

Entrega 3 - grupo 4

septiembre 15- 2024

Geronimo Correa
Tomás Peña
Felipe Rivero

Listado

- 1. Crear un nuevo proyecto de npm e instalar todas las dependencias que necesitarás.**
(1 Punto)
- 2. Utilizar el paquete pg para gestionar la comunicación con la base de datos PostgreSQL.**
(3 Puntos)
- 3. Implementar la autenticación y autorización de usuarios con JWT.**
(2 Puntos)
- 4. Usar el paquete CORS para permitir las consultas de orígenes cruzados.**
(1 Punto)
- 5. Utilizar middlewares para validar las credenciales o token en cabeceras en las rutas que aplique.**
(2 Puntos)
- 6. Realizar test de por lo menos 4 rutas de la API REST comprobando los códigos de estados de diferentes escenarios.**
(1 Punto)

1) Crear un nuevo proyecto de npm e instalar todas las dependencias que necesitarás.

Creamos una carpeta especifica para el backend y ejecutamos npm install

Dependencias Clave

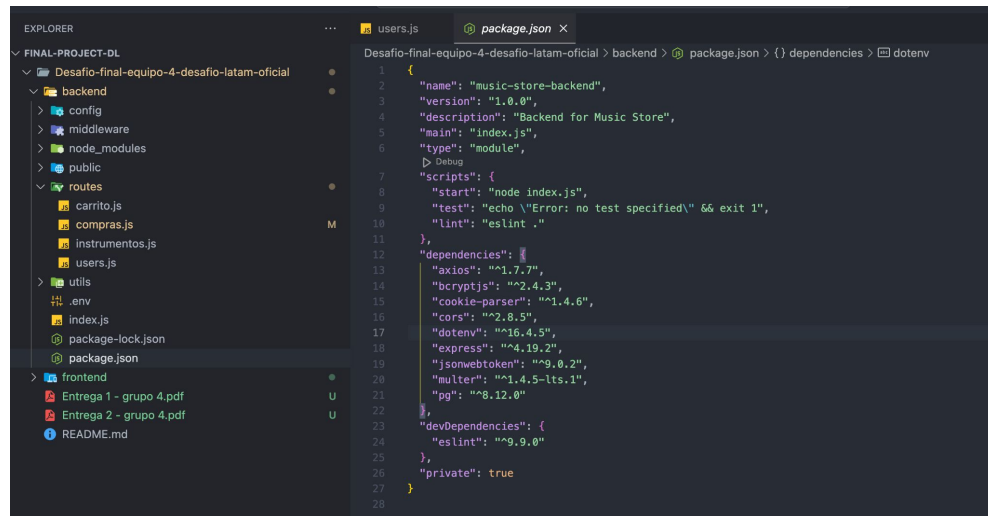
express: Framework para crear el servidor web.

pg: Paquete para interactuar con PostgreSQL.

jsonwebtoken: Para manejo de autenticación JWT.

cors: Para habilitar solicitudes de orígenes cruzados.

