Memoria 1 (rev3). Backend - WS CRUD

Investigación

1. Servicios Web: REST y RESTful

Un **Servicio Web (WS)** es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

- **WS tipo REST:** Se basa en una arquitectura cliente-servidor sin estado (*stateless*), donde cada petición debe contener toda la información necesaria para ser procesada.
- **WS tipo RESTful:** Es una implementación rigurosa de los principios REST que aprovecha al máximo los **métodos HTTP** (verbos) para definir las operaciones sobre los recursos, evitando verbos innecesarios en la URL.

2. Operaciones CRUD y Correspondencia HTTP

El acrónimo **CRUD** define las funciones básicas de una base de datos: Crear, Leer, Actualizar y Eliminar. En una API RESTful, estas se corresponden directamente con los métodos HTTP:

Operación CRUD	**Método HTTP**	**Acción en el Servicio**
Create (Crear)	**POST**	Crea un nuevo elemento en una colección.
Read (Leer)	**GET**	Obtiene colecciones o elementos específicos.
Update (Actualizar)	**PUT**	Modifica un elemento existente por su ID.
Delete (Eliminar)	**DELETE**	Elimina un elemento específico de la tabla.

3. Bases de Datos No Estructuradas: MongoDB

A diferencia de las bases de datos relacionales, **MongoDB** es una DB no estructurada (NoSQL) orientada a documentos.

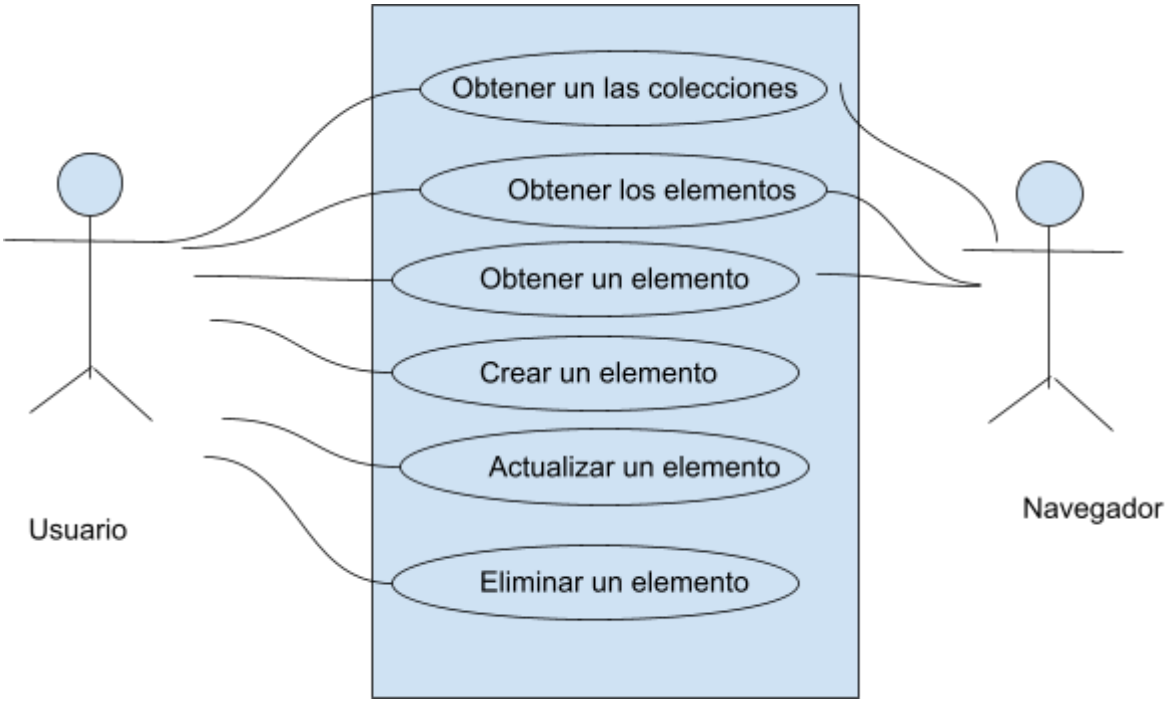
- **Estructura Abierta:** No requiere esquemas o modelos rígidos; cada elemento puede tener una estructura distinta.
- **Formato JSON:** La interacción y el almacenamiento de datos se realizan mediante objetos **JSON**.
- **Conceptos Clave:** Las "tablas" tradicionales se denominan **colecciones** y los "registros" se denominan **elementos** o documentos.

