

# ESLint, Sonar y control de calidad del código.

<b>ESLint, lint-staged y husky siempre al pie del cañón.</b>	<b>2</b>
ESLint, nuestro código como una patena.	2
Manchandonos las manos. Instalación de nuestro entorno.	3
ESLint.config.ts: personalizando Eslint.	4
Análisis de nuestras normas.	6
Lint-staged, no dejes que el tiempo vuele.	6
Instalando lint-staged.	7
Husky, el ojo de Sauron.	7
Instalando Husky.	7
<b>SonarQube. Tu entrada a la excelencia.</b>	<b>8</b>
Primeros pasos. Instalación.	9
Generando un entorno de trabajo.	10
Haciendo nuestro SonarQube.	15
SonarQube desde arriba, ¿para qué vale cada cosa?.	15
Security. La prioridad absoluta.	16
Reliability. De aquellos barro, estos lodos.	18
Maintainability. Invertir en nuestro futuro.	18
Coverage. Código con sello de calidad.	19
Duplications. Evita repetirte como un loro.	19
<b>Quality Gates. No todo lo que compila merece pasar.</b>	<b>20</b>
Creando nuestras propias Quality Gates.	21
<b>Quality Profiles. Las normas son para todos... pero estas son las mías.</b>	<b>23</b>
Creando nuestras propias reglas.	24
<b>El rincón del friki: afinando hasta lo invisible.</b>	<b>25</b>
Alertas por correo. Que no se te escape nada.	26
Analizadores externos. Subcontratando ayudantes.	26

Para poder desarrollar y mantener proyectos grandes de forma sostenible, es fundamental **estandarizar la forma en la que programamos**. Esto es igual de importante cuando trabajamos solos como cuando hay varios programadores implicados (aunque en este último caso es absolutamente crítico).

Definir patrones de organización de carpetas, distribución de ficheros y su nomenclatura es esencial para facilitar el mantenimiento. Pero más allá de cómo distribuimos nuestro código... El verdadero reto está en **cómo lo escribimos**.

Hay muchas formas de programar, algunas válidas y otras que (por decirlo suavemente) no deberían repetirse nunca. Incluso entre buenas prácticas, si dos desarrolladores usan enfoques distintos, la colaboración se resiente: cuesta más intercambiar tareas, incorporar nuevos miembros o garantizar coherencia a lo largo del tiempo.

En proyectos pequeños, con poca carga y un responsable técnico muy experimentado, se podría revisar manualmente cada commit, cada variable, cada fichero... Pero eso rara vez es realista (y cuando lo es, dura poco). Por eso, necesitamos herramientas automáticas que nos ayuden a vigilar la calidad de nuestro código sin depender de revisiones manuales constantes.

Este documento nace precisamente para eso. Estará dividido en dos bloques:

- En la primera parte, nos centraremos en **ESLint**: cómo integrarlo en el proyecto, establecer nuestras propias normas de codificación, y reforzarlo con herramientas como **Husky** y **lint-staged** para impedir que se suba código no validado a nuestras ramas principales.
- En la segunda parte, veremos cómo llevar la inspección del código a un nivel superior con **SonarQube**, desde su configuración hasta su integración en un entorno de **CI/CD**, garantizando así que cada despliegue pase por un filtro de calidad técnica automatizado.

El objetivo no es solo tener un código que funcione, sino tener un código mantenible, coherente y confiable. Y si logramos un 10/10 en el análisis, mejor que mejor (aunque aceptamos un 9... si viene acompañado de un buen commit).

## ESLint, lint-staged y husky siempre al pie del cañón.

### ESLint, nuestro código como una patena.

Si has leído el documento “Arquitectura para Frontales con Angular 19+”, ya sabes que es ESLint y por qué deberías utilizarlo, pero por si se te ha pasado, te dejo un pequeño resumen: ESLint es una herramienta de análisis estático que valida el código según un conjunto de reglas creadas por la comunidad o por nosotros mismos. Permite definir una metodología de codificación común a todos los programadores unificando el código. Esta herramienta es muy potente gracias a su ejecución continua, ya que aunque tienes diferentes scripts que puedes ejecutar para realizar diferentes labores (revisar todo el código en busca de errores, arreglarlos...) ESLint se puede configurar para que esté en constante ejecución, permitiendo al programador ver mientras programa que su código no sigue los estándares de la empresa.

Para que esta herramienta cumpla su función de una manera muy eficiente, no puede funcionar sola. necesita apoyarse en sus compañeras “lint-staged y husky”, pero... ¿para qué valen?.

Husky es una librería que nos va a permitir ejecutar scripts usando como disparadores eventos de git, como un commit. Esto nos garantiza que podremos ejecutar nuestros comandos de ESLint antes de hacer el commit, condicionando la subida a los resultados positivos.

Por otro lado tenemos Lint-staged, librería la cual nos permite ejecutar este comando solo en ficheros deseados, generalmente serán los modificados, pero se le pueden añadir otras condiciones muy interesantes. Esto permite reducir drásticamente el tiempo necesario para ejecutar estas comprobaciones, ya que en proyectos muy grandes, cambiar un solo componente pero pasar el script por otros 200 es muy ineficiente.

### Manchandonos las manos. Instalación de nuestro entorno.

Lo primero que necesitamos es un proyecto, en nuestro caso, uno en Angular. Con esta base, podemos comenzar.

- 1. Instalar ESLint: Para esto es muy recomendable hacerlo a través de “angular-eslint/schematics”, ya que no solo instala los paquetes, también crea el fichero de configuración y elimina TSlint (en caso de que el proyecto ya lo tuviese). Puedes obtener esta extensión desde [aquí](#) o mediante el comando: “ng add @angular-eslint/schematics” para hacerlo todo de manera automática. Si todo se ha instalado correctamente, nos mostrará un mensaje similar a este.

```

✓ Determining Package Manager
  > Using package manager: npm
✓ Searching for compatible package version
  > Found compatible package version: @angular-eslint/schematics@19.4.0.
✓ Loading package information from registry
✓ Confirming installation
✓ Installing package

All angular-eslint dependencies have been successfully installed 🎉

Please see https://github.com/angular-eslint/angular-eslint for how to add ESLint configuration to your project.

We detected that you have a single project in your workspace and no existing linter wired up, so we are configuring ESLint for you automatically.

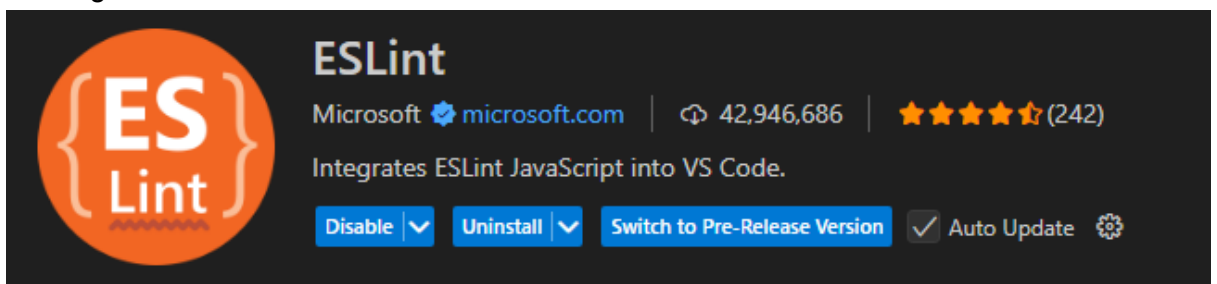
Please see https://github.com/angular-eslint/angular-eslint for more information.

CREATE eslint.config.js (969 bytes)
UPDATE package.json (1522 bytes)
UPDATE angular.json (3415 bytes)
✓ Packages installed successfully.

```

Esto nos indica que todo se ha ejecutado correctamente, también nos informa que ha creado un fichero llamado `eslint.config.js`. Este fichero será el cerebro de **ESLint**, permitiéndonos añadir todas las normas que deseemos.

- 2. Probar que todo ha funcionado correctamente con el comando: `ng lint`. Este comando nos permite ejecutar una batida por todo nuestro código en busca de errores de lindeo (código que no cumple nuestras normas). Es muy probable que te aparezcan varios errores (que debemos solucionar cuanto antes para evitar que al añadir nuestras propias normas acaben apilandose).
- 3. Facilitarnos la vida antes de empezar: en este paso lo que haremos es instalar la extensión de vsCode que nos ayudará a ver de manera más clara los errores. Para esto abrimos la pestaña de descarga de extensiones y buscamos por eslint y descargamos la extensión oficial de Microsoft. A continuación reiniciamos el vsCode.



