

# Esta clase va a ser

- grabada

Certificados oficialmente por

 PedidosYa

**CODERHOUSE**

Clase 39. PROGRAMACIÓN BACKEND

# Documentación de API

Certificados oficialmente por



**CODERHOUSE**

# Temario

38

## Seguridad

- ✓ [Cultura de seguridad](#)
- ✓ [OWASP](#)
- ✓ [OWASP Top 10](#)

39

## Documentación de API

- ✓ [Importancia de la documentación](#)
- ✓ [Documentar con Swagger](#)

40

## Testing Unitario

- ✓ Módulos de testing
- ✓ Testing con Mocha
- ✓ Testing con Chai

# Objetivos de la clase

- **Comprender** la importancia de un proceso de documentación
- **Conocer** Swagger como herramienta para documentar.
- **Aplicar** Swagger para documentar nuestros endpoints

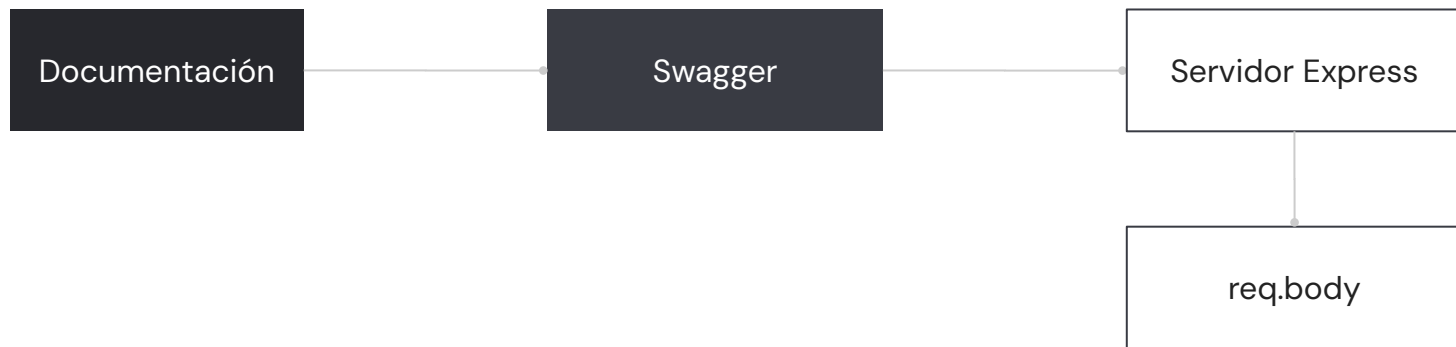
# Glosario

**OWASP:** Open Web Application Security Project. Es un proyecto de código abierto internacional, sin fines de lucro que brinda información referente a la seguridad general de aplicaciones web.

**OWASP top 10:** es un documento que muestra las 10 vulnerabilidades principales y de impacto más crítico en las aplicaciones web a lo largo de un determinado tiempo

**Vulnerabilidad:** cualquier tipo de debilidad en un sistema que permita ser aprovechada por una persona malintencionada para comprometer la seguridad del sistema o el usuario.

## MAPA DE CONCEPTOS



# Documentación

# Documentar

Documentar significa brindar suficiente información sobre algún proceso, con el fin de que éste sea lo suficientemente entendible para quien lo lea.

La documentación puede darse a nivel simple (comentarios sobre el código) o bien a nivel más complejo (herramienta de documentación para un aplicativo en general).





# ¿Por qué es importante documentar?

Vamos a contextualizar algo bastante común en un ambiente laboral no controlado:

Estás trabajando en una empresa que se encarga de la venta de diferentes tipos de mueblería para el hogar, para esto, la empresa sólo procesa los muebles contra solicitud, de manera que manda a hacerlos a su fábrica productora sólo hasta que hayan solicitado una pieza con dichas características. Éste es todo el contexto que necesitamos.

Actualmente sólo trabajan dos desarrolladores en la empresa:

- ✓ Mauricio
- ✓ Tú

Cada quien está desarrollando diferentes módulos para la empresa: Mauricio se está encargando de la gestión de los muebles y el proceso de compra, y tú de usuarios y mantenimiento de CRM y CMS de la empresa.

Mauricio no tiene contexto de lo que estás armando, ni tú tienes contexto de lo que hace él.

# ¡Comienzan los problemas!

Por X cuestiones, Mauricio ha tomado la decisión de romper relaciones con la empresa, así que han decidido contratar a otra persona para apoyarte en el mantenimiento. ¿Cuál es el problema entonces? ¡Te has quedado a cargo de un código que no hiciste y, por lo tanto, del cual no tienes conocimiento!

Te solicitan entonces dar mantenimiento al primer módulo ajeno:

¿Confuso? ¡Espera a ver el código!

