

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS

Escuela profesional de Ciencia de la Computación



INFORME DE ARQUITECTURA DE INTEGRACIÓN CONTINUA

INGENIERÍA DE SOFTWARE II: Trabajo final de curso

Profesor: SARMIENTO CALIZAYA, Edgar

Integrantes:

PERCA QUISPE, Joel Cristian

HUANCA PARQUI, Elizabeth Yasmin

ESPINOZA PEÑALOZA, Edgar Alfonso

VILCHEZ MOLINA, Misael Svante

MOSCOSO APAZA, Imanol Brayan

VALENCIA ARANA, Gabriel Adriano

AQP - 2021

Índice

Índice	2
El proyecto	3
Funcionalidades	3
Tecnologías usadas	4
GitHub	4
Pipeline Jenkins	5
Construcción automática	6
Análisis estático	8
Pruebas unitarias	9
Pruebas de integración	11
Pruebas de seguridad	13
Pruebas de performance	15
Gestión de Issues	17
Despliegue automático	18
Conclusiones	20

1. El proyecto

El proyecto sobre el que se desarrollará el presente trabajo tiene por nombre **takenote** es una aplicación web de notas orientada para los desarrolladores.

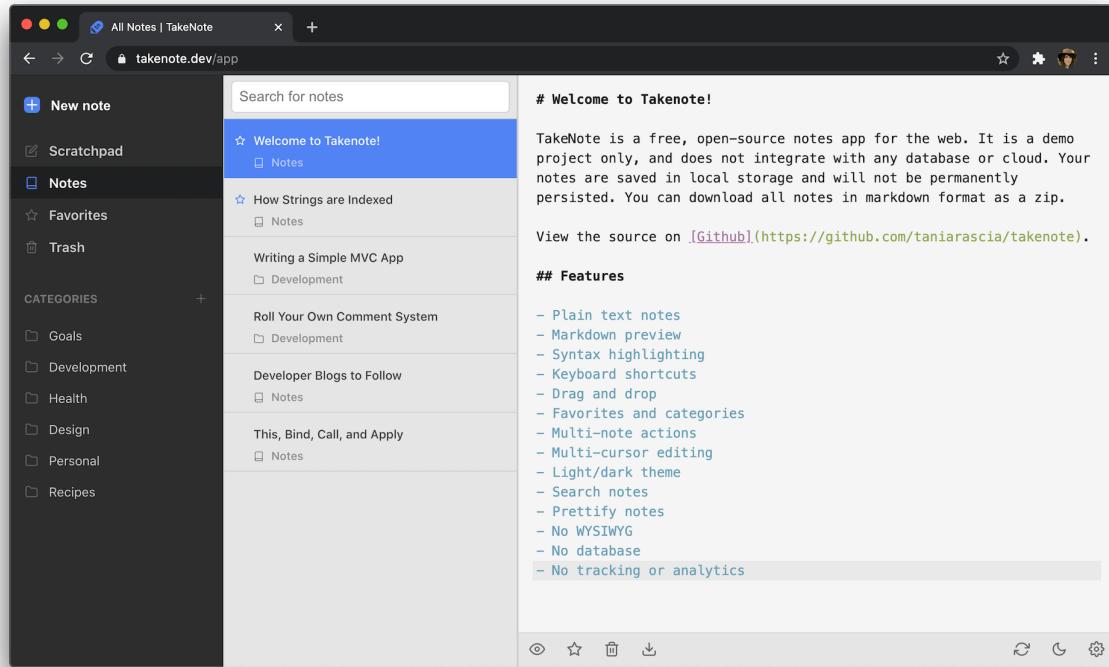


Imagen 1: Vista principal de la aplicación.

2. Funcionalidades

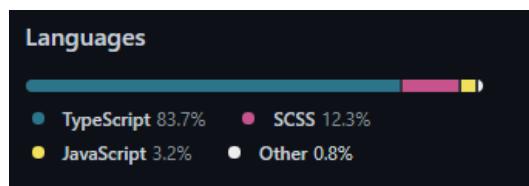
1. **Notas de texto sin formato:** tome notas en un entorno similar al IDE que no hace suposiciones.
2. **Vista previa de Markdown:** ver HTML renderizado.
3. **Notas vinculadas:** use la sintaxis `{{uuid}}` para vincular notas dentro de otras notas.
4. **Resaltado de sintaxis:** modo claro y oscuro disponible (basado en el hermoso *New Moon theme*).
5. **Atajos de teclado:** use el teclado para todas las tareas comunes: creación de notas y categorías, configuración de alternancia y otras opciones.
6. **Arrastrar y soltar:** arrastre una nota o varias notas a categorías, favoritos o papelera.
7. **Edición de varios cursores:** admite varios cursores y otras opciones de Codemirror.

8. **Buscar notas:** busque fácilmente todas las notas o notas dentro de una categoría.
9. **Embellece las notas:** usa Prettier sobre la marcha para tu *Markdown*.
10. **Sin WYSIWYG:** hecho para desarrolladores, por desarrolladores.
11. **Sin base de datos:** las notas solo se almacenan en el almacenamiento local del navegador y están disponibles para su descarga y exportación.
12. **Integración con GitHub:** la opción auto hospedada está disponible para la sincronización automática con un repositorio de GitHub (no disponible en la demostración).

3. Tecnologías usadas

Para el presente trabajo se empleó las tecnologías que se mencionan a continuación:

1. **Lenguajes de programación:** TypeScript, JavaScript, etc:



2. Jenkins para la integración de las diferentes herramientas <http://localhost:8080/>.
3. SonarQube para el análisis estático de código, ejecutándose en <http://localhost:9000/>.
4. NodeJS como entorno de ejecución para el lenguaje JavaScript y TypeScript.
5. NPM para manipular los paquetes de Node además de permitir la construcción.
6. Git para el manejo de versiones y trabajo colaborativo.
7. IDE VSCode con sus extensiones de soporte para Docker.
8. Jest, Cypress para las pruebas unitarias así como para las *end-to-end testing*.
9. Docker para contenerización.
10. OWASP ZAP CLI y JMeter para las pruebas de seguridad y performance respectivamente.

4. GitHub

Como herramienta de gestión de versiones se usó Git de forma local y como repositorio online se usó GitHub alojado en el siguiente enlace: <https://github.com/requiem/takenote>. Las ventajas de esta han sido que se ha podido colaborar de forma grupal entre todo el equipo de desarrollo.