Gracias a esto, nuestros errores ya se verán claramente en nuestra interfaz. Además, cuando tengamos un error, nos permite buscar la referencia de la norma para saber qué está pasando exactamente e incluso una solución automática.

```

public editClientData(clientData: ClientModel): Promise<number> {
  return new Promise((resolve, reject) => {
    this.clientService
      .putClient(environment.testAPI, Number(clientData.id), clientData)
      .then((saveClient) => {
        resolve(saveClient);
      })
      .catch((message: any) => {
        reject(message instanceof Error ? message : new Error(message));
      });
  });
}

```

Unexpected any. Specify a different type. eslint(@typescript-eslint/no-explicit-any)

View Problem (Alt+F8) Quick Fix... (Ctrl+.) Fix using Copilot (Ctrl+I)

- 4. Arreglar todos los errores que tengamos en nuestro proyecto (para empezar con buen pie). Nuestro objetivo será conseguir este mensaje, el cual indica que nuestro código pasa todas las normas.

```
Linting "arquitectura-front"...  
All files pass linting.
```

Ahora mismo tenemos instalado ESLint, con avisos en tiempo real en caso de que alguna norma no se cumpla, sin embargo... solo tenemos las normas “convencionales” o dicho de otra manera, no son personalizadas y adaptadas a nuestras necesidades.

ESLint.config.ts: personalizando Eslint.

Una vez tenemos todo configurado, podemos empezar a definir nuestras propias normas, y para ello es necesario entender cómo funciona este fichero. La primera vez que entremos, veremos algo similar a esto(puede tener ligeras variaciones en función de la versión instalada):

```

const eslint = require("@eslint/js");
const tseslint = require("typescript-eslint");
const angular = require("angular-eslint");

module.exports = tseslint.config(
  {
    files: ["**/*.ts"],
    extends: [
      eslint.configs.recommended,
      ...tseslint.configs.recommended,
      ...tseslint.configs stylistic,
      ...angular.configs.tsRecommended,
    ],
    processor: angular.processInlineTemplates,
    rules: {
      "@angular-eslint/directive-selector": [
        "error",
        {
          type: "attribute",
          prefix: "app",
          style: "camelCase",
        },
      ],
      "@angular-eslint/component-selector": [
        "error",
        {
          type: "element",
          prefix: "app",
          style: "kebab-case",
        },
      ],
    },
  },
  {
    files: ["**/*.html"],
    extends: [
      ...angular.configs.templateRecommended,
      ...angular.configs.templateAccessibility,
    ],
    rules: {},
  }
);

```

Con un primer vistazo podemos ver que es realmente sencillo. Tendremos diferentes zonas, una para cada tipo de fichero o agrupación de los mismos, como .ts, .html, .scss... cada una de ellas con sus propias normas.

- **Files:** que ficheros se van a analizar, esto puede ser tanto para tipos como por zonas (componentes). Esta última opción no es recomendable, sin embargo, si el

proyecto en el que añades eso ya se comenzó sin normas, puede ser una buena idea añadir los ficheros antiguos a una zona con menos normas mientras se van refactorizando poco a poco.

- **Extends:** nos permite añadir normas ya creadas estándar. Esto no solo permite comenzar a usar ESLint muy rápido, si no que además nos permite tener una base muy sólida sin necesidad de investigar qué normas son más importantes o añadirlas manualmente.
- **Rules:** serán todas las normas personalizadas que nosotros queramos añadirle. Para la arquitectura que estamos utilizando, podéis obtener estas normas del fichero de configuración del proyecto, sin embargo, si consideráis necesario añadir o eliminar algunas, podéis usar como referencia la documentación oficial de [ESLint](#), pudiendo elegir la versión de ESLint que se usa para evitar problemas de versiones.

Si nos metemos un poco más profundo en las reglas, vamos a ver cómo está estructurada una norma:

- Nombre: el primer campo es el nombre de la regla, es la que identifica las condiciones que vamos a poner. Estas se pueden encontrar en la documentación oficial. Ej: `@angular-eslint/template/label-has-associated-control`
- Tipo de "respuesta": hace referencia a cómo quieres gestionar el error de esta norma, es decir, el nivel de severidad de la misma. Off -> desactiva la norma, Warn -> muestra una alerta no bloqueante, es decir, el comando terminará su ejecución sin errores pero mostrará la alerta. Error -> mostrará el error como bloqueante.
- Condiciones: en la última parte podemos especificar como queremos que se aplique la norma. No todas tienen esta funcionalidad, y las que la tienen, cada una es diferente. Para poder trabajar con esta sección, es necesario revisar la documentación oficial y seleccionar los parámetros deseados. Ej:

```
{
  "default": ["field", "constructor", "method", "signature"]
}
```

### Análisis de nuestras normas.

En el proyecto de ejemplo tenemos una batería de normativas diseñadas para garantizar que el código se hace de manera correcta, ordenada pero sin ser muy estrictos, ya que hay algunas normas que, si bien son muy buenas prácticas, si el grueso de programadores no tiene mucha experiencia, pueden llegar a ralentizar enormemente el desarrollo. Para no poner en este documento una por una todas las normas con su respectiva explicación, todo lo necesario para entender cómo funcionan y qué hace cada uno, se encontrará como comentario en el propio fichero [eslint.config.js](#) en la raíz del proyecto de ejemplo.

### Lint-staged, no dejes que el tiempo vuele.

ESLint es muy potente y nos permite analizar prácticamente cualquier aspecto de nuestro código, con cientos de normas, las cuales, a su vez pueden tener pequeñas condiciones adaptadas a nosotros. No solo eso, podemos revisar Html y css, dando a nuestros equipos de maquetación el orden que necesitan, sin embargo, en un proyecto grande, ejecutar el comando "ng lint" puede convertirse en un dolor debido al tiempo que lleva ejecutarlo

(llegando a un par de minutos). Para esto vale lint-staged, que nos permitirá ejecutar el comando de linteo solo sobre los ficheros que se han modificado en vez de en todos.

### Instalando lint-staged.

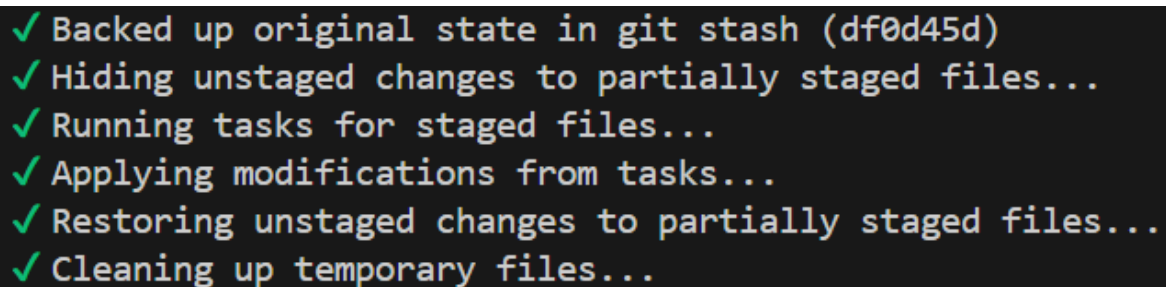
Para comenzar con la instalación solo tenemos que utilizar el comando “npm install --save-dev lint-staged” para instalar la librería como librería de desarrollo. Si quieres ampliar información, puedes acceder a su página de npm [aquí](#).

Una vez tenemos instalada la librería, tenemos que configurarla en el package.json. De momento solo vamos a comenzar por los ficheros .ts para facilitar la instalación.

Añadimos en la base del json la referencia al comando deseado, siendo nuestro caso

```
"lint-staged": {  
  "*.ts": ["eslint" ]  
}
```

Para comprobar que todo ha quedado correctamente instalado , solo tenemos que ejecutar el comando “npx lint-staged”. Si todo ha ido bien, nos devolverá un comando similar a este, dejando claro que todos nuestros ficheros están correctos.



```
✓ Backed up original state in git stash (df0d45d)  
✓ Hiding unstaged changes to partially staged files...  
✓ Running tasks for staged files...  
✓ Applying modifications from tasks...  
✓ Restoring unstaged changes to partially staged files...  
✓ Cleaning up temporary files...
```

### Husky, el ojo de Sauron.

Por un lado, ESLint nos permite poner unas normas para unificar el código, y lint-staged nos otorga la eficiencia de ejecutar dicho control sólo sobre lo que hemos tocado. Ahora ya solo nos queda asegurarnos que esto no es ejecutado por programadores que quieran hacerlo, si no antes de subir a cualquier rama, garantizando así que todas nuestras ramas tienen un código validado.

### Instalando Husky.

Para instalar Husky, basta con ejecutar el comando “npm install --save-dev husky”, el cual lo instalará como una librería de desarrollo. Si quieres más información sobre la librería, puedes verla [aquí](#).

Una vez instalado, tenemos que inicializarlo. husky cuenta con un comando que genera todo lo que necesitamos para dejarlo preparado (o casi todo): “npx husky init”.