*“Reajustar el proceso de gestión de órdenes de venta, para que esta vez permitamos agregar una pila de órdenes de compra premium, las cuales tendrán mayor prioridad que las órdenes de compra normales, y en consecuencia las órdenes de venta deberán primero abastecer a las órdenes premium antes que cubrir las órdenes normales. Sin embargo, si la orden de compra tiene más de 2 semanas desde su solicitud, hay que agregarla a las órdenes prioritarias. Actualmente, tienes la gestión de órdenes de compra normales, te toca agregar la consideración premium. ¿Para cuándo puedes entregarlo?”*

# No puede ser tan malo, ¿verdad?

Lo primero que hay que hacer es entrar al código y entender el paso a paso del proceso que ya existe.

Al menos sabemos que tenemos que procesar órdenes de compra y órdenes de venta.

Así que buscas entre todo el código de la empresa, para poder llegar al código que esperabas encontrar, donde se procesa el cálculo de la repartición de órdenes de compra-venta (posteriormente compra/premium -venta)

**¡Echemos un vistazo para poder entender el proceso!**



## Actividad colaborativa

Leemos el código de la distribución de órdenes, después y comentar que es lo que está haciendo el algoritmo y cómo puedes comenzar a modificarlo.

Duración: **10 minutos**

# Función actual para distribuir órdenes de compra/venta

JS allocationAlgorithm.js X

JS allocationAlgorithm.js > allocate

```
1
2 function allocate (salesOrders,purchaseOrders) {
3   if(!Array.isArray(salesOrders)||!Array.isArray(purchaseOrders)) throw new Error('Invalid data types. Both parameters must be strings');
4   const orderedSales = salesOrders.sort((a,b) => new Date(a.created) - new Date(b.created))
5   const orderedPurchases = purchaseOrders.sort((a,b) => new Date(a.receiving) - new Date(b.receiving))
6   const allocatedOrders = [] ;
7   let totalQuantityInStock = 0;
8   while(orderedSales.length>0&&orderedPurchases.length>0){
9     let currentPurchase = orderedPurchases.shift();
10    totalQuantityInStock+=currentPurchase.quantity;
11    while(totalQuantityInStock>=orderedSales[0].quantity)
12      {
13        const salesOrder = orderedSales.shift();
14        allocatedOrders.push({
15          id:salesOrder.id,
16          date:currentPurchase.receiving
17        })
18        totalQuantityInStock-=salesOrder.quantity;
19        if(orderedSales.length===0) break;
20      }
21  }
22  return allocatedOrders;
23 }
```

# Nuestro trabajo no solo es entender ¡Falta modificar!

Hasta cierto punto, podemos decir que el código se explica por sí solo, ¿verdad?

Sin embargo, para poder comenzar a modificar este código, no basta con “hacerse a una idea de lo que hace”, sino que tenemos que entender a profundidad cómo lo resuelve, así podemos modificar código para agregar nuestra pila de órdenes premium, sin afectar tampoco la lógica principal del flujo.

Para realizar las modificaciones, comienzan a llegarnos muchas dudas.

- ✓ No hay comentarios que me puedan guiar.
- ✓ No hay ningún otro recurso que me apoye a saber cómo funciona o si hay algún punto en el que tenga que tener especial cuidado al momento de modificar.
- ✓ No hay algún ejemplo de input de salesOrders ni purchaseOrders, tendremos que buscarlo probando petición desde frontend (en caso de que se active desde front) o armar nuestro propio mock a partir de la base de datos.

# En conclusión

Notarás cómo un código que no está correctamente documentado, básicamente es una bomba de tiempo que, si en algún momento necesita modificación, presentará múltiples complicaciones para el encargado de dicho mantenimiento.

El ejemplo anterior también puede aplicarse para:

- ✓ **Código de otras áreas de la empresa.**
- ✓ **Código propio que dejamos durante meses** (En ocasiones volver a un código que no hemos tocado en meses o años puede ser un infierno, aun cuando este sea nuestro).

# Swagger



# ¿Qué es Swagger?

Swagger es una herramienta de documentación de código, la cual nos permitirá mantener cada módulo de nuestra API dentro de un **espectro de entendimiento sólido**, es decir, todo se mantendrá en un contexto suficientemente alimentado de información, para poder ser entendido por futuros desarrolladores (O para una versión tuya del futuro), cuando tenga que revisar el código más adelante.

Con esta herramienta podremos hacer nuestra propia **Open API specification**



# Open API specification

También conocida como **Swagger specification**, es un formato de descripción de REST APIs.

Estas especificaciones pueden ser escritas en yaml o en json, y permitirán profundizar sobre un módulo, ruta o esquema específico de nuestra API

Por ejemplo, si queremos realizar la documentación de un módulo de usuarios ¿Qué habría que documentar?

