Diseño para una API REST

Contenido

Recursos y Endpoints:

1. Usuarios:

1. *Obtener todos los usuarios:*

- *Método:*
  GET
- *Ruta:*
  /api/users
- *Descripción:*
  Devuelve una lista de todos los usuarios.
- *Código de Respuesta:*
  200 OK cuando la lista de usuarios se obtiene correctamente.

2. *Obtener un único usuario:*

- *Método:*
  GET
- *Ruta:*
  /api/users/{userId}
- *Descripción:*
  Devuelve los detalles de un usuario específico.
- *Código de Respuesta:*
  200 OK cuando el usuario se obtiene correctamente.
  404 Not Found si el usuario no existe.

3. *Crear un usuario:*

- *Método:*
  POST
- *Ruta:*
  /api/users
- *Descripción:*
  Crear un nuevo ususario
- *Código de Respuesta:*
  200 OK cuando el usuario se obtiene correctamente.
  404 Not Found si el usuario no existe.

4. *Actualizar parcialmente un usuario:*
   - *Método:*
     PATCH
   - *Ruta:*
     /api/users/{userId}
   - *Descripción:*
     Actualiza parcialmente los detalles de un usuario.
   - *Código de Respuesta:*
     200 OK cuando el usuario se actualiza correctamente.
     400 Bad Request si los datos proporcionados son inválidos.
     404 Not Found si el usuario no existe.

5. *Actualizar completamente un usuario:*
   - *Método:*
     PUT
   - *Ruta:*
     /api/users/{userId}
   - *Descripción:*
     Actualiza completamente los detalles de un usuario
   - *Código de Respuesta:*
     200 OK cuando el usuario se actualiza correctamente.
     400 Bad Request si los datos proporcionados son inválidos.
     404 Not Found si el usuario no existe.

6. *Eliminar un usuario:*
   - *Método:*
     DELETE
   - *Ruta:*
     /api/users/{userId}
   - *Descripción:*
     Elimina un usuario si no tiene viviendas asociadas.
   - *Código de Respuesta:*
     204 No Content cuando el usuario se elimina correctamente.
     400 Bad Request si el usuario tiene viviendas asociadas.
     404 Not Found si el usuario no existe.

2. Viviendas:
   1. *Obtener todas las viviendas de un usuario:*
      - *Método:*
        GET
      - *Ruta:*
        /api/users/{userId}/houses
      - *Descripción:*
        Devuelve todas las viviendas de un usuario específico.
      - *Código de Respuesta:*
        200 OK cuando se obtienen las viviendas correctamente.
        404 Not Found si el usuario no existe.

2. *Filtrar viviendas de un usuario por ciudad, calle y país:*
   - *Método:*
     GET
   - *Ruta:*
     /api/users/{userId}/houses
   - *Descripción:*
     Devuelve las viviendas de un usuario filtradas por ciudad, calle y país.
   - *Query Parameters:*
     City
     Street
     Country
     Se usan query parameters para el filtrado porque permiten una fácil combinación de criterios de búsqueda y son más flexibles.
   - *Código de Respuesta:*
     200 OK cuando se obtienen las viviendas filtradas correctamente.
     404 Not Found si el usuario no existe.

3. *Crear una vivienda para un usuario:*
   - *Método:*
     POST
   - *Ruta:*
     /api/users/{userId}/houses
   - *Descripción:*
     Crea una nueva vivienda para un usuario.
   - *Cuerpo de la Solicitud:*
     JSON con los detalles de la vivienda.
   - *Código de Respuesta:*
     201 Created cuando la vivienda se crea correctamente.
     400 Bad Request si los datos proporcionados son inválidos.
     404 Not Found si el usuario no existe.

4. *Actualizar una vivienda de un usuario:*
   - *Método:*
     PUT
   - *Ruta:*
     /api/users/{userId}/houses/{houseId}
   - *Descripción:*
     Actualiza los detalles de una vivienda específica de un usuario.
   - *Cuerpo de la Solicitud:*
     JSON con los detalles actualizados de la vivienda.
   - *Código de Respuesta:*
     200 OK cuando la vivienda se actualiza correctamente.
     400 Bad Request si los datos proporcionados son inválidos.
     404 Not Found si el usuario o la vivienda no existen.

5. *Eliminar una vivienda de un usuario:*
   - *Método:*
     DELETE

- *Ruta:*
  /api/users/{userId}/houses/{houseId}
- *Descripción:*
  Elimina una vivienda específica de un usuario.
- *Código de Respuesta:*
  204 No Content cuando la vivienda se elimina correctamente.
  404 Not Found si el usuario o la vivienda no existen.