5. Pipeline Jenkins

```
1 pipeline {
2     agent any
3     stages {
4         stage('Construccion') {
5             steps {
6                 echo "Instalando dependencias..."
7                 powershell "npm install"
8
9                 echo "Compilando la aplicacion..."
10                powershell "npm run build"
11            }
12        }
13        stage('Analisis estatico') {
14            steps {
15                echo 'SonarQube...'
16                withSonarQubeEnv('SonarQube') {
17                    bat "C:\\sonar\\sonar-scanner\\bin\\sonar-scanner.bat"
18                }
19            }
20        }
21        stage('Pruebas unitarias') {
22            steps {
23                echo 'Ejecutando pruebas unitarias...'
24                powershell "npm run test"
25                echo "Generando reporte de pruebas..."
26                echo "Publicando reporte de pruebas..."
27                publishHTML (target: [
28                    allowMissing: false,
29                    alwaysLinkToLastBuild: false,
30                    keepAll: true,
31                    reportDir: 'reports/jest',
32                    reportFiles: 'test-report.html',
33                    reportName: "Reporte de pruebas unitarias",
34                ])
35            }
36        }
37        stage('Pruebas funcionales') {
38            steps {
39                echo 'Ejecutando pruebas funcionales...'
40                powershell "npm run test:e2e"
41
42                echo "Generando reporte de pruebas..."
43                powershell "rmdir -r mochawesome-report"
44                powershell "npm run create:html:report"
45
46                echo "Publicando reporte..."
47                publishHTML (target: [
48                    allowMissing: false,
49                    alwaysLinkToLastBuild: false,
50                    keepAll: true,
51                    reportDir: 'mochawesome-report',
52                    reportFiles: 'cypress-combined-report.html',
53                    reportName: "Reporte de pruebas funcionales",
54                ])
55            }
56        }
57    }
58}
```

```

57         stage('Pruebas de seguridad') {
58             steps {
59                 echo "OWASP Security Tests"
60                 powershell "cd E:\\dev\\is\\"
61                 ZAP; ./zap.bat -cmd -quickurl https://dvwa.co.uk/ -quickout E:\\dev\\is\\test\\
62                 reportForDVWA.html"
63             }
64             echo "Publicando reporte"
65             publishHTML (target: [
66                 allowMissing: false,
67                 alwaysLinkToLastBuild: false,
68                 keepAll: true,
69                 reportDir: 'E:\\dev\\is\\test\\',
70                 reportFiles: 'reportForDVWA.html',
71                 reportName: "OWASP ZAP Report",
72             ])
73         }
74         stage('Pruebas de Performance') {
75             steps {
76                 echo "Performance Tests"
77
78                 bat "C:\\\\jmeter\\\\bin\\\\jmeter -n -t E:\\\\dev\\\\is\\\\test\\\\
79                 takenote_jmeter.jmx -l E:\\\\dev\\\\is\\\\test\\\\takenote_jmeter_report.jtl"
80                 perfReport "E:\\\\dev\\\\is\\\\test\\\\takenote_jmeter_report.jtl"
81             }
82         }
83         stage('Despliegue') {
84             steps {
85                 echo "Building Docker Image..."
86                 powershell "
87 docker build --build-arg CLIENT_ID=a7520b5205a31ddb8438 -t reqhiem/takenote:v1 ."
88
89                 echo "Deploying Docker Image..."
90                 powershell "docker push reqhiem/takenote:v1"
91
92             }
93         }
94     }

```

Imagen 2: Pipeline general del proyecto de CI/CD.

6. Construcción automática

Para la etapa de la construcción automática se emplea el gestor de paquetes y construcción de NodeJS llamada **NPM**. Esta se basa en un archivo de **JSON** para declarar todas las características del proyecto así como los comandos soportados, con nombre de archivo **package.json**.

```
"name": "takenote",
"version": "0.7.2",
"description": "A web-based notes app for developers.",
"author": "Pocho's Inc.",
"license": "MIT",
"private": false,
"main": "src/server/index.ts",
"scripts": {
  "dev": "concurrently \"npm run server\" \"npm run client\"",
  "client": "cross-env NODE_ENV=development webpack serve --config config/webpack.dev.js",
  "server": "nodemon --config config/nodemon.config.json",
  "build": "cross-env NODE_ENV=production webpack --config config/webpack.prod.js",
  "prod": "node -r ts-node/register/transpile-only src/server/index.ts",
  "start": "npm run client",
  "test": "jest --config config/jest.config.js",
  "test:e2e": "cypress run --config-file config/cypress.config.json",
  "test:e2e:open": "cypress open --config-file config/cypress.config.json",
  "test:coverage": "jest --config config/jest.config.js --coverage --watchAll=false",
  "test:coverage:ci": "
    jest --config config/jest.config.js --ci --coverage --watchAll=false && cat ./coverage/lcov.info | coveralls",
  "format": "prettier --write './**/*.{js,jsx,ts,tsx,css,scss,md}'",
  "eslint": "eslint src/**/*.{ts,tsx}"
},
"repository": {
  "type": "git",
  "url": "git+https://github.com/reqhiem/takenote"
},
```

```
  },
"dependencies": {
  "@reduxjs/toolkit": "^1.4.0",
  "axios": "^0.21.1",
  "clipboard-polyfill": "^3.0.1",
  "codemirror": "^5.58.1",
  "compression": "^1.7.4",
  "cookie-parser": "^1.4.5",
  "cors": "^2.8.5",
  "dayjs": "^1.9.3",
  "express": "^4.17.1",
  "helmet": "^4.1.1",
  "jszip": "^3.5.0",
  "mousetrap": "^1.6.5",
  "mousetrap-global-bind": "^1.1.0",
  "path-browserify": "^1.0.1",
  "prettier": "^2.1.2",
  "process": "^0.11.10",
  "react": "^16.14.0",
  "react-beautiful-dnd": "^13.0.0",
  "react-codemirror2": "^7.2.1",
  "react-device-detect": "^1.14.0",
  "react-dom": "^16.14.0",
  "react-feather": "^2.0.8",
  "react-helmet-async": "^1.0.7",
  "react-markdown": "^4.3.1",
  "react-redux": "^7.2.1",
  "react-router-dom": "^5.2.0",
  "react-split-pane": "^0.1.92",
  "redux": "^4.0.5",
  "redux-saga": "^1.1.3",
  "stream-browserify": "^3.0.0",
  "unist-util-visit": "^2.0.3",
  "uuid": "^8.3.1"
},
```

Imagen 3: Configuración del archivo `package.json` con la metadata del proyecto.

Y el paso que le corresponde en el Pipeline de Jenkins está definido en la siguiente etapa. Estos se manipulan mediante la línea de comandos **PowerShell** para lograr que este opere con la versión del las herramientas de NodeJS y NPM de nuestra máquina.



```
stage('Construccion') {
    steps {
        echo "Instalando dependencias..."
        powershell "npm install"

        echo "Compilando la aplicacion..."
        powershell "npm run build"
    }
}
```

Imagen 4: Configuración del *Pipeline* en la fase de construcción automática (Build).

7. Análisis estático

Para el análisis estático del código fuente se usó la herramienta de software **SonarQube** y su aditivo **SonarQube Scanner**. Los pasos a seguir para un correcto empleo de esta herramienta es como sigue:

1. Desde el panel de Jenkins, navegue hasta **Manage Jenkins > Manage Plugins** e instala el plugin **SonarQube Scanner**.
2. Desde el panel de Jenkins, navegue hasta **Manage Jenkins > Configure System**.
3. En la sección **SonarQube Servers**, le da click en **Add SonarQube**. Agrega la siguiente información:
 - a. **Name:** Dé un nombre único a su instancia de SonarQube.
 - b. **Server URL:** La URL de su instancia de SonarQube.
4. Le da click en **Save**.

Realizado dichos paso se procede a realizar la siguiente configuración en el archivo **Jenkinsfile**.



