Programación Web 2 Servidor MVC y Bases de Datos NoSQL

U-TAD

Entorno MongoDB (NoSQL)

- Cuenta en Mongo Attas
- DB_URI (end point a la base de datos)
- https://www.mongodb.com/atlas/database
  - Sign-in
  - Crear nueva organización
  - Crear nuevo proyecto
  - Construye una base de datos (Shared - Free)
  - Crear Cluster
  - Crear User de BD (username y password)
  - Cualquier IP puede acceder 0.0.0.0/0
  - Finalizar y cerrar
  - Cuando esté desplegada, pulsar conectar (desde una aplicación).
  - Copiar el db_uri y cambia el usuario, la pwd y el nombre de tu base de datos (creadla tb).

Iniciar Proyecto

Crear nuevo directorio

npm init -y

npm i **express nodemon cors dotenv** mongoose multer

Crear fichero **.gitignore**:

- node_modules/
- .env

Crear fichero **.env**

- PORT=3000
- DB_URI=

Crear **index.js**

Iniciamos app.js (indicad en package.json que main=app.js)

```
const express = require("express")
const cors = require("cors")
require('dotenv').config();

const app = express()

//Le decimos a la app de express() que use cors para evitar el error Cross-Domain (XD)
app.use(cors())
app.use(express.json())

const port = process.env.PORT || 3000

app.listen(port, () => {
    console.log("Servidor escuchando en el puerto " + port)
})
```

Scaffold

Estructura de directorios siguiendo el patrón Modelo Vista Controlador (MVC):

- **models**
- **controllers**
- **routes**
- config
- utils
- storage
- test

Es un buen momento para inicializar tu proyecto git:

git init

Conexión a mongo

Creamos el fichero **config/mongo.js**

```
const mongoose = require('mongoose')

const dbConnect = () => {
    const db_uri = process.env.DB_URI
    mongoose.set('strictQuery', false)
    mongoose.connect(db_uri, {
        useNewUrlParser: true,
        useUnifiedTopology: true
    }, (err, res) => {
        if(!err) {
            console.log("Conectado a la BD")
        }else {
            console.err("No se ha podido establecer la conexión a la BD" )
        }
    })
}

module.exports = dbConnect
```

Conexión a mongo

Invocamos la conexión desde **app.js**

```
...
const dbConnect = require('./config/mongo')
...
dbConnect()
```

Models

Creamos el subdirectorio models/**nosql**

Creamos ahí el fichero **users.js** (y todos los otros modelos de esquema que necesitemos, cada uno en un fichero distinto):

https://github.com/rpmaya/u-tad-Server/tree/main/MVC/models/nosql

```
const mongoose = require("mongoose")
const UserScheme = new mongoose.Schema(
    {
        name: {
            type: String
        },
        age: {
            type: Number
        },
        email: {
            type: String,
            unique: true
        },
        password:{
            type: String  // TODO Guardaremos el hash
        },
        role:{
            type: ["user", "admin"], // es el enum de SQL
            default: "user"
        }
    },
    {
        timestamp: true, // TODO createdAt, updatedAt
        versionKey: false
    }
)
module.exports = mongoose.model("users", UserScheme) // "users" es el nombre de la colección en mongoDB (o de la tabla en SQL)
```

Router

http://localhost:3000/users

http://localhost:3000/tracks

http://localhost:3000/storages

routes/index.js (el fichero debe llamarse igual que la ruta)

```
const express = require("express")
const fs = require("fs")
const router = express.Router()
const removeExtension = (fileName) => {
    //Solo la primera parte del split (lo de antes del punto)
    return fileName.split('.').shift()
}
fs.readdirSync(__dirname).filter((file) => {
    const name = removeExtension(file) // index, users, storage, tracks
    if(name !== 'index') {
        router.use('/' + name, require('./'+name)) // http://localhost:3000/api/tracks
    }
})
module.exports = router
```

Router

routes/tracks.js, routes/users, routes/storage

```
const express = require("express")
const router = express.Router()
router.get("/", (req, res) => {
    const data = ["hola", "mundo", "tracks"]
    res.send({data})
})
module.exports = router
```

app.js

…

```
app.use("/api", require("./routes")) //Lee routes/index.js por defecto
```

