Documentación de la Página Web de Gestión de Propiedades y Reservas

Esta aplicación web se basa en Laravel y utiliza Tailwind CSS para el diseño; integra funcionalidades de listado y gestión de propiedades, autenticación de usuarios, carrito de compra para seleccionar propiedades y la realización de reservas a través de una API. Además, se aprovecha Laravel Sanctum para la protección de endpoints API, gestionando tokens de autenticación en la sesión.

1. Descripción General

- **Objetivo:** Permitir la gestión de propiedades (crear, leer, actualizar, eliminar) y ofrecer a los usuarios autenticados la posibilidad de agregar propiedades a un carrito de compra, para posteriormente confirmar reservas.
- **Público:** Usuarios que administran propiedades o interesados en reservar propiedades, quienes deben autenticarse para acceder a funciones avanzadas (añadir al carrito, checkout, etc.).
- **Tecnologías:**
    - **Backend:** Laravel (PHP), con rutas web y API.
    - **Frontend:** Blade templates y Tailwind CSS.
    - **Seguridad:** Laravel Sanctum para autenticación por token y protección de endpoints API.

2. Estructura del Proyecto

a. Rutas

- **Rutas Web (routes/web.php):** Contienen los endpoints para la interfaz de usuario:
    - `/` → Página principal con listado de propiedades.
    - `/properties/{id}` → Detalle de propiedad (Funcionalidades CRUD para actualización y eliminación).
    - `/shopcart` → Vista del carrito de compra con opciones para agregar, eliminar y confirmar reservas.
    - `/login`, `/register` y `/logout` → Autenticación de usuarios.
- **Rutas API (routes/api.php):** Se definen los endpoints para la comunicación con la API:
    - `/api/properties` (GET, POST, PUT, DELETE) → Gestión de propiedades en la API.
    - `/api/reservations` (POST) → Creación de reservas.
    - `/api/login`, `/api/register` y `/api/user` → Funciones de autenticación y obtención de datos del usuario (usando Sanctum).

b. Controladores

- **HomeController:** Muestra la lista de propiedades en la página principal (`home.blade.php`).
- **PropertyController (cliente):** Gestiona la visualización, actualización y eliminación de propiedades en la interfaz del cliente.
- **ShopcartController (cliente):** Maneja el carrito de compra:
    - **add():** Recibe propiedades desde la vista y las almacena en la variable de sesión `cart`.
    - **remove():** Permite eliminar propiedades del carrito.
    - **checkout():** Toma las propiedades del carrito, obtiene el token de sesión, consulta la API para obtener el usuario autenticado y envía una solicitud POST a `/api/reservations` para crear las reservas. Luego vacía el carrito.
- **ReservationController (API):** Recibe la solicitud POST con el array de reservas, valida los datos y crea una reserva por cada propiedad, utilizando el ID del usuario obtenido mediante la autenticación (Sanctum) en la API.
- **LoginController, RegisterController, LogoutController:** Gestionan la autenticación. En particular, el LogoutController elimina el token de sesión (y de ser necesario se revoca el token en la API, si se implementa esa funcionalidad).

c. Vistas

- **home.blade.php:** Muestra el listado de propiedades con la opción de "Añadir al carrito" para usuarios autenticados. Incluye enlaces al carrito, dashboard, y un botón (o enlace a través de un formulario) para hacer logout.
- **property.blade.php:** Detalla la información de cada propiedad y permite actualizar o borrar la propiedad (cuando el usuario está autenticado).
- **shopcart.blade.php:** Visualiza los elementos agregados al carrito; ofrece la posibilidad de eliminar elementos individualmente y de confirmar la reserva (checkout).

3. Flujo de Uso de la Aplicación

1. **Listado y Visualización de Propiedades:**
    - El usuario visita la página principal (`/`), donde se muestra un listado de propiedades.
    - Cada propiedad se muestra en una *card* con detalles básicos (título, imagen, precio y descripción).
2. **Interacción para Usuarios Autenticados:**
    - Si el usuario está autenticado (se verifica la presencia de `api_token` en la sesión):
        - Se muestran opciones adicionales como el "Carrito", "Dashboard" y "Logout".

