



**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERÍA**

**Escuela Profesional de Ingeniería de Sistemas**

**Proyecto “Sistema administrativo SoftVet”**

Curso: Calidad y Pruebas de Software

Docente: Ing Patrick Jose Cuadros Quiroga

Integrantes:

<b><i>Chambi Cori, Jerson Roni</i></b>	<b><i>(2021072619)</i></b>
<b><i>Flores Quispe, Jaime Elias</i></b>	<b><i>(2021070309)</i></b>
<b><i>Leyva Sardon, Elvis Ronald</i></b>	<b><i>(2021072614)</i></b>

**Tacna – Perú**  
**2024**

# **Sistema “Sistema administrativo SoftVet” Informe de Análisis**

**Versión {1.0}**

CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	MPV	ELV	ARV	10/10/2020	Versión Original

## ÍNDICE GENERAL

1.	Descripción del Proyecto	4
2.	Riesgos	5
3.	Análisis de la Situación actual	6
4.	Estudio de Factibilidad	7
4.1	Factibilidad Técnica	7
4.2	Factibilidad económica	8
4.3	Factibilidad Operativa	11
4.4	Factibilidad Legal	11
4.5	Factibilidad Social	12
4.6	Factibilidad Ambiental	12
5.	Análisis Financiero	13
6.	Conclusiones	16

## **Informe de Factibilidad**

### **1. Descripción del Proyecto**

#### **1.1 Nombre del proyecto**

“Sistema administrativo SoftVet”

#### **1.2 Duración del proyecto**

La duración del proyecto consta aproximadamente de 1 mes y 2 semanas

#### **1.3 Descripción**

El "Sistema Administrativo SoftVet" ha sido desarrollado como una solución completa para la gestión eficiente de clínicas veterinarias. Este software integral abarca una amplia variedad de funciones y herramientas diseñadas específicamente para optimizar las operaciones diarias en entornos clínicos. Conformado por módulos especializados, como la gestión de citas programadas, la edición de agendas, el seguimiento del historial clínico y de vacunación, así como funciones esenciales como el inicio de sesión, programación de citas, y la capacidad de registrar desparasitaciones y vacunaciones. Además, cuenta con herramientas para generar informes de horarios personales y pacientes atendidos, así como funciones de recepción. Permite también la gestión completa de clientes, empleados y pacientes, facilitando su registro, edición y eliminación en el sistema.

Este sistema ha sido desarrollado con una base sólida respaldada por la implementación de SQL Server. La utilización de SQL Server Management Studio (SSMS) contribuye a la eficiencia y robustez del sistema al proporcionar una gestión segura y estructurada de la base de datos. La integración de SQL Server garantiza la confiabilidad en el almacenamiento y recuperación de datos, mejorando la capacidad del sistema para gestionar información crítica, como historiales clínicos, citas programadas y registros de pacientes, con eficacia y seguridad.

#### **1.4 Objetivos**

##### **1.4.1 Objetivo general**

Implementar un sistema integral de gestión de servicios veterinarios en Visual Studio, utilizando C# con Programación Orientada a Objetos (POO) y SQL Server

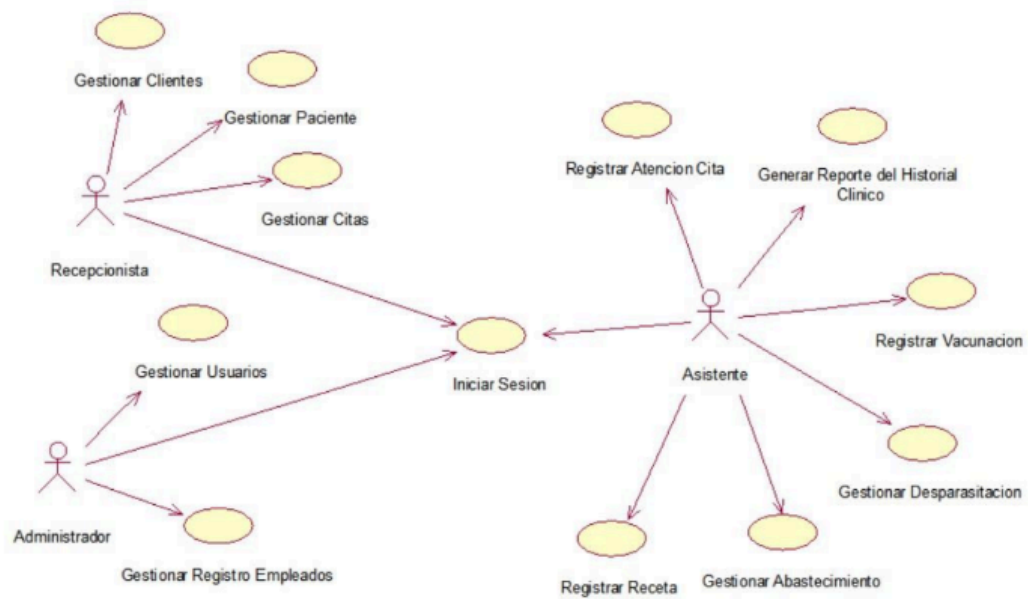
Management Studio (SSMS). Este sistema buscará optimizar la eficiencia y calidad de los servicios veterinarios mediante la integración de herramientas de análisis de código y seguridad (Snyk, SonarCloud y Semgrep) en el flujo de trabajo de GitHub Actions, garantizando así un código seguro y de alta calidad en la administración de citas, historias clínicas y operaciones diarias en la clínica veterinaria.