Al desglosar el módulo, podríamos separarlo en la siguiente fórmula:

- ✓ Un esquema que represente al usuario.
- ✓ Un conjunto de rutas referentes a los usuarios.
  - ✓ Posibles queries para cada ruta
  - ✓ Parámetros para las rutas que sean necesarias.
  - ✓ Consideraciones especiales de cada endpoint
- ✓ Un conjunto de posibles Inputs para operaciones con el usuario.



# Antes de comenzar, necesitamos un proyecto a trabajar.

Te presentamos **Adoptme**, un proyecto destinado a la adopción de mascotas, éste, en su versión inicial, cuenta con las siguientes características.

- Un sistema de usuario:
  - Obtiene todos los usuarios.
  - Obtiene un usuario a partir de su Id.
  - Crea un usuario con base en un método de registro.
  - Actualiza un usuario.
  - Elimina un usuario.
- Un sistema de mascotas:
  - Obtiene todas las mascotas.
  - Crea una mascota.
  - Actualiza una mascota.
  - Elimina una mascota.
- Un sistema de adopción:
  - Obtiene todas las adopciones.
  - Obtiene la adopción por su id.
  - Crea una adopción a partir de un usuario y una mascota.



# ¿Qué haremos con este proyecto?

Este proyecto nos será útil para múltiples clases del futuro, de manera que te recomendamos tener el link para clonar el proyecto a la mano. No maneja entornos para mayor comodidad, así que solo tienes que reemplazar el link de mongo en el archivo app.js y ejecutarlo.

Puedes [clonar el proyecto aquí](#)

El objetivo es que en las clases donde utilicemos este proyecto, no tengamos que preocuparnos en armar las soluciones. En el peor de los casos solo tendremos que modificar algunas cosas.



# Checkpoint: Revisión de código

Destina este tiempo para ejecutar el proyecto que acabas de clonar, da una leída a los endpoints y entiende el funcionamiento general. ¡Puedes comenzar a utilizarlo para sentirte más cómodo al documentarlo!

Tiempo estimado: **5 minutos**



# ¡Hora de documentar!

## Instalación de Swagger

Lo primero será tener instalado Swagger, para ello, habrá que instalar dos dependencias:

- ✓ **swagger-jsdoc:** Nos permitirá escribir nuestro archivo .yaml o .json, y a partir de ahí generará un apidoc
- ✓ **swagger-ui-express:** Nos permitirá linkear una interfaz gráfica que represente la documentación a partir de una ruta de nuestro servidor de express.

```
npm install swagger-jsdoc swagger-ui-express
```



# Creamos las opciones principales de swagger

Desglosemos de qué se trata cada propiedad:

- ✓ **openapi:** Sirve para especificar las reglas específicas que seguirá la openapi generada.
- ✓ **title:** Título de la API que estamos documentando.
- ✓ **description:** Descripción de la API que estamos documentando.
- ✓ **apis:** Aquí especificamos la ruta a los archivos que contendrán la documentación. la sintaxis utilizada indica que utilizaremos una carpeta **docs**, la cual contendrá subcarpetas con cada módulo a documentar.

```
const swaggerOptions = {  
  definition: {  
    openapi: '3.0.1',  
    info: {  
      title: "Documentación del poder y del saber",  
      description: "API pensada para clase de Swagger"  
    }  
  },  
  apis: [`${__dirname}/docs/**/*.yaml`]  
}
```



# Conectamos Swagger a nuestro servidor de Express

Con las opciones ya generadas, falta hacer la conexión final. tomaremos las opciones indicadas y colocaremos las siguientes líneas

```
const specs = swaggerJsdoc(swaggerOptions);  
app.use('/apidocs',swaggerUiExpress.serve,swaggerUiExpress.setup([specs]))
```