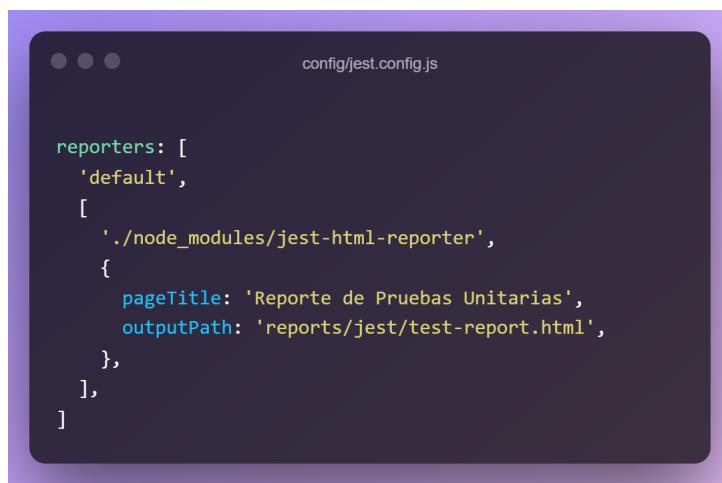
```
stage('Analisis estatico') {
    steps {
        echo 'SonarQube...'
        withSonarQubeEnv('SonarQube') {
            bat "C:\\sonar\\sonar-scanner\\bin\\sonar-scanner.bat"
        }
    }
}
```

Imagen 5: Configuración del *Pipeline* en la fase de análisis estático (*SonarQube*).

8. Pruebas unitarias

Para las pruebas unitarias se usó el marco de pruebas de JavaScript **Jest** debido a su simplicidad de uso y versatilidad de integración con las tecnologías emergentes. Los pasos necesarios para su completa automatización son:

1. Instalar las dependencias con el comando `npm install jest-html-reporter mocha mochawesome-merge mochawesome-report-generator`. El cual nos ayudará a generar reportes HTML sobre las *test suites* y caso de prueba.
2. Posteriormente se configura en el archivo `config/jest.config.js` adicionando a este como *reporter* por defecto al que se acaba de instalar, añadiendo también sus parámetros como el título de la página y el directorio de salida.



```
reporters: [
    'default',
    [
        './node_modules/jest-html-reporter',
        {
            pageTitle: 'Reporte de Pruebas Unitarias',
            outputPath: 'reports/jest/test-report.html',
        },
    ],
]
```

Imagen 6: Configuración del marco de pruebas para el *reporter* en HTML.

3. En el *pipeline* del proyecto ejecutamos el comando **NPM** que ejecuta las pruebas y además genera el reporte, para posteriormente publicarlo en la herramienta Jenkins y disponerlo en el panel de control de la construcción en Jenkins.



```
stage('Pruebas unitarias') {
    steps {
        echo 'Ejecutando pruebas unitarias...'
        powershell "npm run test"
        echo "Generando reporte de pruebas..."
        echo "Publicando reporte de pruebas..."
        publishHTML (target: [
            allowMissing: false,
            alwaysLinkToLastBuild: false,
            keepAll: true,
            reportDir: 'reports/jest',
            reportFiles: 'test-report.html',
            reportName: "Reporte de pruebas unitarias",
        ])
    }
}
```

Imagen 5: Configuración del *Pipeline* en la fase de Pruebas unitarias con Jest.

Si ejecutamos el comando en el directorio del proyecto obtenemos los siguientes resultados.

```
> takenote@0.7.2 test E:\dev\is\takenote
> jest --config config/jest.config.js

PASS  tests/unit/client/slices/sync.test.ts
PASS  tests/unit/client/slices/auth.test.ts
PASS  tests/unit/client/slices/settings.test.ts
PASS  tests/unit/client/slices/category.test.ts
PASS  tests/unit/client/utils/index.test.ts (5.104 s)
PASS  tests/unit/client/slices/note.test.ts (5.228 s)
PASS  tests/unit/client/components/NoteList/NoteListButton.test.tsx
PASS  tests/unit/client/components/editor/EditorEmpty.test.tsx
PASS  tests/unit/client/components/SettingsModal/IconButtonUploader.test.tsx
PASS  tests/unit/client/components/NoteList/SearchBar.test.tsx (5.267 s)
PASS  tests/unit/client/components/SettingsModal/IconButton.test.tsx
PASS  tests/unit/client/components/LastSyncedNotification.test.tsx (5.281 s)
PASS  tests/unit/server/middleware/checkAuth.test.ts
PASS  tests/unit/client/components/AppSidebar/FolderOption.test.tsx
PASS  tests/unit/client/components/AppSidebar/AddCategoryButton.test.tsx
PASS  tests/unit/client/components/AppSidebar/ActionButton.test.tsx
PASS  tests/unit/client/components/Switch.test.tsx
PASS  tests/unit/client/components/AppSidebar/CollapseCategoryButton.test.tsx
PASS  tests/unit/client/components/AppSidebar/ScratchpadOption.test.tsx
PASS  tests/unit/client/components/AppSidebar/AddCategoryForm.test.tsx
PASS  tests/unit/client/containers/ContextMenuOptions.test.tsx (5.903 s)
PASS  tests/unit/client/components/editor/PreviewEditor.test.tsx (6.283 s)
PASS  tests/unit/client/containers/TakeNoteApp.test.tsx (9.263 s)

Test Suites: 23 passed, 23 total
Tests:       101 passed, 101 total
Snapshots:   0 total
Time:        13.249 s
```

Imagen 7: Resultados de la ejecución de las pruebas unitarias en línea de comando.

Dichos resultados pueden ser plasmados en un informe generado por la herramienta `jest-html-generator`.

Reporte de Pruebas Unitarias

Started: 2021-12-29 23:22:03

Suites (23)	Tests (101)	
23 passed	101 passed	
0 failed	0 failed	
0 pending	0 pending	
E:\dev\is\takenote\tests\unit\client\slices\settings.test.ts		2.146s
settings slice	should return the initial state on first run	passed 0.002s
settings slice	should toggle open state	passed 0.001s
settings slice	should update code mirror option	passed 0s
settings slice	should toggle preview markdown state	passed 0s
settings slice	should toggle dark theme state	passed 0.001s

Imagen 8: Resultados de las pruebas unitarias en un reporte generado.

9. Pruebas de integración

Para las pruebas de integración se usó **Cypress**, una herramienta de prueba de interfaz de usuario de próxima generación creada para la web moderna, que ofrece pruebas rápidas, fáciles y confiables para cualquier cosa que se ejecute en un navegador.

```
Jenkinsfile - e2e testing

stage('Pruebas funcionales') {
    steps {
        echo 'Ejecutando pruebas funcionales...'
        powershell "rmdir -r mochawesome-report"
        powershell "npm run test:e2e"

        echo "Generando reporte de pruebas..."
        powershell "npm run create:html:report"

        echo "Publicando reporte..."
        publishHTML (target: [
            allowMissing: false,
            alwaysLinkToLastBuild: false,
            keepAll: true,
            reportDir: 'mochawesome-report',
            reportFiles: 'cypress-combined-report.html',
            reportName: "Reporte de pruebas funcionales",
        ])
    }
}
```

Imagen 9: Configuración del *Pipeline* en la fase de Pruebas de integración.

Los resultados de esta prueba se puede obtener de dos formas, en línea de comandos y una interfaz gráfica, además de poder generarse un reporte para tener persistencia de la última ejecución de las pruebas. En cuanto a la primera forma se obtiene una salida del tipo:

```
(Run Starting)

Cypress: 5.4.0
Browser: Electron 85 (headless)
Specs: 3 found (category.test.ts, note.test.ts, settings.test.ts)

Running: category.test.ts
(1 of 3)

Categories
  ✓ should hide the category list on click of category (1930ms)
  ✓ should show category list on add new category (587ms)
  ✓ creates a new category with the current time (461ms)
  ✓ should add a note to new category (1127ms)
  ✓ should rename existing category after defocusing edit state (871ms)
```

Imagen 10: Resultados de la prueba de integración en línea de comando.