        - El usuario puede hacer clic en "Añadir al carrito" para incluir propiedades en el carrito, utilizando la función `addToCart()` (con peticiones Fetch mediante JSON).
3. **El Carrito y Checkout:**
    - Los elementos del carrito se almacenan en la sesión bajo la clave `cart`.
    - Desde la vista del carrito (`/shopcart`), el usuario puede eliminar individualmente propiedades o confirmar la reserva.
    - Al presionar "Confirmar reserva", se ejecuta la función `checkout()` del ShopcartController, la cual:
        - Verifica que el carrito no está vacío.
        - Obtiene el token y consulta la API para identificar el usuario.
        - Prepara un array de reservas y envía una solicitud POST a la API (a través del endpoint `/api/reservations`).
        - Si la reserva se procesa correctamente, se vacía el carrito y se redirige al usuario (ahora con un mensaje de éxito).
4. **Autenticación y Logout:**
    - El sistema proporciona rutas para login y registro.
    - La funcionalidad de Logout se implementa mediante un formulario que envía una solicitud POST al endpoint `/logout`, eliminando el token de sesión.

4. Notas Técnicas y Recomendaciones

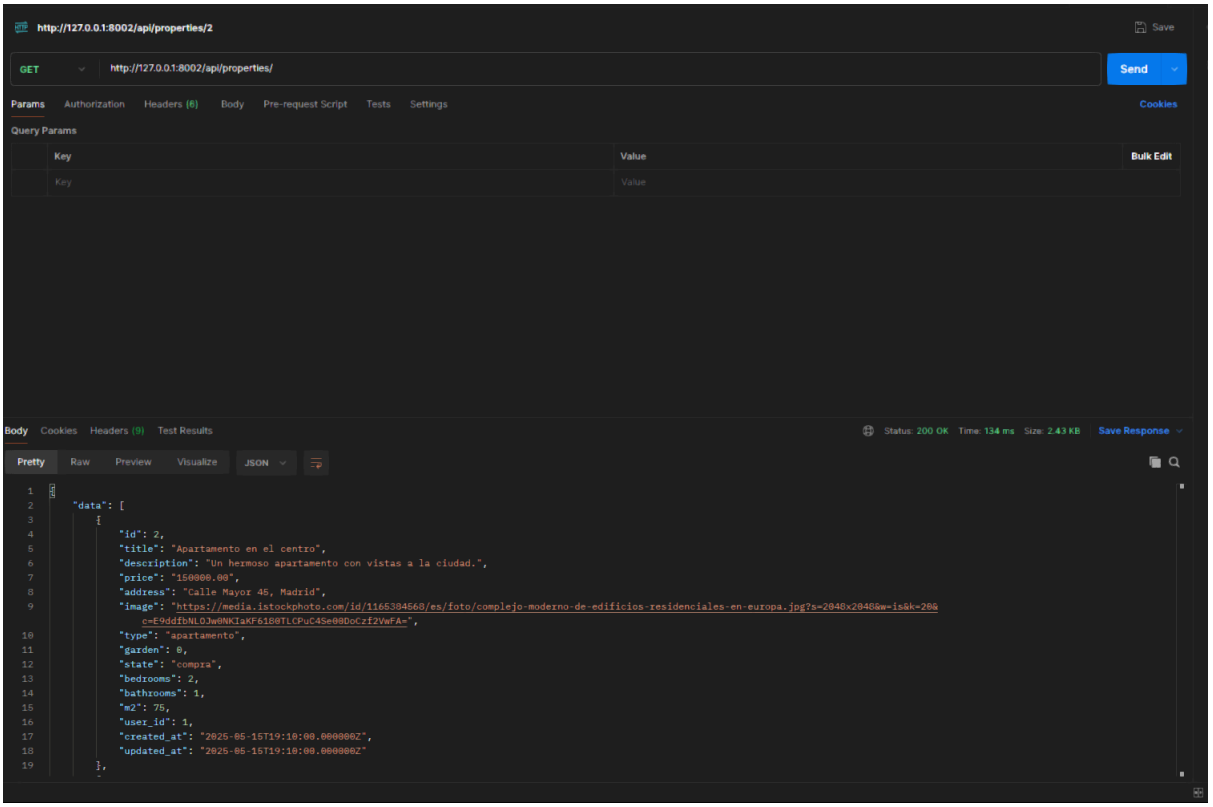
- **Control del Flujo de Autenticación:** Es importante que el header `Accept: application/json` se incluya en las peticiones API (por ejemplo, en Postman y en las solicitudes Fetch) para evitar redirecciones inesperadas.
- **Seguridad y Validación:** La API valida todos los campos requeridos mediante `validate()` y utiliza el token de autenticación para asociar reservas al usuario correcto. Se recomienda manejar errores con respuestas JSON para mantener la consistencia.
- **Manejo de Sesiones y CSRF:** Las peticiones realizadas desde el frontend (especialmente aquellas hechas con Fetch) incluyen el token CSRF para protegerse ante ataques CSRF.
- **Integración Frontend y Backend:** La comunicación entre el cliente y la API está gestionada a través de solicitudes HTTP (GET, POST, etc.), aprovechando el paquete HTTP Client de Laravel. Es importante que se revisen las respuestas (y se manejen correctamente los errores) tanto en el frontend como en el backend para mejorar la experiencia del usuario.