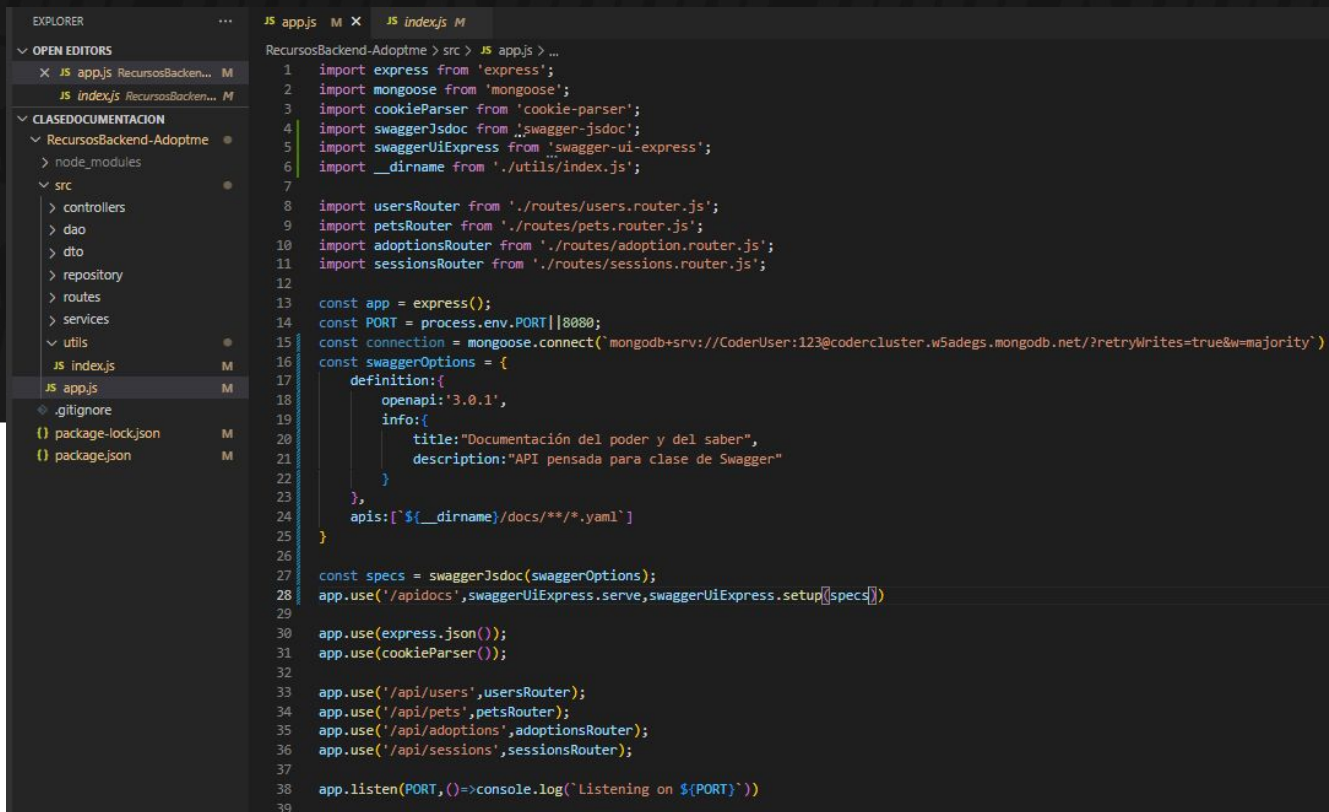
Ejecutemos el servidor y visitemos la ruta que recién definimos.