Y en cuanto a la segunda forma se obtiene lo siguiente:

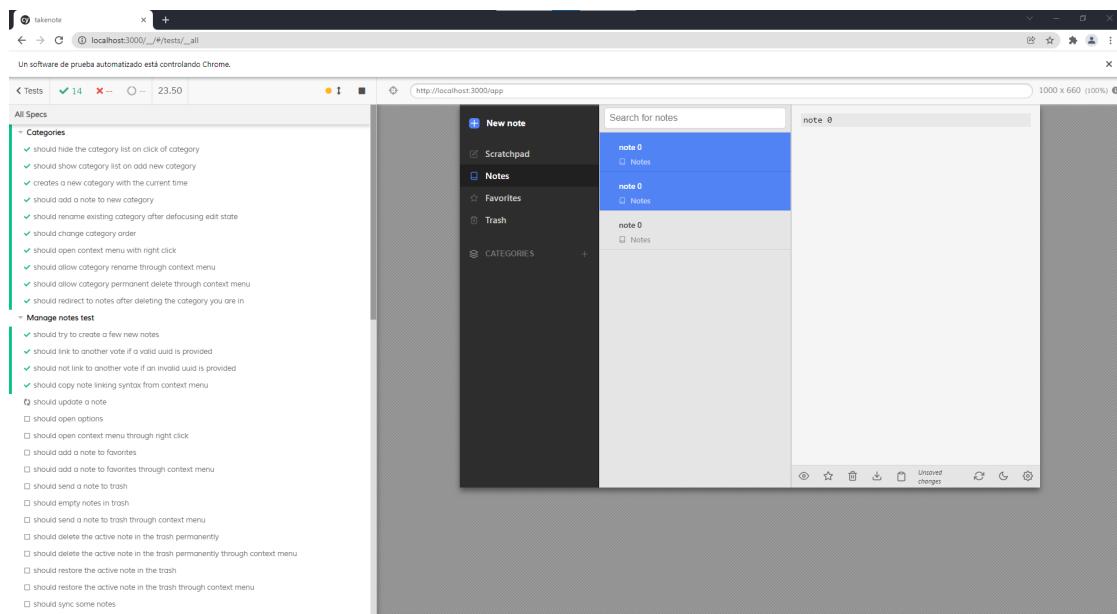


Imagen 11: Resultados de las pruebas de integración con la herramienta [Cypress.io](#) en una interfaz web.

Sin embargo para efectos de automatización se logra obtener un reporte personalizado con las herramientas de mocha, con resultados del tipo:

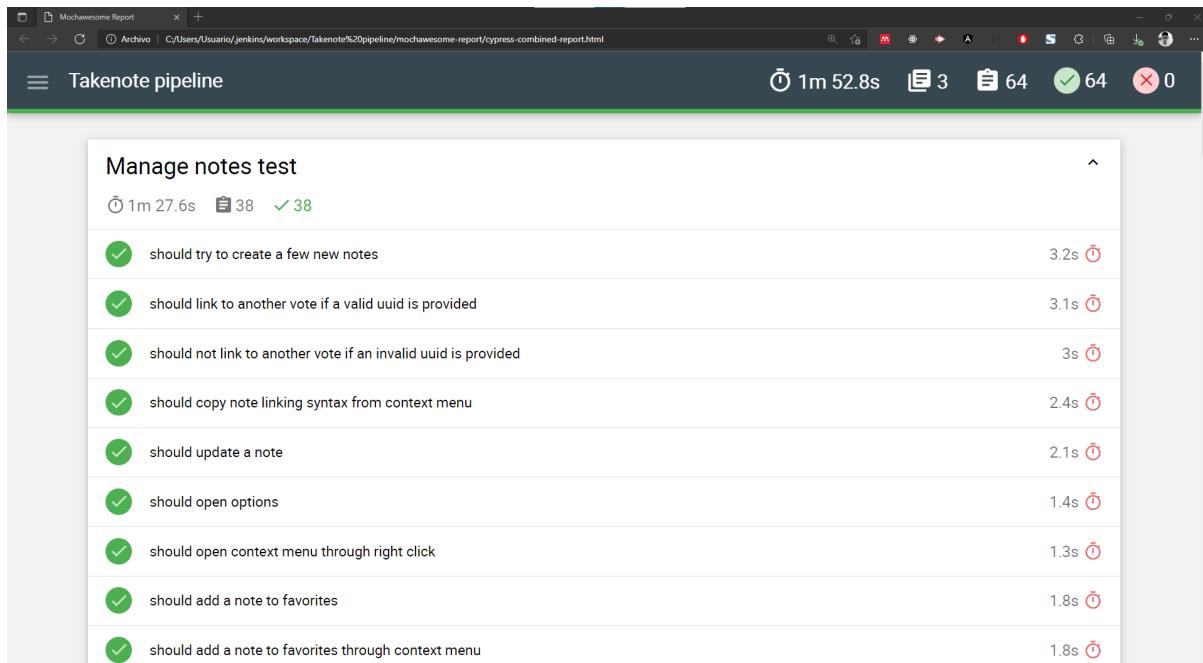


Imagen 12: Resultados de las pruebas de integración con la herramienta **Cypress.io** en un reporte generado

10. Pruebas de seguridad

Las pruebas de seguridad del software son el proceso de evaluar y probar un sistema para descubrir los riesgos de seguridad y las vulnerabilidades del sistema y sus datos. Para tal propósito se usó la herramienta **OWASP ZAP** el cual es un aplicación Java que se puede ejecutar en versión Standalone como un software que dispone de una interfaz gráfica de usuario; Sin embargo, también nos provee herramientas de automatización tales como:

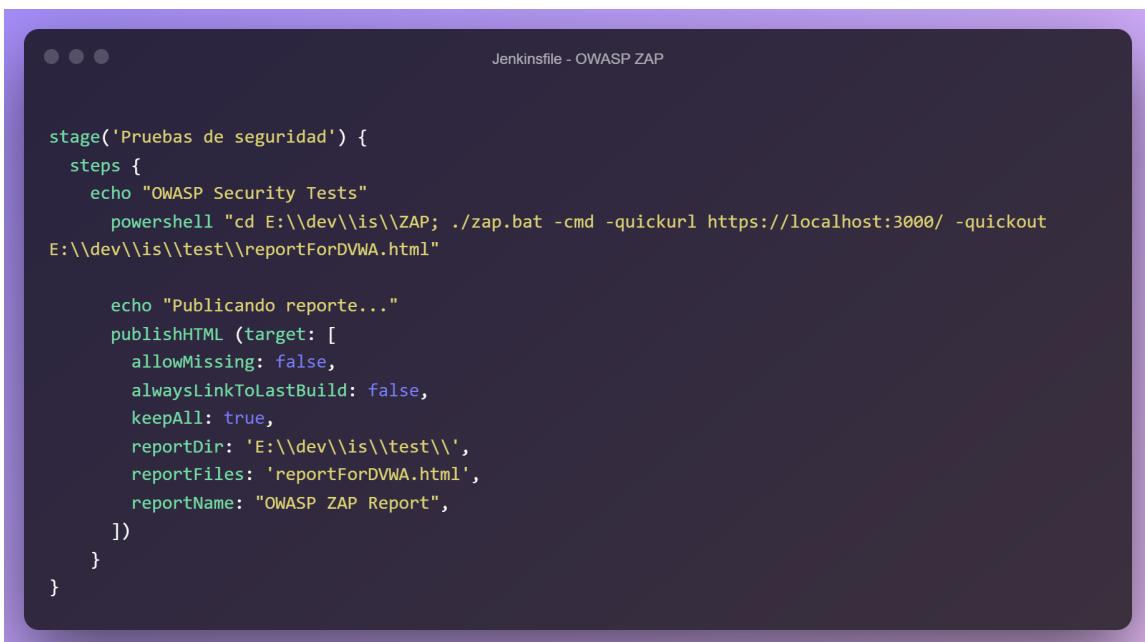
- Docker Packaged Scans
- GitHub Actions
- Automation Framework
- API and Daemon mode
- **Quick Start command line**

Este último nos provee un método de ejecución minimalista con parámetros que van desde lo sencillo hasta lo más elaborado.

