



Desarrollo de FullStack

AI07064821

Diego Villarreal Martinez

Profesor: Francisco Gomez Rubio

Link del github:

[https://github.co](https://github.com)

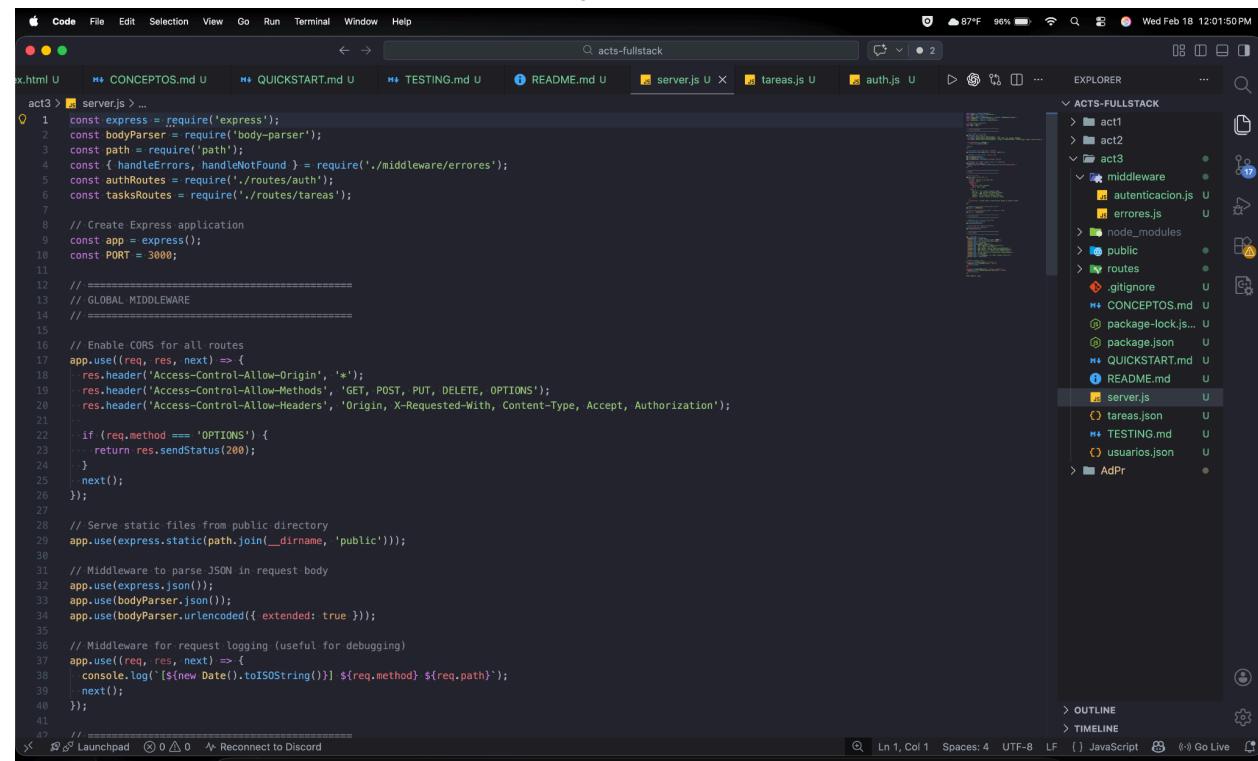
m/WebbiestStew/

acts-fullstack.git

Se inicio el npm y se descargaron las dependencias necesarias

```
● cd "/Users/diegovillarreal/Desktop/sc  
hool/fourth  
semester/fullstack/acts-  
fullstack/act3" && npm install  
  
added 83 packages, and audited 84 packages in  
15 packages are looking for funding  
  run `npm fund` for details  
found 0 vulnerabilities
```

Despues se hizo un archivo llamado Server.js



The screenshot shows a Mac OS X desktop with VS Code open. The title bar says "acts-fullstack". The left sidebar shows files like CONCEPTOS.md, QUICKSTART.md, README.md, server.js, tareas.js, auth.js, and TESTING.md. The right sidebar shows a tree view of the project structure under "ACTS-FULLSTACK", including act1, act2, act3 (with middleware, autenticacion.js, errores.js), node\_modules, public, routes, .gitignore, and several JSON files. The bottom status bar shows "Ln 1, Col 1" and "Spaces: 4".