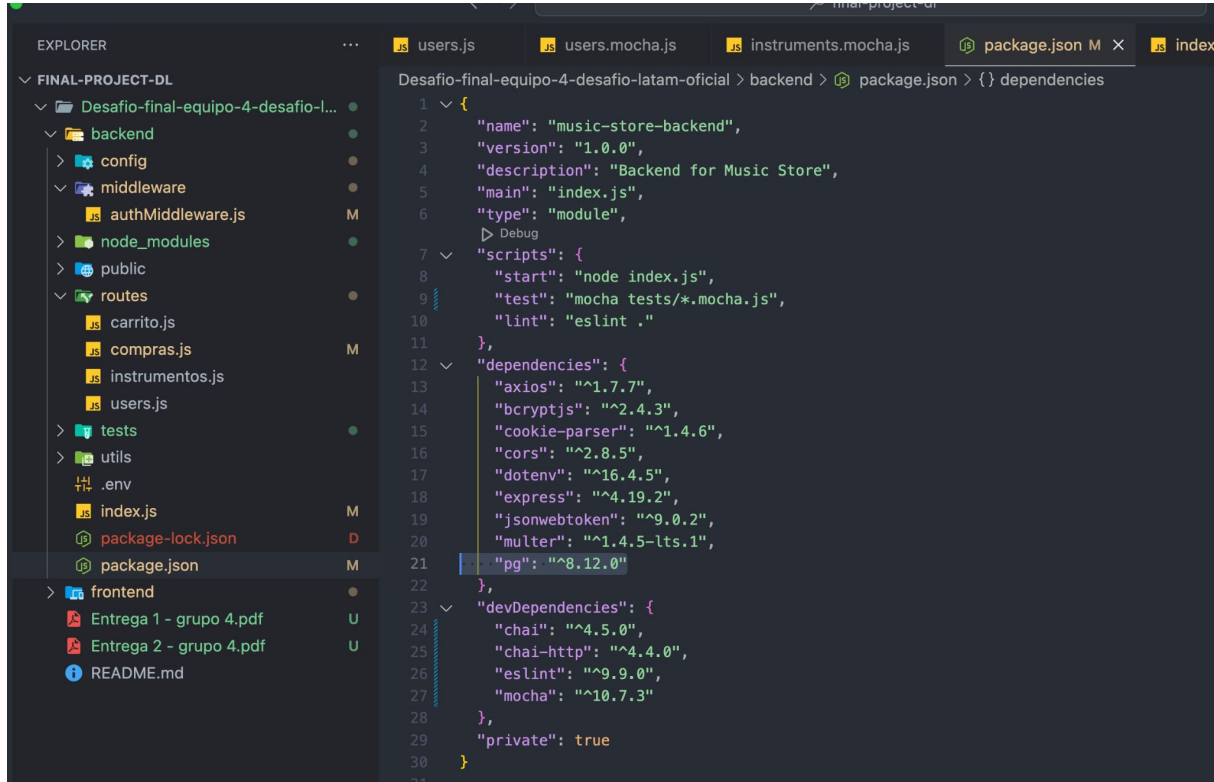
dotenv: Para manejar variables de entorno.



```
1 {
2   "name": "music-store-backend",
3   "version": "1.0.0",
4   "description": "Backend for Music Store",
5   "main": "index.js",
6   "type": "module",
7   "scripts": {
8     "start": "node index.js",
9     "test": "echo \"Error: no test specified\" && exit 1",
10    "lint": "eslint ."
11  },
12  "dependencies": {
13    "axios": "^1.7.7",
14    "bcryptjs": "^2.4.3",
15    "cookie-parser": "^1.4.6",
16    "cors": "^2.8.5",
17    "dotenv": "^16.4.5",
18    "express": "^4.19.2",
19    "jsonwebtoken": "^9.0.2",
20    "multer": "^1.4.5-lts.1",
21    "pg": "^8.12.0"
22  },
23  "devDependencies": {
24    "eslint": "^9.9.0"
25  },
26  "private": true
27 }
```