Diseño en swagger:

En el correo adjunto, esta también metida el YALM con el diseño de esta api y mejorada hasta con más códigos de respuestas.

Conocimientos Adicionales:

¿Qué es OpenAPI?

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-openapi/

Estándar para la descripción de las interfaces de programación que define un formato de descripción abierto e independiente de los fabricantes par los servicios API.

¿Qué es Swagger y Swagger UI (editor online)?

https://www.chakray.com/es/swagger-y-swagger-ui-por-que-es-imprescindible-para-tus-apis/

Swagger es cunado nos referimos a una serie de reglas, especificaciones y herramientas que nos ayudan a documentar nuestras APIs.

Swagger UI es una de las herramientas que nos permite documentar de manera navegable y organizada la información de la api con un fácil acceso.

¿Qué es gherkin y para qué srive?

https://profile.es/blog/que-es-gherkin/

Es un lenguaje de dominio especifico muy parecido al lenguaje natural. Resuelve problemas muy específicos, mas en concreto la comunicación y perfiles técnicos de negocio a la hora de trabajar con desarrollo guiado por comportamiento (BDD).

Elementos del lenguaje:

- Feature: inicio de sesión en la tienda online.
- Scenario: inicio de sesión mediante usuario y contraseña.
- Given: el usuario ha de introducir de forma correcta su usuario y su contraseña, que ha registrado previamente.
- When: el usuario clica sobre el botón de iniciar sesión.
- Then: el usuario puede iniciar sesión de forma correcta.

¿Qué es POSTMAN?

https://www.encora.com/insights/what-is-postman-api-test

Aplicación usada para API testing. Es un cliente HTTP que hace testeos de HTTP request. Tiene una aplicación UI donde se pueden realizar diferentes colecciones para realizar las diferentes pruebas de las responses de los diferentes métodos de los endpoints de la API. Además, puedes añadir factor de seguridad con auth mediante Tokens o certificados.

Métodos:

- GET: Obtener información
- POST: Añadir información
- PUT: Remplazar información
- PATCH: Actualizar cierta informacion
- DELETE: Eliminar información

Códigos de Respuestas:

- 100 series:
  Respuestas temporales
  Ejemplos:
  - 100 CONTINUE
  - 101 SWITCHING PROTOCOLS
  - 102 PROCESSING

- 200 series:
  Respuestas cuando el cliente acepta la petición y el servidor la procesa adecuadamente
  Ejemplo:
  - 200 OK
  - 201 CREATE
  - 202 Accepted
  - 203 Non-Authoritative Information
  - 204 No content
  - 205 Reset de contenido
- 300 series:
  Respuesta relacionada con la URL y su redirección
  Ejemplo:
  - 300 Multiple Choices
  - 301 Moved Permanently
  - 302 Found
  - 303 See Other
  - 304 Not Modified
  - 305 Use Proxy
  - 307 Temporaly Redirect
  - 308 Permanent Redirect
- 400 series:
  Error en respuesta en cliente.
  Ejemplo:
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbibden
  - 404 Not Found
  - 409 Conflict
- 500 series:
  Respuesta fallida por error en serve.
  Ejemplo:
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable
  - 504 Gateway Timeout
  - 599 Network Timeout

¿Qué es SOAP UI? https://www.guru99.com/introduction-to-soapui.html

(leer hasta el punto 4 incluído)

Herramienta de código abierto diseñada para probar servicios web SOAP y REST. Facilita la creación, envío y validación de solicitudes y respuestas de servicios web, permitiendo a los desarrolladores y testers verificar la funcionalidad, rendimiento y seguridad de las APIs.

Soporta pruebas funcionales, de carga y de cumplimiento de estándares. Además, permite la simulación de servicios y la generación de pruebas automatizadas mediante scripts. Utilizado ampliamente en el desarrollo y mantenimiento de aplicaciones web, SOAP UI mejora la eficiencia y precisión en la validación de servicios web.

¿Qué es OAuth 2? ¿Cómo funciona?

OAuth 2.0 es un protocolo de autorización que permite a aplicaciones obtener acceso limitado a los recursos de un usuario sin compartir sus credenciales. Involucra cuatro roles: el usuario (propietario del recurso), la aplicación (cliente), el servidor de recursos y el servidor de autorización.

Cómo funciona:

1. El usuario da su consentimiento para que la aplicación acceda a sus datos.
2. La aplicación obtiene un código de autorización del servidor de autorización.
3. La aplicación intercambia este código por un token de acceso.
4. La aplicación usa el token para solicitar datos al servidor de recursos.

OAuth 2.0 utiliza tokens de acceso para permitir el acceso seguro y controlado a los recursos del usuario.