

Manual de Instalación y Uso

Edwin Iván Saboya Echeverry

MERN = My Sql + Express + React + NodeJs

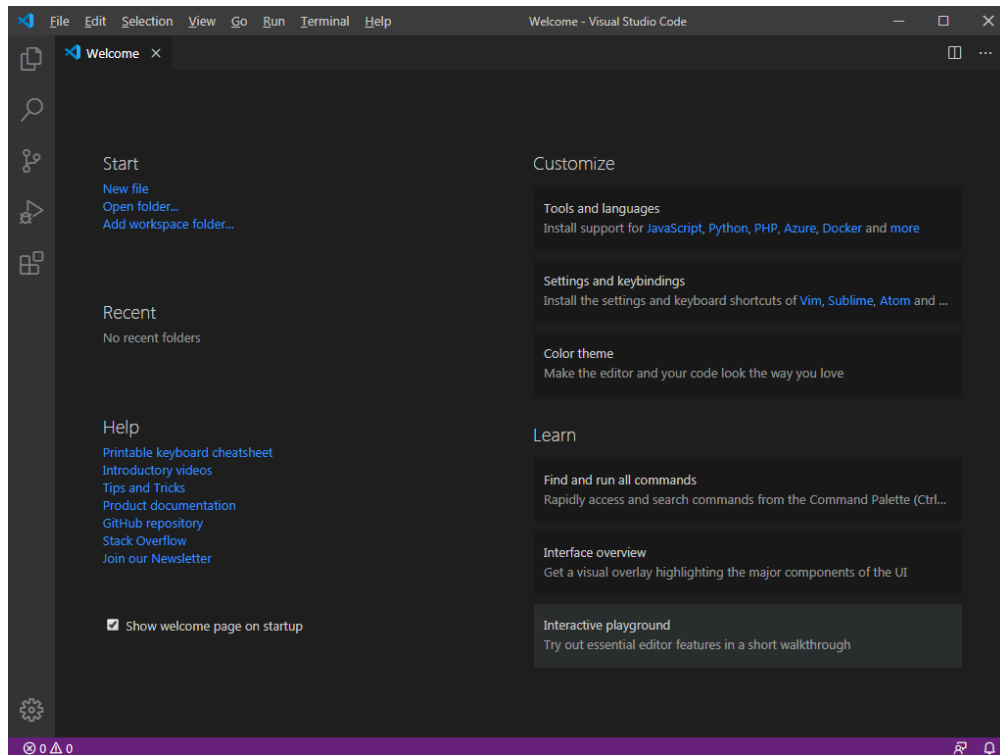
1. NodeJs

Descargar nodejs de <https://nodejs.org/es/> siguiendo instrucciones del installer.

* Debe dejar incluida la sección de agregar al PATH del S.O.

2. IDE de Desarrollo

Se trabaja mediante **visual studio code** descargable de <https://code.visualstudio.com>



a. Crear nueva carpeta para el proyecto.

Proyectos_FrontEndMentor	15/09/2022 7:38 p. m.	Carpeta de archivos
ProyectoUniversoMarvelComics	25/10/2022 10:56 p. m.	Carpeta de archivos
Test-Practico-Chgbk	2/09/2022 6:05 p. m.	Carpeta de archivos

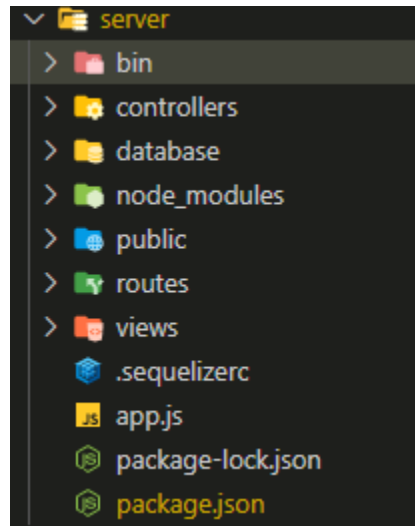
b. Para montar el servidor se requiere instalar express con el siguiente comando:

