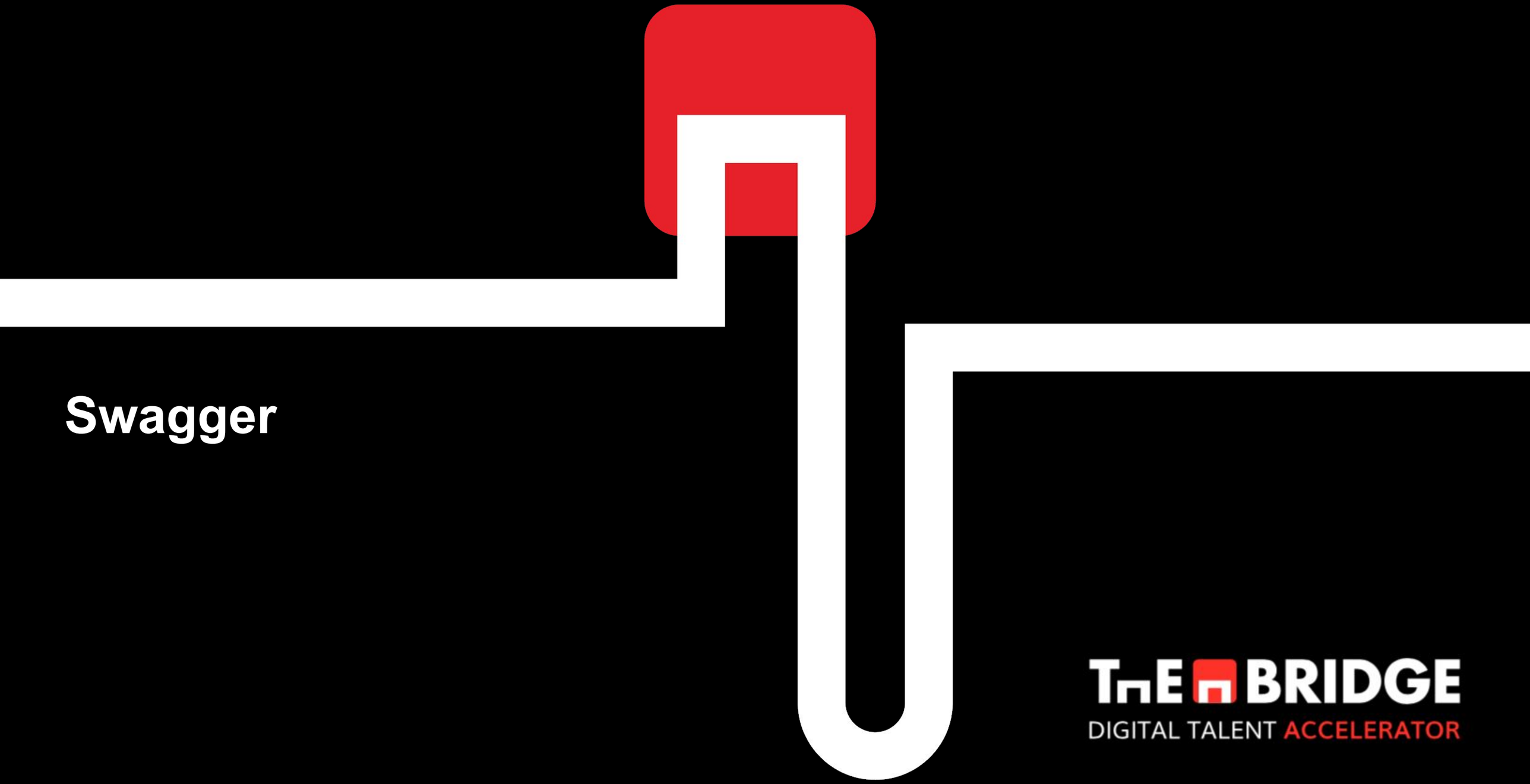
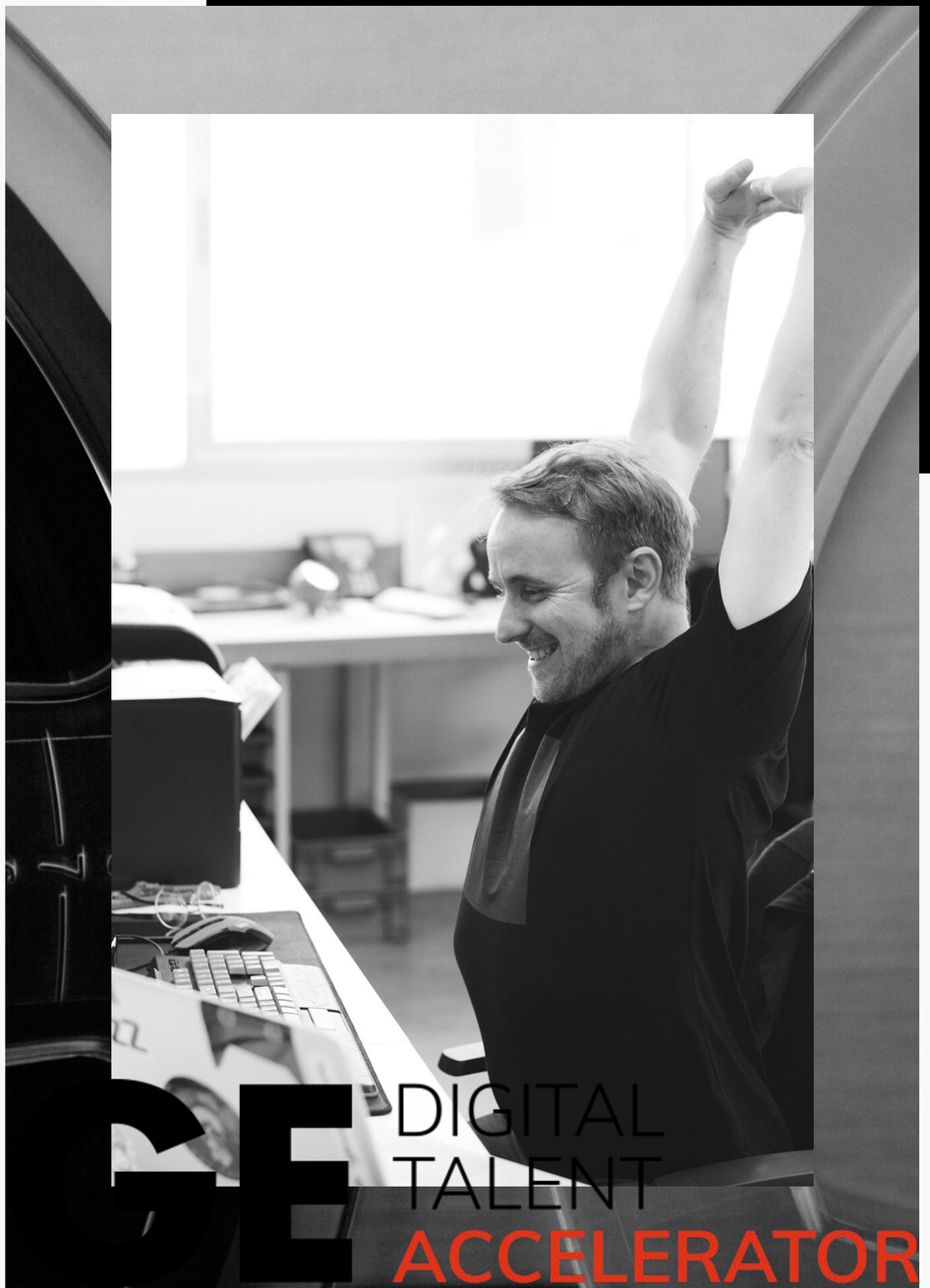
Swagger



