PJS2.

Creació d'una API amb Node.js Express.js i Knex.js

1.- Introducció

Node.js, Express i Knex són eines comuns en el desenvolupament d'aplicacions web, especialment aquelles basades en JavaScript i Node.js.

1. **Node.js**: És un entorn d'execució de JavaScript al costat del servidor. Permet als desenvolupadors escriure codi JavaScript per al servidor, la qual cosa els permet construir aplicacions web escalables i d'alt rendiment.
2. **Express.js (o simplement Express)**: És un marc d'aplicació web per a Node.js. Express simplifica el procés de desenvolupament d'aplicacions web proporcionant una sèrie de funcions i utilitats per a gestionar peticions HTTP, definir rutes i configurar middleware.
3. **Knex.js (o simplement Knex)**: És un generador de consultes SQL per a Node.js. Knex proporciona una forma senzilla i flexible d'interactuar amb bases de dades relacionals des d'aplicacions Node.js. Permet als desenvolupadors escriure consultes SQL utilitzant JavaScript, proporcionant un nivell d'abstracció sobre les diferències entre els diferents motors de bases de dades com PostgreSQL, MySQL, SQLite, etc.

En resum Node.js és l'entorn d'execució de JavaScript al costat del servidor, Express.js és un marc web que s'executa sobre Node.js per simplificar el desenvolupament d'aplicacions web, i Knex.js és una biblioteca que facilita la interacció amb bases de dades relacionals des d'aplicacions Node.js.

2.- API- API Rest

Una **API** (Interfície de Programació d'Aplicacions) és un conjunt de definicions i protocols que permeten la comunicació entre diferents aplicacions de programari. En altres paraules, una **API defineix com diferents components del programari han de comunicar-se entre ells**.

Una **API REST** (Representational State Transfer) és un tipus d'API que **utilitza els protocols HTTP** per a les operacions **CRUD** (Crear, Llegir, Actualitzar, Esborrar) sobre els recursos. Els recursos s'identifiquen mitjançant **URIs** (**Uniform Resource Identifiers**), i les operacions sobre aquests recursos es realitzen mitjançant els **mètodes HTTP com GET, POST, PUT, DELETE,** etc. Aquestes operacions solen ser "**estatless**" (sense estat), el que significa que **cada sol·licitud a l'API conté tota la informació necessària per processar-la**, i no depèn de cap estat de sessió emmagatzemat al servidor.

Amb **Node.js,** es pot crear fàcilment una **API RESTful utilitzant el mòdul Express.js**, que és un marc d'aplicació web per a Node.js. **Express** proporciona una forma senzilla i elegant de definir rutes i controladors per a les diferents operacions HTTP.

3.- Peticions HTTP

Les peticions HTTP són els missatges que es fan entre el client (normalment un navegador web) i el servidor per tal de sol·licitar o enviar dades. Estes peticions segueixen el **protocol HTTP** (Hypertext Transfer Protocol) i permeten que el client i el servidor es comuniquen de manera estàndard.

Les parts bàsiques d'una petició HTTP són:

1. **Mètode HTTP**: Indica l'acció que el client vol realitzar sobre el recurs especificat.

   - **GET**: Sol·licita dades del servidor.
   - **POST**: Envia dades al servidor per crear nous recursos.
   - **PUT**: Envia dades al servidor per actualitzar un recurs existent.
   - **DELETE**: Sol·licita al servidor que elimine un recurs.

2. **URI (Uniform Resource Identifier)**: Indica la ubicació del recurs que el client vol accedir. És com una adreça que identifica de manera única el recurs en el servidor.

3. **Capçaleres (Headers)**: Proporcionen informació addicional sobre la petició o el client. Per exemple, les capçaleres poden incloure informació sobre el tipus de contingut que s'envia o es sol·licita, la longitud del cos del missatge, l'autenticació del client, etc.