- `express server ejs`

3. Inicialización de Proyecto

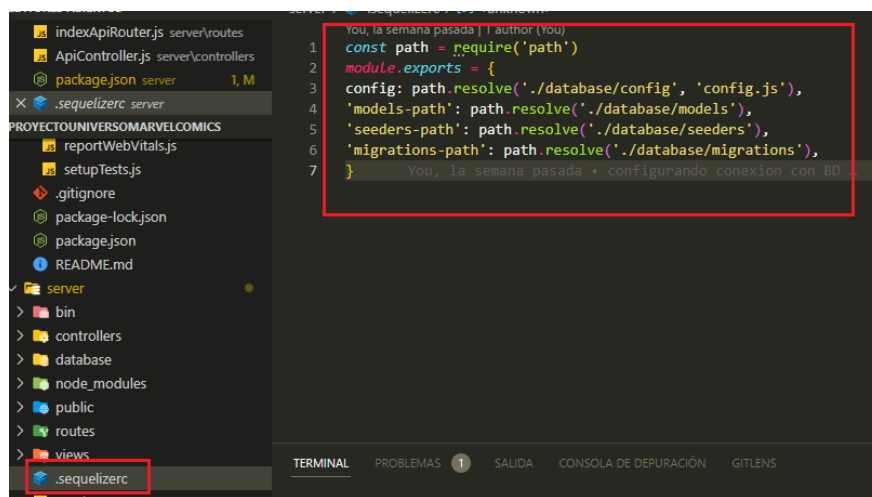
Es necesario realizar la inicialización del proyecto ejecutando el comando **npm init --yes** desde el terminal de PowerShell y con esto queda inicializado el proyecto, dejando creado el archivo package.json

Con la generación del comando anterior, el sistema crea la estructura de carpetas necesarias para empezar a crear nuestros servicios ApiRest



4. Configuración acceso a My Sql con sequelize

Para conectarse a la base de datos lo realizaremos con Sequelize, para ello, es necesario dentro de la carpeta server crear un archivo que se llama **.sequelizerc** que contendrá la configuración necesaria para crear los modelos donde se implementara la estructura de nuestra base de datos.



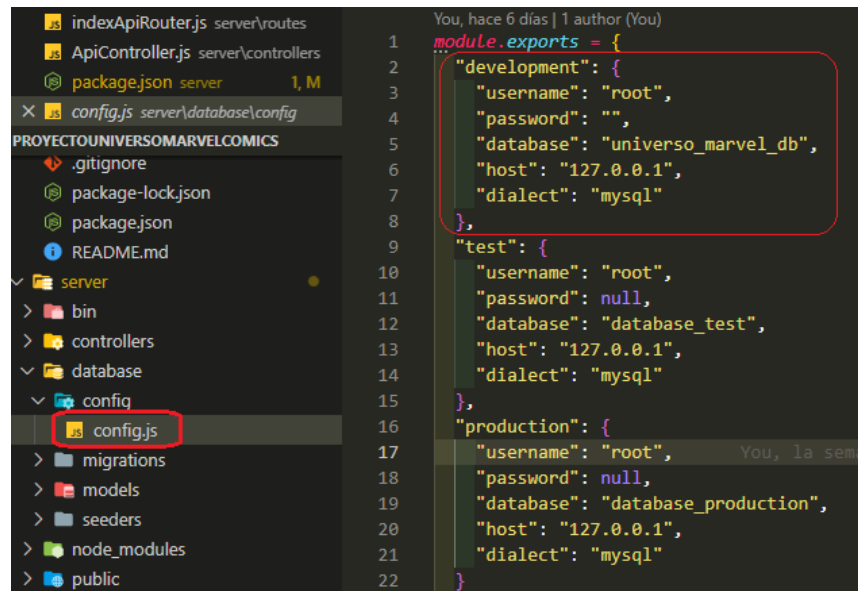
Una vez implementado esta configuración en el archivo, se debe ejecutar el siguiente comando:

- `Sequelize init`

Configurar la conexión a my sql con el siguiente conector:

- `npm i mysql2`

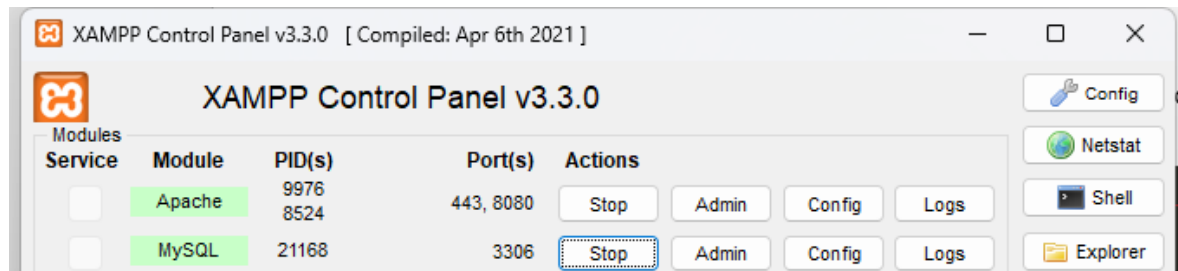
Al realizar esta operación se creará una estructura de carpetas donde se podrá realiza la configuración de las credenciales para cada ambiente.



```

1  module.exports = {
2
3    "development": {
4      "username": "root",
5      "password": "",
6      "database": "universo_marvel_db",
7      "host": "127.0.0.1",
8      "dialect": "mysql"
9    },
10
11    "test": {
12      "username": "root",
13      "password": null,
14      "database": "database_test",
15      "host": "127.0.0.1",
16      "dialect": "mysql"
17    },
18
19    "production": {
20      "username": "root",
21      "password": null,
22      "database": "database_production",
23      "host": "127.0.0.1",
24      "dialect": "mysql"
25    }
26  }
  