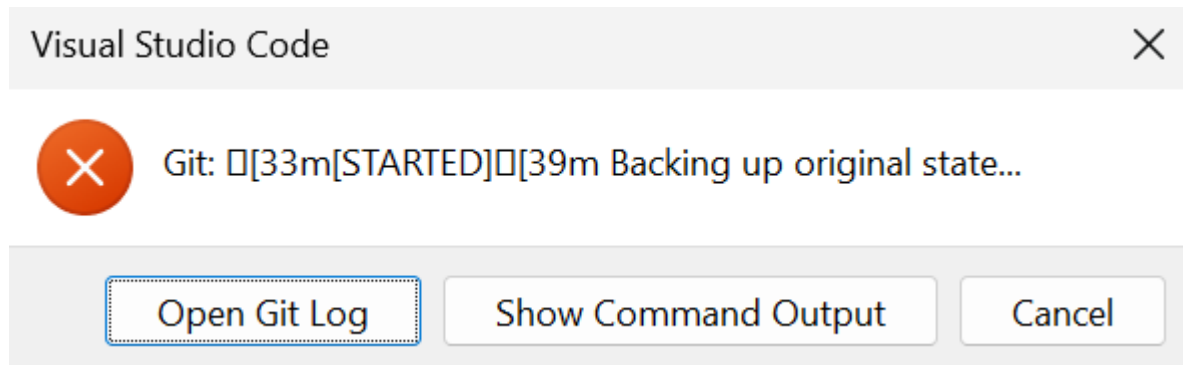
Esto genera una carpeta en la raíz del proyecto la cual contiene el fichero de pre-commit.

Aquí es donde indicaremos a husky que es lo que queremos ejecutar antes de hacer el



commit (y condicionarlo). Para este caso, vamos a comenzar ejecutando el comando de lint que hemos visto anteriormente “npx lint-staged”.

Ahora solo debemos probar que todo funciona, para ello, basta con ir a cualquier fichero .ts y hacer algo que no debamos (aprovecha esta oportunidad para declarar tu última variable sin especificar ámbito) e intenta hacer commit. Si todo está bien configurado, recibirás un error similar a este.



Para obtener más información, podemos pulsar sobre el botón Show Command Output o bien ejecutar en nuestra terminal el comando de ESLint para ver exactamente que nos ha fallado.

Gracias a estos pasos, tu proyecto ya tiene instalado y configurado un entorno de control de código básico. Esta herramienta no es la más completa y bajo ningún concepto debemos aceptar que solo con esto ya hemos terminado. ESLint es una gran herramienta de detección y control de código de primer nivel. Es rápida, continua y muy sencilla de manejar, al no requerir una supervisión (exceptuando cuando se quieren cambiar normas).

Si queremos tener un control completo de nuestro código no nos queda otra alternativa que utilizar herramientas más potentes. para esto tenemos SonarQube, la cual nos permite no solo comprobar que nuestro código es correcto, si no hacer auditorias de calidad, de seguridad, ver cobertura de pruebas y establecer lo que se conoce como “Quality gates” o dicho de otra manera, si no pasas por los mínimos establecidos, no subes a main.

## SonarQube. Tu entrada a la excelencia.

**ESLint** nos permite controlar de forma rápida y directa nuestro código. Sin embargo, esto no es suficiente cuando hablamos de grandes proyectos en los que están implicados múltiples equipos. Tener ciertas normas de nomenclatura, tamaño, etc., ayuda, pero aún así pueden cometerse errores que ESLint no tiene la capacidad de detectar.

Además, es importante destacar que herramientas como **SonarQube** pueden ser necesarias para cumplir con algunos estándares o certificaciones ISO, lo cual puede hacer que, a futuro, te merezca la pena comenzar a utilizarla cuanto antes.

Pero entonces, ¿qué es SonarQube?. SonarQube es una herramienta de análisis estático de código fuente que permite evaluar automáticamente la calidad, seguridad y mantenibilidad del software durante el desarrollo.

## Primeros pasos. Instalación.

Será necesario tener instalado Java JDK 17 o superior (SonarQube no funciona con versiones anteriores o sólo JRE).

La instalación de SonarQube varía en función de cómo queramos utilizarlo. Podemos optar por una instalación local, ideal para comenzar a probarlo. Esta opción monta SonarQube dentro de un contenedor Docker. Los datos pueden guardarse en volúmenes para conservarse entre reinicios. Con esta versión, puedes desplegar tu propia instancia en un contenedor y hacer que todos los proyectos se conecten a ella para registrar los análisis. Ojo: si eres una empresa, deberás adquirir una licencia para ciertas funcionalidades, pero para desarrolladores individuales o pruebas, la edición Community es gratuita. Esta opción permite un control total sobre la instancia, incluyendo la asignación de recursos según tus necesidades.

También existe una opción en la nube (**SonarCloud**), con la que te desentiendes de toda la infraestructura: SonarQube te proporciona la licencia, la máquina, la memoria... tú sólo debes preocuparte de conectar tus proyectos al servicio (bueno, y de arreglar tu código después).

Aquí vamos a seguir el ciclo completo, comenzando con la descarga de la imagen de SonarQube Community (la edición gratuita, y también la más “compleja” de montar localmente).

Lo primero que debemos hacer es descargar la imagen oficial desde Docker Hub (asegúrate de tener Docker instalado previamente).

Usaremos el siguiente comando: `docker pull sonarqube:community`.  
(debemos haber instalado docker antes).

Es una descarga un poco pesada, así que paciencia.

Una vez completada la descarga, levantamos el contenedor y accedemos al puerto correspondiente. Al iniciar, veremos los logs en consola. Para acceder al panel, usamos como credenciales por defecto:

Usuario: admin

Contraseña: admin



## Log in to SonarQube

Login \*

Password \*

[Go back](#)

[Log in](#)

Nada más entrar, se nos pedirá cambiar la contraseña por una nueva. Tras eso, ya estaremos dentro de nuestro panel de control.

Ahora nos queda ir a nuestro proyecto e instalar el paquete npm para poder enviar nuestras métricas. Para ello ejecutamos el comando: `npm i sonar-scanner`.

Ahora mismo ya tenemos todo lo necesario para comenzar a utilizar SonarQube. En el próximo apartado se verá cómo se sincroniza el proyecto con el repositorio, como se configura...

### Generando un entorno de trabajo.

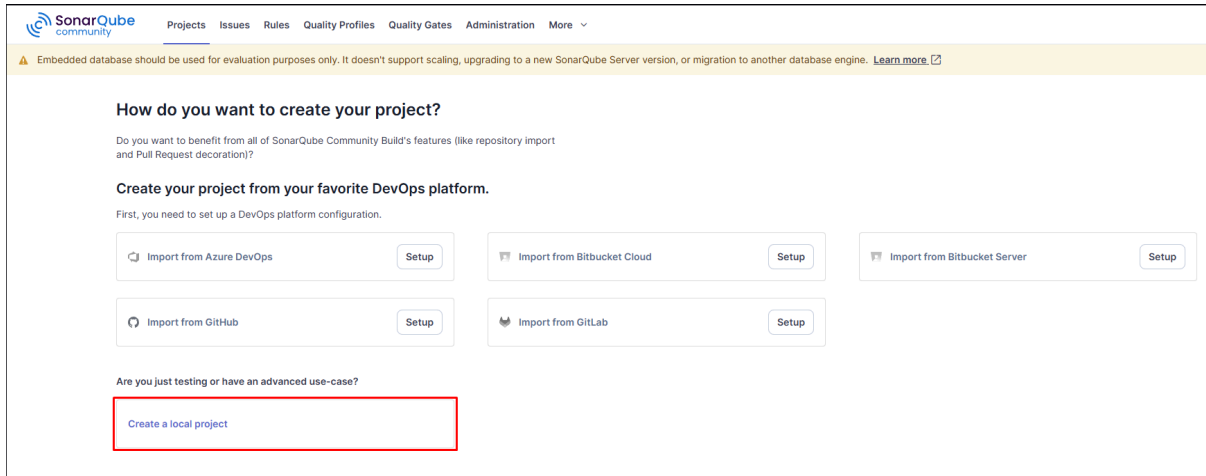
Una vez está todo instalado, el siguiente paso es configurar nuestro proyecto para poder enviar las métricas del mismo a nuestro servidor (en este caso es una imagen de docker local).

Para configurar SonarQube en nuestro proyecto debemos crear en la raíz un fichero con nombre sonar-project.properties. A continuación pongo un ejemplo muy básico con una configuración mínima para comenzar, aunque luego se irá ampliando:

```
sonar.projectKey=Arquitectura-Front
sonar.projectName=Arquitectura-Front
sonar.projectVersion=1.0
sonar.sourceEncoding=UTF-8
sonar.sources=src
sonar.host.url=http://localhost:9000
sonar.token=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
sonar.javascript.lcov.reportPaths=coverage/lcov.info
sonar.exclusions=**/*.spec.ts, **/*.module.ts, **/*.config.ts, **/environments/*.ts, **/*.mock.ts
```

Aquí, los campos más importantes son **sonar.host.url** y **sonar.token**, los cuales se encargan de indicar a sonar en donde debe almacenar los datos (url) y un token de autorización individual por proyecto (token). los demás campos permiten añadir configuraciones básicas a sonar. Claramente la url la obtendremos del puerto que hayamos elegido en nuestra imagen de docker (si estamos haciéndolo en local) o bien la url del servidor en el que lo tengamos alojado.

