

Requisitos para el siguiente tutorial

- ✓ Conocimiento en POO
- ✓ Conocimiento en JavaScript
- ✓ Conocimientos básicos en TypeScript
- ✓ Conocimientos básicos en Node JS

PASOS A SEGUIR:

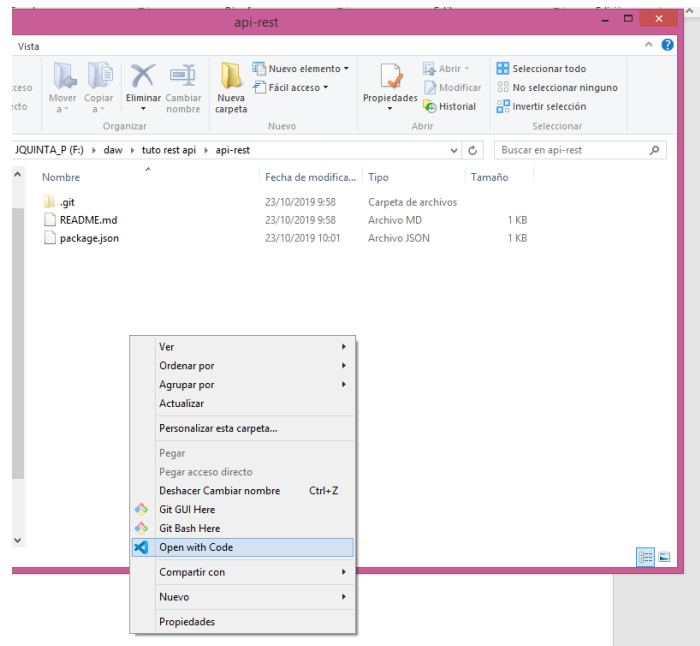
1. Se crea una carpeta para los archivos necesarios.
2. Se abre una consola en la ubicación del proyecto.
3. Se ejecuta el comando "npm init --yes".

```
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

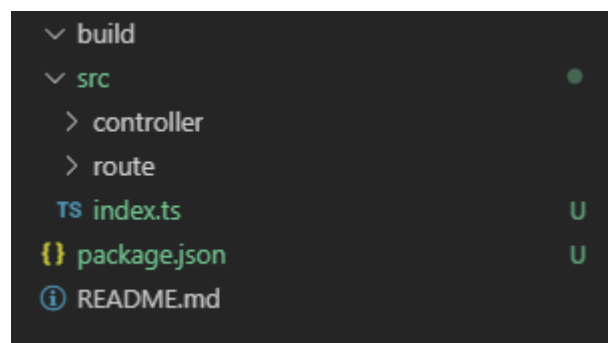
F:\daw\tuto rest api\api-rest>npm init --yes
Wrote to F:\daw\tuto rest api\api-rest\package.json:

{
  "name": "api-rest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/jquintanas/api-rest.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/jquintanas/api-rest/issues"
  },
  "homepage": "https://github.com/jquintanas/api-rest#readme"
}
```

4. Abrir el proyecto con su editor de código favorito. Para este ejemplo se usará visual studio code.



5. Se procede a crear la siguiente estructura:



- ✓ La carpeta build es donde se situará el código transpilado para que Node Js pueda correr el servidor.
- ✓ En la carpeta src se procederá a crear toda la estructura del proyecto con código TypeScript.
- ✓ La carpeta controller es donde se ubicarán los controladores del api.
- ✓ La carpeta route es donde estará la configuración de las rutas del api.
- ✓ El archivo index.ts es el archivo de arranque de nuestra api, aquí se ubicará toda la lógica del sistema.

6. Luego, se procede a configurar el archivo tsconfig.json, que es el archivo de configuración del proceso de transpilación. No olvidar que para dar paso a la creación de dicho archivo debemos tener instalado TypeScript ¹ y usar el comando “tsc –init”.

```
F:\daw\tuto rest api\api-rest>tsc --init
message TS6071: Successfully created a tsconfig.json file.
```

7. A continuación, se configura el archivo tsconfig.json los siguiente.

¹ [Instalación de TypeScript](#)

