



**Entrega final del proyecto: ECOTRIBUTARIO**

**Fundamentos de ingeniería de software**

**Juan Sebastián Rodríguez Pabon**

**Juan Nicolas Parra Santos**

**Sara Muñoz**

**Entregado a: Luis Moreno Sandoval**

## **Resumen**

ECOTRIBUTARIO es una app que ofrece descuentos fiscales a empresas las cuales contribuyan al cuidado del medio ambiente y se pueda evidenciar mediante archivos los cuales un experto analizará y determinará si se obtendrá o no el descuento. Dicho descuento podrá ser del 100% según sean las evidencias, se otorgará menor descuento. Para usar la app de ECOTRIBUTARIO se pedirá el registro con datos según corresponda (Si es administrador o si es empresa).

Para poder usar la app de ECOTRIBUTARIO se puede acceder mediante dos roles: Administrador y Empresa.

- El Administrador cuenta con funcionalidades como: revisar solicitudes de descuento, visualizar el ranking de empresas, consultar el historial de solicitudes y cerrar sesión.
- La Empresa puede: crear nuevas solicitudes, consultar su historial y volver al inicio. Cada funcionalidad se encuentra debidamente representada en una interfaz adaptada al rol correspondiente.

ECOTRIBUTARIO es una aplicación desarrollada con el objetivo de incentivar a las empresas a contribuir activamente con el cuidado del medio ambiente, ofreciendo descuentos fiscales proporcionales al impacto positivo que estas generen. Ha sido diseñado e implementado utilizando métricas de calidad, patrones de diseño del catálogo GOF y pruebas unitarias que aseguran el correcto funcionamiento de cada módulo del software.

## **Introducción**

### **Objetivo principal del proyecto**

- Diseñar una app que otorgue descuentos fiscales a empresas que demuestren su contribución con el medio ambiente.

### **Objetivos específicos del proyecto**

- Implementar métricas de calidad para evaluar aspectos como: mantenibilidad, complejidad y calidad del código.
- Implementar patrones GOF (Singleton, Strategy y Decorator).
- Implementar pruebas unitarias para evaluar que cada componente funcione correctamente.
- Implementar un servicio de mensajería según la base de datos a Telegram.

## **Contexto y relevancia**

Actualmente el cuidado del medio ambiente es uno de los pilares mas importantes en la sociedad, puesto que se debe de tener un entorno saludable y próspero para tener así una calidad de vida buena. Frente a esta situación se conoce que las empresas en la ciudad de Bogotá son gran fuentes considerables de emisiones de Co2, contaminación, residuos orgánicos, etc...

Por esta razón se implemento una app “ECOTRIBUTARIO” la cual su función será ofrecer descuentos fiscales a empresas que demuestren que contribuyen notoriamente en el cuidado del medio ambiente subiendo evidencias de dichos cambios. En este contexto se desarrolla la aplicación ECOTRIBUTARIO, una app que busca recompensar a aquellas empresas que demuestran un compromiso real con el medio ambiente.

ECOTRIBUTARIO no solo busca ofrecer un beneficio económico a las empresas comprometidas, sino también consolidarse como una herramienta tecnológica innovadora que promueve activamente el desarrollo sostenible y la conciencia ambiental corporativa.

Según el artículo de “Impacto ambiental de las organizaciones” se sabe que en una empresa alrededor del 42% de los trabajadores identifican que si existen medidas de cuidado al medio ambiente. Esta cifra es muy baja puesto que se debería tener más del 80% para que no haya una contaminación en el medio ambiente, también se conoce que en un año el 31% de los trabajadores informan que no tienen charlas informativas ni iniciativas frente al medio ambiente y solo el 24% se conoce que tienen charlas mensualmente. Entorno a esto se realizo la app de ECOTRIBUTARIO, implementando un software que otorgue un descuento fiscal a empresas mediante implementaciones métricas de calidad, patrones GOF y pruebas unitarias.