Para obtener el token debemos entrar a nuestro panel de control y acceder a la sección de proyectos. Ahi encontraremos todos los repositorios de los que podemos obtener el proyecto, pero en nuestro caso, seleccionamos la opción de local.



The screenshot shows the SonarQube Community web interface. At the top, there is a navigation bar with links: Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. Below the navigation bar, there is a warning message: "Embedded database should be used for evaluation purposes only. It doesn't support scaling, upgrading to a new SonarQube Server version, or migration to another database engine. [Learn more](#)".

The main section is titled "How do you want to create your project?". It asks: "Do you want to benefit from all of SonarQube Community Build's features (like repository import and Pull Request decoration)?"

Below this, there is a section titled "Create your project from your favorite DevOps platform." with the instruction: "First, you need to set up a DevOps platform configuration." This section contains five buttons with "Setup" links: "Import from Azure DevOps", "Import from Bitbucket Cloud", "Import from Bitbucket Server", "Import from GitHub", and "Import from GitLab".

At the bottom, there is a question: "Are you just testing or have an advanced use-case?". Below this question, the "Create a local project" button is highlighted with a red rectangular box.

A continuación rellenamos los datos de nuestro proyecto en el formulario y pinchamos en **next**.

1 of 2

## Create a local project

Project display name \* ⓘ

Arquitectura-Front

Project key \* ⓘ

Arquitectura-Front

Main branch name \*

main

The name of your project's default branch [Learn More](#) ⓘ

Cancel

Next

En la siguiente ventana nos pide que le digamos cómo queremos configurar las “versiones de nuestro proyecto”, es decir, que consideramos una versión nueva relevante y cual simplemente es un pequeño cambio que no merece la pena almacenar.

Choose the baseline for new code for this project

☒ Use the global setting

Previous version

Any code that has changed since the previous version is considered new code.

Recommended for projects following regular versions or releases.

☐ Define a specific setting for this project

☐ Previous version

Any code that has changed since the previous version is considered new code.

Recommended for projects following regular versions or releases.

☐ Number of days

Any code that has changed in the last x days is considered new code. If no action is taken on a new issue after x days, this issue will become part of the overall code.

Recommended for projects following continuous delivery.

☐ Reference branch

Choose a branch as the baseline for the new code.

Recommended for projects using feature branches.

Back

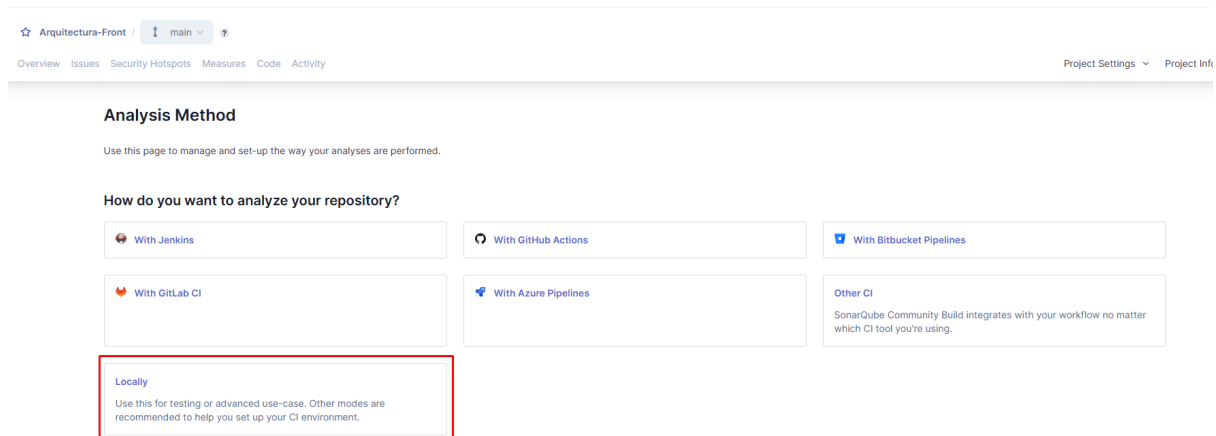
Create project

Vamos a hacer un repaso muy breve por cada una de estas opciones para ver cual deberías utilizar en tu proyecto:

- **Use the global setting:** Usa la configuración general del sistema (normalmente la opción “Previous version”). Útil si no quieres configurar cada proyecto manualmente.
- **Previous version:** Usa la última versión analizada como referencia. Ideal para proyectos con versiones o releases regulares.
- **Number of days:** Considera nuevo todo el código modificado en los últimos X días. Útil para entregas continuas o CI/CD frecuente.
- **Reference branch:** Compara con una rama base (por ejemplo main). Recomendado para proyectos con muchas feature branches.

En función a esto, cada uno elegirá cual es la más apropiada. En mi opinión, si se controla el número de merges contra main y están correctamente agrupados, **Previous version** es la que más me gusta, ya que tienes un histórico general de los cambios y evitas que una persona o equipo añada mucho código de golpe, dificultando así su arreglo.

Una vez tenemos nuestro proyecto creado, volvemos a la sección de proyectos y lo seleccionamos. En este caso, debemos especificar de donde queremos sacar los registros. En este ejemplo, seleccionamos la opción **locally**.



Ahora pulsamos sobre la opción de generar un token, el cual deberá ser actualizado con el tiempo por seguridad.

## 1 Provide a token

Generate a project token

Use existing token

Token name \* ⓘ  
Analyze "Arquitectura-Front"

Expires in  
30 days ▼

Generate

ⓘ Please note that this token will only allow you to analyze the current project. If you want to use the same token to analyze multiple projects, you need to generate a global token in your [user account](#) ⓘ. See the [documentation](#) for more information.

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#) ⓘ.

Una vez tengas el token generado, dirígete al fichero sonar-project.properties que hemos creado anteriormente y pégalo en el atributo correspondiente.

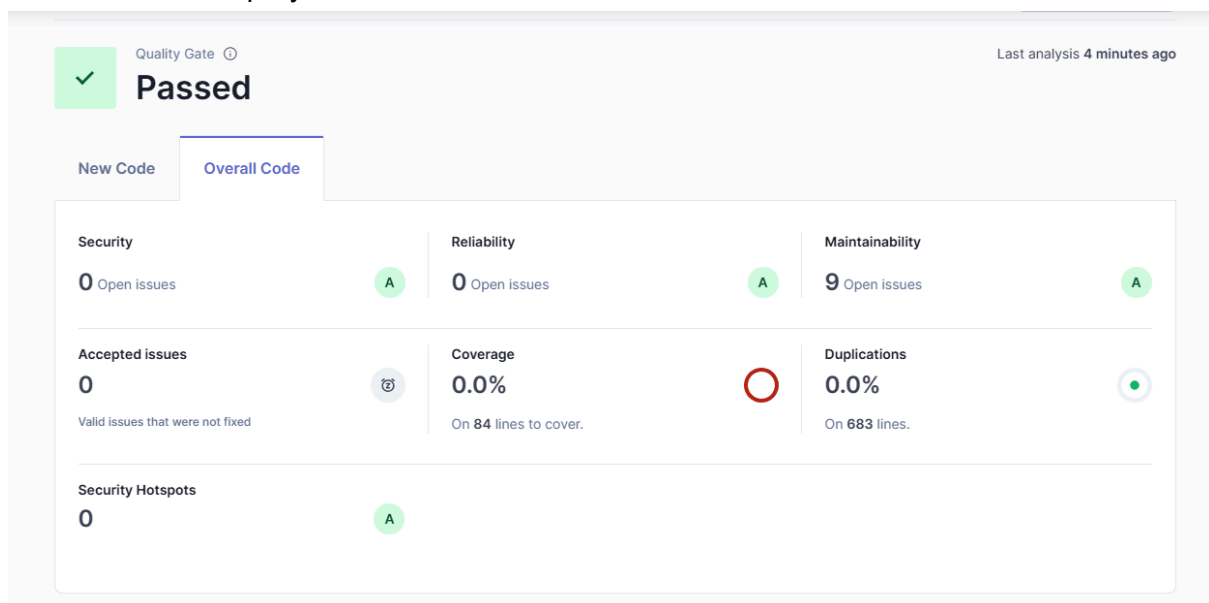
*OJO:* En este ejemplo, y con el fin de hacerlo lo más visual posible (y dado que todo estará en local), voy a incluir el token directamente en el fichero.

Sin embargo, **en proyectos públicos o compartidos**, este token **debe almacenarse como un secreto** en tu proveedor cloud de confianza.

Nunca expongas este token en internet si quieres evitar problemas de seguridad como accesos indebidos o ataques.

Por último, ya solo queda ejecutar por primera vez el escáner de sonar. Por comodidad, podemos añadir este script al package.json: "sonar": "sonar-scanner".

Tras esto, una vez termine el proceso de análisis, podremos ver en nuestro dashboard el estado de nuestro proyecto.