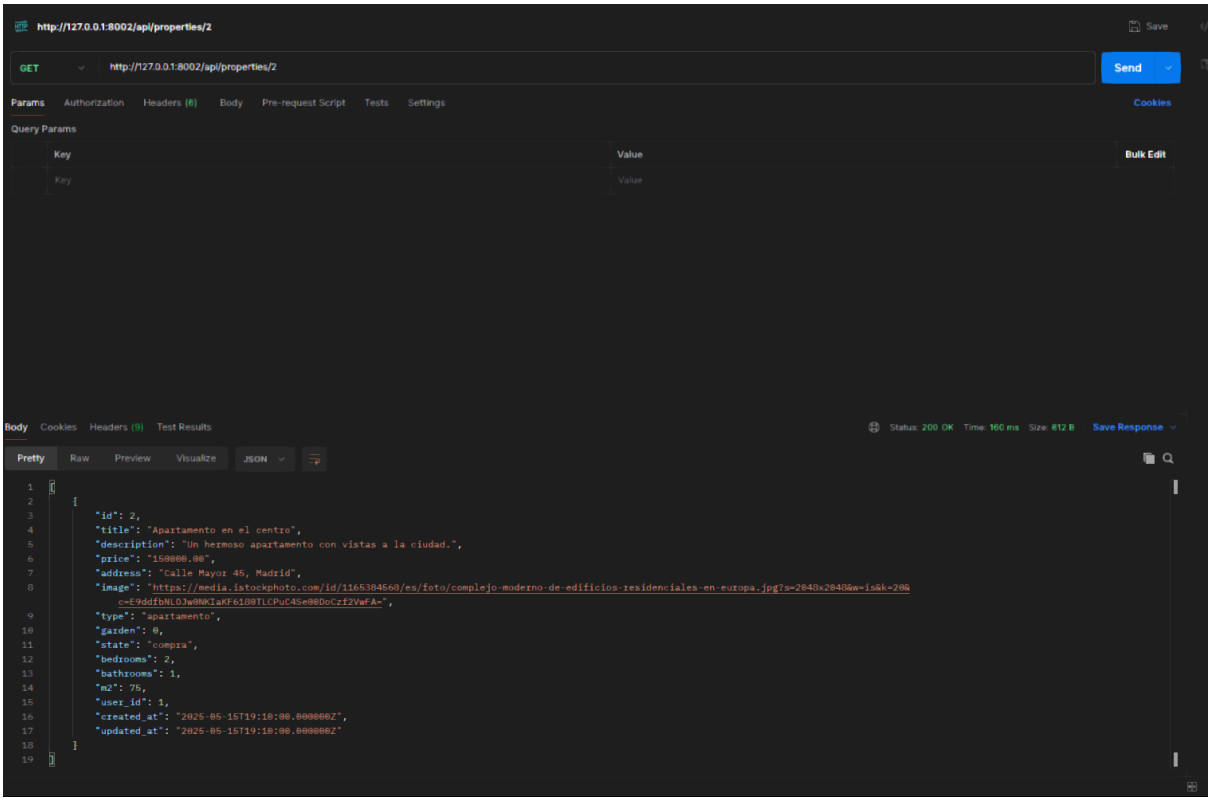
5. Consideraciones Finales

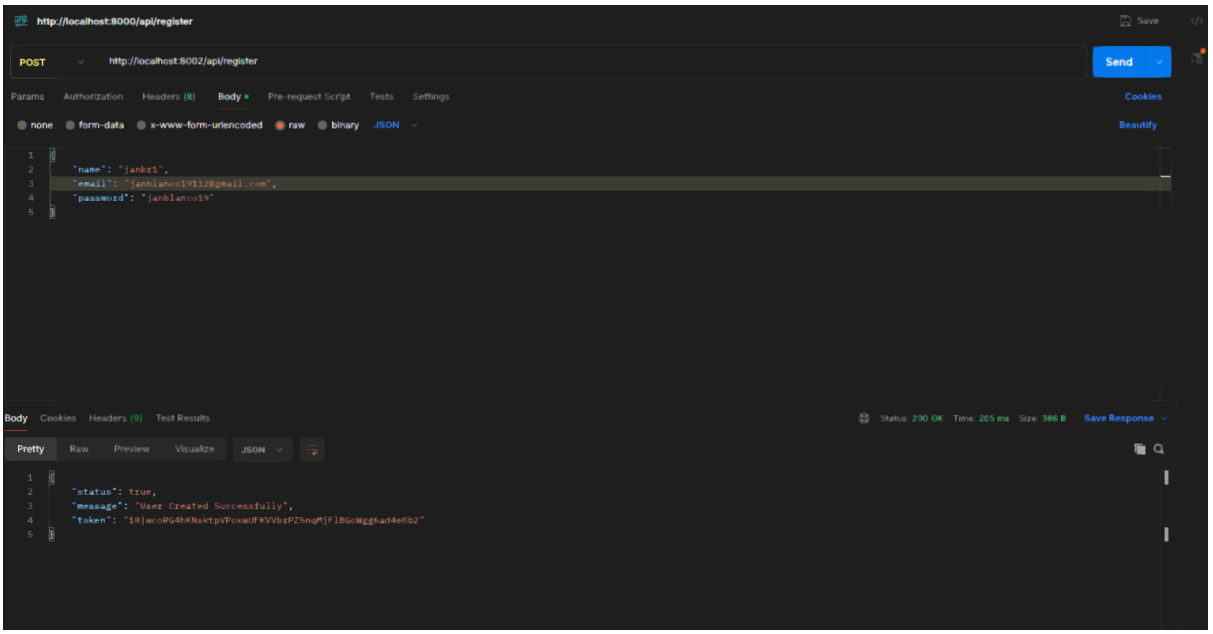
Esta documentación brinda un panorama del funcionamiento general de la aplicación, su estructura y flujo de usuario. Se recomienda mantener el código modular para facilitar futuras ampliaciones o modificaciones y garantizar que la comunicación entre el cliente y la API se maneje mediante respuestas JSON, para evitar errores de formato (como los "Unexpected token" en el caso de obtener HTML).

Si surgen más dudas o se necesitan aclaraciones adicionales, ¡avísame! Espero que esta documentación te sirva como guía para entender y mantener la aplicación.

CAPTURAS DE DEVELOPMENT Y TESTING EN POSTMAN.




http://127.0.0.1:8002/api/properties/2
Save
GET
http://127.0.0.1:8002/api/properties/2
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Bulk Edit
Body
Cookies
Headers (9)
Test Results
Status: 200 OK
Time: 160 ms
Size: 812 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
[
"id": 2,