La importancia del articulo mencionado anteriormente es que solo el 42% de los trabajadores en empresas de Bogotá reconocen la existencia de medidas ambientales dentro de sus organizaciones, y únicamente un 24% reporta la realización de charlas informativas mensuales.

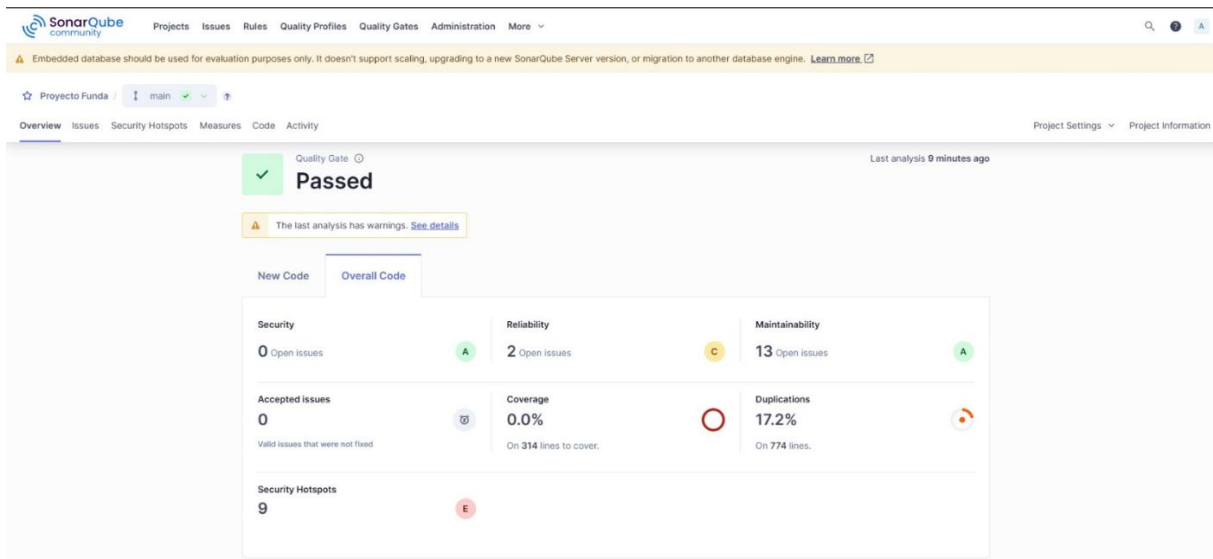
Durante el desarrollo del sistema, se aplicaron métricas de calidad para evaluar aspectos como la eficiencia del código, la cobertura de pruebas y el cumplimiento de requisitos funcionales y no funcionales. Además, se incorporaron patrones de diseño GOF para garantizar una arquitectura robusta y flexible, facilitando la reutilización de componentes y la extensión futura del sistema. Finalmente, se implementaron pruebas unitarias exhaustivas que aseguran la fiabilidad de cada una de las funcionalidades, contribuyendo a la estabilidad general del software.

## **Desarrollo**

### **implementación de métricas de calidad**

En la implementación de métricas de calidad es fundamental entender que se usan para poder entender el funcionamiento del código y si es fácil de mantener y ver si el rendimiento es aceptable. Para ello se aplicaron las siguientes métricas:

A continuación se mostraron los resultados del análisis automático realizado con una herramienta llamada SonarQube, que nos ayuda a detectar errores, duplicaciones, malas prácticas y oportunidades de mejora en el código de nuestra app de ECOTRIBUTARIO.