Más adelante entraremos en cada una de estas opciones y veremos cómo podemos configurarlas, adaptarlas y controlarlas para exprimir al máximo el potencial de SonarQube.

## Haciendo nuestro SonarQube.

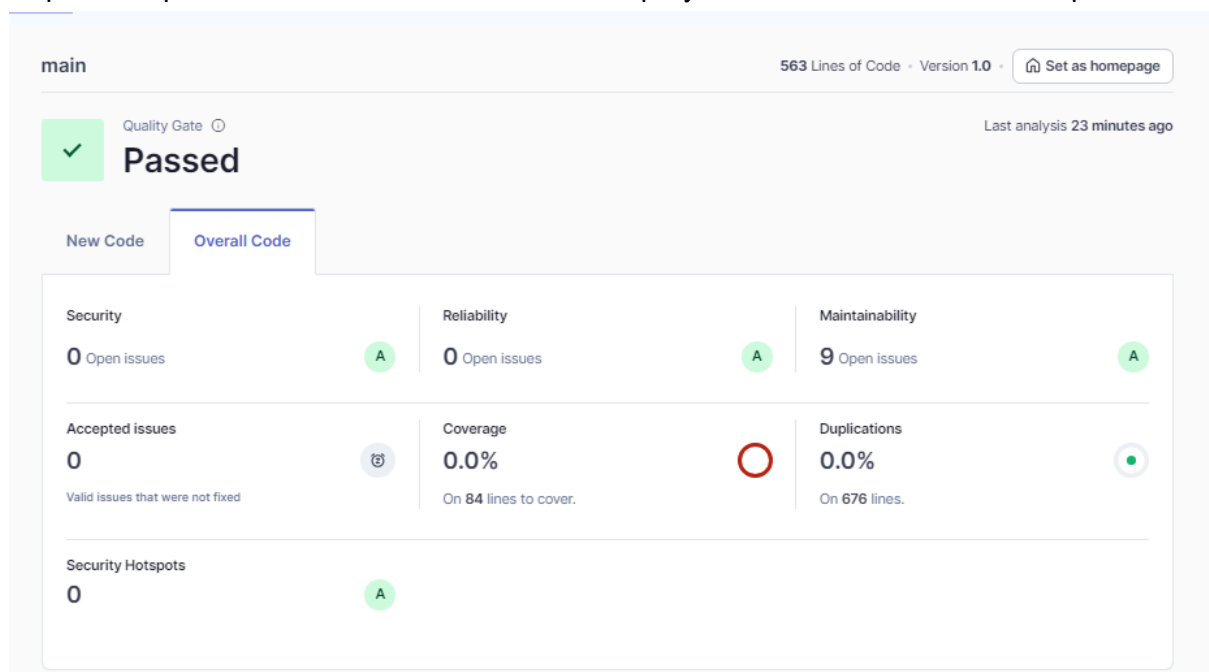
Después de 5 páginas trabajando con **SonarQube** no hemos hecho nada más que instalarlo, y aunque esto ya nos permite comenzar a mejorar la calidad de nuestro código, todo esto es completamente genérico, no está adaptado a nuestras necesidades. Tampoco aporta nada más lejos de la satisfacción de ver si nuestro código es bueno o no.

Ahora vamos a comenzar a dar forma a nuestra instancia de **SonarQube**, creando nuestras propias reglas, generando la documentación correspondiente, pasándole la cobertura de pruebas y lo más importante, configurando las **Quality Gates** (condiciones para determinar si el proyecto cumple con los mínimos para ser considerado válido o si por el contrario, debemos refactorizar).

Pero lo primero es lo primero. De nada nos vale comenzar a crear normas, condiciones y otros elementos si no sabemos para qué vale cada opción del panel o no entendemos lo que estamos leyendo.

## SonarQube desde arriba, ¿para qué vale cada cosa?.

Lo primero que vemos cuando accedemos a un proyecto en **SonarQube** es esta pantalla



Esta pantalla será nuestro centro de operaciones. El que se va a encargar de informarnos de cualquier anomalía, problema o incidente a resolver. Podemos tener tanto una visión general como solo la de la última auditoría. A continuación se describe brevemente qué indica cada una de ellas.



- **Security:** Evalúa la presencia de vulnerabilidades de seguridad en el código. Incluye problemas que podrían ser explotados por atacantes, como inyecciones, uso inseguro de APIs, etc.
- **Reliability:** Mide la robustez del código detectando errores que pueden causar fallos en tiempo de ejecución (bugs). Se centra en garantizar que la aplicación funcione correctamente.
- **Maintainability:** Indica la facilidad con la que el código puede ser modificado. Se basa en la detección de code smells (malas prácticas) que no afectan el funcionamiento, pero sí la legibilidad o escalabilidad.
- **Accepted Issues:** Son incidencias que han sido marcados como "aceptados" por el equipo, es decir, problemas conocidos que se ha decidido no corregir por ahora. Se excluyen del análisis activo pero se mantienen visibles.
- **Coverage:** Representa el porcentaje de código cubierto por pruebas unitarias. Evalúa tanto líneas ejecutadas como condiciones lógicas (ramas). Cuanto más alto, más confianza hay en la estabilidad del código ante cambios.
- **Duplications:** Detecta bloques de código duplicado. Un alto porcentaje indica riesgo de errores al modificar código redundante y aumenta el esfuerzo de mantenimiento.
- **Security Hotspots:** Son fragmentos de código que podrían implicar riesgos de seguridad dependiendo de su contexto. No son vulnerabilidades confirmadas, pero requieren revisión manual.

Todos y cada uno de estos puntos son importantes y requieren una supervisión constante para evitar que se acumulen los problemas. Sin embargo, está claro que los problemas de seguridad, son mucho más críticos que los problemas de mantenimiento.

Llegados este punto y antes de empezar a meternos en profundidad en cada uno de ellos, quiero hacer un llamamiento a la calma, ya que al haber instalado y ejecutado **SonarQube** en un proyecto (y más si no es nuevo)... verás que hay muchos (o muchísimos) fallos, errores, vulnerabilidades... Claramente deben ser subsanados con el paso del tiempo, pero esto debe hacerse de manera escalonada, sin dedicar todos los recursos de un equipo y sobre todo, haciendo primero los más graves e ir poco a poco ampliando las zonas con código de calidad del proyecto.

## Security. La prioridad absoluta.

Cuando hablamos del apartado **security** la norma es clara, solucionar las incidencias cuanto antes. Todo lo que aparece aquí son problemas que podrían dejar expuesto nuestro proyecto, poniendo en peligro nuestra infraestructura, datos de clientes...

En esta ventana veremos todas nuestras incidencias de seguridad

0.0% Security Hotspots Reviewed

To review Acknowledged Fixed Safe

2 Security Hotspots to review

Review priority: Medium

Code Injection (RCE)

Make sure that this dynamic injection or execution of code is safe.

Weak Cryptography

Make sure that using this pseudorandom number generator is safe here.

2 of 2 shown

Make sure that this dynamic injection or execution of code is safe.

Dynamically executing code is security-sensitive typescript:S1523

Status: To review

This security hotspot needs to be reviewed to assess whether the code poses a risk.

Review

Where is the risk? What's the risk? Assess the risk How can I fix it? Activity

src/app/examples/sonarqube-vulnerabilities.example.ts

Open in IDE

```
14 // VULNERABLE: Inserta HTML sin sanitizar
15 return `<div>${userInput}</div>`;
16 }
17
18 // Ejemplo 3: Hardcoded Credentials
19 const DB_PASSWORD = "superSecret123!"; // VULNERABLE: Credenciales hardcodeadas
20
21 // Ejemplo 4: Unsafe Deserialization
22 function vulnerableDeserialization(data: string): any {
23   // VULNERABLE: Usa eval para deserializar
24   return eval(data);
25 }
26
27 // Ejemplo 5: Insecure Direct Object Reference
28 function vulnerableIDOR(userId: string): string {
29   // VULNERABLE: No verifica permisos
30   return `api/users/${userId}/data`;
31 }
32
33 // Ejemplo 6: Command Injection
34 function vulnerableCommandExec(command: string): void {
```

Como se puede ver, se han creado unas cuantas vulnerabilidades de seguridad. En este caso, Aunque ambas se relacionan con la seguridad, Security muestra vulnerabilidades confirmadas, mientras que Security Hotspots señala fragmentos que requieren revisión manual para determinar si suponen o no un riesgo real. Como son muy parecidas, para el ejemplo, la usaremos de manera indistinta.

Lo primero que vemos es un listado a la izquierda con todas las vulnerabilidades con su prioridad asociada.

En la parte central de la ventana podemos ver en donde se ha producido dicha incidencia, indicando el documento, línea y columnas afectadas con una breve descripción del problema.

También tenemos diferentes pestañas que nos ayudarán a solucionar este problema. En muchos casos, las vulnerabilidades serán bastante fáciles de entender, pero otras es posible que se compliquen, para eso tenemos la pestaña **What's the risk**. Aquí nos indicará la referencia a nuestra vulnerabilidad y una explicación de por qué es problemático. Junto con esta pestaña encontramos **How can I fix it?**, la cual nos ayudará a arreglarla.