4. Herramientas de Desarrollo y Conectividad

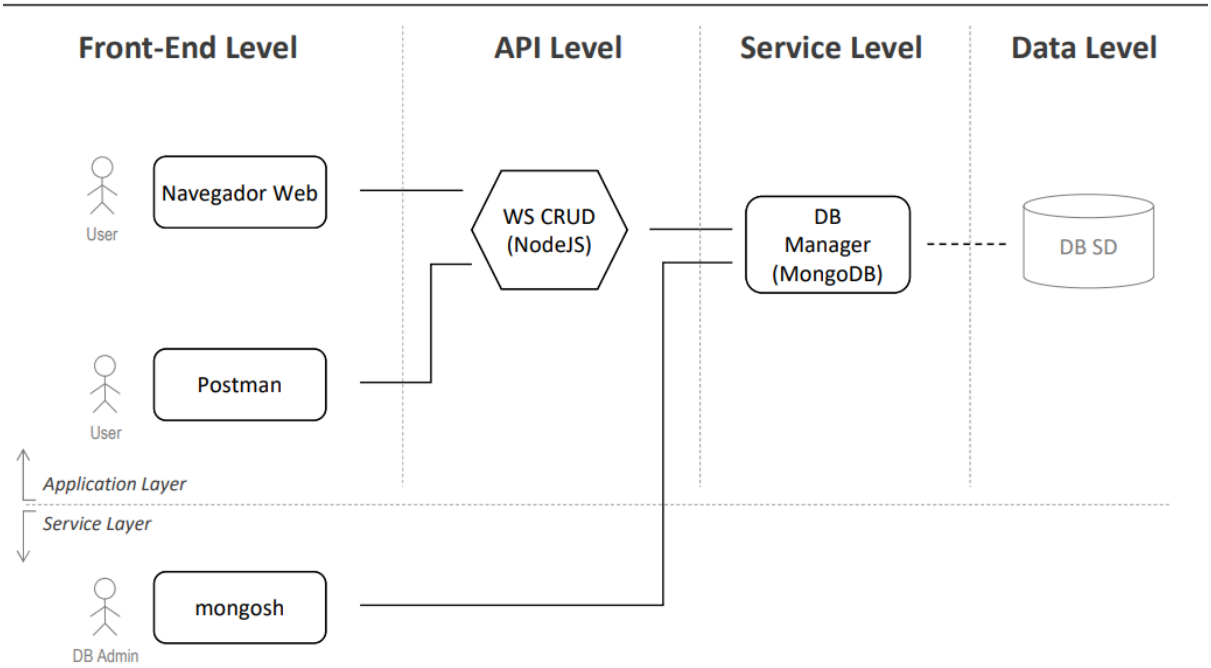
- **mongojs:** Es una biblioteca para **Node.js** que simplifica el acceso a MongoDB, emulando la sintaxis del cliente nativo de la base de datos dentro del código JavaScript.
- **Middleware app.param:** Técnica utilizada para interceptar parámetros en la URL (como :coleccion) y facilitar el soporte multi-colección de forma dinámica.
- **Persistencia con Docker:** El uso de **volúmenes** en contenedores Docker permite que los datos de la DB no se pierdan al detener o reiniciar el servicio

Documentación del Servicio (o Aplicación)

Diagrama de Casos de Uso,



Arquitectura Distribuida del sistema,



Definición del API,

1. Crear producto nuevo (POST)

Crea un nuevo recurso dentro de la colección especificada.

- **Solicitud (HTTP Request):** `POST /api/product`.
- **Cabeceras (Headers):** `Content-Type: application/x-www-form-urlencoded` o `application/json`.
- **Cuerpo (Body):**
  - `name`: "mi producto"
  - `price`: "200"
  - `photo`: "miProducto.png"
  - `category`: "general"
- **Descripción:** Este endpoint inserta un objeto JSON con la estructura abierta en la base de datos "SD". El sistema valida que se incluya al menos el campo `nombre` (o `name`) antes de guardar.

---

2. Obtener todos los productos (GET)

Recupera el listado completo de elementos de la tabla.

- **Solicitud (HTTP Request):** `GET /api/product`.
- **Descripción:** Llama al método `find()` de la biblioteca `mongojs` para devolver todos los registros de la colección en formato JSON.

- **Respuestas:** Código `200 OK` con un array de objetos.

---

3. Obtener un producto (GET por ID)

Busca un elemento específico mediante su identificador único.