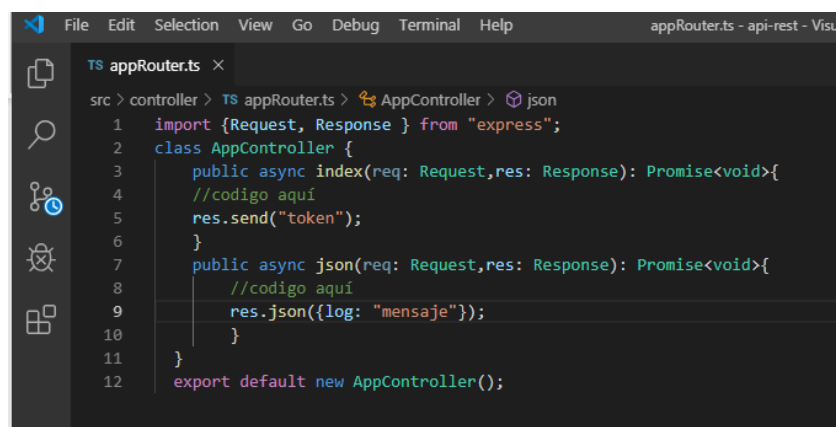
```
"target": "es6",  
"outDir": "./build",
```

8. Luego de esto, se instala express para crear el servidor con el comando “npm i express” y como dependencia de desarrollo “@types/express”.

```
F:\daw\tuto rest api\api-rest>npm i express  
npm notice created a lockfile as package-lock.json. You should commit this file.  
npm WARN api-rest@1.0.0 No description  
  
+ express@4.17.1  
added 50 packages from 37 contributors and audited 126 packages in 25.737s  
found 0 vulnerabilities
```

```
F:\daw\tuto rest api\api-rest>npm i @types/express -D  
npm WARN api-rest@1.0.0 No description  
  
+ @types/express@4.17.1  
added 8 packages from 57 contributors and audited 139 packages in 6.731s  
found 0 vulnerabilities
```

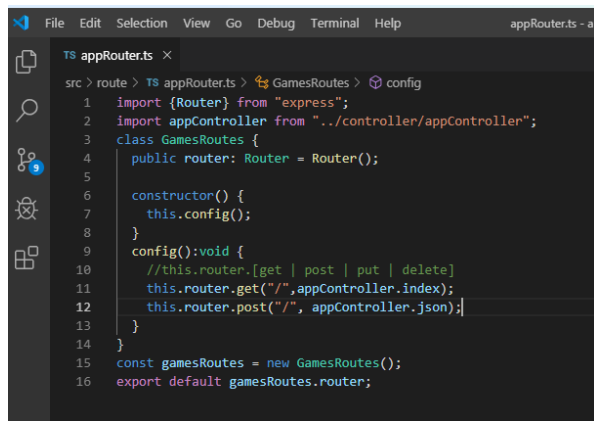
9. después, se debe configurar el controller del api.



```
src > controller > TS appRouter.ts > AppController > json  
1 import {Request, Response } from "express";  
2 class AppController {  
3     public async index(req: Request,res: Response): Promise<void>{  
4         //codigo aquí  
5         res.send("token");  
6     }  
7     public async json(req: Request,res: Response): Promise<void>{  
8         //codigo aquí  
9         res.json({log: "mensaje"});  
10    }  
11 }  
12 export default new AppController();
```

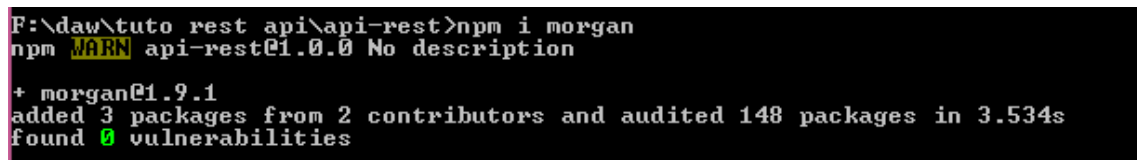
- En este caso solo se creará un método para devolver un mensaje cuando se acceda al mismo, también se puede devolver un json o renderizar un template, pero para nuestro caso en la api solo se usarán res.json(aquí el json).

10. Se procede a configurar el archivo de router, las rutas pueden ser la misma, pero con distintos métodos ejemplo get("/") y post ("/").



```
src > route > TS appRouter.ts > GamesRoutes > config
1 import {Router} from "express";
2 import appController from "../controller/appController";
3 class GamesRoutes {
4   public router: Router = Router();
5
6   constructor() {
7     this.config();
8   }
9   config():void {
10    //this.router.[get | post | put | delete]
11    this.router.get("/", appController.index);
12    this.router.post("/", appController.json);
13  }
14 }
15 const gamesRoutes = new GamesRoutes();
16 export default gamesRoutes.router;
```

