SERVER NODEJS HERRERA

- Inicio con npm init -y
- Instalo Express i dotenv
- En realidad vamos a construir el servidor basado en **CLASES**
- Creo el servidor básico en /app.js (y el script necesario en el package.json)
    - Recuerda que node ya incorpora --watch para estar escuchando los cambios
    - Recuerda también poner type: module para usar imports y exports
- El servidor básico es el que muestran en la documentación
- app.js:

```
import express from 'express'


const app = express()

const port = 3000

app.get('/', (req,res)=>{
    res.send('Hello world')
})

app.listen(port, ()=>{
    console.log(`Server corriendo en puerto ${port}`)
} )
```

- Configuro dotenv. Creo un archivo .env en la raíz y le añado PORT = 8080
- Importo dotenv en app.js
- app.js

```
import express from 'express'
import dotenv from 'dotenv'



const app = express()

dotenv.config()


app.get('/', (req,res)=>{
    res.send('Hello world')
})

app.listen(process.env.PORT, ()=>{
    console.log(`Server corriendo en puerto ${process.env.PORT}`)
} )
```

- De esta manera tenemos un servidor básico. Vamos a construirlo con clases

---

Express basado en clases

- Creo en la raíz /models/server.js
- server.js

```
import express from 'express'

export class Server {

    constructor(){
        this.app = express()
        this.port = process.env.PORT

        this.routes()
    }

    routes(){
        this.app.get('/', (req,res)=>{
            res.send('Hello world')
        })
    }

    listen(){
        this.app.listen(this.port, ()=>{
            console.log(`Server corriendo en puerto ${this.port}`)
        })
    }
}
```

- Creo una nueva instancia de Server en app.js y ejecuto el método listen para levantar el servidor

```
import dotenv from 'dotenv'
import { Server } from './models/server.js'


dotenv.config()

const server = new Server()

server.listen()
```

- Configuro la carpeta pública utilizando este mismo mecanismo
- Creo la carpeta /public/index.html
- Le añado un h1 al index.html sólo para comprobar que renderiza bien (aquí pondría mi sitio web)

- index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Acceso Denegado</title>
</head>
<body>

    <h1>Acceso Denegado</h1>
</body>
</html>
```

- En Server creo el método middlewares (y lo ejecuto en el constructor)
    - La palabra use es clave para saber que es el uso de un middleware
    - Configuro para poder recibir json en middlewares()

```
import express from 'express'

export class Server {

    constructor(){
        this.app = express()
        this.port = process.env.PORT

        //Middlewares ( en el constructor van a ejecutarse al levantar el servidor
)
        this.middlewares()

       //Rutas
        this.routes()
    }

    middlewares(){
        this.app.use(express.static('public')) //Esto es lo que se va a servir en
'/'
        this.app.use(express.urlencoded({extended: false}))
        this.app.use(express.json())
    }

    routes(){
        this.app.get('/api', (res,req)=>{
            res.json({
                message: "get from API"
            })
        })
    }
```

```
    listen(){
        this.app.listen(this.port, ()=>{
            console.log(`Server corriendo en puerto ${this.port}`)
        })
    }
}
```

Peticiones GET-POST-PUT-DELETE

- Instalo ThunderClient (para no usar POSTMAN)
- Devuelvo la respuesta en formato json con res.json
- En el método routes del Server

```
    routes(){
        this.app.get('/api', (req,res)=>{
            res.status(200).json({
                msg: 'get from API'
            })
        })
    }
```

Usando códigos de respuesta HTTP

- Status Code
  - Succesfull:
    - 200 --> OK
    - 201 --> Created
    - 202 --> Accepted
    - 203 --> Non-Authoritative Information
    - 204 --> No content
    - 205 --> Reset Content
    - 206 --> Partial Content
  - Redirection:
    - 300 --> Multiple choices
    - 301 --> Moved Permanently
    - 302 --> Found
    - 303 --> See Other
    - 304 --> Not modified
    - 305 --> Use Proxy
    - 306 --> Unused
    - 307 --> Temporary redirected
  - Client Error:
    - 400 --> Bad Request //server no entendió la url que se le dió

        - 401 --> Unauthorized //necesitas estar autenticado
        - 402 --> Payment Required //no usado realmente
        - 403 --> Forbidden //el server rechaza darte un archivo, la autenticación no te ayudará
        - 404 --> Page Not Found
        - 405 --> Method Not Allowed
        - 406 --> Not Acceptable
        - 407 --> Proxy Authentication Required
        - 408 --> Request TimeOut //el navegador ha tardado demasiado en pedir algo
        - 409 --> Conflict
        - 410 --> Gone
        - 411 --> Length Required
        - 412 --> Precondition Failed
        - 413 --> Request Entity Too Large
        - 415 --> Unsuported Media Type
        - 416 --> Request Range Not Satisfiable
    - Server Error:
        - 500 --> Internal Server Error //algo en el servidor no fue bien
        - 501 --> Not implemented
        - 502 --> Bad Gateway
        - 503 --> Service Unavaliable
        - 504 --> Gateway TimeOut
        - 505 --> HTTP Version Not Supported

CORS Middleware

- No tiene mucho sentido configurarlo internamente dónde todas las peticiones salen del mismo lugar
- Permite proteger el servidor de una manera superficial dándole el acceso sólo a quién yo lo permita
- Es un middleware, lo añado en el server