2. Utilizar el paquete pg para gestionar la comunicación con la base de datos PostgreSQL.

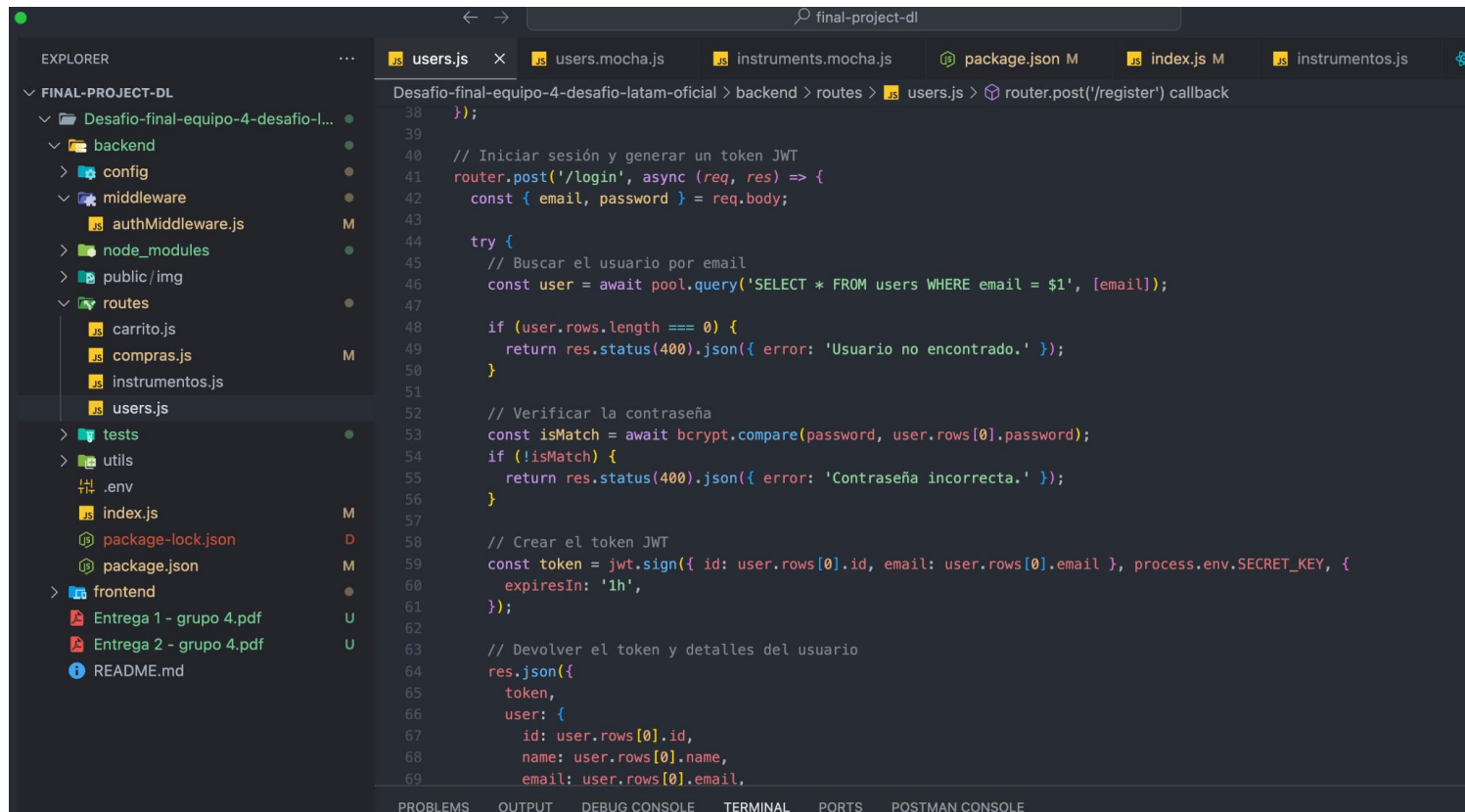
Está en el backend indicado



```
1 {
2   "name": "music-store-backend",
3   "version": "1.0.0",
4   "description": "Backend for Music Store",
5   "main": "index.js",
6   "type": "module",
7   "scripts": {
8     "start": "node index.js",
9     "test": "mocha tests/*.mocha.js",
10    "lint": "eslint ."
11  },
12  "dependencies": {
13    "axios": "^1.7.7",
14    "bcryptjs": "^2.4.3",
15    "cookie-parser": "^1.4.6",
16    "cors": "^2.8.5",
17    "dotenv": "^16.4.5",
18    "express": "^4.19.2",
19    "jsonwebtoken": "^9.0.2",
20    "multer": "^1.4.5-lts.1",
21    "pg": "^8.12.0"
22  },
23  "devDependencies": {
24    "chai": "^4.5.0",
25    "chai-http": "^4.4.0",
26    "eslint": "^9.9.0",
27    "mocha": "^10.7.3"
28  },
29  "private": true
30 }
```