-quickurl	Especifica la URL de la aplicación de destino que será atacada.
------------------	---

-quickout	Especifica el archivo en el que escribir el informe. El formato del informe dependerá de la extensión del archivo; las extensiones de archivo admitidas son .html , .json , .md , .xml
-quickprogress	Muestra barras de progreso ascii, si se inicia con el indicador -cmd.

Un ejemplo de uso puede ser el siguiente comando: cd E:\\dev\\is\\ZAP; ./zap.bat -cmd -quickurl https://localhost:3000/ -quickout E:\\dev\\is\\test\\reportForDVWA.html, que se desplaza al directorio donde se encuentra el ejecutable de OWASP ZAP y luego ejecuta una prueba de seguridad a nuestro servidor local. Para ello el programa de ZAP debería estar disponible en la dirección que se indica en el ejemplo.



```

stage('Pruebas de seguridad') {
    steps {
        echo "OWASP Security Tests"
        powershell "cd E:\\dev\\is\\ZAP; ./zap.bat -cmd -quickurl https://localhost:3000/ -quickout E:\\dev\\is\\test\\reportForDVWA.html"

        echo "Publicando reporte..."
        publishHTML (target: [
            allowMissing: false,
            alwaysLinkToLastBuild: false,
            keepAll: true,
            reportDir: 'E:\\dev\\is\\test\\',
            reportFiles: 'reportForDVWA.html',
            reportName: "OWASP ZAP Report",
        ])
    }
}

```

Imagen 13: Configuración del *Pipeline* en la fase de Pruebas de seguridad con la herramienta OWASP ZAP.

Esto nos genera un reporte con la siguiente estructura:

 OWASP ZAP Report

Site: <http://localhost:3000>

Generated on jue., 30 dic. 2021 04:26:23

Summary of Alerts

Nivel de riesgo	Number of Alerts
Alto	0
Medio	3
Bajo	3
Informativo	1
False Positives:	0

Alertas

Nombre	Nivel de riesgo	Number of Instances
CSP: Wildcard Directive	Medio	1
Exploración de directorios	Medio	1
X-Frame-Options Header Not Set	Medio	2
Divulgación de la marca de hora - Unix	Bajo	79
El servidor divulga información mediante un campo(s) de encabezado de respuesta HTTP ""X-Powered-By""	Bajo	6
X-Content-Type-Options Header Missing	Bajo	5
Divulgación de información - Comentarios sospechosos	Informativo	11

Alert Detail

Medio	CSP: Wildcard Directive
Descripción	The following directives either allow wildcard sources (or ancestors), are not defined, or are overly broadly defined: frame-ancestors, form-action The directive(s), frame-ancestors, form-action are among the directives that do not fallback to default-src, missing/excluding them is the same as allowing anything
URL	http://localhost:3000/sitemao.xml
Método	GET
Parameter	
Atacar	
Evidence	default-src 'none'
Instances	1
Solución	Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.
Reference	http://www.w3.org/TR/CSP2/ http://www.w3.org/TR/CSP/ http://caniuse.com/#search=content+security+policy http://content-security-policy.com/ https://github.com/shapesecurity/salvation https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources
CWE Id	693
WASC Id	15
Plugin Id	10056

Imagen 14: Reporte generado de las pruebas de seguridad con la herramienta ZAP CLI.

11. Pruebas de performance

Para las pruebas de performance, en principio definimos el plan pruebas mediante la interfaz gráfica de la herramienta **JMeter**, allí podemos agregar el grupo de hilos (Usuarios) configurando sus parámetros en una cantidad de usuarios gigante para estresar la dirección URL, el número de bucles a realizar, además configuramos el objeto HTTP Request con sus parámetros necesarios para testear la dirección URL <http://localhost:3000/> y por último se le añade objetos listener para captar la salida de los resultados añadiendo aserciones para corroborar el contenido de la página principal al ser un aplicativo de tipo SPA (*Single Page Application*).

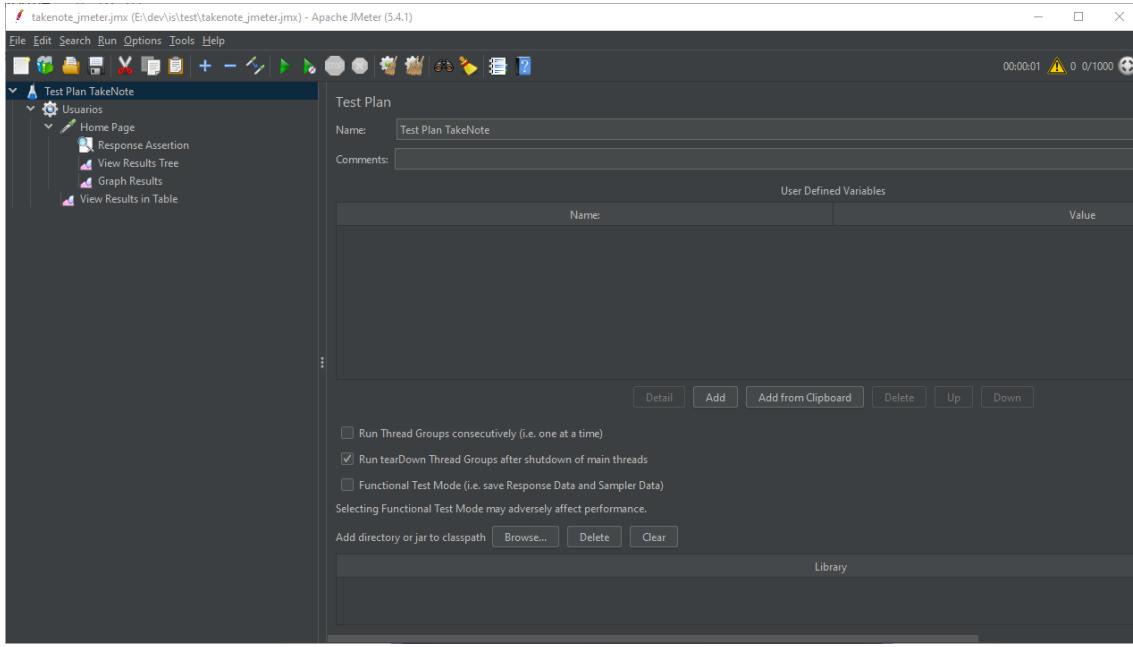


Imagen 15: Configuración del plan de pruebas en JMeter con *HTTP Requests*.