## APROXIMACIÓN AL PROCESO

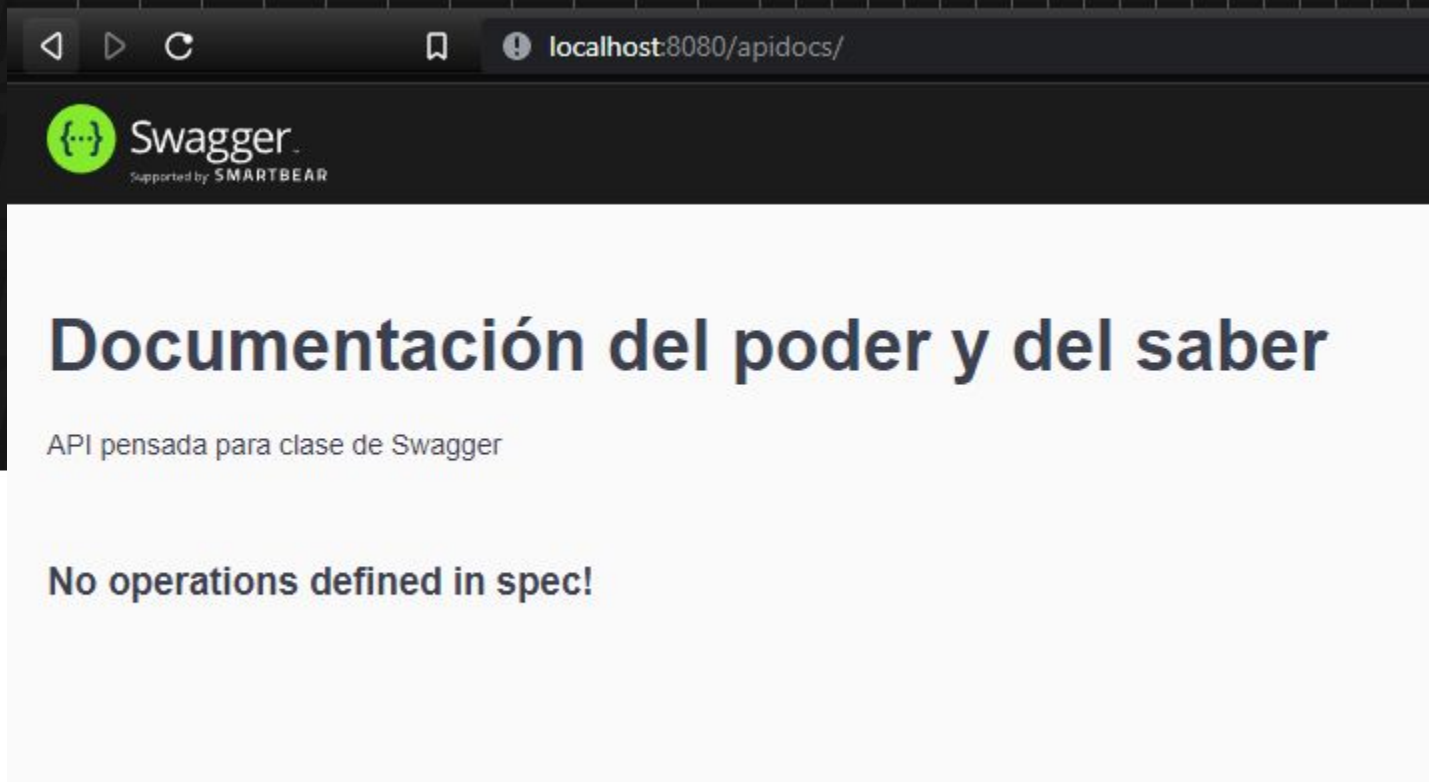
Así debería verse el archivo app.js en panorama completo:



```
1 import express from 'express';
2 import mongoose from 'mongoose';
3 import cookieParser from 'cookie-parser';
4 import swaggerJsdoc from 'swagger-jsdoc';
5 import swaggerUiExpress from 'swagger-ui-express';
6 import __dirname from './utils/index.js';
7
8 import usersRouter from './routes/users.router.js';
9 import petsRouter from './routes/pets.router.js';
10 import adoptionsRouter from './routes/adoption.router.js';
11 import sessionsRouter from './routes/sessions.router.js';
12
13 const app = express();
14 const PORT = process.env.PORT || 8080;
15 const connection = mongoose.connect('mongodb+srv://CoderUser:123@codercluster.w5adegs.mongodb.net/?retryWrites=true&w=majority');
16 const swaggerOptions = {
17   definition: {
18     openapi: '3.0.1',
19     info: {
20       title: "Documentación del poder y del saber",
21       description: "API pensada para clase de Swagger"
22     },
23   },
24   apis: [`${__dirname}/docs/**/*.yaml`]
25 }
26
27 const specs = swaggerJsdoc(swaggerOptions);
28 app.use('/apidocs', swaggerUiExpress.serve, swaggerUiExpress.setup(specs));
29
30 app.use(express.json());
31 app.use(cookieParser());
32
33 app.use('/api/users', usersRouter);
34 app.use('/api/pets', petsRouter);
35 app.use('/api/adoptions', adoptionsRouter);
36 app.use('/api/sessions', sessionsRouter);
37
38 app.listen(PORT, () => console.log(`Listening on ${PORT}`))
39
```



## APROXIMACIÓN AL PROCESO





# Break

¡10 minutos y volvemos!

# Escribiendo nuestro archivo de configuración

# Swagger reportándose al servicio

Una vez llegando a la vista principal, podremos visualizar una interfaz gráfica lista para comenzar a mostrar los elementos que queramos documentar.

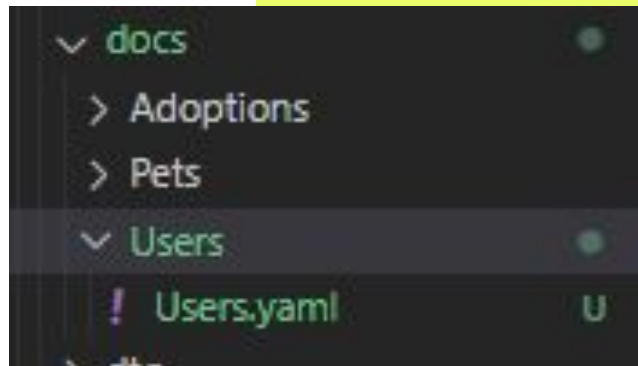
El resto consiste en escribir nuestros .yaml para comenzar con las especificaciones. Una documentación como mínimo debe contar con los siguientes elementos:

- ✓ Schemas.
- ✓ Routes.
- ✓ Inputs.
- ✓ Responses.

