

ESLint, Sonar y control de calidad del código.

ESLint, lint-staged y husky siempre al pie del cañón.	2
ESLint, nuestro código como una patena.	2
Manchandonos las manos. Instalación de nuestro entorno.	2
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
SonarQube. Tu entrada a la excelencia.	8

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.

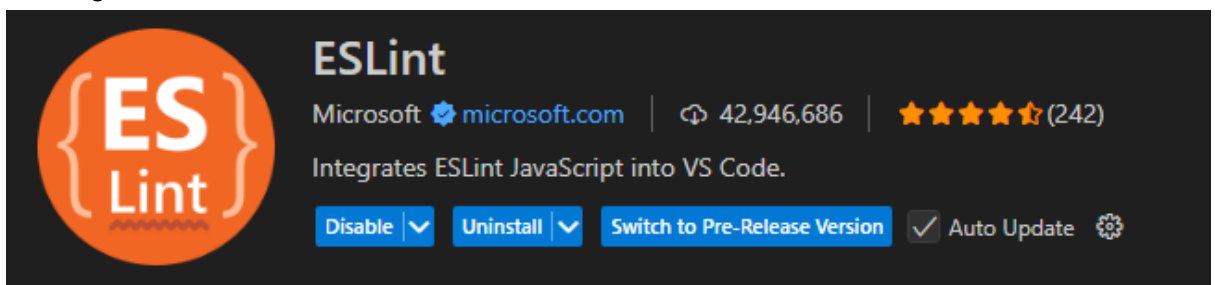
We detected that you have a single project in your workspace and no existing linter wired up, so we are configuring ESLint for you automatically.

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.

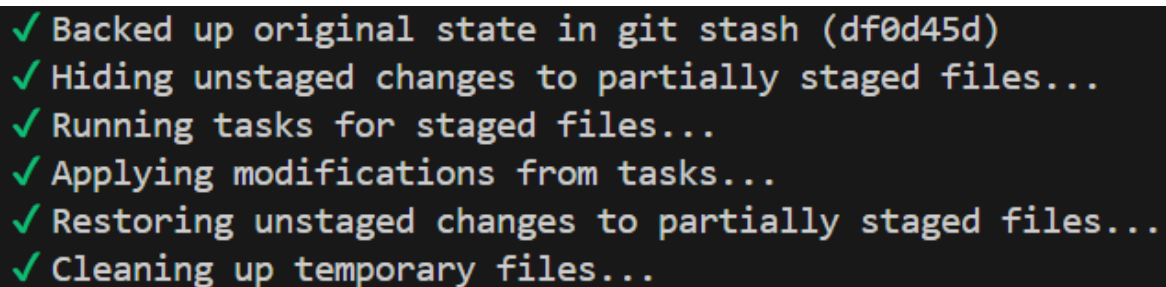
Para comenzar con la instalación solo tenemos que utilizar el comando “npm install --save-dev lint-staged” para instalar la librería como librería de desarrollo. Si quieres ampliar información, puedes acceder a su página de npm [aquí](#).

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](#).

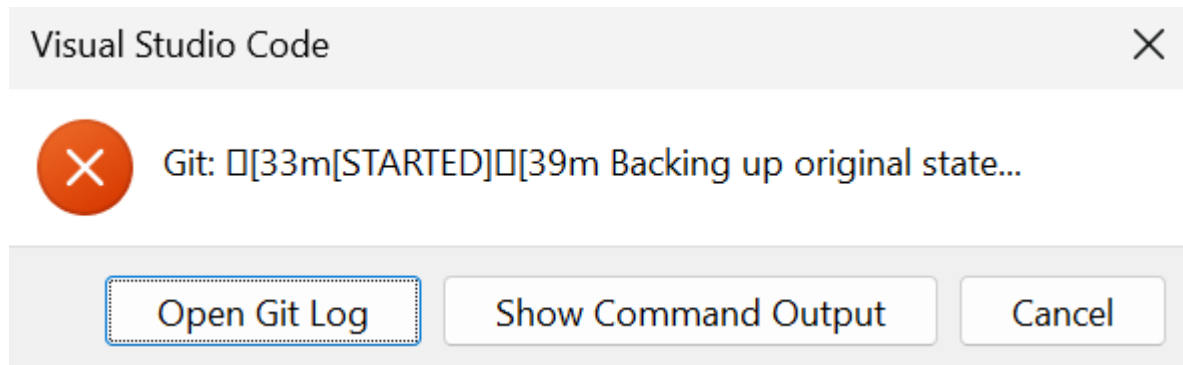
Una vez instalado, tenemos que inicializarlo. husky cuenta con un comando que genera todo lo que necesitamos para dejarlo preparado (o casi todo): “npx husky init”.

