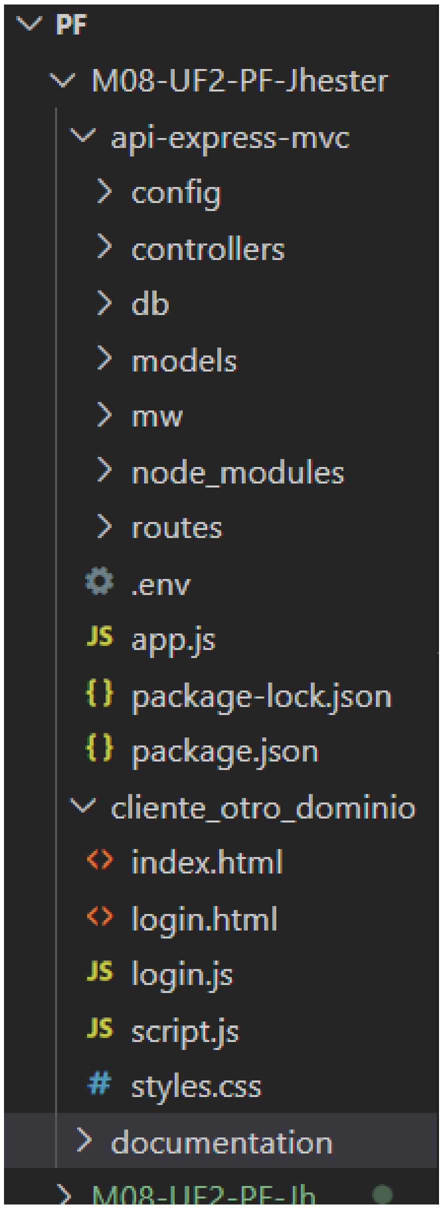
M8-UF2-PF
by Jhester John Calma

Structure



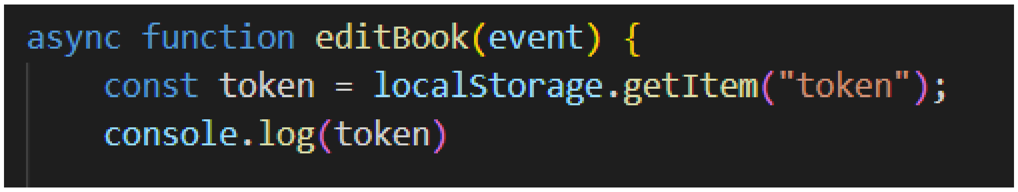
```
PF
  M08-UF2-PF-Jhester
    api-express-mvc
      config
      controllers
      db
      models
      mw
      node_modules
      routes
      .env
      app.js
      package-lock.json
      package.json
    cliente_otro_dominio
      index.html
      login.html
      login.js
      script.js
      styles.css
    documentation
  M08-UF2-PF-Jh
```

- **config:** Contains the configuration data for MongoDB/SQL.
- **controllers:** Handles the requests made by the user.
- **models:** Interacts with the database.
- **mw:** Verifies requests before they reach the controllers.
- **routes:** Defines the routes to the controllers.

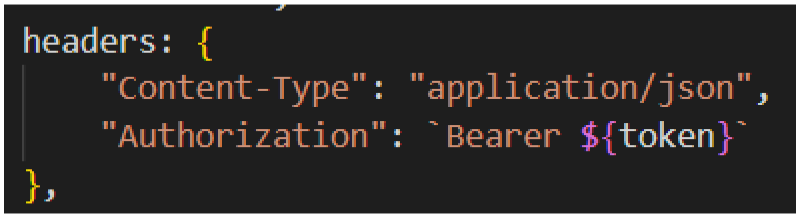
Changes for JWT Authentication:

Frontend:

- **login.html: The page where the user inputs their credentials.**
- **login.js: Sends a request to the backend to generate a token based on the user's credentials.**



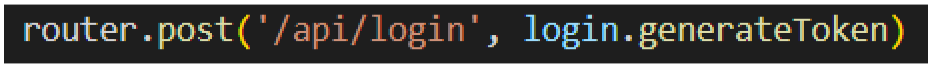
```
async function editBook(event) {
    const token = localStorage.getItem("token");
    console.log(token)
```



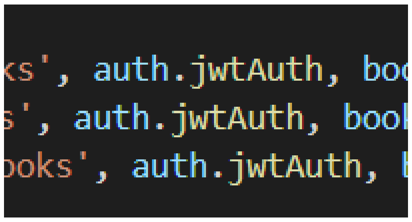
```
headers: {
    "Content-Type": "application/json",
    "Authorization": `Bearer ${token}`
},
```

Backend:

- **controller/login.js: It is responsible for verifying that the user/password is correct, and only there that the system will generate a token.**
- **mw/auth.js: Checks that the token inside the Authorization header is valid to perform an operation.**
- **.env: Contains the secret key. (1234) in this case**



```
router.post('/api/login', login.generateToken)
```