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3 const path = require('path');
4 const { handleErrors, handleNotFound } = require('./middleware/errores');
5 const authRoutes = require('./routes/auth');
6 const taskRoutes = require('./routes/tareas');
7
8 // Create Express application
9 const app = express();
10 const PORT = 3000;
11
12 // =====
13 // GLOBAL MIDDLEWARE
14 // =====
15
16 // Enable CORS for all routes
17 app.use((req, res, next) => {
18   res.header('Access-Control-Allow-Origin', '*');
19   res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');
20   res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, Authorization');
21
22   if (req.method === 'OPTIONS') {
23     return res.sendStatus(200);
24   }
25   next();
26 });
27
28 // Serve static files from public directory
29 app.use(express.static(path.join(__dirname, 'public')));
30
31 // Middleware to parse JSON in request body
32 app.use(bodyParser.json());
33 app.use(bodyParser.urlencoded({ extended: true }));
34
35 // Middleware for request logging (useful for debugging)
36 app.use((req, res, next) => {
37   console.log(`${new Date().toISOString()} ${req.method} ${req.path}`);
38   next();
39 });
40
41 // =====
```

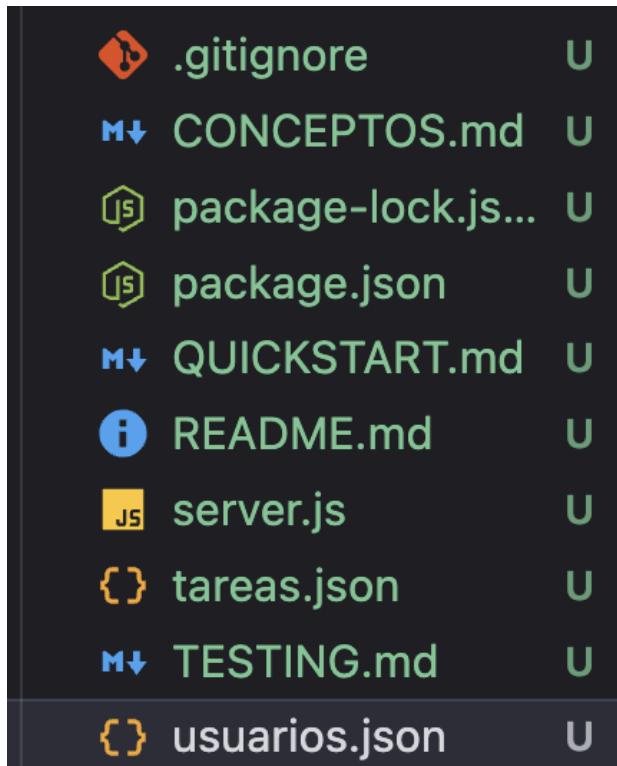
A screenshot of the Visual Studio Code interface. The title bar shows 'acts-fullstack'. The left sidebar has tabs for 'x.html U', 'CONCEPTOS.md U', 'QUICKSTART.md U', 'TESTING.md U', 'README.md U', 'server.js U', 'tareas.js U', 'auth.js U', and 'AdPr'. The main editor area displays the 'server.js' file with code for setting up an API. The right sidebar shows the 'ACTS-FULLSTACK' folder structure with files like 'act1', 'act2', 'act3', 'middleware', 'public', 'routes', and various JSON and MD files. The bottom status bar shows 'Ln 1, Col 1' and other standard info.

```
act3 > server.js > ...
45 // API info route
46 app.get('/api', (req, res) => {
47   res.json({
48     message: 'Welcome to the Tasks API',
49     version: '1.0.0',
50     endpoints: {
51       auth: {
52         register: 'POST /register',
53         login: 'POST /login'
54       },
55       tasks: {
56         'get all': 'GET /tareas (requires token)',
57         'get one': 'GET /tareas/:id (requires token)',
58         'create': 'POST /tareas (requires token)',
59         'update': 'PUT /tareas/:id (requires token)',
60         'delete': 'DELETE /tareas/:id (requires token)'
61       }
62     },
63     instructions: 'Include token in Authorization header as: Bearer <token>'
64   });
65 });
66 );
67 );
68 // Authentication routes (no additional prefix, already in router)
69 app.use('/', authRoutes);
70 );
71 // Task routes (no additional prefix, already in router)
72 app.use('/', tasksRoutes);
73 );
74 =====
75 // ERROR HANDLING
76 =====
77 );
78 // Middleware for routes not found (404)
79 // Must go after all routes
80 app.use(handleNotFound);
81 );
82 // Centralized error handling middleware
83 // Must be the last middleware
84 app.use(handleErrors);
85 );
86 =====
```

A screenshot of the Visual Studio Code interface, identical to the first one but with a few changes in the 'server.js' code. Lines 86 through 119 have been added to handle port listening and unhandled rejections.