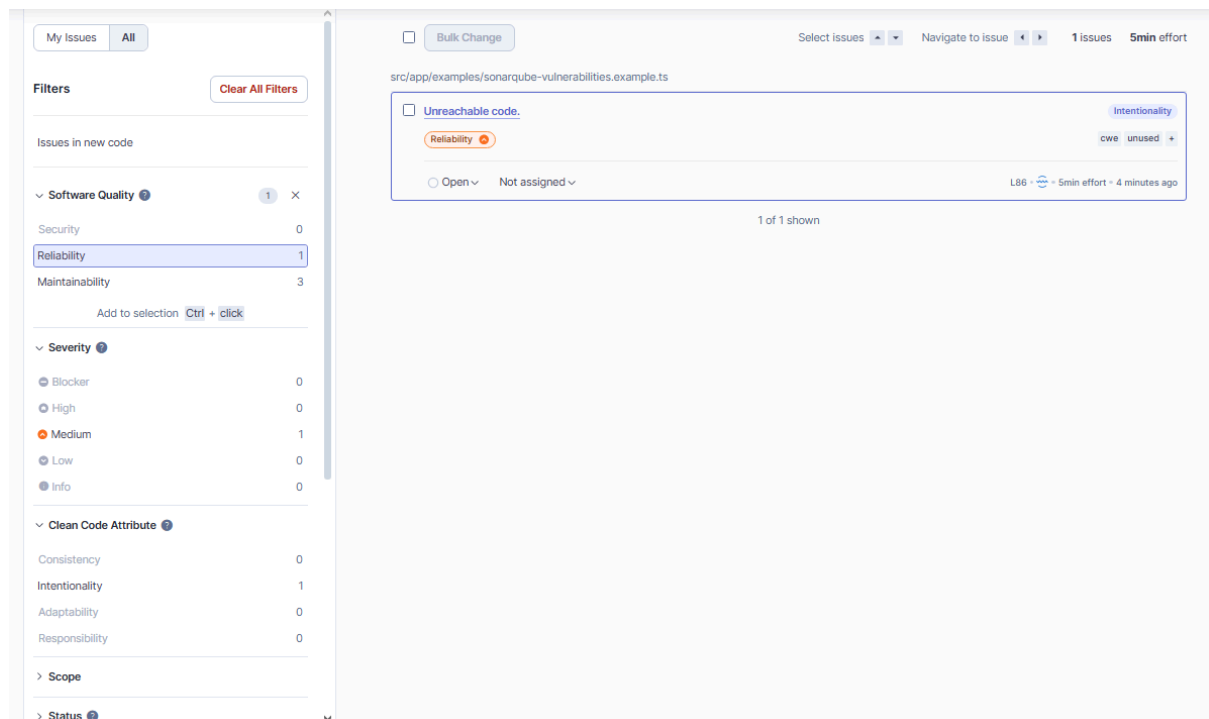
Por último, tenemos la pestaña de **Activity**, en la cual podremos añadir comentarios en los que contemos el estado en el que está o incluso explicaciones de por que se ha hecho de esa manera, en caso de que sea muy complicado solucionarla y se requiera de un cambio más profundo en la infraestructura de la aplicación.

Cuando estamos dentro de la opción **Security Hotspots**, también podremos cambiar el estado en el que se encuentra dicha incidencia para que el supervisor pueda ver claramente la evolución de la misma.

Por último, encontramos la información de a quién está asignado. Esta permite a los supervisores asignar a sus equipos las incidencias para que todo el mundo sepa que debe hacer o cuál es su responsabilidad.

## Reliability. De aquellos barros, estos lodos.

Como hemos visto anteriormente, esta pestaña nos indica fragmentos de código los cuales, si bien no tienen por qué fallar, hay flujos de ejecución que no están bien controlados, por lo que el día de mañana pueden convertirse en errores.



La ventana es muy similar a la vista anteriormente en **Security**(y como todas las de **SonarQube**). En el apartado de la izquierda nos permite filtrar por distintas casuísticas, nivel de gravedad...

Una vez entramos en el detalle de la incidencia, podremos hacer lo mismo que en las de **Security**. Revisar el por qué, obtener ayuda a como resolverla, asignarla a otros programadores o marcarla con diferentes tipologías (falso positivo, resuelta...).

Es necesario destacar que todas las incidencias que aparezcan aquí, deben ser tratadas con urgencia, ya que si bien no siempre van a dar error, el día de mañana podrían convertirse en problemas graves.

## Maintainability. Invertir en nuestro futuro.

Este apartado se centra en detectar código de baja calidad. Este código no va a fallar, no supone un problema en la seguridad ni va a ralentizar el programa, sin embargo, es un código que es difícil de **leer, mantener o escalar**. En cuanto más grande sea nuestro proyecto o más programadores trabajen simultáneamente en él (más aún cuando hay

juniors en los equipos) este apartado irá cobrando importancia. Mantener un nivel bajo de incidencias aquí hará nuestro código más versátil y rápido de entender.

No debemos obsesionarnos con este apartado, ya que es seguramente el menos grave de todos, sin embargo, siempre debemos tener un ojo puesto sobre él para asegurarnos que no crece demasiado. Además, para programadores nuevos que entren a la aplicación, es un gran método de entrenamiento, ya que comenzarán a familiarizarse con el código, con las buenas prácticas y el impacto que pueden tener en la aplicación es bajo (al no ser cambios muy complejos, la posibilidad de romper algo es baja).

## Coverage. Código con sello de calidad.

Este apartado se encarga de validar la cobertura que tiene la aplicación en pruebas unitarias. Este apartado es bastante complejo de tratar, ya que si bien, una cobertura alta es sinónimo de un código a prueba de errores, alcanzar una cobertura alta es un reto para cualquier proyecto grande. Es necesario un equipo formado en pruebas unitarias y con tiempo suficiente para hacerlas y mantenerlas.

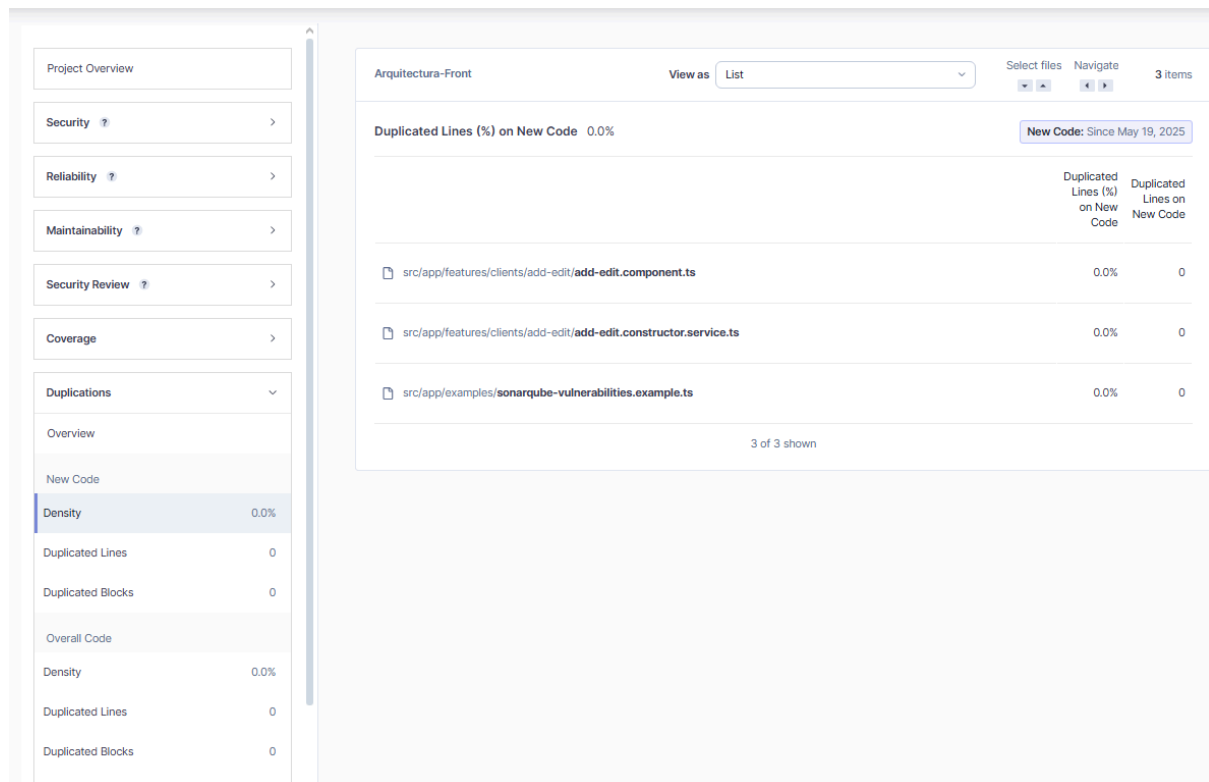
Además hay que destacar que en el caso de frontales, si bien las pruebas unitarias son útiles, en muchos casos puede ser más rentable centrar los esfuerzos en pruebas e2e.