### Descripción

Métrica	Valor	Nivel	Interpretación
Bugs	0	Muy bien	No se encontraron errores críticos de lógica.
Vulnerabilidad	0	Excelente	No hay fallas de seguridad registradas.
Code Smells (mal olor de código)	13	Medio	Hay problemas de mantenibilidad a revisar.
Cobertura	0.0%	Grave	No hay pruebas unitarias ejecutadas.
Duplicación	17.2%	Muy alto	Demasiado código duplicado, requiere refactor.
Seguridad	9	Crítico	Posibles puntos de riesgo sin mitigar.
Punto de control	Pasado	Cumple	En general, pasa el mínimo requerido.



☐ Bulk Change

Select issues

Navigate to issue

15 issues

2h 50min effort

src/main/java/com/ecotributario/MainApp.java

☐ Make the enclosing method "static" or remove this set.

Intentionality

Maintainability

multi-threading

Open

Not assigned

L19 • 20min effort • 20 minutes ago

☐ Replace this use of System.out by a logger.

Adaptability

Maintainability

bad-practice cert

Open

Not assigned

L25 • 10min effort • 20 minutes ago

☐ Replace this use of System.err by a logger.

Adaptability

Maintainability

bad-practice cert

Open

Not assigned

L38 • 10min effort • 20 minutes ago

src/.../com/ecotributario/controllers/CrearSolicitudController.java

☐ Replace this use of System.out by a logger.

Adaptability

Maintainability

bad-practice cert

Open

Not assigned

L58 • 10min effort • 20 minutes ago

☐ Replace this use of System.out by a logger.

Adaptability

Maintainability

bad-practice cert

Open

Not assigned

L58 • 10min effort • 20 minutes ago

Proyecto Funda

src/main/java/com/ecotributario/MainApp.java

Open in IDE

See all issues in this file

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

```
return mainStage;
}

@Override
public void start(Stage primaryStage) throws Exception {
    mainStage = primaryStage;

    cambiarVista("/com/ecotributario/views/inicio.fxml",
"Ecotributario - Inicio");
}

public static void cambiarVista(String rutaFXML, String tituloVentana) {
    try {
        System.out.println("Intentando cargar: " + rutaFXML);



        FXMLLoader loader = new FXMLLoader(MainApp.class.getResource(
rutaFXML));
        Parent root = loader.load();
    }
}
```

Make the enclosing method "static" or remove this set.

Replace this use of System.out by a logger.





Proyecto Funda

> src/main/java/com/ecotributario/MainApp.java  [Open in IDE](#) [See all issues in this file](#) 

```
20      cambiarVista("/com/ecotributario/views/inicio.fxml",
21      "Ecotributario - Inicio");
22  }
23
24  public static void cambiarVista(String rutaFXML, String tituloVentana) {
25      try {
26          System.out.println("Intentando cargar: " + rutaFXML);
27
28          FXMLLoader loader = new FXMLLoader(MainApp.class.getResource(rutaFXML
29          ));
30          Parent root = loader.load();
31
32          if (mainStage == null) {
33              mainStage = new Stage();
34          }
35
36          mainStage.setTitle(tituloVentana);
37          mainStage.setScene(new Scene(root));
```

Replace this use of System.out by a logger.

Proyecto Funda

> src/main/java/com/ecotributario/MainApp.java  [Open in IDE](#) [See all issues in this file](#) 

```
33      mainStage.setTitle(tituloVentana);
34      mainStage.setScene(new Scene(root));
35      mainStage.show();
36
37      } catch (Exception e) {
38          System.err.println("✖ ERROR: No se pudo cargar la vista: " +
39          rutaFXML);
40
41          e.printStackTrace();
42      }
43
44      public static void main(String[] args) {
45          launch(args);
46      }
47  }
```

Replace this use of System.err by a logger.