…

Controllers

controllers/tracks.js

```
/**
 * Obtener lista de la base de datos
 * @param {*} req
 * @param {*} res
 */
const getItems = (req, res) => {
    const data = ["hola", "mundo"]
    res.send({data})
}
const getItem = (req, res) => {...}
const createItem = (req, res) => {...}
const updateItem = (req, res) => {...}
const deleteItem = (req, res) => {...}

module.exports = { getItems, getItem,
                   createItem, updateItem,
                   deleteItem };
```

routes/tracks.js

```
const { getItems, getItem } = require("../controllers/tracks")
router.get("/", getItems)
router.get("/:id", getItem)
```

Models

models/index.js

```
const models = {
    usersModel: require('./nosql/users'),
    tracksModel: require('./nosql/tracks'),
    storageModel: require('./nosql/storage')
}
module.exports = models
```

routes/index.js

```
const { getItems, getItem, createItem } =
require("../controllers/tracks")

...

router.post("/", createItem)
```

controllers/index.js

```
const { tracksModel } = require('../models')
const getItems = async (req, res) => {
    const data = await tracksModel.find({})
    res.send(data)
}
const createItem = async (req, res) => {
    const { body } = req
    //console.log(body)
    const data = await
tracksModel.create(body)
    res.send(data)
}
```

Upload File

Multer: dependencia que funciona como middleware entre la ruta y el controlador.

En el directorio “routes”, nos creamos un nuevo fichero “storage.js”

```
const express = require("express")
const router = express.Router();
//Lo moveremos a otro archivo en utils (helpers)
const multer = require("multer")
const storage = multer.diskStorage({
    destination:function(req, file, callback){ //Pasan argumentos automáticamente
        const pathStorage = __dirname+"/../storage"
        callback(null, pathStorage) //error y destination
    },
    filename:function(req, file, callback){ //Sobreescribimos o renombramos
        //Tienen extensión jpg, pdf, mp4
        const ext = file.originalname.split(".").pop() //el último valor
        const filename = "file-"+Date.now()+"."+ext
        callback(null, filename)
    }
})
const uploadMiddleware = multer({storage}) //Middleware entre la ruta y el controlador
//hasta aquí
router.post("/", uploadMiddleware.single("image"), (req, res) => { //solo enviamos uno con .single, sino .multi
    res.send("test")
})
module.exports = router;
```

Upload File

Creamos controllers/storage.js a partir de models/tracks.js y cambiamos “tracksModel” por “storageModel”

Creamos utils/handleStorage.js

```
const multer = require("multer")
const storage = multer.diskStorage({
    destination:function(req, file, callback){ //Pasan argumentos automáticamente
        const pathStorage =   dirname+"/../storage"
        callback(null, pathStorage) //error y destination
    },
    filename:function(req, file, callback){ //Sobreescribimos o renombramos
        //Tienen extensión jpg, pdf, mp4
        const ext = file.originalname.split(".").pop() //el último valor
        const filename = "file-"+Date.now()+"."+ext
        callback(null, filename)
    }
})
const uploadMiddleware = multer({storage}) //Middleware entre la ruta y el controlador
module.exports = uploadMiddleware
```

En routers/storage.js

```
const express = require("express")
const router = express.Router();
const uploadMiddleware  = require("../utils/handleStorage" )
const { createItem } = require("../controllers/storage")
router.post("/", uploadMiddleware.single("image"), createItem)
module.exports = router;
```

Upload file

En .env: `PUBLIC_URL=http://localhost:3000`

En app.js: `app.use(express.static("storage")) // http://localhost:3000/file.jpg`

Echamos un vistazo a models/nosql/storage.js (campos url y filename)

En controllers/storage.js modificamos la función createItem()

```
const createItem = async (req, res) => {
    const { body, file } = req
    const fileData = {
        filename: file.filename,
        url: process.env.PUBLIC_URL+"/"+file.filename
    }
    const data = await storageModel.create(fileData)
    res.send(data)
}
```

Ejercicio

Siguiendo el ejemplo de “tracks”, completa “users”