3) Implementar la autenticación y autorización de usuarios con JWT

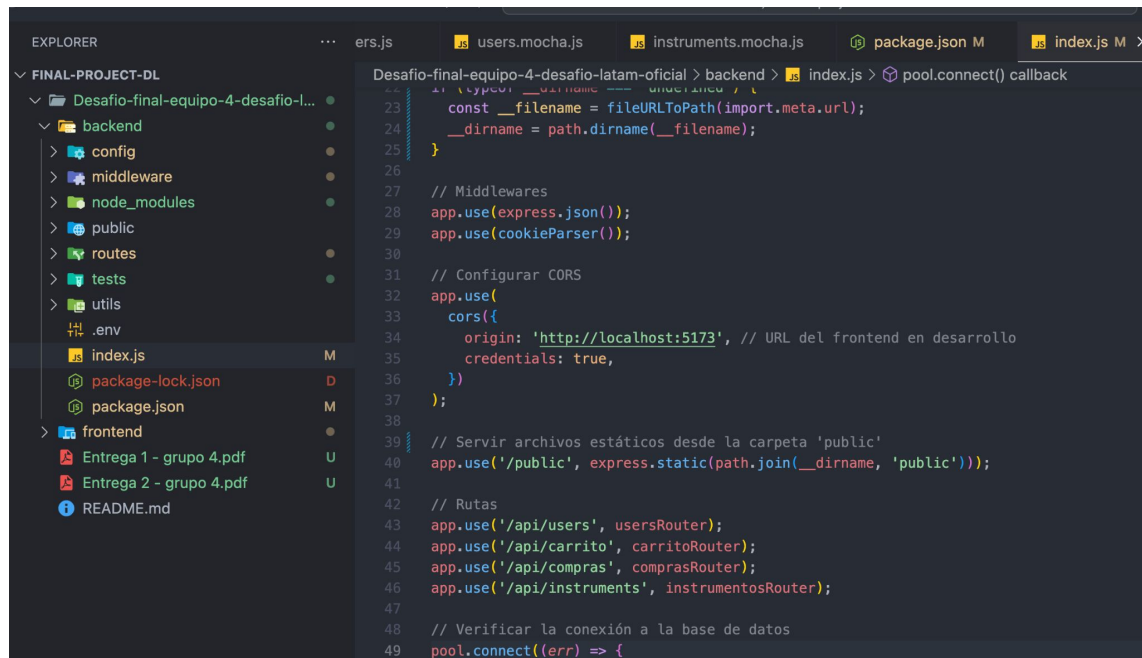
(users.js)



```
Desafio-final-equipos-4-desafio-latam-oficial > backend > routes > users.js > router.post('/register') callback
38  });
39
40  // Iniciar sesión y generar un token JWT
41  router.post('/login', async (req, res) => {
42    const { email, password } = req.body;
43
44    try {
45      // Buscar el usuario por email
46      const user = await pool.query('SELECT * FROM users WHERE email = $1', [email]);
47
48      if (user.rows.length === 0) {
49        return res.status(400).json({ error: 'Usuario no encontrado.' });
50      }
51
52      // Verificar la contraseña
53      const isMatch = await bcrypt.compare(password, user.rows[0].password);
54      if (!isMatch) {
55        return res.status(400).json({ error: 'Contraseña incorrecta.' });
56      }
57
58      // Crear el token JWT
59      const token = jwt.sign({ id: user.rows[0].id, email: user.rows[0].email }, process.env.SECRET_KEY, {
60        expiresIn: '1h',
61      });
62
63      // Devolver el token y detalles del usuario
64      res.json({
65        token,
66        user: {
67          id: user.rows[0].id,
68          name: user.rows[0].name,
69          email: user.rows[0].email,
```

