

MySQL

U-TAD

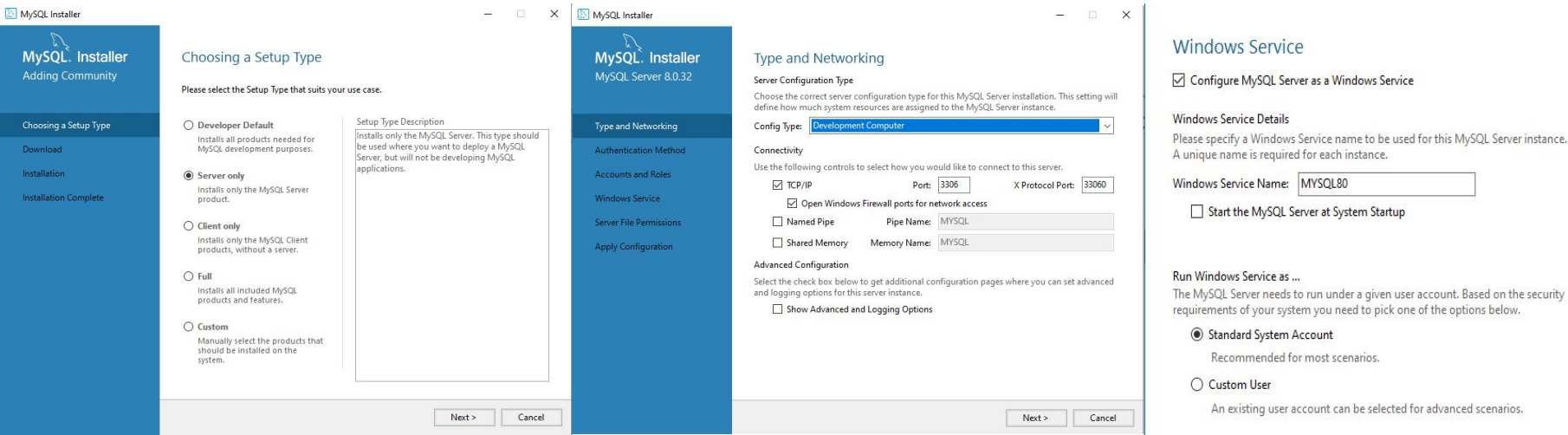
<https://github.com/rpmaya/apiserver.git>

MySQL Sequelize

ORM: Modelo que permite mapear estructuras de una base de datos relacional
`npm i sequelize mysql2` (MySQL, Postgres, SQL Server, ...)

<https://dev.mysql.com/downloads/installer/> (web-community)

No thanks, just start my download



MySQL Sequelize

Creamos el archivo **config/mysql.js**

```
const { Sequelize } = require("sequelize")

const database = process.env.MYSQL_DATABASE
const username = process.env.MYSQL_USER
const password = process.env.MYSQL_PASSWORD
const host = process.env.MYSQL_HOST

const sequelize = new Sequelize(
  database,
  username,
  password,
  {
    host,
    dialect: "mysql"
  }
)

const dbConnectMySQL = async () => {
  try {
    await sequelize.authenticate()
    console.log("MySQL conexión correcta")
  } catch (err) {
    console.log("MySQL error de conexión:", err)
  }
}

module.exports = { sequelize, dbConnectMySQL }
```

MySQL Sequelize

Crea tus variables en .env

```
MYSQL_DATABASE=apiserver
```

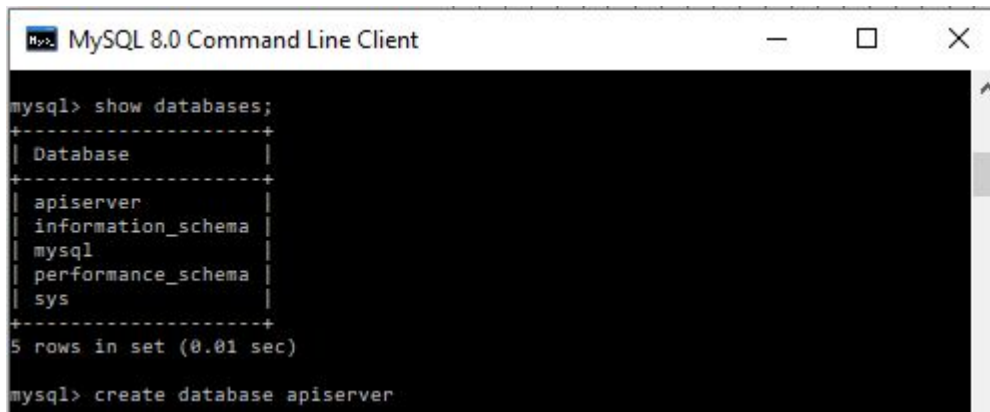
```
MYSQL_USER=root
```

```
MYSQL_PASSWORD=<tuPasswd>
```

```
MYSQL_HOST=localhost
```

```
ENGINE_DB=mysql
```

Desde la consola de MySQL crea la base de datos “apiserver”, o como la quieras llamar:



```
MySQL 8.0 Command Line Client
mysql> show databases;
+-----+
| Database |
+-----+
| apiserver |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql> create database apiserver
```

MySQL Sequelize

Y en **app.js**

...

```
const { sequelize, dbConnectMySQL } = require("../config/mysql")
```

...

```
if (process.env.ENGINE_DB === 'nosql'){
  dbConnect()
  // Crea las colecciones por defecto si no existieran
}else{
  dbConnectMySQL()
  sequelize.sync() // Crea las tablas en la base de datos si no existieran
}
```

Reinicia el servidor!

