integradas para seguridad y mensajería.

4.1 Base de Datos

- URL: definida en *DATABASE_URL*.
- Motor: MySQL con PyMySQL.
- ORM: SQLAlchemy.
- Sesiones: *SessionLocal()*.

4.2 Correo Electrónico

- Envío mediante *FastAPI-Mail*.
- Configuración por variables de entorno: - Usuario, contraseña, puerto, servidor SMTP.
- Soporta TLS y SSL.

4.3 Seguridad

- *JWT*: para autenticación de usuarios.
- *bcrypt*: para encriptar contraseñas.
- *OAuth2PasswordBearer*: extracción automática del token.

4.4 Dependencias reutilizables

- *get_db()*: genera una sesión DB por request.
- *obtener_usuario_actual()*: extrae el usuario desde el token JWT.

main.**get_db**()

Dependencia de FastAPI para obtener una sesión de base de datos.

Se utiliza con *Depends(get_db)* para abrir una sesión, cederla al endpoint y cerrarla automáticamente.

Índice de Módulos Python

m

Índice

S

T

U

V

Y