BackEnd

A. Instalar nodemon. —>npm install -g nodemon

1. Crear server módulo nativo nodejs HTTP

Servidor debe ESCUCHAR el puerto 8080
Servidor nativo Node:

```
const http= require("http");

const server = http.createServer((request, response)=>{
    response.end("Hola mundo, desde el Back");
})

const PORT=8080;

server.listen(PORT, ()=>{
    console.log(`server listening on port: ${PORT}`);
}).on('error', (error)=>{
    console.error(`error in server on port ${PORT}`);
});
```

2. Probar diferencias entre :

```
node nameFile.js.
```

ó

```
nodemon nameFile.js
```

equivalencia

```
node --watch nameFile.js
```

EXPRESS

Framework desarrollo de Servidores

Como express no es nativo necesitamos un package.json para Inicializar un ambiente Node al que le cargaremos la dependencia

```
npm init -y    // crea el package.json
```

Se puede importar una dependencia instalada con
common.js

```
const express= require(“express”);
```

O bien por

module. (En el type del package.json debe ir “type”: ”module”)

```
import express from ‘express’
```

Servidor con express.

```
import express from "express";

const app=express();

app.get("/",(req,res)=>{
    res.send(`Hola mundo inicial`);
})

const PORT=8080;
app.listen(PORT,()=>{console.log(`server active on http://
localhost:${PORT}`)})
```

Objetos req

<< req.params >>

```
app.get("/user/:nombre",(req,res)=>{
    res.send(`Hola ${req.params.nombre}`);
})

app.get("/user/:nombre/:apellido",(req,res)=>{
    res.send(`Hola ${req.params.nombre} ${req.params.apellido}
`);
})
```

Ejemplo con datos:

```
const datos=[
    {"id":1, "name":"name1", "age":31},
    {"id":2, "name":"name2", "age":32},
    {"id":3, "name":"name3", "age":33},
    {"id":4, "name":"name4", "age":34},
]

app.get("/datos",(req,res)=>{
    res.send(datos);
})

app.get("/datos/:id",(req,res)=>{
    const id=req.params.id;
    const usuario=datos.find(elem=>elem.id==id);
```

```
    if(usuario){
        res.send(usuario);
    }else{
        res.status(404).send("No se encontró el usuario");
    }
})
```

<< req.query >>

Para que el servidor cuente con la capacidad de procesar datos complejos desde la URL se debe usar:
`app.use(express.urlencoded({extended:true}))`

Recupera todos los datos que lleguen a la ruta luego del ? Usando clave=valor y concatenándolos con & si se desean más datos

```
app.get("/query",(req,res)=>{
    const consulta=req.query;
    res.send(name);
})
```

Si se desea recuperar sólo ciertos valores se debe desestructurarlos el req.query

```
app.get("/query",(req,res)=>{
    const {name, lastName}=req.query;
    res.send(name, lastName);
})
```

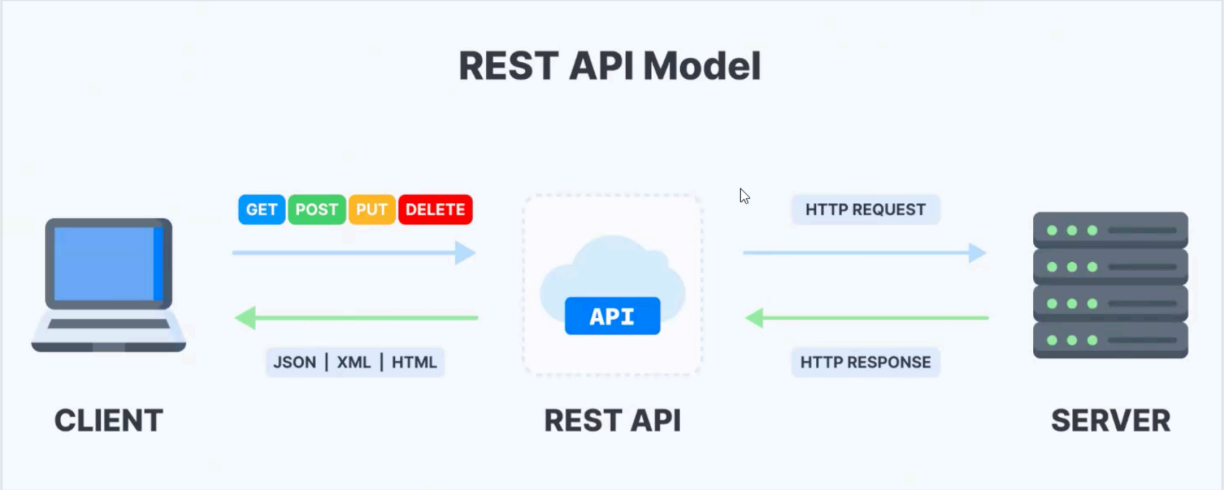
Ejemplo de una url compleja:
`http://localhost:8080/query?name=wilmer&apellido=patiño`