Con estos elementos podemos comenzar a estructurar una documentación sólida para un proyecto.

# Crearemos una carpeta para cada respectiva entidad

Según los elementos que haya que documentar, cada entidad tendrá una carpeta con un archivo correspondiente a dicha entidad. Sobre este archivo escribiremos todos los elementos que enlistamos anteriormente.



# Definiendo rutas

La palabra reservada **paths** sirve para colocar cada ruta que se encuentre en nuestro respectivo router. Ésta contendrá, de manera indentada, todos los métodos que estén relacionados con esa ruta (en este proyecto sólo está relacionado con el método get)

Cada método puede tener una breve descripción de la intención del endpoint, así también como una etiqueta para agrupar en la documentación.

Una vez escrita, podemos visualizar los cambios directamente en la interfaz gráfica que tenemos de Swagger

```
! Users.yaml U x
RecursosBackend-Adoptme > src > docs > Users > ! Users.yaml > ...
1  paths:
2    /api/users/:
3      get:
4        summary: Obtiene todos los usuarios
5        tags:
6          - Users
7
```

## Documentación del poder y del saber

API pensada para clase de Swagger

Users

GET

/api/users/ Obtiene todos los usuarios

```
get:
  summary: Obtiene todos los usuarios
  tags:
    - Users
  responses:
    "200":
      description: Los usuarios fueron obtenidos satisfactoriamente de la base de datos
    "500":
      description: Error interno del servidor, usualmente generado por una query defectuosa o un fallo de conexión con la base de datos
```

GET /api/users/ Obtiene todos los usuarios			Try it out
Parameters			
No parameters			
Responses			
Code	Description	Links	
200	Los usuarios fueron obtenidos satisfactoriamente de la base de datos	No links	
500	Error interno del servidor, usualmente generado por una query defectuosa o un fallo de conexión con la base de datos	No links	

# Respuestas

Cada método puede devolver diferentes statuses al momento de procesarse, de modo que es bueno informar qué representa cada uno de estos errores, para ser más informativo.

Podemos notar cómo, si abrimos el endpoint generado por Swagger, vienen dichas descripciones ligadas al método y al status.



GET /api/users/ Obtiene todos los usuarios

Parameters

No parameters

Try it out

Responses

Code	Description	Links
200	Los usuarios fueron obtenidos satisfactoriamente de la base de datos	No links
500	Error interno del servidor, usualmente generado por una query defectuosa o un fallo de conexión con la base de datos	No links

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/users/' \
  -H 'accept: */*'
```

Request URL

http://localhost:8080/api/users/

Server response

Code	Details
200	<p>Response body</p> <pre>{   "status": "success",   "payload": [     {       "role": "user",       "id": "638b0c323f3e4b3be5dc397e",       "first_name": "Mauricio",       "last_name": "Flores",       "email": "correo@correo.com",       "password": "\$2b\$10\$ZjC7z6o1l0FikbKyI1ly.ks410octgt3Ddu/VhIdrsd5I0p2jka",       "v": 0,       "pets": [         {           "_id": "638fe71cc0c8756a598791cf"         }       ]     }   ] }</pre>

# Try it out

¿Notas el botón en la esquina superior derecha que dice “Try it out”? ¡Maravilloso! Swagger no sólo se encarga de informar sobre el endpoint, sino que también nos da la posibilidad de ejecutar dicha consulta al alcance de un botón.

Al presionar el botón, obtendremos más abajo información de la petición que se realizó, además del resultado obtenido.

# Generando componentes

Sería maravilloso contar con un ejemplo de un usuario, es decir, en múltiples ocasiones tenemos problemas para abstraer y proyectar un objeto, utilizando Swagger podemos generar un **schema** de usuario, para tener una idea más clara de **qué es lo que realmente representa esa entidad en nuestro aplicativo**.

Ahora, debajo de **paths** en nuestro archivo users.yaml, procederemos a definir otro elemento llamado **components**.

Los componentes pueden contener:

- ✓ Esquemas de una entidad.
- ✓ Modelos de respuestas.
- ✓ Esquemas de Inputs para un método particular

Definiendo un schema de tipo User. Nota cómo cada propiedad de este esquema cuenta con un tipo de dato y una descripción de lo que representa. Además, un example de cómo debería verse un objeto en la realidad.