Para estas situaciones, **SonarQube** nos permite establecer un porcentaje deseado de cobertura o mejor aún, indicar que ficheros queremos que entren dentro de la cobertura (por ejemplo, hacer obligatorio incluir en la cobertura los servicios responsables de validaciones o transformación de datos en la cobertura, pero excluir ficheros con métodos como abrir un modal, los cuales, tiene más sentido que sean probados por pruebas e2e al ser mucho más fieles a la realidad).

## Duplications. Evita repetirte como un loro.

En cuanto más grande es el proyecto, más probable es encontrar bloques duplicados, ya sea por copia/pega o falta de refactorización del código. Con el paso del tiempo esto se vuelve un problema, ya que todo código que tengamos duplicado multiplica el trabajo por cada repetición, haciendo que el código sea muy difícil de mantener (y que en muchos casos se nos olvide cambiar algo en algún sitio y falle).

Este apartado analiza todo el código y encuentra estas zonas de código problemático. Este apartado es un poco más complicado de trabajar, ya que generalmente requiere refactorizar el código, crear servicios o funciones comunes... pero ten claro que a la larga merece mucho la pena.



Como se puede ver, la vista es muy parecida a las otras, con su menú lateral para poder filtrar todas las incidencias y el listado central. Si entramos a la vista individual, veremos el componente con las líneas duplicadas que deben ser modificadas.

## Quality Gates. No todo lo que compila merece pasar.

Gracias al bloque anterior, ya tenemos un seguimiento bastante preciso de nuestro código. Sabemos ubicar incidencias, priorizarlas, gestionirlas por usuario y entendemos para qué sirve cada métrica. Sin embargo, todavía nos falta algo clave: **una forma de frenar el despliegue de código de baja calidad.**

Todo el trabajo que hemos hecho hasta ahora puede quedarse en nada si no hay una barrera clara que impida que el código malo llegue a producción.

Las Quality Gates son una serie de condiciones mínimas que podemos configurar en cada proyecto para decidir si el código es válido o no. Estas condiciones se pueden personalizar tanto a nivel de proyecto como por tipo de incidencia.

Por ejemplo, podemos establecer límites sobre:

- La cobertura mínima de tests
- El porcentaje de código duplicado
- La cantidad máxima de incidencias de seguridad
- La gravedad de los errores permitidos

## Creando nuestras propias Quality Gates.

Lo más importante antes de empezar a configurar es **entender la lógica y la estrategia** detrás de estas reglas.

Esto no es simplemente un trabajo de administración. Antes hay que hacer un análisis realista del proyecto y del equipo:

- ¿Es un proyecto recién creado o uno con más de un año de desarrollo?
- ¿Está desarrollado por un equipo de juniors o por analistas senior?

La clave está en que **las Quality Gates son controles estrictos**: están para cumplirse, no para esquivar (Por supuesto, se pueden modificar, pero **solo** con el consenso de arquitectos, tech leads y dirección. Nunca como solución rápida a una urgencia).

Exigir una cobertura del 100% o tolerancia cero a errores de maintainability puede parecer ideal, pero en la práctica puede:

- Ralentizar el desarrollo
- Limitar la flexibilidad para solucionar bugs en producción
- E incluso llevar a abandonar SonarQube por parecer “inviable”

Para crear una nueva, debemos seleccionar la opción de **Quality Gates** en la barra superior de **SonarQube**. Aquí encontraremos todas las que tengamos configuradas y la por defecto.

Quality Gates ⌵ Create

Sonar way ⌵ DEFAULT BUILT-IN ⌵

**Sonar way** DEFAULT BUILT-IN Copy

⌵ The only quality gate you need to practice [Clean as You Code](#)

**Conditions** ⌵

Your new code will be clean if:

- New code has 0 issues
- All new security hotspots are reviewed
- New code has sufficient test coverage Coverage is greater than or equal to 80.0% ⌵
- New code has limited duplications Duplicated Lines (%) is less than or equal to 3.0% ⌵

**Projects** ⌵

Every project not specifically associated to a quality gate will be associated to this one by default.

Ahora pulsamos sobre Create. le damos un nombre y nos aparecerá la configuración de la misma.

## Standar Gate



### ✓ This quality gate complies with Clean as You Code

This quality gate is [configured for Clean as You Code](#). It ensures that:

- ✓ New code has 0 issues
- ✓ All new security hotspots are reviewed
- ✓ New code has sufficient test coverage
- ✓ New code has limited duplications

### Conditions ⓘ

#### Conditions on New Code

Metric	Operator	Value	
Issues	is greater than	0	
Security Hotspots Reviewed	is less than	100%	
Coverage	is less than	80.0%	
Duplicated Lines (%)	is greater than	3.0%	

You may click unlock to edit this quality gate. Adding extra conditions to a compliant quality gate can result in drawbacks. Are you reconsidering [Clean as You Code](#)? We strongly recommend this methodology to achieve a Clean Code status.

Unlock editing

### Projects ⓘ

With

Without

All

Search

Verás que la nueva Quality Gate utiliza las normas “**Clean as You Code**”, un estándar que impone SonarQube.

Estas reglas por defecto solo permiten editar la **cobertura** y la **duplicidad de código**.

Pero como comentábamos antes, esto debe ser un **objetivo a largo plazo**, no algo inmediato. Por eso, lo ideal es **modificar los valores iniciales** para que se ajusten a la realidad del equipo y proyecto. Para ello pulsa **Unlock Editing**.

Como extra, podemos añadir condiciones personalizadas pulsando sobre el botón “Add Condition”, el cual, nos permite elegir entre muchas normas predefinidas por SonarQube y asignarles un valor, o bien numérico absoluto o en porcentaje.

**Add Condition** ×

Where?

☒ On New Code  
☐ On Overall Code

Quality Gate fails when

Duplicated Blocks ▾

Operator

is greater than

Value \*

Add Condition Close

A continuación debemos añadir los proyectos que queramos vincular a esta **Quality Gate** y por último los usuarios o grupos de usuarios que van a tener acceso a la edición de esta.

Aunque lo ideal sería tener una única Quality Gate estándar para toda la empresa, en la práctica esto rara vez funciona. La mejor estrategia es tener varias versiones:

- **Inicial:** para proyectos grandes y antiguos, con muchas incidencias.
- **Intermedia:** para proyectos en transición o con cierto trabajo de limpieza hecho.
- **Final:** la que sigue al 100% las normas de Clean as You Code.

Así conseguimos que todos los equipos se adapten progresivamente al estándar deseado sin generar frustración.

Las *Quality Gates* no son solo filtros, son compromisos. Si se diseñan bien, se convierten en un pilar para garantizar la calidad continua del código. Si se diseñan mal, pueden ser una barrera o incluso una excusa para dejar de usar SonarQube.

Empieza con cabeza, escala con estrategia y nunca las uses como castigo.

## Quality Profiles. Las normas son para todos... pero estas son las más.

Cuando hablamos de normas de codificación en **SonarQube**, esto no es un capricho de los desarrolladores de la plataforma o una conspiración para hacer que tardes más en programar. Son normativas, muchas de ellas diseñadas por los autores del lenguaje en cuestión, y otras aportes de la comunidad con el objetivo de estandarizar la forma en la que se desarrolla. Casi siempre, seguir estas normas al pie de la letra va a facilitar enormemente la vida como desarrollador, pero es posible que la empresa decida añadir



algunas nuevas para tener más control sobre el código o cambiar la importancia de algunas. Para esto tenemos los **Quality Profiles**. nos permiten coger los conjuntos de normas que tiene un lenguaje, por ejemplo Typescript y hacer pequeñas modificaciones que nos permitan trabajar más fácilmente.

## Creando nuestras propias reglas.

Para poder editar las normas predefinidas que tiene SonarQube, debemos hacer click en la opción de Quality Profiles que encontraremos en la barra superior de la ventana.

**Quality Profiles**

Quality profiles are collections of rules to apply during an analysis. For each language there is a default profile. All projects not explicitly assigned to some other profile will be analyzed with the default. Ideally, all projects will use the same profile for a language. Learn more about [Quality Profiles](#).

