Node.js

Qué es node.js

- Es un entorno de ejecución de JS que corre sobre Chrome V8
- Open source
- Cross-plataforma
- Lo mantiene OpenJS Foundation
- Permite ejecutar JS del lado del servidor
- No tiene librerías ni APIs para interactuar con el navegador sino con la máquina

¿Para qué se usa?

- Procesamiento de datos en tiempo real
- Herramientas de línea de comando
- Programación del lado del servidor: servidores web, APIs, microservicios, etc

Lo básico

- Instalar localmente
  - nvm
- Docker
  - docker run -it -d --name myNode node[:tag]
- Ejecutar consola (Read Eval Print Loop, REPL)
  - Es como la consola del navegador
  - Permite ejecutar los comandos que aprendimos anteriormente
- node <path_file>
  - Ejecutar un script, tal como el interprete de python

Módulo http

- Modulo http → import http from “http”
  - Parte de la librería estandard de node
  - “http” sin “./” porque no está en la carpeta actual sino en un modulo externo
- let server = http.createServer([options][,requestListener]); → Crea un servidor HTTP.
  - RequestListener → (request, response) => {}
- server.listen(port[,callback]); → El servidor escucha en un puerto y se llama al callback. El servidor sigue escuchando aún cuando el callback termine inmediatamente.

Módulo http

- Manejando peticiones (respuestas)
  - response.writeHead(statudCode[, statusMessage][, headers])
  - response.write(statudCode[, statusMessage][, headers])
  - response.end([data[,encoding][,callback]])

- Mucho más en https://nodejs.org/api/http.html

Módulos externos

- Npm: Node Package Manager
  - Gestiona dependencias para los proyectos de node.
  - Usa un archivo package.json/package-lock.json que contiene metadatos
- npm init [-y] → “type”: “module” (ES6); “commonjs” (por defecto)
- npm install <package>
  - Busca el paquete en el repositorio remoto y lo instala como dependencia del proyecto
  - Node Package Registry (https://www.npmjs.com/)
  - node_modules

Express

- Framework web para node (https://expressjs.com/en/5x/api.html)
- Rápido y minimalista
  - npm intall express –save
- `app.METHOD(PATH, HANDLER)`
  - **app.get(path, callback [, callback …])**
- **app.listen([port[, host[, backlog]]][, callback])**
- **app.all(path, handler) → todos los métodos**
- **Path = ‘*’ → cualquier ruta**

Express

- req.params
  - app.get(‘/users/:id’)
    - curl **-i** “localhost:8080/users/**10**” → {id: 10}
- req.query
  - app.get(‘/users’)
    - Curl **-i** “localhost:8080/users?a**=10&b=Carlos**”→{a: 10, b: “Carlos”}
- node script.js
  - Commonjs → const nombre = require(‘modulo’)
  - ES6 → import {nombre} from ‘modulo’;
    - package.json → type: “module” → ES6

Express - Middlewares

- Función que tiene acceso a la petición y respuesta HTTP
  - Pueden terminar la petición
  - Pueden pasar el control al siguiente middleware (nex())
  - Pueden modificar la petición y/o la respuesta
  - Se ejecutan en el orden en que se agregan a la aplicación
- Nativos:
  - Express.static(), express.json()
- De terceros
  - Cookie-parser, body-parser, morgan y muchos más

Express - Middlewares

- **app.use(middleware);**
  - **(req, res, next) => {**

```
// middleware function implementation
next();
```

}

Separa lógica – primera aproximación

- app.js
  - Express()
  - Importa módulos (router, middleware, etc)
- router.js (mjs)
  - Funciones que gestionan los llamados
- middleware.js (mjs)
  - Middlewares propios

Ejecutar aplicacion

- node [--watch] app.js
  - --watch permite recargar la aplicación automáticamente cuando hay cambios en el código
  - Ver node –help para más opciones de ejecución

express-validator

- npm install express-validator
- (v7) → Node.js 14+, Express.js 4.x
- Puede funcionar con otros frameworks (no express) que modele la petición HTTP similar a express, con las estas propiedades:
  - req.body, req.cookies, req.headers, req.paramas, req.query
- Cadena de validaciones
  - Son middleware, se pueden pasar a los manejadores de ruta
  - Tipos de métodos: Validadores, sanitizadores, modificadores
  - Validator.js → strings

express-validator

- query, body, params, cookies → partes de la petición a validar
- validationResult → array con los errores de la validación

```
import express from "express";
import { query, validationResult } from "express-validator";

const app = express();
app.use(express.json());
app.get("/hello", query("person").notEmpty(), (req, res) => {
  const result = validationResult(req);
  if (result.isEmpty()) {
    return res.send(`Hello, ${req.query.person}!`);
  }

  res.send({ errors: result.array() });
});
app.listen(3000, ()=>{
  console.log('listening!')
});
```

```
root@e6e62fcf5c18:/# curl -l localhost:3000/hello -i
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 86
ETag: W/"56-MMFOaIKN9Sjgfd1F7t7a133i2C4"
Date: Wed, 13 Nov 2024 22:00:12 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"errors":[{"type":"field","msg":"Invalid value","path":"person","location":"query"}]}
```