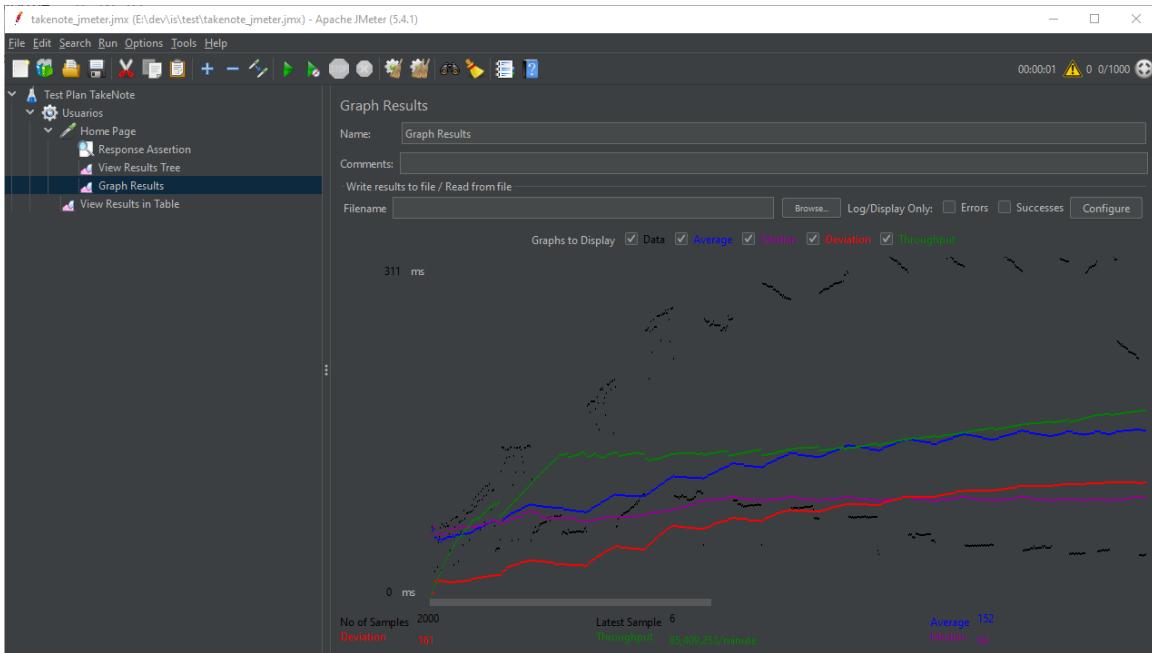


Imagen 16: Configuración del plan de pruebas en JMeter con *Listeners*.

Posteriormente se procede a guardar el plan de prueba en un directorio específico al que se pueda acceder posteriormente con formato .jmx. Luego podemos ejecutar dichas pruebas mediante la terminal con los comandos que se describen a continuación:



```

Jenkinsfile - Performance test

stage('Pruebas de Performance') {
    steps {
        echo "Performance Tests"
        bat "C:\\\\jmeter\\\\bin\\\\jmeter -n -t E:\\\\dev\\\\is\\\\test\\\\takenote_jmeter.jmx
-1 E:\\\\dev\\\\is\\\\test\\\\takenote_jmeter_report.jtl"

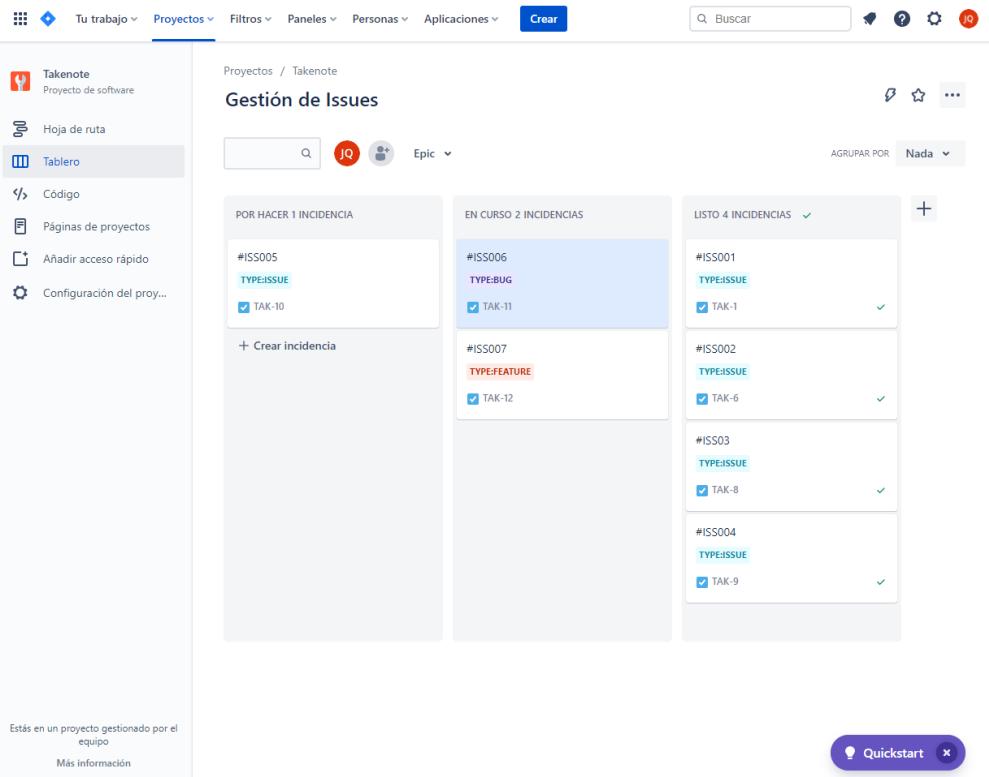
        perfReport "E:\\\\dev\\\\is\\\\test\\\\takenote_jmeter_report.jtl"
    }
}

```

Imagen 17: Configuración del *Pipeline* en la fase de Pruebas de performance con JMeter.

12. Gestión de Issues

Para la gestión de *Issues* se optó por usar la herramienta Jira debido a su versatilidad para la administración de tareas de un proyecto, el seguimiento de errores e incidencias y para la gestión operativa de proyectos.