4. **Cos del missatge (Message Body)**: És una part opcional de la petició que conté dades enviades al servidor. Normalment s'utilitza amb mètodes com POST o PUT per enviar dades de formularis o JSON al servidor.

Quan el servidor rep una petició HTTP, processa la informació inclosa en la petició i envia una resposta al client. La resposta HTTP també inclou un conjunt de capçaleres i, opcionalment, un cos de missatge amb les dades sol·licitades o altres continguts.

• Exemples

1. Mètode HTTP:

- **GET**: Per obtindre la informació d'un usuari. Realitzarem una petició **GET**:

```
GET /api/users/123
```

Sol·licitaria al servidor les dades de l'usuari amb l'ID 123.

- **POST**: Si volem crear un nou usuari, utilitzarem una petició **POST** amb les dades del nou usuari al servidor:

```
POST /api/users
Content-Type: application/json

{
"name": "Vicent Andreu",
"email": "vicent@example.com"
}
```

- **PUT**: Suposem que volem actualitzar les dades d'un usuari existent amb **l'ID 123**. Enviaríem una petició **PUT** amb les dades actualitzades al servidor:

```
PUT /api/users/123
Content-Type: application/json

{
"name": "Tica Andreu",
"email": "tica@example.com"
}
```

- **DELETE**: Per eliminar un usuari amb l'ID 123, faríem una petició **DELETE** com la següent:

```
DELETE /api/users/123
```

2. URI (Uniform Resource Identifier):

En els exemples anteriors, les URIs com `/api/users` i `/api/users/123` identifiquen els recursos (en este cas, els usuaris) a què es refereixen les peticions.

3. Capçaleres (Headers):

A les peticions **POST** i **PUT**, s'afegeix una capçalera `Content-Type` per indicar el tipus de contingut del cos del missatge. En aquests casos, s'usa `application/json` per indicar que el cos del missatge és en format JSON.

4. Cos del missatge (Message Body):

En les peticions **POST** i **PUT**, el cos del missatge conté les dades del nou usuari (per POST) o les dades actualitzades de l'usuari (per PUT), en format **JSON**. Estes dades s'envien al servidor perquè puga processar-les i actuar en conseqüència.

Exemple API Rest Simple:

https://node-comarques-rest-server-production.up.railway.app/api/comarques/provincies

https://node-comarques-rest-server-production.up.railway.app/api/comarques/comarquesAmbImatge/$provincia

https://node-comarques-rest-server-production.up.railway.app/api/comarques/infoComarca/$comarca

Projecte API amb Node.js, Express.js i Knex.js

El projecte constarà de 4 parts on es realitzaran les 4 operacions **CRUD**: **READ, DELETE, INSERT i UPDATE.**

Utilitzarem **Node.js per crear la nostra aplicació web**, aprofitant les seves capacitats per **executar codi JavaScript al servidor**. A més, farem servir el framework **Express.js per gestionar les rutes, les peticions HTTP i les respostes** de manera eficient. Amb **Express.js**, simplificarem el desenvolupament de la nostra **API REST**. Finalment, **utilitzarem Knex.js per interactuar amb la base de dades relacional SQLite**. Esta biblioteca ens proporciona una manera senzilla de construir **consultes SQL i realitzar operacions CRUD** a la nostra taula de dades.

Amb **Node.js** com a motor d'execució, **Express.js** com a framework web i **Knex.js** com a capa d'accés a dades, podrem crear una API REST completa per gestionar les dades de la nostra taula en SQLite.

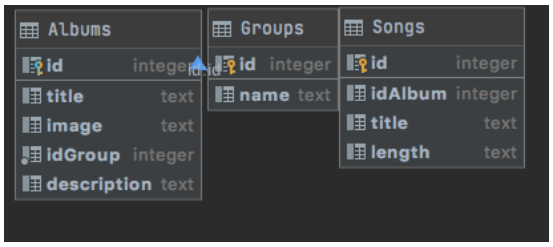
SQLite - Configuració i Creació del Datasource

Hem de crear una base de dades amb **SQLite** que reflectisca el nostre projecte. La idea és que més endavant **el client farà una petició a la API**, **es consultarà la base de dades i se li retornarà un JSON**. Quan el client faça una crida al lloc web principal **se li presentarà una pàgina web feta dinàmicament amb jsrender**.