- **Solicitud (HTTP Request):** `GET /api/product/{id}`.

- **Parámetros:** `id`: El `ObjectID` generado por MongoDB.
  +**Descripción:** Utiliza `findOne()` para localizar el documento que coincida con el `_id` proporcionado.

---

4. Modificar un elemento (PUT)

Actualiza los campos de un producto existente.

- **Solicitud (HTTP Request):** `PUT /api/product/{id}`.
- **Cuerpo (Body):** Objeto JSON con los campos a actualizar (ej. `{ "price": "250" }`).

- **Descripción:** Emplea el operador `$set` para modificar solo los campos enviados, manteniendo el resto del documento intacto.
- **Respuestas:** Devuelve un objeto con el número de documentos modificados (`nModified`).

---

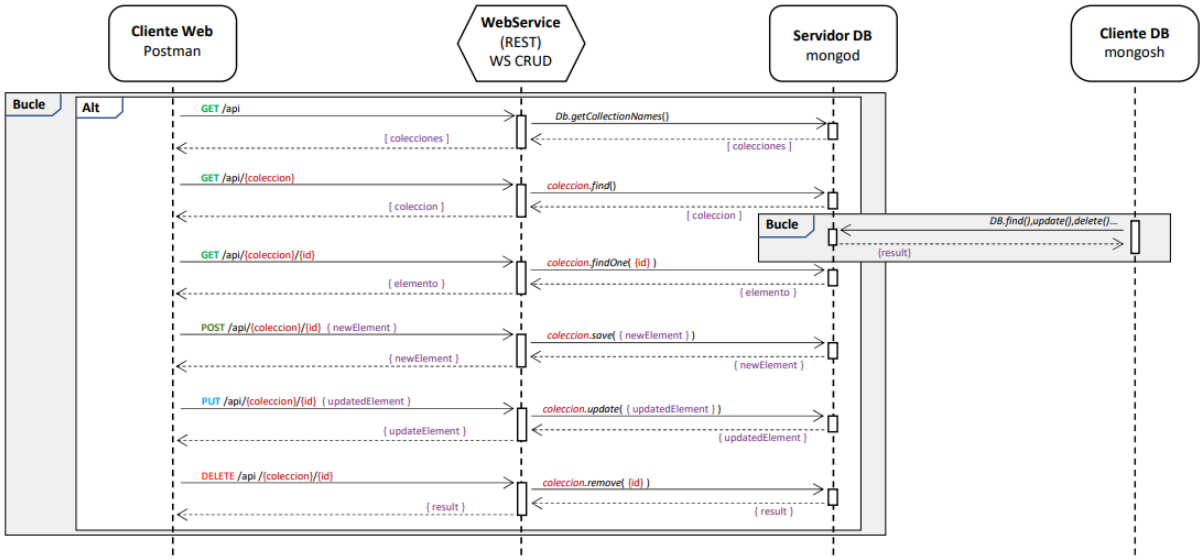
5. Eliminar un elemento (DELETE)

Borra un recurso de la colección de forma permanente.

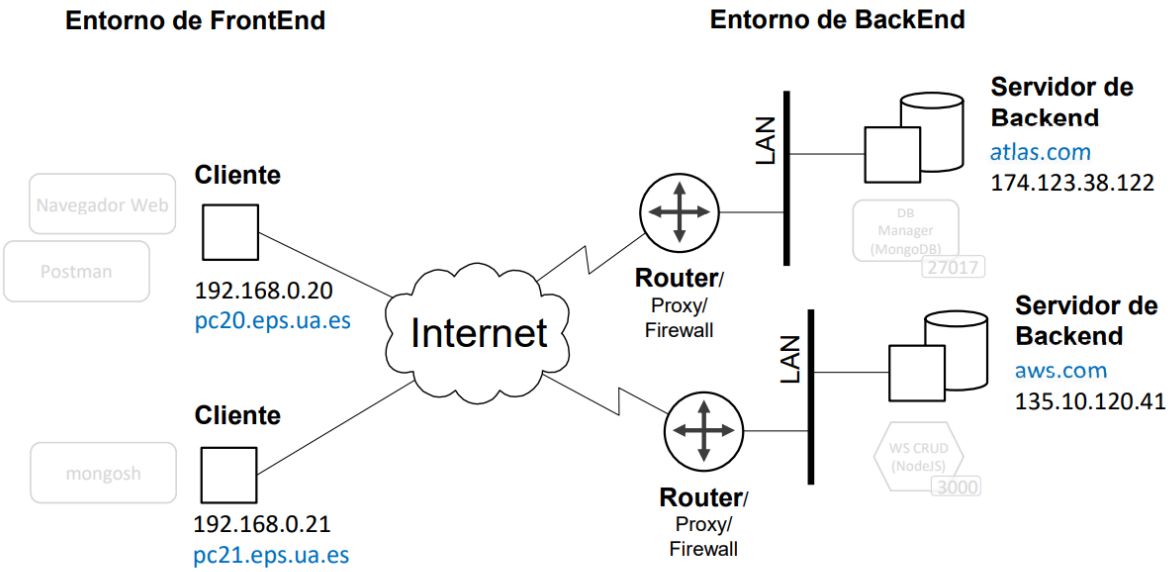
- **Solicitud (HTTP Request):** `DELETE /api/product/{id}`.