```
! Users.yaml U x JS app.js M
RecursosBackend-Adoptme > src > docs > Users > ! Users.yaml > {} components > {} schemas > {} User > {} example > password
description: LOS usuarios fueron obtenidos satisfactoriamente de la base de datos
10 | | | "500":
11 | | | description: Error interno del servidor, usualmente generado por una query defectuosa o un fallo de conexión con la base de datos
12 |
13 components:
14   schemas:
15     User:
16       type: object
17       properties:
18         _id:
19           type: ObjectId
20           description: Id autogenerado de mongo
21         first_name:
22           type: String
23           description: Nombre del usuario
24         last_name:
25           type: String
26           description: Apellido del usuario
27         email:
28           type: String
29           description: Correo del usuario, este campo es único
30         password:
31           type: String
32           description: Contraseña Hasheada del usuario.
33       example:
34         _id: ObjectId("638b8c323f3a4b3be6dc397e")
35         first_name: Mauricio
36         last_name: Espinosa
37         email: correomau@correo.com
38         password: $2b$10$Zjc7z6oTE0FikbhKyIUy.ks4IOoctgt3Ddzu/VwxUrsdSIDp2jka
```

Swagger reconoce el schema, muestra los campos bien explicados, además de un ejemplo de cómo debería verse un valor real.

## Schemas

### User ▾ {

<code>_id</code>	<code>ObjectId</code> Id autogenerado de mongo
<code>first_name</code>	<code>String</code> Nombre del usuario
<code>last_name</code>	<code>String</code> Apellido del usuario
<code>email</code>	<code>String</code> Correo del usuario, este campo es único
<code>password</code>	<code>String</code> Contraseña Hasheada del usuario.

}

example: `OrderedMap { "_id": "ObjectId(\"638b8c323f3a4b3be6dc397e\")", "first_name": "Mauricio", "last_name": "Espinosa", "email": "correomau@correo.com", "password": "$2b$10$Zjc7z6oTE0FikbhKyILUy.ks4IOoctgt3Ddzu/VWxUrsdSIDp2jka" }`

# Referenciando esquemas en nuestro documento

Ahora, tenemos un esquema ya definido en nuestros componentes, ¿cómo hacer para indicar que el método `get` en `api/users/` devuelva un array de usuarios?

Para esto, podemos indicar en nuestro respectivo status un valor **content**, el cual contendrá el cuerpo de ejemplo de la respuesta, usaremos un **\$ref**, con el fin de referenciar a algún componente más del documento.

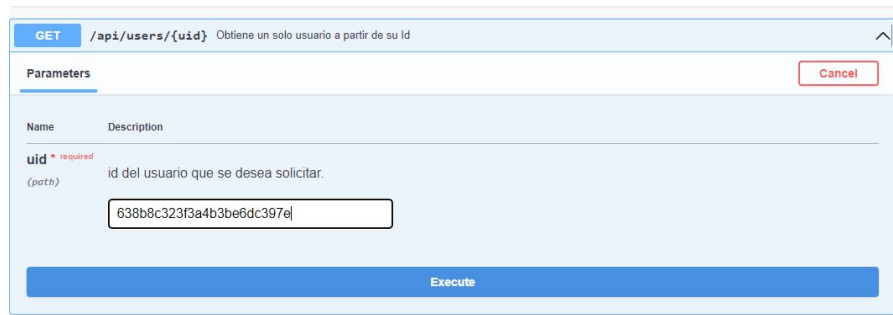
Revisa cómo se quedó la documentación una vez añadido el schema

```
responses:
  "200":
    description: Los usuarios fueron obtenidos satisfactoriamente de la base de datos
    content:
      application/json:
        schema:
          type: array
          items:
            $ref: '#components/schemas/User'
  "500":
    description: Error interno del servidor, usualmente generado por una query defectu
```

# Parametrizando

Notarás que, en su mayoría, los endpoints del router de usuarios tienen un `:uid`, haciendo referencia a la id del usuario. ¿Cómo se agrega ésto a la documentación? Para ello, usaremos la sintaxis `{param}`

Una vez indicado, Swagger habilita un campo para poder pasarlo como parámetro



The screenshot shows the Swagger UI for a GET endpoint `/api/users/{uid}` with the description "Obtiene un solo usuario a partir de su id". Under the "Parameters" tab, a table lists the parameter `uid` as a required path parameter. Below the table, a text input field contains the value `638b8c323f3a4b3be6dc397e`. A blue "Execute" button is at the bottom, and a red "Cancel" button is in the top right corner of the parameters section.

Name	Description
<code>uid</code> * required (path)	id del usuario que se desea solicitar.

```
    "500":  
      description: Error interno del servidor, usualmente g  
/api/users/{uid}:  
  get:  
    summary: Obtiene un solo usuario a partir de su Id  
    tags:  
      - Users  
    parameters:  
      - name: uid  
        in: path  
        required: true  
        description: id del usuario que se desea solicitar.  
        schema:  
          $type: String  
    responses:  
      "200":  
        description: Usuario encontrado  
        content:  
          application/json:  
            schema:  
              $ref: '#components/schemas/User'
```

# Definiendo el req.body

Pensemos en nuestro req.body como un input o un **requestBody** como gustan llamarlo en Swagger. Al final, lo primero es tener este Input inicial separado en un componente nuevo.

En nuestro mismo archivo, debajo del componente "schemas", hay que crear otro componente llamado requestBodies, sobre el cual guardaremos un input "updateUser".

updateUser entonces representará el usuario que llega por req.body, para poder hacer la actualización.