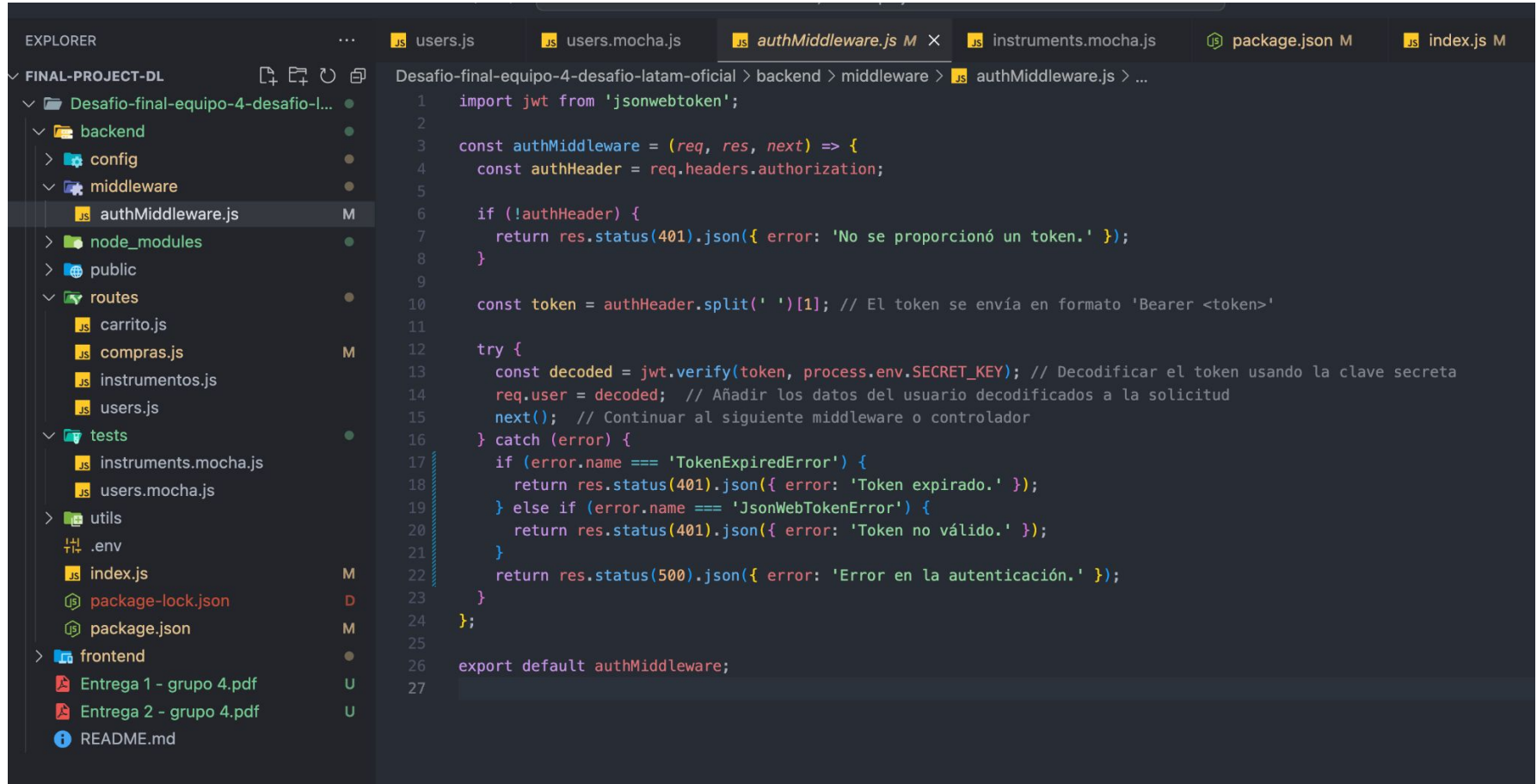
4) Usar el paquete CORS para permitir las consultas de orígenes cruzados. (index.js)