```
middlewares(){
    this.app.use(express.static('public'))
    this.app.use(express.urlencoded({extended: false}))
    this.app.use(express.json())
    this.app.use(cors()) //
    }
```

- De esta manera nos ahorramos el error de cross-origin

Separar las rutas y el controlador de la clase

- Creo /routes/user.routes.js
- router.routes.js

```
import express from 'express'

const router = express.Router()

router.get('/', (req,res)=> res.json({msg: "get from API"}))
router.post('/', (req,res)=> res.json({msg: "post from API"}))
router.patch('/', (req,res)=> res.json({msg: "put from API"}))
router.delete('/', (req,res)=> res.json({msg: "delete from API"}))

export default router
```

- En el server, al hacer la exportación por default del router, puedo llamarlo como yo quiera al importarlo

```
    routes(){

        this.app.use('/api/usuarios', userRouter)
    }
```

- Ahora desde Thunder Client puedo hacer una petición POST a http://localhost:8080/api/usuarios
- Puedo declarar las rutas en el constructor y usarlas con el this

```
import express from 'express'
import cors from 'cors'
import userRouter from '../routes/user.routes.js'

export class Server {

    constructor(){
        this.app = express()
        this.port = process.env.PORT
        this.usuariosPath = '/api/usuarios'

        //Middlewares ( en el constructor van a ejecutarse al levantar el servidor
)
        this.middlewares()

       //Rutas
        this.routes()
    }

    middlewares(){
        this.app.use(express.static('public')) //Esto es lo que se va a servir en
'/'
        this.app.use(express.urlencoded({extended: false}))
        this.app.use(express.json())
        this.app.use(cors())
    }
```

```
    routes(){

        this.app.use(this.usuariosPath, userRouter)
    }

    listen(){
        this.app.listen(this.port, ()=>{
            console.log(`Server corriendo en puerto ${this.port}`)
        })
    }
}
```

- Puedo separar los controladores en otro archivo
- Creo la carpeta controllers/user.controller.js

```
export const getUser = (req,res) => res.json({msg: "get from API"})
export const addUser = (req,res) => res.json({msg: "post from API"})
export const updateUser = (req,res) => res.json({msg: "put from API"})
export const deleteUser =  (req,res) => res.json({msg: "delete from API"})
```

- En el user.routes

```
import express from 'express'
import {getUser, addUser, updateUser, deleteUser} from
'../controllers/user.controller.js'

const router = express.Router()

router.get('/', getUser) //no ejecuto la función, mando la referencia a la misma
router.post('/', addUser)
router.patch('/', updateUser)
router.delete('/', deleteUser)


export default router
```

- También puedo usar la exportación por defecto de los controladores y usar la dotación de punto
- user.controller.js

```
const getUser = (req,res) => res.json({msg: "get from API"})
const addUser = (req,res) => res.json({msg: "post from API"})
const updateUser = (req,res) => res.json({msg: "put from API"})
const deleteUser =  (req,res) => res.json({msg: "delete from API"})

export default {
    getUser,
    addUser,
```

```
    updateUser,
    deleteUser
}
```

- user.routes

```
import express from 'express'
import userController from '../controllers/user.controller.js'

const router = express.Router()

router.get('/', userController.getUser)
router.post('/', userController.addUser)
router.patch('/', userController.updateUser)
router.delete('/', userController.deleteUser)


export default router
```

Obtener datos de un POST

- Selecciono JSON en el body de Thunder Client
- Escribo un JSON válido

```
{
  "nombre": "Miguel",
  "edad": 41,
}
```

- Para recibir JSON necesito usar un middleware en el server

> this.app.use(express.json())

- Ahora puedo recibir JSON
- Para captar la data uso req.body en el controlador. La extraigo con desestructuración
- user.controller

```
const addUser = (req,res) => {
 const {nombre, edad} = req.body;
 res.json({
    nombre,
    edad
 })
}
```

Parámetros de segmento y query

- Uso :id para utilizar ese parámetro de segmento válido de la url
- user.routes.js

```
router.patch('/:id', userController.updateUser)
```

- La url sería

> http://localhost:8080/api/usuarios/10 10 === /:id del router

- Para obtener el parámetro viene en req.params
- Uso desestructuración para sacar el id
- user.controller

```
const updateUser = (req,res) =>{
    const {id} = req.params
    res.json({
        id    //10
    })
}
```

- Con las query pasa algo parecido

> http://localhost:8080/api/usuarios/10?q=hola&nombre=clara¬apikey=123456

- Express parsea las query
- Puedo especificar en la desestructuración las query y darles un valor en caso de que vengan vacías

```
const updateUser = (req,res) =>{
    const {id} = req.params
    const {q, nombre="no name", apiKey} = req.query

    res.json({
        id,
        q,
        nombre,
        apiKey
    })
}
```