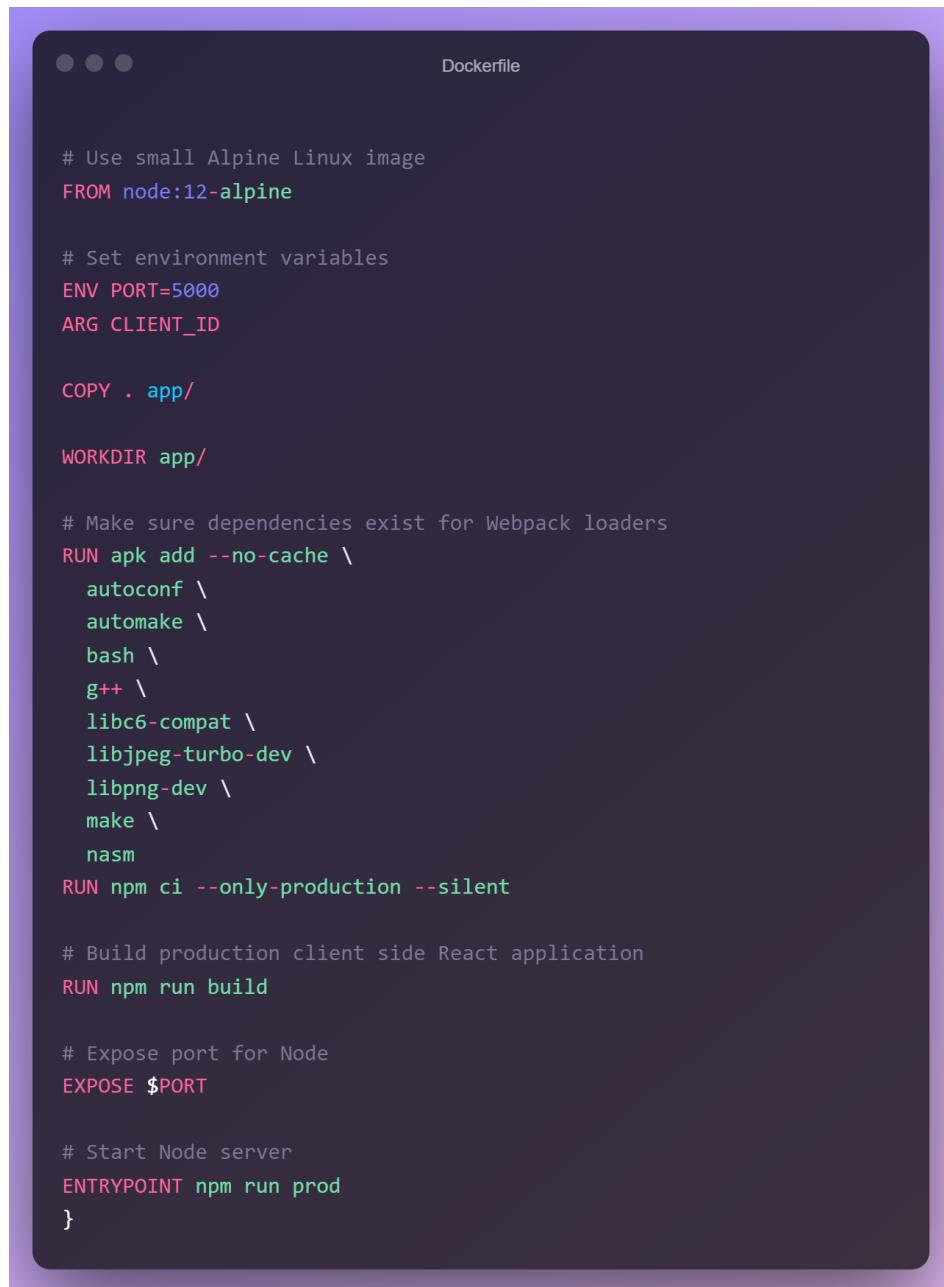
The screenshot shows the Jira main dashboard for the 'Takenote' project. On the left sidebar, there are navigation links: 'Tu trabajo', 'Proyectos' (selected), 'Filtros', 'Panesles', 'Personas', 'Aplicaciones', 'Crear', 'Buscar', and a settings gear icon. The main area is titled 'Gestión de Issues' and displays a board with three columns: 'POR HACER 1 INCIDENCIA', 'EN CURSO 2 INCIDENCIAS', and 'LISTO 4 INCIDENCIAS'. Each column contains a list of issues with their IDs, types (e.g., #ISS005, TYPE:ISSUE, TAK-10), and status. A '+' button is available to add new issues. At the bottom left, a message says 'Estás en un proyecto gestionado por el equipo' and 'Más información'. A 'Quickstart' button is at the bottom right.

Columna	Issue ID	Tipo	Estado
POR HACER 1 INCIDENCIA	#ISS005	TYPE:ISSUE	TAK-10
	#ISS006	TYPE:BUG	TAK-11
EN CURSO 2 INCIDENCIAS	#ISS007	TYPE:FEATURE	TAK-12
	#ISS006	TYPE:BUG	TAK-11
LISTO 4 INCIDENCIAS	#ISS001	TYPE:ISSUE	TAK-1
	#ISS002	TYPE:ISSUE	TAK-6
	#ISS003	TYPE:ISSUE	TAK-8
	#ISS004	TYPE:ISSUE	TAK-9

Imagen 18: Vista del tablero principal para la gestión de *Issues* con la herramienta Jira.

13. Despliegue automático

Para el despliegue, específicamente la contenerización del software se usó la herramienta Docker en el sistema operativo Windows empaquetar la versión de NodeJS que se emplea en el proyecto así como sus respectivos paquetes que emplea el programa. Dicha configuración está sujeta a los siguientes parámetros:



A screenshot of a code editor showing a Dockerfile. The file contains the following content:

```
# Use small Alpine Linux image
FROM node:12-alpine

# Set environment variables
ENV PORT=5000
ARG CLIENT_ID

COPY . app/

WORKDIR app/

# Make sure dependencies exist for Webpack loaders
RUN apk add --no-cache \
    autoconf \
    automake \
    bash \
    g++ \
    libc6-compat \
    libjpeg-turbo-dev \
    libpng-dev \
    make \
    nasm
RUN npm ci --only-production --silent

# Build production client side React application
RUN npm run build

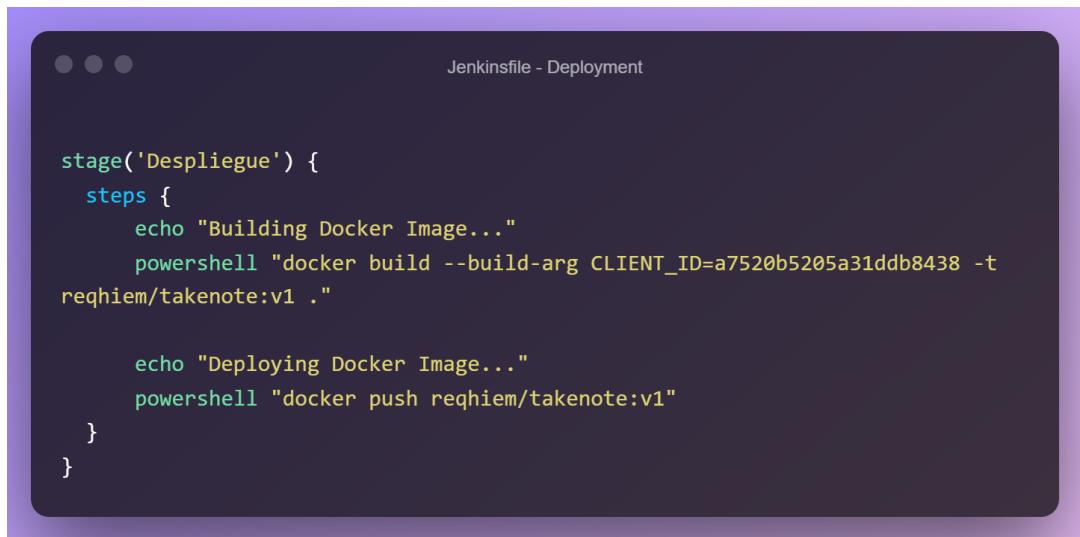
# Expose port for Node
EXPOSE $PORT

# Start Node server
ENTRYPOINT npm run prod
}
```

Imagen 19: Archivo de configuración *Dockerfile* con los *scripts* necesarios para construir una imagen del software a partir del directorio principal.