Per a treballar amb SQLite a **WebStorm** hem de fer:

- Instalar **Pluggin en WebStorm**- > "**Database Tools and SQL for WebStorm**"
- Crear **Datasource** SQLITE.
- Crear base de dades **(1 taula**). **Cal modificar el vostre json** per a que tinga un sol objecte per tal de simplificar el projecte a una taula.
- Omplir la taula amb les dades (a ma).

- **Exemple definición de taules SQLite.** En el projecte inicial **música.json** l'estructura de les taules Sqlite seria el següent:



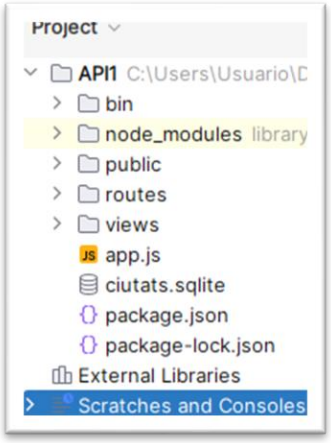
NOTA SOBRE EL JSON: Per tal de simplificar l'elaboració de les 4 parts del projecte API, heu de modificar el **vostre json** a un sol objecte amb dades per tal de crear una sola taula SQLite.

Configuració dels Mòduls

Per a poder realitzar les 4 parts del projecte API necessitem instal·lar en **Webstorm** els mòduls per a Node.js, npm de knex i els mòdul d'sqlite3, hi ha diferències amb la versió de Webstorm del videom per tant:

- **Passos:**
  1. Instal·lar node.js https://nodejs.org/en/download
  2. Crear projecte **Express**.
     Des de consola (si hi ha problemes):
     ```
     $ npm install
     $ npm audit fix –force
     ```
  3. Comprovar que s'ha creat el projecte Express:
     `localhost:3000`
  4. Modificar **app.js**

- **Estructura del projecte Express.js**



→ **`bin/:`** Conté scripts executables relacionats amb la gestió i execució de l'aplicació Express.js.

→ **`node_modules/:`** Esta carpeta conté totes les **dependències** del projecte, incloses les biblioteques i els mòduls externs instal·lats a través de **npm** o **Yarn**.

→ **`public/:`** Conté tots els **arxius estàtics** de l'aplicació web, com arxius CSS, JavaScript, imatges, etc. Estos arxius són accessibles directament pel client.

→ **`routes/:`** En esta carpeta es defineixen els fitxers de les **rutes de l'aplicació**. Cada fitxer pot contenir conjunts de rutes relacionades amb una funcionalitat específica de l'aplicació.

→ **`views/:`** Aci trobarem els fitxers de les vistes de l'aplicació, que normalment estan escrits en un motor de plantilles com **EJS, Pug o Handlebars**.

→ **`app.js:`** Este **és el fitxer principal de l'aplicació Express.js**. Aci és on s'inicialitza l'aplicació, s'estableixen configuracions globals i s'inclouen els fitxers de les rutes.

→ **`package.json:`** Conté la **configuració del projecte**, incloses les dependències, scripts d'execució i metadades del projecte. També és utilitzat per npm o Yarn per gestionar les dependències del projecte.

- La base de dades (**`ciutats.sqlite`**) ha d'estar a l'arrel del projecte

- Exemple d'arxiu **`package.json`** on trobem les configuracions dels mòduls necessaris, si falta algun, el podem posar a ma.