	Lines of Code	Security	Reliability	Maintainability	Security Hotspots	Coverage	Duplications
Projecto Funda	618	0	2	13	9	0.0%	17.2%
src/main/java/com/ecotributario	618	0	2	13	9	0.0%	17.2%

1 of 1 shown

Project Overview

Security ? >

Reliability ? >

Maintainability ? >

Security Review ? >

Coverage >

Duplications >

Size >

Complexity ?  
Cyclomatic Complexity 80  
Cognitive Complexity 30

Issues >

Projecto Funda > src/main/java/com/ecotributario View as Select files Navigate 3 items

Cyclomatic Complexity 80

controllers 71

models 4

MainApp.java 5

3 of 3 shown

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

src/.../java/com/ecotributario/controllers/CrearSolicitudController.java Open in IDE

```
63 @FXML
64 private void onVolverAlInicioEmpresa() {
65     try {
66         FXMLLoader loader = new FXMLLoader(getClass().getResource(
67             "/com/ecotributario/views/inicioEmpresa.fxml"));
68         Parent root = loader.load();
69         Stage stage = (Stage) btnVolver.getScene().getWindow();
70         stage.setScene(new Scene(root));
71         stage.setTitle("Panel Empresa");
72         stage.show();
73     } catch (Exception e) {
74         e.printStackTrace();
75     }
76 }
77
```

Make sure this debug feature is deactivated before delivering the code in production.



src/.../java/com/ecotributario/controllers/HistorialSolicitudesController.java [Open in IDE](#)

```
43
44
45     @FXML
46     private void onVolverAlPanel() {
47         try {
48             FXMLLoader loader = new FXMLLoader(getClass().getResource(
49                 "/com/ecotributario/views/inicioEmpresa.fxml"));
50             Parent root = loader.load();
51             Stage stage = (Stage) btnVolver.getScene().getWindow();
52             stage.setScene(new Scene(root));
53             stage.setTitle("Ecotributario - Inicio Empresa");
54         } catch (IOException e) {
55             e.printStackTrace();
56         }
57     }
58 }
```

Make sure this debug feature is deactivated before delivering the code in production.

src/.../java/com/ecotributario/controllers/InicioAdministradorController.java [Open in IDE](#)

```
22
23
24     @FXML
25     private void cerrarSesion(ActionEvent event) {
26         try {
27             FXMLLoader loader = new FXMLLoader(getClass().getResource(
28                 "/com/ecotributario/views/inicio.fxml"));
29             Parent root = loader.load();
30             panelContenido.getScene().setRoot(root);
31         } catch (IOException e) {
32             mostrarError("Error al cerrar sesión");
33             e.printStackTrace();
34         }
35     }
36
37     private void cargarVista(String rutaFXML) {
38         try {
39             FXMLLoader loader = new FXMLLoader(getClass().getResource(rutaFXML));
40             Node contenido = loader.load();
41             panelContenido.getChildren().setAll(contenido);
42         } catch (IOException e) {
43             mostrarError("No se pudo cargar la vista: " + rutaFXML);
44         }
45     }
46 }
```

Make sure this debug feature is deactivated before delivering the code in production.



- La primera que se va a evaluar es la métrica errores (Bugs), el cual se tienen 0, es decir no se encuentran errores críticos de lógica en el código que impidan su funcionamiento.
- Después se evaluaron los fallos de Seguridad, el cual se tienen 0, es decir no se detectaron problemas que pongan en riesgo la información.
- Analizando la métrica de mantenimiento se pueden evidenciar varias partes del código que podríamos simplificar o escribir de mejor manera, en total de tendrían 13.
- En la métrica de mantenimiento se sabe que en la cobertura de pruebas se tiene un 0%, es decir no se tienen, no se han hecho pruebas automáticas para verificar que el programa funcione
- En la métrica de Duplicación, se tiene un 17% puesto que hay muchas partes del código repetidas, lo cual al momento de realizar su proceso de mantenimiento se podría complicar este en su Código
- En la métrica de Seguridad se tienen 9 puntos críticos que podrían causar inconvenientes a futuro si no se revisan a tiempo.
- En la métrica de complejidad del Código se tienen 80 puntos porque en algunas partes del código son más difíciles de entender de lo necesario debido a algunas funciones importantes.