```
email: correo mau@correo.com
password: $2b$10$Zjc7z6oTE0FikbhKyILUy.ks4IOoctgt3Ddzu/VWxUrSD5IDp2jka
requestBodies:
  updateUser:
    type: object
    properties:
      first_name:
        type: String
        description: Nombre del usuario
      last_name:
        type: String
        description: Apellido del usuario
      email:
        type: String
        description: Correo del usuario, este campo es único
      password:
        type: String
        description: Contraseña del usuario, posteriormente será hasheada
    example:
      first_name: Marisol
      last_name: Cadena
      email: correoMarisol@correo.com
      password: 123
```

# Uso de req.body

Una vez que ya está definido el req.body, sólo basta con referenciarlo en el endpoint **put** que armaremos en el mismo path **api/users/{uid}**

Recuerda que, debido a que sólo es un método diferente al **get**, pero sigue siendo la misma ruta, hay que colocar el método put al nivel de get para que la documentación lo reciba correctamente.

```
application/json.  
  schema:  
    $ref: '#components/schemas/User'  
put:  
  summary: Actualiza un usuario a partir de su id  
  tags:  
    - Users  
  parameters:  
    - name: uid  
      in: path  
      required: true  
      description: id del usuario que se desea solicitar.  
      schema:  
        $type: String  
  requestBody:  
    required: true  
    content:  
      application/json:  
        schema:  
          $ref: '#components/requestBodies/updateUser'
```



# Prueba de req.body

Nota algo interesante, cuando definimos un body y presionamos **try it out**, nota cómo en el cuadro de abajo se genera automáticamente un json de ejemplo listo para enviar.

Esto permite entonces que no nos perdamos en la idea de “¿Cómo probar un servicio en particular?” Ya que no tenemos que buscar el input de prueba que corresponde, pues Swagger ya nos genera el objeto de prueba automáticamente, y sólo necesitamos cambiar los campos del body

**PUT** /api/users/{uid} Actualiza un usuario a partir de su id

**Parameters** Cancel

Name	Description
<b>uid</b> * required (path)	id del usuario que se desea solicitar.

uid

**Request body** required application/json

```
{
  "first_name": "Marisol",
  "last_name": "Cadena",
  "email": "correoMarisol@correo.com",
  "password": "123"
}
```

# Importante

Seguramente estarás pensando: “Se hace bastante robusto ese archivo!” y querrás separar la lógica de documentación en más sub-archivos, sin embargo, la lógica de múltiples archivos actualmente se encuentra inestable por parte del repositorio principal, lo que está presentando múltiples errores.

**¡Se recomienda que sigas trabajando con un archivo por entidad!**



## Hands on lab

En esta instancia de la clase **repasaremos** algunos de los conceptos vistos en clase con una aplicación

### ¿De qué manera?

El profesor demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Tiempo estimado: **30 minutos**

# Proceso de adopción desde Swagger

¿Cómo lo hacemos? **Se crearán los endpoints correspondientes en la documentación para llevar a cabo un proceso de adopción**

- ✓ Definir la documentación para mascotas, sólo es necesario definir el método get y el post para crear y ver las mascotas.
- ✓ Definir la documentación para procesar el método "register" que se encargará de crear un usuario. No es necesario implementar el login, sólo el registro.
- ✓ Definir la documentación de adopción, la cual deberá recibir doble parámetro para poder llevar a cabo el proceso de adopción.
- ✓ Corroborar en la base de datos que las entidades se estén creando correctamente.



# Documentar API



# Documentar API

### Consigna

- ✓ Realizar la configuración necesaria para tener documentado tu proyecto final a partir de Swagger.

### Aspectos a incluir

- ✓ Se debe tener documentado el módulo de productos.
- ✓ Se debe tener documentado el módulo de carrito
- ✓ No realizar documentación de sesiones

### Formato

- ✓ Link al repositorio de Github sin `node_modules`

### Sugerencias

- ✓ Recuerda que es un proceso de documentación, ¡Hay que ser lo más claros posibles!

¿Preguntas?

**Muchas gracias.**



# Resumen de la clase hoy

- ✓ Importancia de la documentación
- ✓ Swagger
- ✓ Manejo de CRUD documentado con Swagger

**Opina y valora**  
**esta clase**

**#DemocratizandoLaEducación**