```
act3 > server.js > ...
86 // =====
87 // START SERVER
88 // =====
89 app.listen(PORT, () => {
90   console.log(`=repeat(50)`);
91   console.log(` Server running on port ${PORT}`);
92   console.log(` URL: http://localhost:${PORT}`);
93   console.log(`=repeat(50)`);
94   console.log(`Available endpoints:`);
95   console.log(` POST /register - Register new user`);
96   console.log(` POST /login - User login`);
97   console.log(` GET /tareas - Get all tasks (authenticated)`);
98   console.log(` POST /tareas - Create new task (authenticated)`);
99   console.log(` PUT /tareas/:id - Update task (authenticated)`);
100  console.log(` DELETE /tareas/:id - Delete task (authenticated)`);
101  console.log(`=repeat(50)`);
102  console.log(` For debugging, run: node --inspect server.js`);
103  console.log(`=repeat(50)`);
104  console.log(`=repeat(50)`);
105 });
106 );
107 // Handle uncaught errors
108 process.on('uncaughtException', (error) => {
109   console.error(`✖ Uncaught error:`, error);
110   process.exit(1);
111 });
112 );
113 process.on('unhandledRejection', (reason, promise) => {
114   console.error(`✖ Unhandled promise rejection:`, reason);
115   process.exit(1);
116 });
117 );
118 module.exports = app;
119 );
```

Se hicieron los JSON correspondientes



Se hizo el middleware tambien

A screenshot of the Visual Studio Code interface. The code editor shows the file `autenticacion.js` with the following content:

```
act3 > middleware > autenticacion.js > ...
1 const jwt = require('jsonwebtoken');
2
3 // Secret key for JWT (in production should be in environment variables)
4 const JWT_SECRET = 'clave_secreta_super_segura_2024';
5
6 /**
7 * Middleware to authenticate JWT tokens
8 * Verifies that the token in the Authorization header is valid
9 */
10 function authenticateToken(req, res, next) {
11     // Get token from Authorization header
12     const token = req.headers['authorization'];
13
14     // Check if token is missing
15     if (!token) {
16         return res.status(401).json({
17             error: 'Access denied',
18             message: 'No authentication token provided'
19         });
20     }
21
22     try {
23         // Verify and decode the token
24         // Token can come as "Bearer TOKEN", so we extract only the token
25         const cleanToken = token.startsWith('Bearer ') ? token.slice(7) : token;
26
27         const decoded = jwt.verify(cleanToken, JWT_SECRET);
28
29         // Add user data to request object
30         req.user = decoded;
31
32         // Continue to next middleware function or route
33         next();
34     } catch (err) {
35         return res.status(403).json({
36             error: 'Invalid token',
37             message: 'The provided token is not valid or has expired'
38         });
39     }
40 }
41
42 module.exports = {
```

The Explorer sidebar on the right shows the project structure:

- ACTS-FULLSTACK
- act1
- act2
- act3
- middleware
  - autenticacion.js U
  - errores.js U
- node\_modules
- public
- routes
- .gitignore U
- CONCEPTOS.md U
- package-lock.json... U
- package.json U
- QUICKSTART.md U
- README.md U
- server.js U
- tareas.json U
- TESTING.md U
- usuarios.json U

VS Code interface showing the errors.js file in the act3 folder. The code handles various error types and sends a JSON response with status code, error, message, stack, and details.

```
act3 > middleware > errors.js > ...
1  /**
2   * Custom middleware for centralized error handling
3   * This should be the last middleware in the application
4   */
5  function handleErrors(err, req, res, next) {
6    // Error logging for debugging
7    console.error('==== ERROR CAUGHT ====');
8    console.error(`Route: ${req.method} ${req.path}`);
9    console.error(`Error: ${err.message}`);
10   console.error(`Stack: ${err.stack}`);
11   console.error('=====');
12
13   // Determine status code
14   let statusCode = err.statusCode || err.status || 500;
15
16   // Error message
17   let message = err.message || 'Internal server error';
18
19   // Custom error response based on type
20   if (err.name === 'ValidationError') {
21     statusCode = 400;
22     message = `Validation error: ${err.message}`;
23   } else if (err.name === 'JsonWebTokenError') {
24     statusCode = 403;
25     message = 'Invalid JWT token';
26   } else if (err.name === 'TokenExpiredError') {
27     statusCode = 401;
28     message = 'Expired JWT token';
29   } else if (err.code === 'ENOENT') {
30     statusCode = 500;
31     message = 'Error accessing data file';
32   }
33
34   // Send error response
35   res.status(statusCode).json({
36     error: true,
37     message: message,
38     ...(process.env.NODE_ENV === 'development' && {
39       stack: err.stack,
40       details: err
41     })
42   });
43 }
44
45 /**
46  * Middleware to handle routes not found (404)
47 */
48 function handleNotFound(req, res, next) {
49   res.status(404).json({
50     error: true,
51     message: `Route not found: ${req.method} ${req.path}`,
52     path: req.path,
53     method: req.method
54   });
55 }
56
57 module.exports = {
58   handleErrors,
59   handleNotFound
60 };
61
```

VS Code interface showing the errors.js file in the act3 folder. The code handles various error types and sends a JSON response with status code, error, message, stack, and details.