11. Para configurar el archivo principal del servidor “index.ts” instalamos un módulo extra “Morgan”, con él podremos ver las rutas e información básica que se solicita a las mismas.



```
F:\daw\tuto rest api\api-rest>npm i morgan
npm WARN api-rest@1.0.0 No description

+ morgan@1.9.1
added 3 packages from 2 contributors and audited 148 packages in 3.534s
found 0 vulnerabilities
```

12. Lo siguiente es configurar el archivo principal



```
File Edit Selection View Go Debug Terminal Help index.ts - api-rest - Visual Studio Code
TS index.ts x
src > TS index.ts > Server > router
1 import express, {Application} from "express";
2 import morgan from "morgan";
3 const bodyParser = require("body-parser");
4 const path = require("path");
5 //imports de rutas personalizadas
6 import appRouter from "../route/appRouter";
7
8
9 class Server {
10 public app:Application;
11 constructor() {
12   this.app = express();
13   this.config();
14   this.router(); }
15
16 config():void {
17   this.app.set("port", process.env.PORT || 3000);
18   this.app.set('view engine', 'ejs');
19   this.app.set('views', path.join(__dirname, 'views'));
20   //static files
21   this.app.use(express.static(path.join(__dirname, '/public')));
22   this.app.use(morgan("dev"));
23   this.app.use(bodyParser.json());
24   this.app.use(bodyParser.urlencoded({ extended: true }));
25 }
26
27
28 router():void {
29   this.app.use("/", appRouter);
30 }
31
32 start(): void {
33   this.app.listen(this.app.get("port"), () => {
34     console.log("server on port: ", this.app.get("port"));
35
36     //db.sequelize.sync();
37   });
38 }
39 }
40 const server = new Server();
41 server.start();
```

12.1. Se realizan los imports.

```
import express, {Application} from "express";
import morgan from "morgan";
const bodyParser = require("body-parser");
const path = require("path");
```

- express y Application corresponden a los módulos requeridos para crear el servidor en express.
- Con Morgan se procede a configurar una interfaz para mostrar por consola las rutas a las que se acceden.
- bodyParser es para poder transmitir JSON por los métodos get, post, etc.
- Finalmente, el módulo Path es para trabajo interno de rutas

13. Para poder empezar con el desarrollo de nuestra api se requiere de una clase que represente a nuestro servidor, la cual será la siguiente:

```
class Server {
  public app:Application;
}
```

14. Luego de esto, se crea el método de configuración del api

```
config():void {
  this.app.set("port", process.env.PORT || 3000);
  //static files
  this.app.use(express.static(path.join(__dirname, '/public')));
  this.app.use(morgan("dev"));
  this.app.use(bodyParser.json());
  this.app.use(bodyParser.urlencoded({ extended: true }));
}
```

- Como se puede observar, tenemos que configurar un puerto en el cual nuestro servidor estará escuchando las peticiones a las rutas configuradas.

```
this.app.set("port", process.env.PORT || 3000);
```

- Cuando se desarrolla en modo local podemos especificar el puerto que más nos convenga, pero cuando se realiza el despliegue de nuestra api en algún VPS (Segarra, 2019) como heroku, por ejemplo, estos servidores nos proporcionan un puerto en concreto por lo cual se accede a este por medio de la variable de entorno (Aosbot, 2019) "process.env.PORT".
- Si nuestra api devuelve algún tipo de archivo estático, necesitamos configurar un directorio de acceso público para que se pueda guardar este tipo de información aquí, lo cual se realiza con:

```
this.app.use(express.static(path.join(__dirname, '/public')));
```

- El "/public" representa la carpeta donde se almacenarán los archivos estáticos, dicha carpeta debe estar ubicada en el directorio "build", ya que aquí es donde se mantendrá la información que será visible para el consumidor de nuestra api.

```
this.app.use(morgan("dev"));
```

- Con esta línea se indica a express "this.app" que utilice el modulo Morgan ² el cual recibe un Sting de entrada, con lo cual podremos observar de manera resumida las rutas que son accedidas en nuestra api.

```
this.app.use(bodyParser.json());
```

² [Documentación de Morgan](#)

- Con esta configuración le indicamos a express que puede recibir por json en la comunicación del api.

```
this.app.use(bodyParser.urlencoded({ extended: true }));
```

- Y en la última configuración, lo que le indicamos a nuestro servidor es que solo recibirá por la url datos planos, ya que al momento no se transmiten archivos por la url en nuestra api.