MySQL Models

Crear en **models/mysql/**: **tracks**, **users** y **storage**. (Copiadlos desde Github)

Importarlos en **models/index.js** de forma dinámica:

```
const pathModels = (process.env.ENGINE_DB === 'nosql') ? './nosql/' : './mysql/'
const models = {
  userModel: require(pathModels+'users'),
  tracksModel: require(pathModels+'tracks'),
  storageModel: require(pathModels+'storage'),
}

module.exports = models
```

MySQL Consideraciones

En Mongo usábamos el `_id`, ahora usaremos `id`.

En `utils/handleJwt` teníamos la propiedad `user._id` y ahora será `user.id`

Por lo que nos crearemos un `/utils/handlePropertiesEngine.js` para modificarlos:

```
const ENGINE_DB = process.env.ENGINE_DB
const getProperties = () => {
  const data = {
    nosql: {
      id: '_id'
    },
    mysql: {
      id: 'id'
    }
  }
  return data[ENGINE_DB]
}
module.exports = getProperties
```

MySQL Consideraciones

En utils/handleJwt.js

```
const jwt = require("jsonwebtoken")
const getProperties = require("../utils/handlePropertiesEngine")
const propertiesKey = getProperties()
```

```
...
```

```
    //_id: user._id,
    [propertiesKey.id]: user[propertiesKey.id],
```

```
...
```


MySQL Consideraciones

Y en middlewares/session.js

```
...
const getProperties = require("../utils/handlePropertiesEngine")
const propertiesKey = getProperties()
...

const authMiddleware = async (req, res, next) => {
  ...

  if(!dataToken){ //Eliminamos el dataToken._id
    handleHttpError(res, "NOT_PAYLOAD_DATA", 401)
    return
  }
  const query = {
    // _id o id
    [propertiesKey.id]: dataToken[propertiesKey.id]
  }
  //const user = await usersModel.findById(dataToken._id) // findById solo para Mongoose
  const user = await usersModel.findOne(query) // findOne válido para Mongoose y Sequelize
  req.user = user // Inyecto al user en la petición
  next()
  ...
}
```

Crear tablas en Mysql

Desde **MySQL 8.0 Command Line Client** o cualquier otro gestor o interfaz:

show databases;

use apiserver;

show tables;

describe users;

describe tracks;

describe storages;

Probamos...

Cambia en .env `ENGINE_DB=nosql` re-inicializa el server y comprueba que no hayas roto nada (funciona el login, etc, con mongoose)

Vuelve a cambiar el .env `ENGINE_DB=mysql` re-inicializa el server

En todos los validators, vamos a quitar la validación **isMongold()** ya que no aplica a SQL, y lo dejamos simplemente con que no sea vacío.

Registra un usuario!

Si todo va bien, compruébalo en la base de datos desde la consola:

SELECT * from apiserver.users;

Probamos...

Antes de probar el login, vamos a **controllers/auth.js** y en la función loginCtrl(), vemos el `.select('password name role email')` que está vinculado al modelo de mongoose, así que lo dejamos comentado (lo eliminamos) para *mysql*, pero habría que ponerlo para *nosql*

```
const loginCtrl = async (req, res) => {  
  try {  
    req = matchedData(req)  
    const user = await userModel.findOne({ email: req.email }) //.select("password name role email")  
  }  
}
```

...

Si quieres que los usuarios puedan subir tracks modifica en routes/tracks:

```
router.post("/", authMiddleware, checkRol([ "user", "admin"]), validatorCreateItem, createItem)
```

...

Para probar el resto de rutas, habría que adaptar los validators, los POST desde el cliente y el `.find()` por `.findAll()`

JOINS (Relaciones entre colecciones)

Primero para mongo (en **models/nosql/tracks.js**) con funciones estáticas:

...

```
TracksScheme.statics.findAllData = function(name) {
  // "this." hace referencia a su propio modelo
  const joinData = this.aggregate([
    {
      $lookup: { // lookup =~ left join
        from: "storages",
        localField: "mediaId", // tracks.mediaId
        foreignField: "_id",   // storages._id
        as: "audio" // Alias audio
      }
    },
    /* { // From LEFT JOIN to INNER JOIN
      $unwind("$audio")
    } */
  ])
  return joinData
}

TracksScheme.plugin(mongooseDelete, {overrideMethods: "all"})
module.exports = mongoose.model("tracks", TracksScheme) // Nombre de la colección
```

JOINS (Relaciones entre colecciones)

Primero para mongo (en **controllers/tracks.js**) y dentro de la función `getItems()`:

...

```
const data = await tracksModel.findAllData()
```

...

JOINS (Relaciones entre colecciones)

Y si solo quiero un item (en models/nosql/tracks.js):

```
TracksScheme.statics.findOneData = function(name) {
  const joinData = this.aggregate([
    {
      $match: {
        _id:mongooseTypes.ObjectId(id)
      }
    },
    {
      $lookup: {
        from: "storages",
        localField: "mediaId", // tracks.mediaId
        foreignField: "_id",   // storages._id
        as: "audio" // Alias audio
      }
    },
    /* {
      $unwind:"$audio"
    }, */
  ])
  return joinData
}
```

Y en controllers/tracks.js, en la función getItem(id):

```
const data = await tracksModel.findOneData(id)
```

JOINS (Relaciones entre tablas)

En `models/mysql/tracks.js`

```
const Storage = require("../Storage")

...

Tracks.findAllData = function () {
  Tracks.belongsTo(Storage, {
    foreignKey: 'mediaId',
    as: "audio"
  })
  return Tracks.findAll({include:'audio'})
}

Tracks.findOneData = function (id) {
  Tracks.belongsTo(Storage, {
    foreignKey: 'mediaId',
    as: "audio"
  })
  return Tracks.findOne({where:{id: id}, include:'audio'})
}

module.exports = Tracks
```