```
act3 > middleware > errors.js > ...
1  /**
2   * Custom middleware for centralized error handling
3   * This should be the last middleware in the application
4   */
5  function handleErrors(err, req, res, next) {
6    // Error logging for debugging
7    console.error('==== ERROR CAUGHT ====');
8    console.error(`Route: ${req.method} ${req.path}`);
9    console.error(`Error: ${err.message}`);
10   console.error(`Stack: ${err.stack}`);
11   console.error('=====');
12
13   // Determine status code
14   let statusCode = err.statusCode || err.status || 500;
15
16   // Error message
17   let message = err.message || 'Internal server error';
18
19   // Custom error response based on type
20   if (err.name === 'ValidationError') {
21     statusCode = 400;
22     message = `Validation error: ${err.message}`;
23   } else if (err.name === 'JsonWebTokenError') {
24     statusCode = 403;
25     message = 'Invalid JWT token';
26   } else if (err.name === 'TokenExpiredError') {
27     statusCode = 401;
28     message = 'Expired JWT token';
29   } else if (err.code === 'ENOENT') {
30     statusCode = 500;
31     message = 'Error accessing data file';
32   }
33
34   // Send error response
35   res.status(statusCode).json({
36     error: true,
37     message: message,
38     ...(process.env.NODE_ENV === 'development' && {
39       stack: err.stack,
40       details: err
41     })
42   });
43 }
44
45 /**
46  * Middleware to handle routes not found (404)
47 */
48 function handleNotFound(req, res, next) {
49   res.status(404).json({
50     error: true,
51     message: `Route not found: ${req.method} ${req.path}`,
52     path: req.path,
53     method: req.method
54   });
55 }
56
57 module.exports = {
58   handleErrors,
59   handleNotFound
60 };
61
```

Se hizo un HTML con todo (debido a sus altos contenidos, estara en el repo de github)

## Ejecucion:

The image shows a MacBook Pro screen with two windows open.

**Top Window (Safari Browser):**

- Title: Tasks API - Web Interface
- URL: localhost
- Content: A "Tasks Manager" application. The header says "Manage your tasks with ease". It shows three counts: 0 Total Tasks, 0 Active Tasks, and 0 Completed. Below is a "Create New Task" form with fields for Title and Description, and a blue "Add Task" button. At the bottom, it says "My Tasks" and "No tasks yet! Create your first task above."

**Bottom Window (VS Code Editor):**

- Title: Code
- File: index.html U
- Code Preview: CONCEPTOS.md U
- Code Preview: QUICKSTART.md U
- Code Preview: TESTING.md U
- Code Preview: README.md U
- Code Preview: server.js U
- Terminal: act3\$ public > index.html ...
- Output:

```
act3$ public > index.html ...
2  <html lang="en">
3    <head>
4      <meta name="viewport" content="width=device-width, initial-scale=1.0">
5      <title>Tasks API - Web Interface</title>
6      <style>
7        * {
8          margin: 0;
9          padding: 0;
10         box-sizing: border-box;
11       }
12     body {
13       font-family: 'Segoe UI', Tahome, Geneva, Verdana, sans-serif;
14       background: linear-gradient(135deg, ##67eea0, ##764ba2 100%);
```
- Problems: 0
- Output: 0
- Debug Console: 0
- Terminal: 0
- Ports: 0
- GitLens: 0
- Explorer: ACTS-FULLSTACK (tree view)
- Terminal Output:

```
act3$ public > index.html ...
2  <html lang="en">
3    <head>
4      <meta name="viewport" content="width=device-width, initial-scale=1.0">
5      <title>Tasks API - Web Interface</title>
6      <style>
7        * {
8          margin: 0;
9          padding: 0;
10         box-sizing: border-box;
11       }
12     body {
13       font-family: 'Segoe UI', Tahome, Geneva, Verdana, sans-serif;
14       background: linear-gradient(135deg, ##67eea0, ##764ba2 100%);
```

Server running on port 3000  
URL: http://localhost:3000

Available endpoints:

  - POST /register - Register new user
  - POST /login - User login
  - GET /tareas - Get all tasks (authenticated)
  - POST /tareas - Create new task (authenticated)
  - PUT /tareas/:id - Update task (authenticated)
  - DELETE /tareas/:id - Delete task (authenticated)

For debugging, run: node --inspect server.js

```
[2026-02-18T17:51:08.557Z] GET /favicon.ico
[2026-02-18T17:54:06.328Z] POST /register
[2026-02-18T17:54:06.426Z] GET /tareas
[2026-02-18T17:54:12.024Z] POST /tareas
[2026-02-18T17:54:14.866Z] DELETE /tareas/1
[2026-02-18T17:54:14.881Z] GET /tareas
```