15. Una vez configurado nuestro servidor debemos asignarle las rutas permitidas de acceso

```
router():void {  
  this.app.use("/",appRouter);  
}
```

- La asignación de rutas cuenta de dos partes, la primera que es la ruta base desde la cual se empezará a trabajar y nuestro archivo donde configuramos las rutas que estarán disponibles en dicha ruta base.

16. Finalmente, para terminar con la programación de nuestra api, debemos realizar el método que levantará nuestro servidor.

```
start(): void {  
  this.app.listen(this.app.get("port"), () => {  
    console.log("server on port: ", this.app.get("port"));  
  });  
}
```

- El método start lo único que realizará es indicarle a nuestro servidor que se ponga a escuchar peticiones en un puerto específico, el mismo que fue configurado previamente y se lo puede acceder mediante el comando "this.app.get('port')", como esto requiere un callback se hará mediante la implementación de una función flecha que lo único que hará es indicar que el servidor escucha en "X" puerto.

17. Métodos adicionales de configuración

```
constructor() {  
  this.app = express();  
  this.config();  
  this.router(); }  
}
```

No olvidemos que todos estos son métodos de la clase `Server` y que para poder usarlos debemos crear una instancia de la misma y llamar a su método `start` que es el encargado de arrancar todo nuestro servidor.

```
const server = new Server();
server.start();
```

Hasta el momento ya tenemos configurado nuestro servidor, pero ¿cómo lo ponemos en ejecución?. Sencillamente, debemos configurar el archivo de arranque de Node JS para que así podamos iniciar nuestra app.

Para lograr esto debemos dirigirnos al archivo “package.json” de nuestra carpeta principal y crear los scripts necesarios para transpilar y otro para arrancar nuestro servidor.

18. El script para transpilar nuestro código.

```
"build": "tsc -w",
```

19. Mientras que el script para arrancar el servidor.

```
"start": "node build/index.js"
```

Con estos últimos ajustes habríamos terminado de configurar toda nuestra api, ahora lo que debemos hacer es ejecutarla mediante la consola y verificar que todo se encuentre bien.

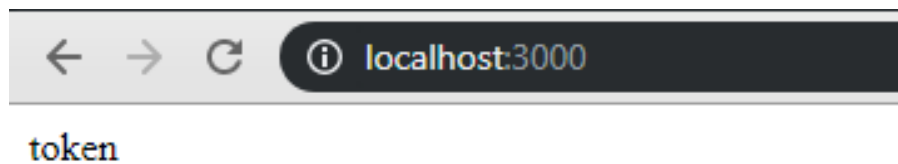
Primero, ejecutamos “npm run build” para transpilar nuestro código en una consola y esperamos a que lo transpile.

```
F:\daw\tuto rest api\api-rest>npm run build
> api-rest@1.0.0 build F:\daw\tuto rest api\api-rest
> tsc -w
[10:45:39 AM] Starting compilation in watch mode...
[10:45:43 AM] Found 0 errors. Watching for file changes.
```


Con el código ya transpilado, ahora debemos proceder a ejecutar nuestro servidor mediante el comando “npm start”.

```
F:\daw\tuto rest api\api-rest>npm start  
  
> api-rest@1.0.0 start F:\daw\tuto rest api\api-rest  
> node build/index.js  
  
server on port: 3000
```

Como podemos observar, nos indica que nuestro servidor está levantado en el puerto 3000, mismo puerto que configuramos previamente, para poder verificar que esté funcionando sin errores abrimos nuestro navegador e ingresamos a la siguiente dirección “<http://localhost:3000/>” debemos obtener el siguiente mensaje en el navegador



Y en la consola podemos observar que ruta fue solicitada.

```
F:\daw\tuto rest api\api-rest>npm start  
  
> api-rest@1.0.0 start F:\daw\tuto rest api\api-rest  
> node build/index.js  
  
server on port: 3000  
GET / 200 6.412 ms - 5  
GET /favicon.ico 404 4.993 ms - 150
```

Si se va a estar en desarrollo del api es tedioso estar pausando y ejecutando a cada momento el comando “npm start” por lo cual se sugiere el uso del middleware nodemon ³.

[Código del repositorio en GitHub.](#)

³ [Página oficial nodemon](#)

Referencias

Aosbot. (22 de Octubre de 2019). *Wikipedia*. Obtenido de Variable de entorno:
https://es.wikipedia.org/wiki/Variable_de_entorno

Segarra, F. (30 de Agosto de 2019). *¿Qué es un VPS? Todo lo que necesitas saber sobre servidores virtuales*. Obtenido de Hostinger: <https://www.hostinger.es/tutoriales/que-es-un-vps>