Y su implementación en el *Pipeline* de Jenkins obedece a la siguiente estructura, en principio se construye la imagen del contenedor con *tag* que hace referencia a una cuenta

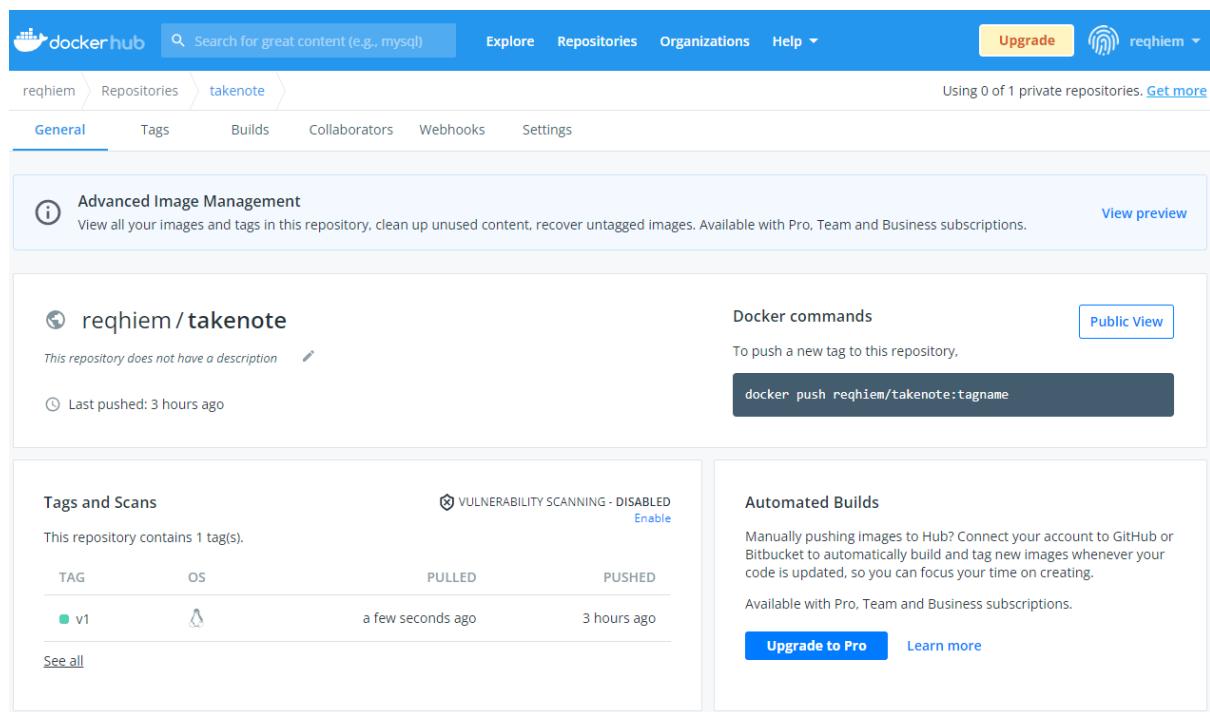
en **DockerHub** para poder posteriormente hacer un *push* de la imagen y liberar su disposición para el público general.



```
stage('Despliegue') {
    steps {
        echo "Building Docker Image..."
        powershell "docker build --build-arg CLIENT_ID=a7520b5205a31ddb8438 -t reqhiem/takenote:v1 ."

        echo "Deploying Docker Image..."
        powershell "docker push reqhiem/takenote:v1"
    }
}
```

Imagen 20: Configuración del *Pipeline* en la fase de contenerización con la herramienta *Docker*.



The screenshot shows the DockerHub interface for the repository `reqhiem/takenote`. At the top, there's a navigation bar with links for Explore, Repositories, Organizations, Help, Upgrade, and a user icon for `reqhiem`. Below the navigation, the repository path `reqhiem/takenote` is shown. The main content area displays the repository details for `takenote`, which was last pushed 3 hours ago. It includes sections for Docker commands (with a button to `docker push reqhiem/takenote:tagname`), Tags and Scans (showing one tag `v1` pulled a few seconds ago and pushed 3 hours ago), and Automated Builds (with a note about connecting to GitHub or Bitbucket). There are also buttons for `Upgrade to Pro` and `Learn more`.

Imagen 21: Captura del repositorio Docker donde se dispone al público una imagen estable del proyecto que se desarrolla.

Producto de la ejecución de los anteriores pasos y al integrar todos los componentes del Pipeline se obtiene el siguiente resultado siendo este la ejecución sucesiva de cada actividad.

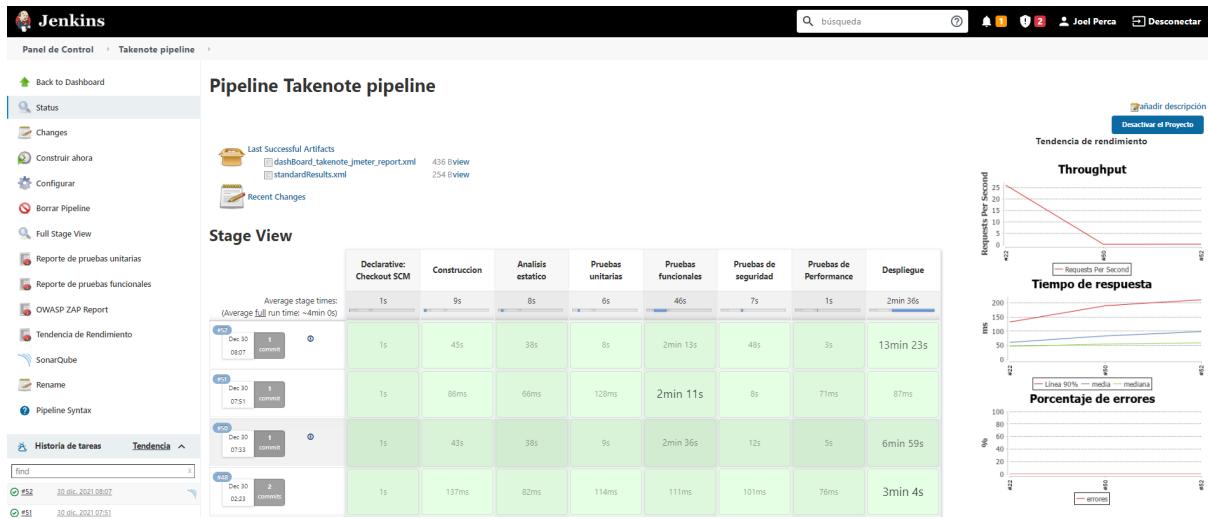


Imagen 22: Vista general de la ejecución del *Pipeline* en *Jenkins* de nuestra arquitectura de CI/CD.

14. Conclusiones

Como se ha podido apreciar a lo largo del presente informe se ha conseguido integrar las diferentes tecnologías vistos a lo largo del curso, se logró entonces un mayor entendimiento de la automatización de los procesos de las metodologías ágiles u otras con el objetivo de reducir tiempo y coste en el proceso del desarrollo de software, se ha conseguido una colaboración activa de los diferentes miembros del equipo teniendo en cuenta que se les asignó actividades específicas por lo que se distribuyó la carga de trabajo. Por último se logró entender con mayor detenimiento nuestro rol como desarrolladores y cómo puede optimizar los resultados siguiendo los estándares de diseño, desarrollo y las tecnologías a disposición.