Ejemplo de una consulta:

```
const datos=[
    {"id":1, "name":"name1", "age":31, "group":"uno"},
    {"id":2, "name":"name2", "age":32, "group":"dos"},
    {"id":3, "name":"name3", "age":33, "group":"uno"},
    {"id":4, "name":"name4", "age":34, "group":"dos"},
]

app.get("/query",(req,res)=>{
    const {group}=req.query;
    const filtrados=datos.filter(user=>user.group==group)
    res.send(filtrados)
})
```

Métodos HTTP Get/obtener - Post/Crear Put/Modificar. Delete/eliminar



API Rest

Para que se reconozca la estructura json del body es necesario integrar no solo la línea de codificación sino de parseado:

```
app.use(express.urlencoded({extended:true}))
app.use(express.json())
```

Ejemplo de un servidor:

```
import express form "express";

const app=express();
const server=app.listen(8080,()=>{console.log("server active on
http://localhost:8080")})

app.use(express.urlencoded({extended:true}))
app.use(express.json())

let userDB=[
    {"id":1, "name":"user1", "group":"admin"},
    {"id":2, "name":"user2", "group":"user"},
]

app.get("/api/data",(req,res)=>{
    res.send(userDB);
})

app.post("/api/data",(req,res)=>{
```

```
    let user=req.body;
    if(!user.name || !user.group || !user.id){
        return res.status(400).send("invalid data")
    }else{
        userDB.push(user)
        res.send({"status":"OK", "message":"User created "})
    }
})

app.put("/api/data",(req,res)=>{
    let info=req.query;
    let id=info.id;
    if(info.id){
        userDB.map(item=>{
            if(item.id==id){
                item.name=info.name?info.name:item.name;
                item.group=info.group?info.group:item.group;
                return res.send({"status":"OK", "message":"User
Updated "})
            }
        })
    }
    res.status(400).send({status:"error", error:"Data not
found"})
})

app.delete("/api/data/:id",(req,res)=>{
    let id=req.params.id;
    let lengthInitial=userDB.length;
    userDB=userDB.filter(item=>item.id!=id);
    if(userDB.length==lengthInitial){
        return res.status(400).send({status:"error",
error:"Data not found"})
    }else{
        res.send({"status":"OK", "message":"User deleted "})
    }
})
```

ROUTER

Se crea la carpeta <routes> en src. Dentro un archivo por cada entidad, ejemplo users.router.js:

```
import { Router } from "express";

const router=Router();

let userDB=[
    {"id":1, "name":"user1", "group":"admin"},
    {"id":2, "name":"user2", "group":"user"},
]

router.get("/",(req,res)=>{
    res.send(userDB);
})

router.get("/:id",(req,res)=>{
    const id=req.params.id;
    const user=userDB.find(item=>item.id==id)
    res.send(user);
})
...
export default router;
```

En el archivo app se importa el archivo user.router.js

```
import userRouter from "./routes/user.router.js";
…
app.use('/api/users',userRouter)  //ruta de los end-points
```

Para disponer de una carpeta pública se requiere crear la siguiente línea (y la carpeta public en la raíz del proyecto)

```
app.use(express.static('public'));
```

Express busca los archivos relativos al directorio estático, por lo que el nombre del directorio estático no forma parte del URL.

Se pueden servir los archivos directamente desde la raíz ej: http://localhost:8080/algo.txt