Filter by: Select language

Language	Projects	Rules	Updated	Used
AzureResourceManager, 1 profile(s)	<a href="#">Projects</a>	<a href="#">Rules</a>	<a href="#">Updated</a>	<a href="#">Used</a>
<a href="#">Sonar way</a> BUILT-IN	DEFAULT	<a href="#">31</a>	21 hours ago	Never
C#, 1 profile(s)	<a href="#">Projects</a>	<a href="#">Rules</a>	<a href="#">Updated</a>	<a href="#">Used</a>
<a href="#">Sonar way</a> BUILT-IN	DEFAULT	<a href="#">321</a>	21 hours ago	Never
CSS, 1 profile(s)	<a href="#">Projects</a>	<a href="#">Rules</a>	<a href="#">Updated</a>	<a href="#">Used</a>
<a href="#">Sonar way</a> BUILT-IN	DEFAULT	<a href="#">24</a>	21 hours ago	29 minutes ago
CloudFormation, 1 profile(s)	<a href="#">Projects</a>	<a href="#">Rules</a>	<a href="#">Updated</a>	<a href="#">Used</a>
<a href="#">Sonar way</a> BUILT-IN	DEFAULT	<a href="#">26</a>	21 hours ago	Never
Docker, 1 profile(s)	<a href="#">Projects</a>	<a href="#">Rules</a>	<a href="#">Updated</a>	<a href="#">Used</a>
<a href="#">Sonar way</a> BUILT-IN	DEFAULT	<a href="#">40</a>	21 hours ago	Never
Flex, 1 profile(s)	<a href="#">Projects</a>	<a href="#">Rules</a>	<a href="#">Updated</a>	<a href="#">Used</a>

**Deprecated Rules**

Deprecated rules are still activated on 2 quality profile(s):

- [Sonar way](#)  
JavaScript, 1 rule(s)
- [Sonar way](#)  
TypeScript, 1 rule(s)

**Recently Added Rules**

- [Unused local variables should be removed](#)  
PHP, activated on 1 profile(s)
- [Return of boolean expressions should not be wrapped into an "if-then-e..."](#)  
PHP, activated on 1 profile(s)
- [Ternary operators should not be nested](#)  
PHP, activated on 1 profile(s)
- [Assignments should not be made from within sub-expressions](#)  
PHP, not yet activated
- [Modifiers should be declared in the correct order](#)  
PHP, activated on 1 profile(s)
- [Boolean literals should not be redundant](#)  
PHP, activated on 1 profile(s)
- [Local variables should not be declared and then immediately returned o...](#)  
PHP, activated on 1 profile(s)
- [Tests should include assertions](#)  
PHP, activated on 1 profile(s)
- [Repeated patterns in regular expressions should not match the empty s...](#)  
PHP, activated on 1 profile(s)
- ["switch" statements should not have too many "case" clauses](#)  
PHP, activated on 1 profile(s)

[See all 3.8k recently added rules](#)

Aquí tenemos varias opciones. Podemos crear nuestro conjunto de reglas propio, sin ninguna base, lo cual salvo que sepas muy bien lo que estás haciendo, no es recomendable, ya que deberás añadir manualmente todas las reglas básicas de buenas prácticas que tiene tu lenguaje. la otra opción es coger el conjunto de normas básicas existentes y extender el tuyo, garantizando así que la base es sólida y tu solo aportas detalles.

Para esto podemos pulsar sobre el botón **Create** y seleccionar la opción de “Extend existing qualityProfile” o bien buscar el lenguaje sobre el que queremos trabajar y hacer click sobre los 3 puntos para abrir el menú desplegable, sobre el cual utilizaremos la opción de “extend”.

Nosotros vamos a extender las normas básicas de **typescript** en una nueva norma llamada Custom typescript.

Quality Profiles / TypeScript / Custom typescript profile

Custom typescript profile

Updated: 1 minute ago

Used: Never

See Changelog

Inheritance

Change Parent

Sonar way BUILT-IN	317 active rules	96 inactive rules	0 overridden rules
Custom typescript profile	317 active rules	96 inactive rules	0 overridden rules

Projects

Change Projects

No projects are explicitly associated to the profile.

Permissions

Users with the global "Administer Quality Profiles" permission and those listed below can manage this quality profile.

Grant permissions to more users

Rule breakdown

Software Qualities	Active	Inactive
Security	13	2
Reliability	110	8
Maintainability	182	75

Clean Code Categories	Active	Inactive
Consistency	67	33
Intentionality	170	39
Adaptability	24	11
Responsibility	6	1

1 deprecated rule ?

Activate More

En la parte de la izquierda podemos ver y modificar los conceptos básicos de las normas, como a qué proyecto están asignadas o quién podrá modificar los permisos. En la parte de la derecha encontramos las normas que tenemos asociadas al mismo.

Para comenzar, vamos a añadir nuevas normas pulsando sobre el botón “Activate more”. Se cambiará la vista a un listado con todas las disponibles, y podremos añadirlas pulsando sobre el botón “Activate”, en donde podremos seleccionar su gravedad en función de la categoría a la que pertenezca.

Si queremos editar alguna de las que ya hay, en la ventana anterior podremos pinchar sobre cualquiera de las categorías que tenemos. Esto nos llevará a la misma pantalla que la anterior pero con la opción de desactivar o modificar.

Esto nos va a permitir customizar **SonarQube** adaptándolo a las necesidades de nuestra empresa.

Como en el punto anterior con las **Quality Gates**, es necesario destacar la importancia de no ser muy laxo (haciendo inutil la auditoría de SonarQube) o demasiado estricto, perdiendo toda flexibilidad y capacidad de adaptación de nuestro código.

## El rincón del friki: afinando hasta lo invisible.

Hasta ahora, todo lo que hemos visto ha estado muy centrado en el código y su integración con SonarQube. Sin embargo, hay una parte un poco menos glamurosa, pero igual de importante: la administración del servidor.

Sí, esa parte del documento que todos evitamos... hasta que tenemos que gestionar 15 proyectos, 6 equipos, múltiples lenguajes, varios Quality Gates y queremos datos veraces en los informes. Entonces nos damos cuenta de que hay que adaptar el funcionamiento de la aplicación a nuestras necesidades reales.

Aquí veremos algunas opciones muy útiles que te permitirán mantener el sistema bajo control sin perder la cabeza.

## Alertas por correo. Que no se te escape nada.

Ponte en la piel de un administrador de SonarQube con una docena de proyectos y 30 personas haciendo **commits**, fixes y pidiendo aprobaciones constantemente. Mantener todo eso bajo control no es tarea sencilla.

Para facilitarte la vida, puedes activar las alertas por correo. Así, en lugar de tener que revisar dashboards constantemente, recibirás notificaciones automáticas sólo cuando haya algo que requiera tu atención.

Para configurarlo, ve a **Administration** en la barra superior y entra en **Email Notifications**. Allí podrás introducir los datos de tu servidor **SMTP** como en cualquier otro sistema de mailing: host, puerto, credenciales, etc.

Aunque la configuración es sencilla, su impacto es enorme: permite que la persona responsable de calidad reaccione más rápido ante incidencias, revisiones o desviaciones que podrían colarse si nadie estaba mirando.

## Analizadores externos. Subcontratando ayudantes.

SonarQube es una herramienta muy potente, con un montón de reglas, métricas y paneles propios. Pero parte de su verdadero poder no está solo en lo que hace, sino en lo que permite que otros hagan por él. Hablamos de los analizadores externos.

Por ejemplo, en nuestra arquitectura, ESLint es una herramienta imprescindible para garantizar la calidad del código JavaScript/TypeScript. Pero sus métricas no siempre coinciden con las de SonarQube. Esto podría obligarnos a elegir entre una u otra... y perder información valiosa sea cual sea la decisión.

Aquí es donde entran los analizadores externos: nos permiten decirle a SonarQube “oye, este informe de calidad generado por otra herramienta también cuenta”. De este modo, SonarQube no solo analiza lo suyo, sino que también integra resultados de terceros, haciendo nuestros dashboards mucho más completos y útiles.

Para configurarlos ve a **Administration**, como ya hemos hecho antes. A continuación busca la opción que permite importar informes externos (puede llamarse External Issues, Generic Issue Import o estar dentro de la configuración del lenguaje). Antes de ejecutar el análisis de SonarQube, genera el informe externo (por ejemplo: `eslint -f json -o eslint-report.json`). Ahora debes indicar en **SonarQube** la url en la que vas a guardar la información generada con el comando anterior.

Con esto, SonarQube tomará ese informe y lo incorporará como parte de la calidad del proyecto, sin necesidad de renunciar a ninguna de las herramientas que ya estás utilizando. Una forma elegante de decirle a SonarQube: “tú analiza, pero escucha a mis otros linters también”.

## Otras funciones.

Desde aquí también es donde se gestionan los usuarios, grupos nuevos proyectos... No vamos a profundizar mucho en estos puntos puesto que son ventanas de gestión básicas que no requieren ningún conocimiento extra.

Con este manual ya tenemos todo lo necesario para integrar sistemas de validación de código tanto en tiempo real (ESlint) como con auditorías(**SonarQube**). Esto supondrá un cambio enorme en la calidad de los proyectos, en su mantenimiento y escalabilidad.

Es cierto que este tipo de herramientas (sobretudo las de auditoría) no son baratas y suponen un desembolso inicial en licencias, máquinas y en tiempo grande, sin embargo, en muy poco tiempo se rentabilizan no solo por la disminución de tiempos de desarrollo, si no también con el aumento de la confianza de los clientes en nuestros productos.

Como también se ha mencionado antes, para implementar estos sistemas es necesario ser paciente y seguir una curva de implementación progresiva para evitar problemas durante la implementación.