En la implementación de aquellas métricas se pudo analizar muchos aspectos, buenos y malos. Para esto las analizamos de la siguiente manera:

### **1) Uso de impresiones para depuración**

En varios archivos se usa el siguiente tipo de código para mostrar errores:

```
e.printStackTrace();
```

```
System.out.println("algo");
```

Esto es bueno cuando se está programando, pero para la versión final de la app de ECOTRIBUTARIO no se deberían de poner, para poder solucionar esto se debería de reemplazar estas líneas por un sistema de registro de mensajes (logger), que permite guardar errores sin mostrar todo en pantalla (En la versión final se vería malo).

### **2) Mucho código repetido**

También se analizó que se repiten muchas veces las mismas líneas para cargar ventanas y manejar la interfaz. Esto hace que el código sea más largo y difícil de actualizar. Para poder solucionar este error se tendría que crear un método general para cargar dichas ventanas y llamarlo desde distintas clases o lugares para que el bloque no se repita muchas veces.

### 3) Código complejo o con muchas decisiones

Hay funciones con muchas condiciones internas, eso aumenta la dificultad para entender o modificar el código, para poder solucionar esto se podría dividir funciones en partes mas pequeñas. Por ejemplo, si una función hace 3 cosas distintas, crear 3 funciones auxiliares mas simples.

### 4) No hay pruebas automatizadas

El sistema no cuenta con pruebas automáticas que confirmen que lo programado funciona bien, es decir no se hicieron pruebas unitarias, para esto pues se tendrían que hacer para los controladores dichas pruebas unitarias porque allí es el funcionamiento base para que la app este bien.

En la implementación de aquellas métricas se pudo analizar ciertos patrones en la organización del código que se podrían aplicar en la plantilla, logger y controller, para esto las analizamos de la siguiente manera:

Problema detectado	Solucion Propuesta	¿Para que sirve?
Repetición de código	Método reutilizable (plantilla)	Permite encapsular fragmentos de código que se repiten en varios lugares, facilitando la reutilización, el mantenimiento y la legibilidad del sistema.
Impresiones en consola	Registro de mensajes (logger)	Reemplaza los <code>print()</code> por un sistema de logging estructurado, lo cual permite hacer un seguimiento adecuado de los eventos del sistema, especialmente útil en ambientes de producción.
Código con varias responsabilidades	Separar responsabilidades (Controller)	Aplica el principio de responsabilidad única, dividiendo el código en capas o módulos con funciones específicas. Esto mejora la escalabilidad, pruebas y comprensión del código.

En la ejecución de dichas métricas se puede evidenciar que hay varias cosas que se pueden mejorar para que el código sea más claro, seguro y fácil de mantener. Estas mejoras también facilitarán que más adelante otras personas puedan trabajar sobre la app de ECOTRIBUTARIO sin complicaciones, eliminar cosas como:

- Código duplicado.
- Reemplazar impresiones por registros adecuados.
- Organizar mejores funciones complejas.
- Agregar pruebas automáticas para asegurar el buen funcionamiento.

## **Códigos y patrones**

Se usaron los patrones de: Singleton, Strategy y Decorator con el objetivo de que con ayuda de estos 3 patrones se pueda construir la app de ECOTRIBUTARIO más limpia, modular y extensible. Es decir, Singleton se usará para tener una única instancia de un controlador. Strategy se usará para elegir comportamientos en tiempo de ejecución. Y por último decorator, sirve para añadir características a un objeto de forma dinámica. Estos 3 ayudan a que en la app se pueda separar responsabilidades claramente, siendo así parecido a los principios SOLID.

### **Singleton**

### **Strategy**

### **Decorator**

Se aplicó el patrón Singleton en conexion.java siendo un patrón creación al que asegura que una clase tenga solo una instancia y proporciona un punto de acceso global a ella.