Índice

¿Qué es Swagger?

Ejemplo práctico con Swagger



THE BRIDGE DIGITAL TALENT ACCELERATOR

Swagger

Swagger es una herramienta de software que se utiliza para diseñar, construir, **documentar** y **probar tu API.**

Instalación

Swagger ui

```
npm i swagger-ui-express
```

Ejemplo práctico con Swagger

Documentación de la API

En nuestra API, documentamos la información general en la carpeta **docs** en el archivo **basicInfo.js**.

La información general de la API comprende la **versión de openAPI**.

El objeto **info** comprende un **title, description, version**, etc.

La información es muy recomendable para las API disponibles públicamente para mejorar la experiencia del desarrollador.

```
module.exports = {
    openapi:"3.0.3",
    info:{
        version:"1.0.0",
        title:"Tasks",
        description: "Task API"
    }
}
```

docs/index.js

Ahora creamos un archivo **index.js** que exportará todos los archivos que vamos a ir creando en la carpeta **docs:**

```
const basicInfo = require('./basicInfo');
module.exports = {
    ...basicInfo
};
```

index.js principal del proyecto

Para probar lo anterior:

- Importamos swaggerUi
- Importamos el **index.js** de la carpeta **docs**
- Creamos una ruta para poder ver la documentación creada
- En el navegador, abrimos nuestra página de documentación desde **http://localhost:3000/api-docs**

```
const express = require("express");
const app = express();
const PORT = process.env.PORT || 3000;
const mongoose = require("mongoose");
const { MONGO_URI } = require("./config/keys");
const swaggerUI = require('swagger-ui-express')
const docs = require('./docs/index')

. . .

app.use("/tasks", require("./routes/tasks"));

app.use('/api-docs', swaggerUI.serve,swaggerUI.setup(docs))

app.listen(PORT, () => console.log("Servidor levantado en
el puerto" + 3000));
```

Componentes de la API

Los componentes se utilizan para contener diferentes **definiciones reutilizables**. Las definiciones reutilizables involucran esquemas, parámetros... Después de su definición, se accede a los componentes mediante **$ref**.