```

Para su correcto funcionamiento se debe validar que xampp este arriba.



Con esto Podemos garantizar la correcta conexión a nuestra base de datos.

5. Inicialización y arranque de proyecto MERN

Crear archivo dentro de carpeta src llamado **app.js**

Los siguientes son paquetes necesarios para esta fase, ejecutar por la terminal.

- **npm install path** (Librería para manejo de Paths)
- **npm install cors** (Controlar el manejo de encabezados)

El archivo `app.js` maneja la inicialización de la aplicación, setea los módulos y arranca el servidor local, para arrancar el servidor local en modo pruebas ejecutar **nodemon index.js**, el cual toma la configuración y arranca el servidor.

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

app.use( ( req, res, next ) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET, POST, PUT, PATCH, DELETE');
  res.setHeader('Access-Control-Allow-Headers', "x-access-token, Origin, X-Requested-With");
  next();
});

app.use('/api', indexRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// correr servidor
app.listen(3001, ()=>{
  console.log('server on port 3001');
})
```

Aplicación queda ejecutada en puerto 3001 y conectado My Sql.

Comandos rápidos y atajos de consola, comando a instalar.

- **npm install nodemon -D** (Comando para poder definir atajos rapidos por consola D=Solo Desarrollo y permite hacer hotdeploy)

Agregar al archivo **package.json** en el tag scripts