En esta parte del código:

```
private static Conexion instancia;

public static Conexion getInstance() {
    if (instancia == null) {
        instancia = new Conexion();
    }
    return instancia;
}
```

Se crea una única instancia por lo que el constructor es privado . Esto Garantiza que se utilice una única conexión a la base de datos durante la ejecución del sistema, lo que ahorra recursos y evita errores de concurrencia.

También se usó el patrón en: SesionAdmin.java y SesionEmpresa.java

Y se evidencia en esta parte del código:

```
private static SesionAdmin instancia;  
  
public static SesionAdmin getInstance() {  
    if (instancia == null) {  
        instancia = new SesionAdmin();  
    }  
    return instancia;  
}
```

Asegura que solo sea un tipo de sesión disponible ya sea administrador o empresa.

Se aplicó el patrón STRATEGY o DECORATOR

Se aplicó en:

- Validador.java (interfaz base)
- Aquí define una interfaz común con el método validar(...).
- ValidadorRequerido.java
- ValidadorNumericoPositivo.java
- Es estas dos se implementan esta estrategia para distintas validaciones.
- CampoValidador.java
- Aca recibe estrategias y las ejecuta según la necesidad:

Por un lado se aplicó STRATEGY, éste define una familia de algoritmos, encapsularlos y hacerlos intercambiables. El algoritmo varía independientemente de los clientes que lo utilizan.

Aquí se puede observar en el siguiente código:

```
public class CampoValidador {  
    private List<Validador> validadores;  
    public boolean validar(String valor) {  
        for (Validador v : validadores) {  
            if (!v.validar(valor)) return false;  
        }  
        return true;  
    }  
}
```

También se podría decir que se usó el patrón de DECORATOR, este consiste en agregar responsabilidades a un objeto de forma dinámica, sin alterar su estructura. Se podría analizar que se usa en estructura tipo Decorator si los validadores se combinan en capas (uno envuelve al otro).

### **Pruebas unitarias**

Las pruebas unitarias ayudan a verificar si alguna función de ECOTRIBUTARIO funciona por sí sola, es decir que no depende de otras funcionalidades, indicando que cada funcionalidad no caerá si otra cae.

### **Servicio integración SMS**

Se implementó un servicio de mensajes el cual mandará mensajes de la base de datos a la aplicación de Telegram, esto se hizo con ayuda de

### **Conclusiones**

La implementación del proyecto ECOTRIBUTARIO demuestra cómo mediante una app puede integrarse eficazmente con iniciativas ambientales para generar un impacto positivo en la sociedad. Al ofrecer descuentos fiscales a empresas comprometidas con prácticas sostenibles, se incentiva una transformación real en el comportamiento corporativo. La aplicación fue desarrollada aplicando métricas de calidad, patrones de diseño GOF y pruebas unitarias, lo que garantiza su robustez, escalabilidad y buen rendimiento. También, se incorporó un servicio de mensajería para fortalecer la comunicación entre el sistema y los usuarios. Este proyecto no solo responde a una necesidad

ambiental urgente, sino que también promueve el uso de buenas prácticas en el desarrollo de los fundamentos de ingeniería de software.

El uso de los patrones de diseño del catálogo GOF (Singleton, Strategy y Decorator) fue fundamental para lograr una arquitectura de software más estructurada, mantenible y flexible en ECOTRIBUTARIO. Estos patrones permitieron encapsular comportamientos, restringir instancias innecesarias y extender funcionalidades de manera dinámica, lo que facilita la escalabilidad del sistema y mejora la calidad del código. Incorporar estos patrones no solo fortaleció el diseño técnico, sino que también promovió la aplicación de principios sólidos de los fundamentos de ingeniería de software como la reutilización y la separación de responsabilidades.

La distribución de responsabilidades y la interacción entre los componentes. Además, facilitaron la comunicación técnica con el equipo y aportaron claridad al proceso de documentación. En conjunto con los patrones GOF.