- Cal comprovar-ho.

```
{
  "name": "api1",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "^4.19.2",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1",
    "pug": "^2.0.4",
    "knex": "^3.1.0",
    "sqlite3": "^5.1.7"
  }
}
```

Descripció principals funcions

```
// Comporvació inicial
app.get('/api', function(req,res){
data='OK';
res.send(data)

});
```

A continuació arranquem l'aplicació posant al navegador: **localhost:3000/api**

Esta funció és un manejador d'expressions (**o endpoint**) per a una ruta específica en una aplicació web construïda amb **Express.js**.
Permet gestionar les peticions **GET** que arriben a l'adreça `'/api'`. En concret, esta funció respon amb la cadena `'OK'` quan es fa una petició **GET** a l'adreça `'/api'` de l'aplicació **Express.js**.

- `app.get('/api', function(req,res){`

  Esta funció s'executarà quan realitzem una petició **GET** a l'adreça `'/api'`.
  - → `app` és l'objecte **d'Express** que s'utilitza per definir les rutes i funcions manejadores per a estes rutes.
  - → `get` indica que només respon a les peticions **HTTP GET**. El primer argument de `get` és la ruta (`'/api'` en este cas) i el segon és una funció de retrollamada que s'executa quan es rep una petició **GET** a esta ruta.

- `function(req,res){`:

  És la funció de retrollamada que s'executa quan es rep una petició **GET** a `'/api'`, conté dos paràmetres, **req i res**, que representen l'objecte de **petició (request)** i l'objecte de **resposta (response)** respectivament.
  - → **req** conté tota la informació de la petició HTTP, com ara els paràmetres de la URL o les dades enviades pel client.
  - → **res** és l'objecte que utilitzem per respondre a la petició.

- `data='OK';` : Aci declarem una variable `data` i se li assigna el valor `'OK'`. Esta variable pot ser qualsevol cosa que vulguem enviar com a resposta al client, però en aquest cas, simplement s'envia la cadena `'OK'`.
- `res.send(data)`: Esta línia envia la resposta al client. Utilitzant l'objecte **res**, cridem la funció `send` i li passem la variable `data` com a paràmetre.
  Això fa que el servidor envïi `'OK'` com a resposta al client que va fer la petició **GET** a `'/api'`.

```
// Carreguem el modul sqlite3

var sqlite3 = require('sqlite3');
```

En este moment podem instal·lar el modul sqlite3

Quan carregues un mòdul a **Node.js** utilitzant la funció `require`, el que retorna aquesta crida és el valor exportat pel mòdul especificat.
En este cas, quan fas `require('sqlite3')`, **Node.js** carrega el mòdul **sqlite3** i retorna el valor exportat per aquest mòdul, que **pot ser una funció, un objecte, o qualsevol altra cosa que el mòdul exporte**.

```
 / Carreguem el módul knex
// Inicialitzem knex amb sqlite3

var db = require('knex')({
client:'sqlite3',
connection:{
filename: nomBDA.sqlite'
},
useNullAsDefault:true // Establir el flag useNullAsDefault a true

});
```

Este codi utilitza el mòdul **knex** per connectar-se a una base de dades **SQLite** i configurar una instància de connexió a la base de dades amb el fitxer `nomBDA.sqlite`

- `var db = require('knex')({`: Utilitzem funció `require` per importar el mòdul **knex**. A continuació, cridem la funció retornada per `require`, la qual ens permet crear una nova instància de **knex**.
- `client:'sqlite3'`: Esta línia especifica el client de base de dades que volem utilizar, en el nostre cas una base de dades **SQLite**.
- `filename: nomBDA.sqlite'`: Especifiquem la configuració de connexió per a la base de dades. A l'exemple declarem el nostre fitxer **SQLite** amb el nom **nomBDA.sqlite**.
- `useNullAsDefault:true`: Establim el flag `useNullAsDefault` a `true`. Per a que quan s'inserten valors **NULL** a la base de dades, **knex** els interpretarà com a valors nuls per defecte.

Nota: Com sols treballem amb una taula, hem d'aconseguir Seleccionar per ID i Seleccionar per nom. Com no es poden fer dos consultes al mateix temps tindrem que posar una estructura IF..ELSE