Routes-Controllers-Services-Models

- Route: Mapea una ruta a un controlador específico para su procesacmiento y que se le responda al cliente
- Controller: Manipula la petición y respuesta (req, res). Valida entradas (query, params, body) y responde al cliente.
- Service: Hace los llamados a la DB y otros cálculos, manipula instancias de los modelos y responde al controlador.
- Model: representa los item que, normalmente, se guardan en la DB

Routes-Controllers-Services-Models

- Puede ser más código, pero es más mantenible. Separa responsabilidades

```
├──models
│   ├──user.model.js
├──routes
│   ├──user.route.js
├──services
│   ├──user.service.js
├──controllers
│   ├──user.controller.js
```

```
this.router = Router();
this.controller = new CourseController();
this.router
  .route("/")
  .get(this.controller.getAll)
  .post(this.controller.createCourse);
        You, 1 second ago • Uncommitted ch
this.router
  .route("/:code")
  .get(this.controller.getOne)
  .put(this.controller.updateCourse)
  .delete(this.controller.deleteCourse);
}
```

```
updateCourse = async (req, res) => {
  const { code } = req.params;
  const { name, credits } = req.body;
  if (!name || !credits) {
    return res.status(400).send({ code: 400, message: "some data missing" });
  }
  const updated = await this.#service.updateCourse(code, name, credits);
  res.status(200).send(updated);
};
```

```
class CourseService {
  getAll = async () => {      You, 7 days ago • Se agregar los 5 recurs
    try {
      console.log("getAll en CursosService");
      const resultados = await new Db().query("SELECT * FROM course");
      return resultados.rows.map(
        ({ name, code, credits }) => new Course(name, code, credits)
      );
    } catch (error) {
      console.log("error al listar cursos", error);
      throw new CustomError(error.code, error.detail);
    }
  };
```

JSON Web Token (JWT)

- IEFT RFC 7519
- Código alfanumérico
- Firmado por una clave privada (prevenir manipulación de datos)
- Stateless session. Se almacena en el cliente y se envía para validar las peticiones
- Header.Payload.Signature
- EyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. → Header
  eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9. → Payload
  TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ → signature

JWT

- Header: Metadatos acerca del token
  - alg: algoritmo con el que se firma (HS256, PS256, PS384, etc)
  - type: tipo de token (JWT por defecto, para JWT standard)
  - kid: identificador de la clave pública con que se firma
- Payload: información real que se va a firmar
  - Tiempo de validez, rol, identificador ,etc
- Signature: Message Authentication Code (MAC)
  - La firma del token, permite validar que no se haya alterado
  - Se necesita el payload y una clave secreta para generarlo/decodificarlo

JWT

- No se guarda el token en el servidor, sino que se verifica que sea correcto el que envía el cliente.
- Está diseñado para ser pequeño y que se pueda transmitir en URL o headers HTTP
- Usa JSON para codificar el header y el payload
- No se encripta, solo se codifica. Hay que tener cuidado con lo que se pone en el payload

JWT- Cómo se usa

- El usuario envía usuario y contraseña
- Se valida que sean correctos
- Se genera un JSON con la información pertinente para usar en la aplicación
- Se firma el JSON con la clave privada y se le manda al cliente el token
- El cliente envía el token para cada consulta que necesite y el servidor verifica que sea válido, usando la misma clave privada

Enlaces de interés

- https://express-validator.github.io/docs/guides/getting-started
- https://blog.angular-university.io/angular-jwt/
- https://jwt.io/ → https://jwt.io/libraries
- https://medium.com/@shubhadeepchat/explaining-json-web-tokens-jwt-with-real-life-examples-62a183889022
- https://medium.com/@diego.coder/autenticaci%C3%B3n-en-node-js-con-json-web-tokens-y-express-ed9d90c5b579
- http://www.cyberchief.ai/2023/05/secure-jwt-token-storage.html
- https://expressjs.com/en/resources/middleware/cors.html