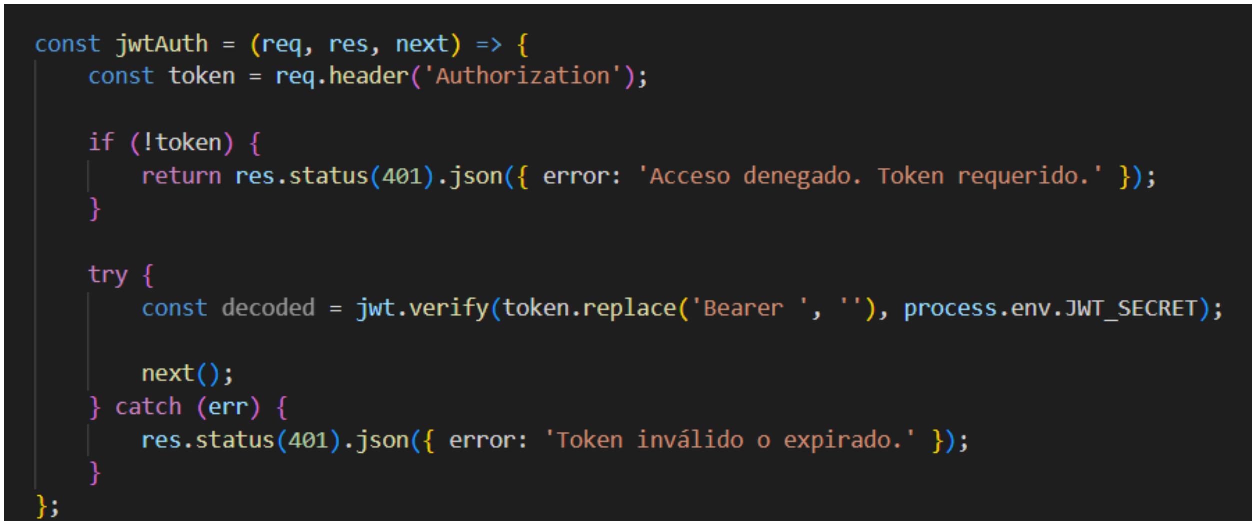


```
ks', auth.jwtAuth, boo
s', auth.jwtAuth, book
oks', auth.jwtAuth, b
```

When a POST request is sent to /api/login, the generateToken function is triggered to authenticate the user.

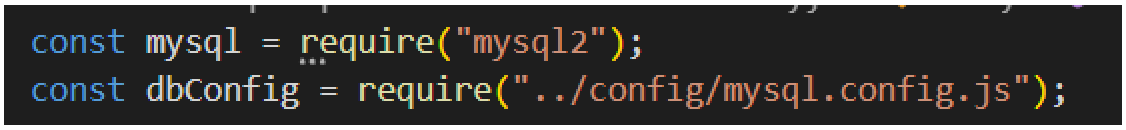
For POST, DELETE, or PUT requests to /api/books, the jwtAuth function is invoked to validate the token.

To delete, edit, or create books, the token is included in the Authorization header and sent with the request. If the token is valid, the operation is executed successfully, and the changes are reflected in index.html.

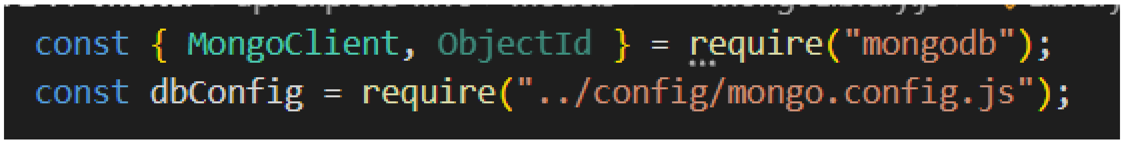


```
const jwtAuth = (req, res, next) => {
    const token = req.header('Authorization');

    if (!token) {
        return res.status(401).json({ error: 'Acceso denegado. Token requerido.' });
    }

    try {
        const decoded = jwt.verify(token.replace('Bearer ', ''), process.env.JWT_SECRET);

        next();
    } catch (err) {
        res.status(401).json({ error: 'Token inválido o expirado.' });
    }
};
```

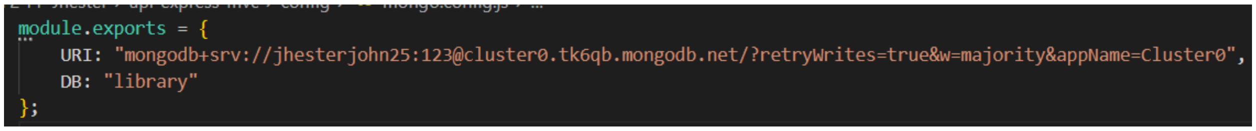
Switching from SQL to MONGO



```js
const mysql = require("mysql2");
const dbConfig = require("../config/mysql.config.js");
```



```js
const { MongoClient, ObjectId } = require("mongodb");
const dbConfig = require("../config/mongo.config.js");
```



```js
module.exports = {
    URI: "mongodb+srv://jhesterjohn25:123@cluster0.tk6qb.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0",
    DB: "library"
};
```

The key difference is that SQL operates synchronously, executing queries sequentially, while MongoDB is asynchronous, allowing operations to run without blocking execution.

Since a constructor cannot be asynchronous, we do not establish the connection within it as we do in the SQL module. Instead, we define an asynchronous function responsible for setting up the connection. This function is then called whenever an operation needs to be performed, ensuring efficient and non-blocking database interactions.

This approach helps maintain performance and scalability, especially in applications handling multiple concurrent requests.