En el proyecto se utiliza el paquete **CORS** para habilitar la funcionalidad de consultas de orígenes cruzados en el servidor backend de Node.js con Express. Esto es esencial cuando el frontend y el backend se ejecutan en dominios diferentes, como en el caso de un frontend React en <http://localhost:5173> y un backend Node.js en <http://localhost:5000>.



```
Desafio-final-equipo-4-desafio-latam-oficial > backend > index.js > pool.connect() callback
23 const __filename = fileURLToPath(import.meta.url);
24 __dirname = path.dirname(__filename);
25 }
26
27 // Middlewares
28 app.use(express.json());
29 app.use(cookieParser());
30
31 // Configurar CORS
32 app.use(
33   cors({
34     origin: 'http://localhost:5173', // URL del frontend en desarrollo
35     credentials: true,
36   })
37 );
38
39 // Servir archivos estáticos desde la carpeta 'public'
40 app.use('/public', express.static(path.join(__dirname, 'public')));
41
42 // Rutas
43 app.use('/api/users', usersRouter);
44 app.use('/api/carrito', carritoRouter);
45 app.use('/api/compras', comprasRouter);
46 app.use('/api/instrumentos', instrumentosRouter);
47
48 // Verificar la conexión a la base de datos
49 pool.connect((err) => {
```

5) Utilizar middlewares para validar las credenciales o token en cabeceras en las rutas que aplique.



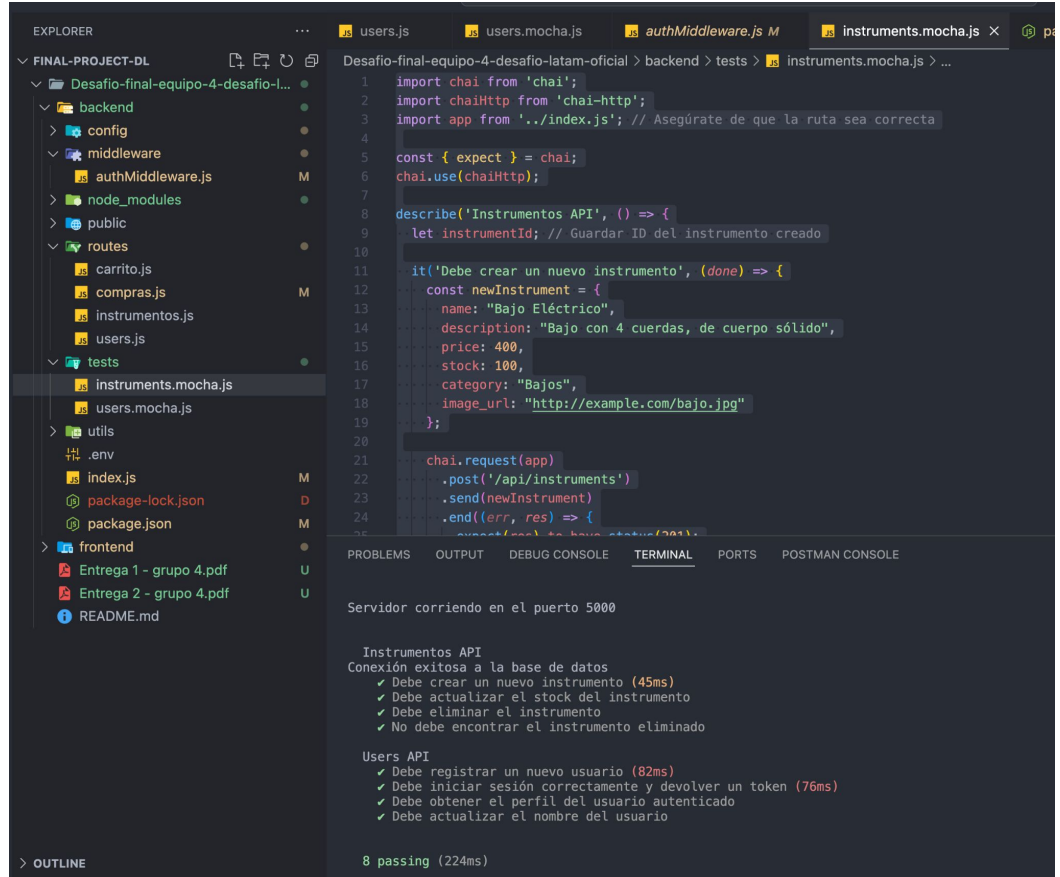
```
EXPLORER
FINAL-PROJECT-DL
  Desafio-final-equipo-4-desafio-l...
    backend
      config
      middleware
        authMiddleware.js
      node_modules
      public
      routes
        carrito.js
        compras.js
        instrumentos.js
        users.js
      tests
        instruments.mocha.js
        users.mocha.js
      utils
      .env
      index.js
      package-lock.json
      package.json
    frontend
      Entrega 1 - grupo 4.pdf
      Entrega 2 - grupo 4.pdf
      README.md

Desafio-final-equipo-4-desafio-latam-oficial > backend > middleware > authMiddleware.js > ...
1  import jwt from 'jsonwebtoken';
2
3  const authMiddleware = (req, res, next) => {
4    const authHeader = req.headers.authorization;
5
6    if (!authHeader) {
7      return res.status(401).json({ error: 'No se proporcionó un token.' });
8    }
9
10   const token = authHeader.split(' ')[1]; // El token se envía en formato 'Bearer <token>'
11
12   try {
13     const decoded = jwt.verify(token, process.env.SECRET_KEY); // Decodificar el token usando la clave secreta
14     req.user = decoded; // Añadir los datos del usuario decodificados a la solicitud
15     next(); // Continuar al siguiente middleware o controlador
16   } catch (error) {
17     if (error.name === 'TokenExpiredError') {
18       return res.status(401).json({ error: 'Token expirado.' });
19     } else if (error.name === 'JsonWebTokenError') {
20       return res.status(401).json({ error: 'Token no válido.' });
21     }
22     return res.status(500).json({ error: 'Error en la autenticación.' });
23   }
24 };
25
26 export default authMiddleware;
27
```