- **Descripción:** Invoca el método `remove()` filtrando por el `_id` textual convertido a `ObjectID`.
- **Verificación:** Una llamada posterior de tipo GET a ese mismo ID debería confirmar que el recurso ya no existe

Diagrama de Secuencia,



Arquitectura física del Escenario de Despliegue.



Diseño de la interfaz del Servicio Web

El diseño de la API se basa en una estructura **RESTful**, lo que permite simplificar las rutas delegando la lógica de la operación al método HTTP utilizado.

Verbo HTTP	**Ruta**	**Descripción**
GET	/api	Obtiene el listado de todas las colecciones existentes en la base de datos.
GET	/api/{coleccion}	Recupera todos los elementos contenidos en la colección especificada.
GET	/api/{coleccion}/{id}	Obtiene un único elemento identificado por su {id} dentro de una {coleccion}.
POST	/api/{coleccion}	Crea un nuevo registro o recurso en la colección indicada.
PUT	/api/{coleccion}/{id}	Modifica los datos del elemento con el {id} especificado.
DELETE	/api/{coleccion}/{id}	Elimina de forma permanente el recurso identificado por {id}.

Instalación y ejecución de MongoDB

Preparación: Es necesario instalar herramientas previas como `gnupg` y `curl`, además de importar la clave pública oficial de MongoDB 8.0.

Instalación: Se añade el repositorio oficial al archivo `sources.list.d` y se procede con `sudo apt-get install -y mongodb-org`.

Gestión del servicio: Se utiliza `systemctl` para iniciar el demonio `mongod`.

Verificación: El funcionamiento se comprueba mediante `sudo systemctl status mongod` o accediendo al cliente de terminal `mongosh` en el puerto local 27017.

Alternativa Dockerizada de MongoDB

Se proponen dos enfoques para desplegar el servidor mediante contenedores, lo que garantiza limpieza y portabilidad.

Opción A: Versión Rápida (Sin Persistencia)

Ideal para pruebas rápidas en entornos temporales. Se levanta el contenedor con:

`docker run --name mongodb-container -d -p 27017:27017 mongo:latest`.

Al no definir volúmenes, los datos se pierden al eliminar el contenedor.

Opción B: Configuración con Persistencia (Recomendada)

Utiliza un archivo `docker-compose.yml` para definir un volumen persistente llamado `mongo_data`.

- **Imagen**: Se emplea `mongo:latest`.
- **Puertos**: Mapeo del puerto estándar `27017`.
- **Volumen**: Se mapea `mongo_data` a `/data/db` dentro del contenedor para asegurar que los datos sobrevivan a reinicios.

Implementación del servicio CRUD

La implementación se realiza en **Node.js** utilizando las bibliotecas `express`, `morgan` y, fundamentalmente, `mongojs` por su simplicidad.