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.

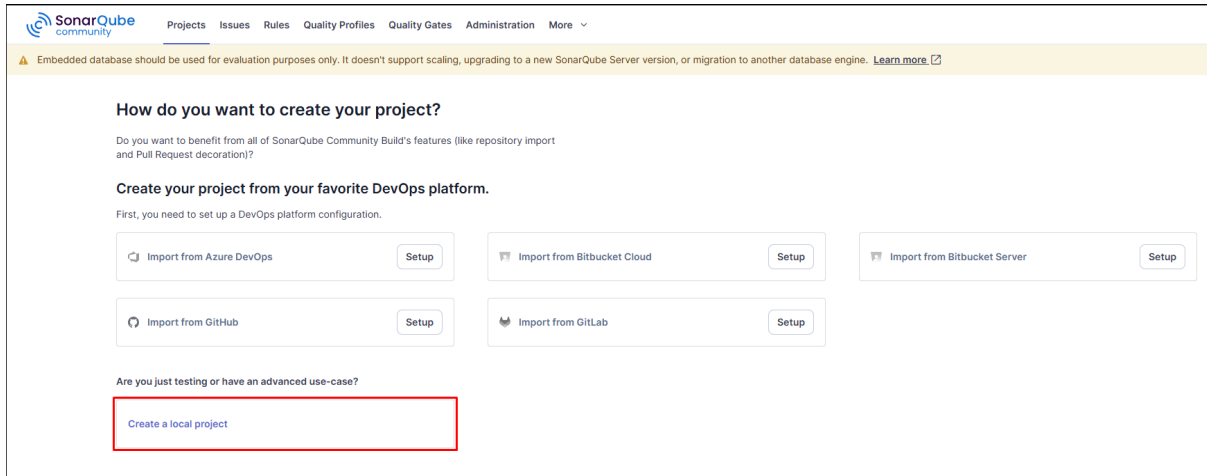
Una vez está todo instalado, el siguiente paso es configurar nuestro proyecto para poder enviar las métricas del mismo a nuestro servidor (en este caso es una imagen de docker local).

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 text: "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
- 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 rectangle.