6) Realizar test de por lo menos 4 rutas de la API REST comprobando los códigos de estados de diferentes escenarios.

funcionaron los test de endpoints de instrumentos y usuario pero, tuvimos que usar una alternativa a jest y supertest que fue mocha y chaiHttp

(Esto por encontrar muchos errores de incompatibilidad entre formato ECMA y CJS)



The screenshot shows a VS Code editor with a project named 'FINAL-PROJECT-DL'. The Explorer sidebar on the left shows the project structure, including folders for 'backend', 'frontend', and 'tests'. The 'tests' folder contains 'instruments.mocha.js' and 'users.mocha.js'. The main editor displays the content of 'instruments.mocha.js', which is a Mocha test file using Chai and ChaiHttp. The test file includes imports, a describe block for 'Instrumentos API', and an it block for 'Debe crear un nuevo instrumento'. The test code is as follows:

```
1 import chai from 'chai';
2 import chaiHttp from 'chai-http';
3 import app from '../index.js'; // Asegúrate de que la ruta sea correcta
4
5 const { expect } = chai;
6 chai.use(chaiHttp);
7
8 describe('Instrumentos API', () => {
9   let instrumentId; // Guardar ID del instrumento creado
10
11   it('Debe crear un nuevo instrumento', (done) => {
12     const newInstrument = {
13       name: "Bajo Eléctrico",
14       description: "Bajo con 4 cuerdas, de cuerpo sólido",
15       price: 400,
16       stock: 100,
17       category: "Bajos",
18       image_url: "http://example.com/bajo.jpg"
19     };
20
21     chai.request(app)
22       .post('/api/instruments')
23       .send(newInstrument)
24       .end((err, res) => {
25         expect(res).to.have.status(201);
26       });
27   });
28
29   // ... other tests ...
30 });
```

The bottom panel of the editor shows the 'TERMINAL' tab, which displays the output of the test runner. It indicates that the server is running on port 5000 and lists the results of the tests:

```
Servidor corriendo en el puerto 5000

Instrumentos API
Conexión exitosa a la base de datos
✓ Debe crear un nuevo instrumento (45ms)
✓ Debe actualizar el stock del instrumento
✓ Debe eliminar el instrumento
✓ No debe encontrar el instrumento eliminado

Users API
✓ Debe registrar un nuevo usuario (82ms)
✓ Debe iniciar sesión correctamente y devolver un token (76ms)
✓ Debe obtener el perfil del usuario autenticado
✓ Debe actualizar el nombre del usuario

8 passing (224ms)
```