```
"iniciar": "nodemon src/index.js"
```

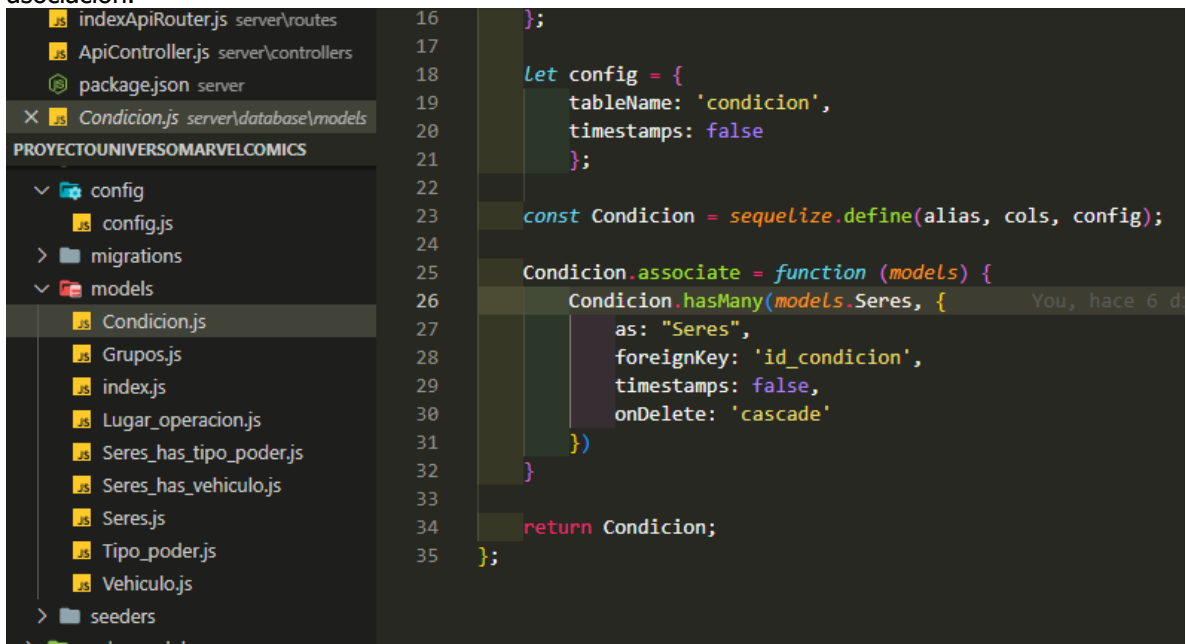
Con esto bastara con ejecutar **npm run iniciar** y el servidor quedara iniciado. Asi también se pueden agregar otros comandos como vayan siendo necesarios.

6. Configuración de Express para almacenamiento de persistencia.

Las **routes** en MERN definen los puntos API de acceso REST a funcionalidades de acceso a la base de datos, para iniciar a acceder se deberá crear el archivo **/src/routes/ indexApiRouter.js** en donde se manejará el CRUD y accesos a las diferentes estructuras, así como definir el modelo de datos o estructura creando el archivo **/src/models/....js**

Nota: En este caso son modelos de nuestra base de datos que serán consumidos por nuestras api's.

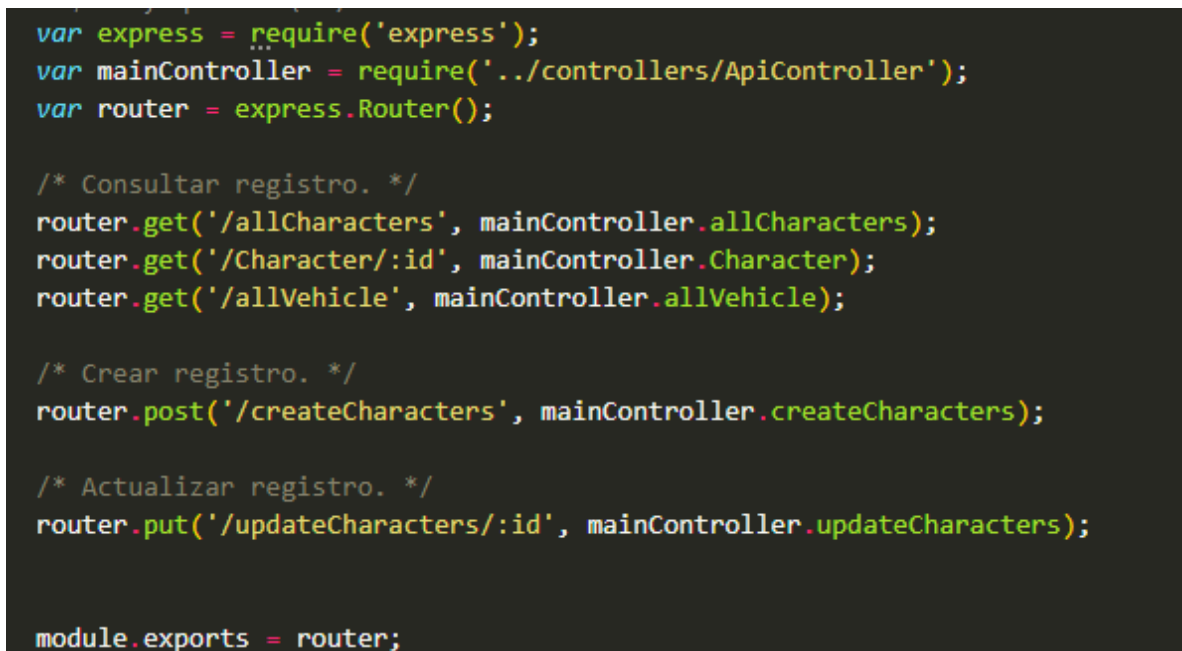
Contiene el Esquema de modelamiento para la estructura de base de datos con su correspondiente asociación.



```
16   };
17
18   let config = {
19     tableName: 'condicion',
20     timestamps: false
21   };
22
23   const Condicion = sequelize.define(alias, cols, config);
24
25   Condicion.associate = function (models) {
26     Condicion.hasMany(models.Seres, {
27       as: "Seres",
28       foreignKey: 'id_condicion',
29       timestamps: false,
30       onDelete: 'cascade'
31     });
32   }
33
34   return Condicion;
35 };
```

The screenshot shows a code editor with a file explorer on the left. The project is named 'PROYECTO UNIVERSO MARVEL COMICS'. The file explorer shows a 'models' folder containing several files: 'Condicion.js', 'Grupos.js', 'index.js', 'Lugar_operacion.js', 'Seres_has_tipo_poder.js', 'Seres_has_vehiculo.js', 'Seres.js', 'Tipo_poder.js', and 'Vehiculo.js'. The 'Condicion.js' file is selected, and its content is displayed in the editor. The code defines a Sequelize model for 'Condicion' with a table name 'condicion' and no timestamps. It also defines an association with the 'Seres' model, where 'Condicion' has many 'Seres' (aliased as 'Seres'), with 'id_condicion' as the foreign key and 'cascade' on delete.

Routes