En nuestra API, documentamos los componentes editando el archivo **docs/components.js** de la siguiente manera:

```
module.exports = {
 components: {
   schemas: {
     task: {
       type: 'object',
       properties: {
         _id: {
           type: 'objectId',
           description: 'task identification number',
           example: '6201064b0028de7866e2b2c4',
         },
         title: {
           type: 'string',
           description: "task's title",
           example: 'Make an excelent readme',
         },
         completed: {
           type: 'boolean',
           description: 'status of the task',
           example: false,
         },
       },
     },
   },
 },
}
```

docs/index.js

Ahora, en nuestro archivo **index.js** importamos components

```
const basicInfo = require('./basicInfo')
const components = require('./components');

module.exports = {
    ...basicInfo,
    ...components
};
```

tasks.js

Creamos el archivo **tasks.js**. En este caso estamos enviando una petición **GET** a **/tasks/getAll**

Para documentar este endpoint, editamos el archivo **/docs/tasks.js** de la siguiente manera:

```
module.exports = {
 paths: {
   '/tasks/getAll': {
     get: {
       tags: {
         Tasks: 'Get Tasks',
       },
       description: 'Get tasks',
       operationId: 'getTasks',
       parameters: [],
       responses: {
         200: {
           description: 'Tasks were obtained',
           content: {
             'application/json': {
               schema: {
                 $ref: '#/components/schemas/task',
               },
             },
           },
         },
       },
     },
   },
 },
}
```

docs/index.js

Ahora, en el archivo **index.js de docs,** añadimos lo siguiente:

```
const basicInfo = require('./basicInfo');
const tasks = require('./tasks');
const components = require('./components');

module.exports = {
    ...basicInfo,
    ...tasks,
    ...components
};
```

Input Model

```
module.exports = {
    components:{
        schemas:{
. . .
            taskInput:{
                type:'object',
                properties:{
                    title:{
                        type:'string',
                        description:"Task name",
                        example:"Make an excelent readme"
                    },
                }
            }
        }
    }
}
```

Componentes de la API

En el caso de querer documentar también endpoints que necesiten de un token, es decir, estar autenticados, editamos el archivo **docs/components.js** de la siguiente manera:

```
module.exports = {
  components: {
    securitySchemes: {
      ApiKeyAuth: {
        type: "apiKey",
        name: "Authorization",
        in: "header"
      }
  },
. . .
```

tasks.js

Editamos el archivo **/docs/tasks.js** y en el endpoint que se necesite estar autenticado añadimos lo siguiente :

```
post: {
        security: [{
          ApiKeyAuth: [ ]
        }],
        tags: {
          Tasks: "Create a task",
        },
        description: "Create a task",
        operationId: "createTask",
. . .
```

Post

Para documentar este endpoint, editar el **/docs/tasks.js** , en este caso estamos enviando una petición **POST** a **/tasks**.

```
. . .
'/tasks/create': {
    post: {
        security: [{ ApiKeyAuth: [ ] }],
        tags: {
          Tasks: "Create a task",
        },
        description: "Create Task",
        operationId: "createTask",
        parameters: [],
        requestBody: {
          content: {
            "application/json": {
              schema: {
                $ref: "#/components/schemas/taskInput",
              },
            },
          },
        },
        responses: {
          201: {
            description: "Task created successfully",
          },
          500: {
            description: "Server error",
          },
        },
      }
}
. . .
```

Id Model

```
module.exports = {
    components:{
        schemas:{
            . . .
            _id:{
                type:'objectId',
                description:"An id of a task",
                example: "6470da3ba50d0ed22dd4ef96"
            },
        }
    }
}
```

Put

Para documentar este endpoint, editar el **/docs/tasks.js** , en este caso estamos enviando una petición **PUT** a **/tasks/id/:_id**.

```
. . .
 "/tasks/id/{_id}": {
      put: {
        security: [{ApiKeyAuth: []}],
        tags: {
          Task: "Update a task",
        },
        description: "Update Task",
        operationId: "updateTask",
        parameters: [
          {
            name: "_id",
            in: "path",
            schema: {
              $ref: "#/components/schemas/_id",
            },
            description: "Id of Task to be updated",
          },
        ],
        requestBody: {
          content: {
            "application/json": {
              schema: { $ref: "#/components/schemas/taskInput" },
            },
          },
        },
        responses: {
          200: { description: "Task updated successfully" },
          404: { description: "Task not found" },
          500: { description: "Server error" },
        },
      },
    },
```

Delete

Para documentar este endpoint, editaremos el archivo **/docs/tasks.js**.
En este caso estamos enviando una petición **DELETE** a **/tasks/id/:_id**.

```
. . .
 delete: {
       security: [{ApiKeyAuth: []}],
         tags: {
           Task: "Delete a task",
         },
         description: "Deleting a Task",
         operationId: "deleteTask",
         parameters: [
           {
             name: "_id",
             in: "path",
             schema: {
               $ref: "#/components/schemas/_id",
             },
             description: "Deleting a Task",
           },
         ],
         responses: {
           200: { description: "Task deleted successfully" },
           404: { description: "Task not found" },
           500: { description: "Server error" },
         },
       },
. . .
```