"title": "Apartamento en el centro",
"description": "Un hermoso apartamento con vistas a la ciudad.",
"price": "150000.00",
"address": "Calle Mayor 45, Madrid",
"image": "https://media.istockphoto.com/id/1165384568/es/foto/complejo-moderno-de-edificios-residenciales-en-europa.jpg?s=2048x2048&w=is&k=20&c=E9ddfbNLOJw0NKIaKF6180TLCPuC4Se08DoCzf2VwFA=",
"type": "apartamento",
"garden": 0,
"state": "compra",
"bedrooms": 2,
"bathrooms": 1,
"m2": 75,
"user_id": 1,
"created_at": "2025-05-15T19:18:00.000000Z",
"updated_at": "2025-05-15T19:18:00.000000Z"
}
]


http://localhost:8000/api/register
Save
POST
http://localhost:8002/api/register
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
JSON
Beautify
{
"name": "janbr1",
"email": "janblanco19112@gmail.com",
"password": "janblanco19"
}
Body
Cookies
Headers (9)
Test Results
Status: 200 OK
Time: 205 ms
Size: 386 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"status": true,
"message": "User Created Successfully",
"token": "18|wcoRG4hKNsktpVPoxwUFKVVbiPZ5nqMjFlBGoWgg6ad4e8b2"
}