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

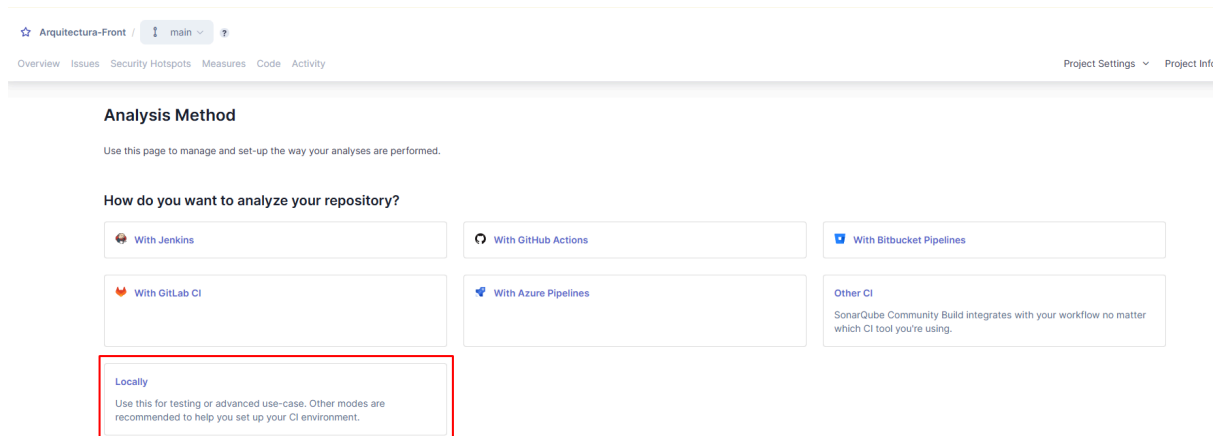
Create project

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 tokenUse existing token

Token name * ⓘ
Analyze "Arquitectura-Front"

Expires in
30 days ▼

Generate

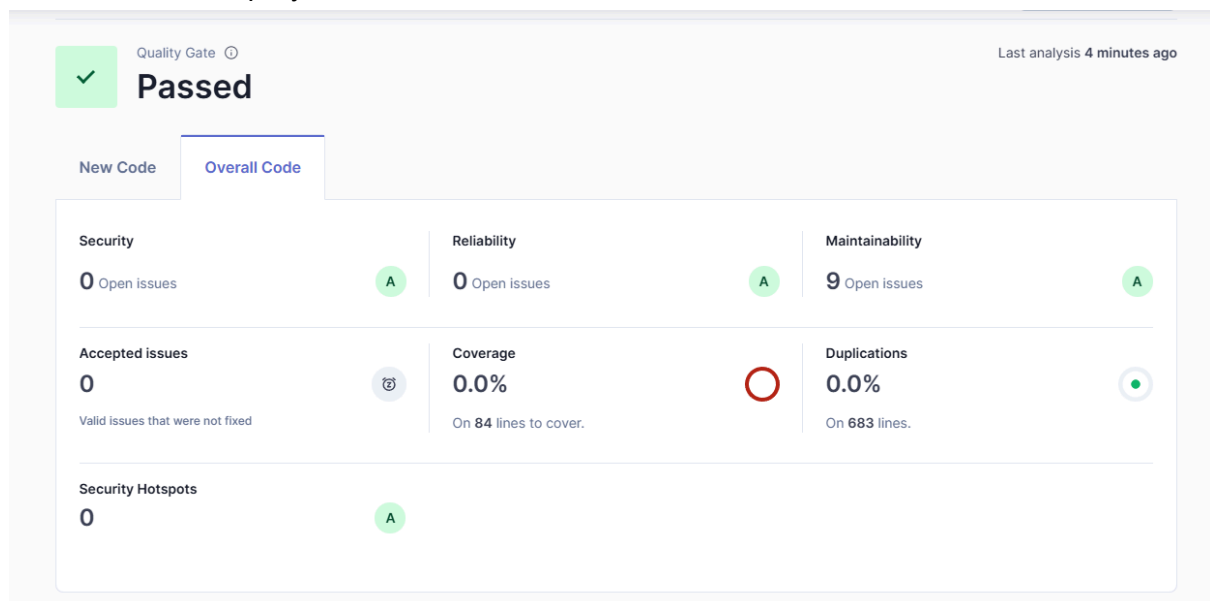
ⓘ Please note that this token will only allow you to analyze the current project. If you want to use the same token to analyze multiple projects, you need to generate a global token in your [user account](#) [\[?\]](#). See the [documentation](#) for more information.

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, vete al fichero **sonar-project.properties** que hemos creado anteriormente y pega el token en su atributo. **OJO:** para este ejemplo y con el fin de hacer todo lo más visual posible (y ya que todo va a estar en local) yo voy a poner el token a pelo en el fichero, sin embargo, si vas a trabajar en proyectos públicos, este token deberá ser almacenado como un secreto en tu distribuidor cloud de confianza, nunca expongas esto a internet si quieres evitar problemas de trolls, hackers y demás basura.

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.