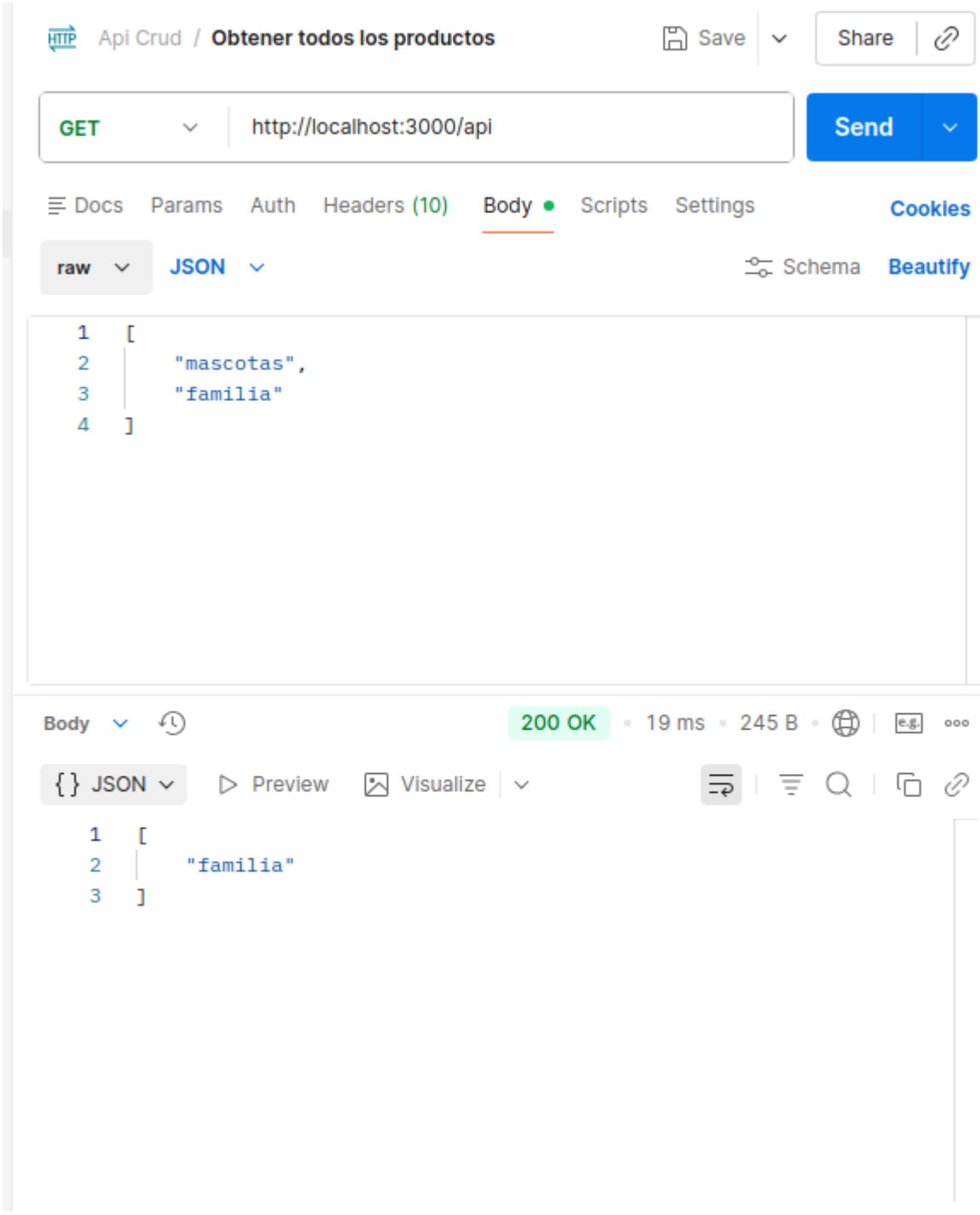
Aspectos Técnicos Relevantes:

- **Conexión**: Se establece el enlace con la base de datos denominada **"SD"**.
- **Soporte Multicolección**: Se implementa un middleware mediante `app.param("coleccion", ...)` que intercepta la ruta y asigna dinámicamente la colección solicitada a `req.collection`.

- **Conversión de ID**: Para las rutas que requieren un identificador (GET por id, PUT, DELETE), se utiliza la función `mongojs.ObjectID` para convertir el string de la URL en un formato compatible con MongoDB.
- **Lógica de Controladores**:
    - **GET**: Utiliza `find()` o `findOne()`.
    - **POST**: Valida la existencia de campos mínimos (como `nombre`) antes de usar `save()`.
    - **PUT**: Emplea el operador `$set` para actualizaciones parciales y seguras.
    - **DELETE**: Ejecuta `remove()` sobre el ID proporcionado

Prueba del servicio

prueba mongodb


Api Crud / Obtener todos los productos
Save
Share
GET
http://localhost:3000/api
Send
Docs
Params
Auth
Headers (10)
Body
Scripts
Settings
Cookies
raw
JSON
Schema
Beautify
1 [
2 "mascotas",
3 "familia"
4]
Body
200 OK
19 ms
245 B
JSON
Preview
Visualize
1 [
2 "familia"
3]