```
var express = require('express');
var mainController = require('../controllers/ApiController');
var router = express.Router();

/* Consultar registro. */
router.get('/allCharacters', mainController.allCharacters);
router.get('/Character/:id', mainController.Character);
router.get('/allVehicle', mainController.allVehicle);

/* Crear registro. */
router.post('/createCharacters', mainController.createCharacters);

/* Actualizar registro. */
router.put('/updateCharacters/:id', mainController.updateCharacters);

module.exports = router;
```

The screenshot shows a code editor with route definitions. It imports 'express', 'mainController' (from '../controllers/ApiController'), and creates a 'router' using 'express.Router()'. There are three GET routes: '/allCharacters' (handler: mainController.allCharacters), '/Character/:id' (handler: mainController.Character), and '/allVehicle' (handler: mainController.allVehicle). There is one POST route: '/createCharacters' (handler: mainController.createCharacters). There is one PUT route: '/updateCharacters/:id' (handler: mainController.updateCharacters). Finally, 'module.exports' is set to 'router'.

Controllers

```

const db = require("../database/models");

const mainController = {
  > allCharacters : ( async (req, res) => { ...
  >   } ),
  > createCharacters : ( async (req, res) => { ...
  >   } ),
  > allVehicle : ( async (req, res) => { ...
  >   } ),
  > Character : ( async (req, res) => { ...
  >   } ),
  > updateCharacters : ( async (req, res) => { ...
  >   } )
}

module.exports = mainController;

```

You, hace 7 días • add models DataBase

Al ejecutar en la url se realiza las siguientes operaciones:

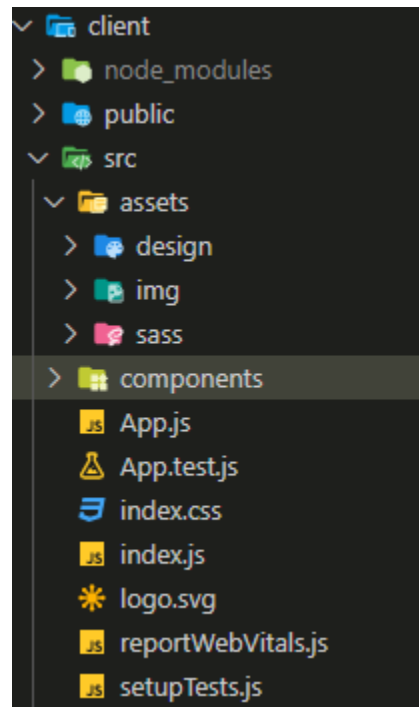
- <http://localhost:3001/api/createCharacters> : Crear un personaje
- <http://localhost:3001/api/allCharacters> : Consultar todos los personajes
- <http://localhost:3001/api/Character/ID> : Consultar un personaje por ID
- <http://localhost:3001/api/updateCharacters/id> : Actualizar un personaje por ID
- <http://localhost:3001/api/allVehicle> : Consultar todos los vehículos

7. Capa de Presentación con REACT

Ya con los servicios web REST expuestos, se inicia el consumo desde React, para eso necesario crear el ambiente para el front, para ello se sugiere crear una carpeta cliente que contendrá toda nuestra estructura

- `npx create-react-app client`

Se creará la siguiente estructura de carpetas:



Para utilizar todas las funcionalidades en los componentes de React se sugiere hacer las siguientes instalaciones:

```
- npm i node-fetch@2
- npm install react-router-dom --save
- npm i sass
- npm i react-bootstrap
```

- a. Crear los siguientes archivos en **/src/Components**. Con el siguiente contenido, son el punto de entrada y el App de react.

```

You, hace 5 días | 1 author (You)
import ContainerPrincipal from './components/ContainerPrincipal';
import { BrowserRouter, Route, Routes } from 'react-router-dom'; 12.6k (gzipped: 5k)
import Header from './components/Header';

function App() {
  return (
    <div className="App">
      <Header/>
      <BrowserRouter>
        <Routes>
          <Route path="/" exact={true} element={<ContainerPrincipal/>} ></Route>
        </Routes>
      </BrowserRouter>
    </div>
  );
}

export default App;

```

- b. todo el contenido generado dinámicamente por React mediante las clases anteriores, el webpack ayuda para que el archivo bundle.js que contiene todo el código transformado de react a JavaScript quede listo para usar.
- c. ejecutar en una nueva terminal sin parar el proceso npm run iniciar, ejecutar **npm start**

al abrir en el navegador por el puerto y raíz de contexto vemos que React está escuchando y procesando en la raíz del contexto.