#### 1.4.2 Objetivos Específicos

- ❖ Evaluar la capacidad técnica y los recursos necesarios para desarrollar e implementar el sistema integral de gestión en el entorno propuesto de Visual Studio, C# y SQL Server Management Studio.
- ❖ Examinar la operatividad del sistema, considerando cómo se integrará con los procesos diarios de la clínica veterinaria, y analizar el impacto de las herramientas Snyk, SonarCloud y Semgrep en la mejora de la eficiencia operativa y la calidad del código en la gestión de citas, historias clínicas y otros aspectos esenciales.
- ❖ Establecer un cronograma para el desarrollo del proyecto, considerando los plazos de entrega, la disponibilidad de recursos y cualquier factor que pueda afectar el tiempo de implementación.

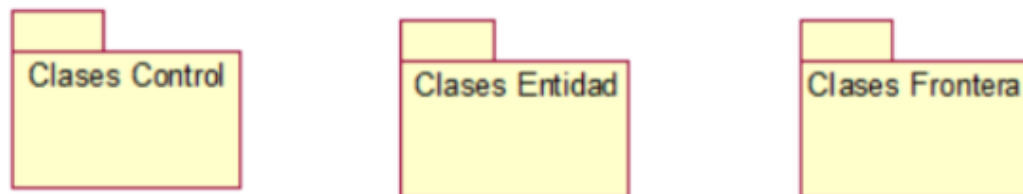
## 2. Diagramas

### 2.1 Diagrama de casos de uso

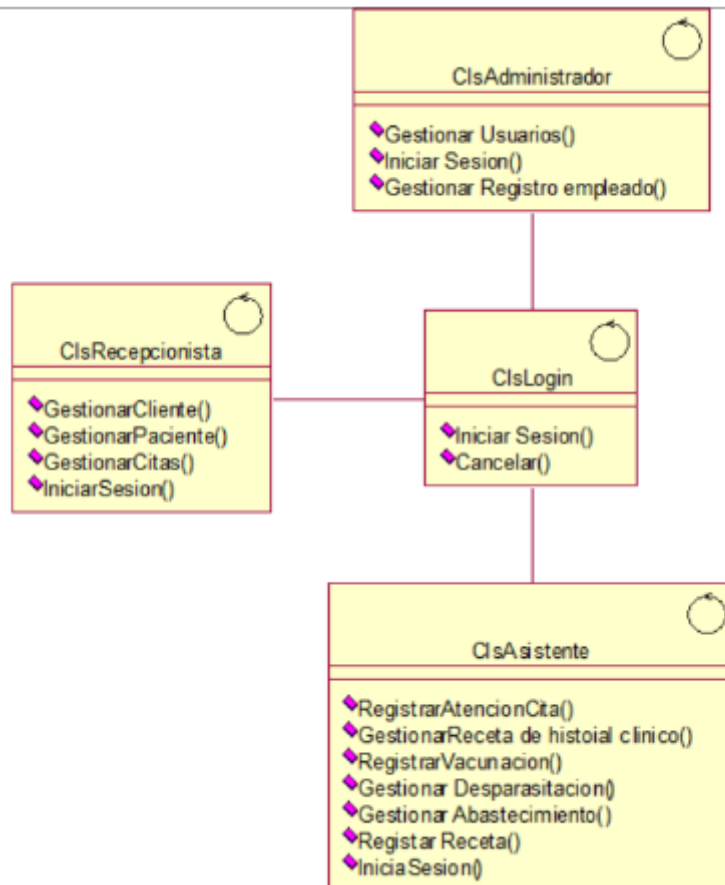


## 2.2 Diagrama de clases

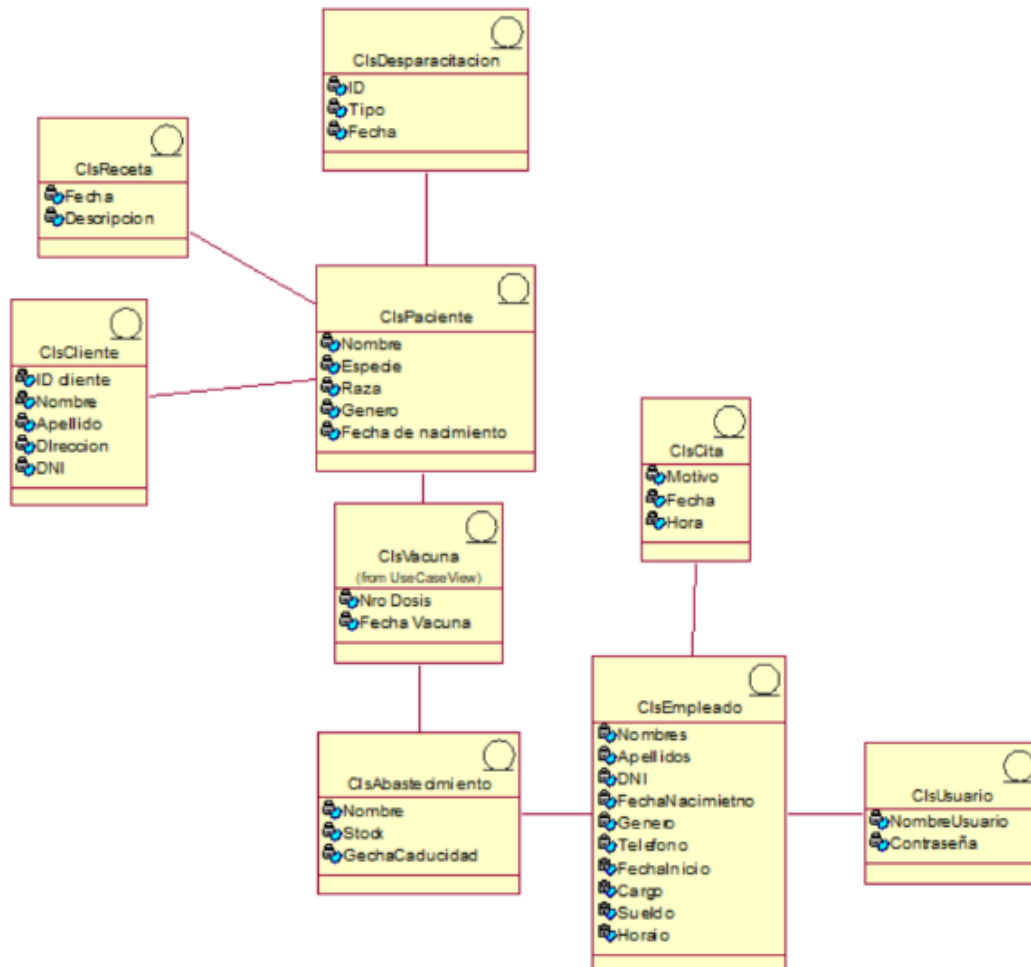
## 2.3 Paquete



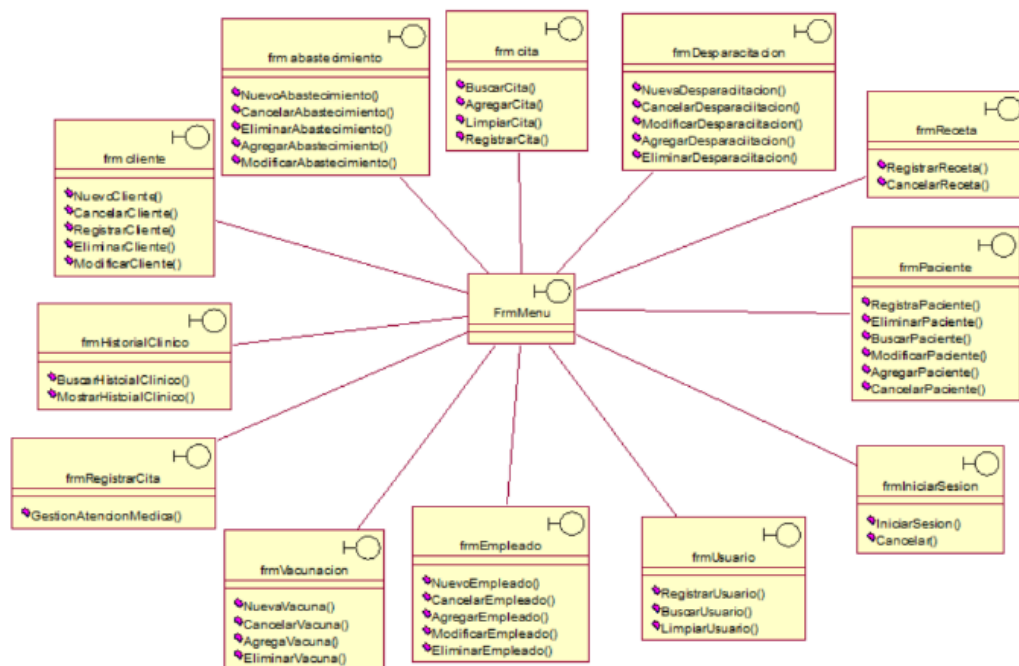
## 2.4 Diagrama de clases de control



## 2.5 Diagrama de clases de entidad

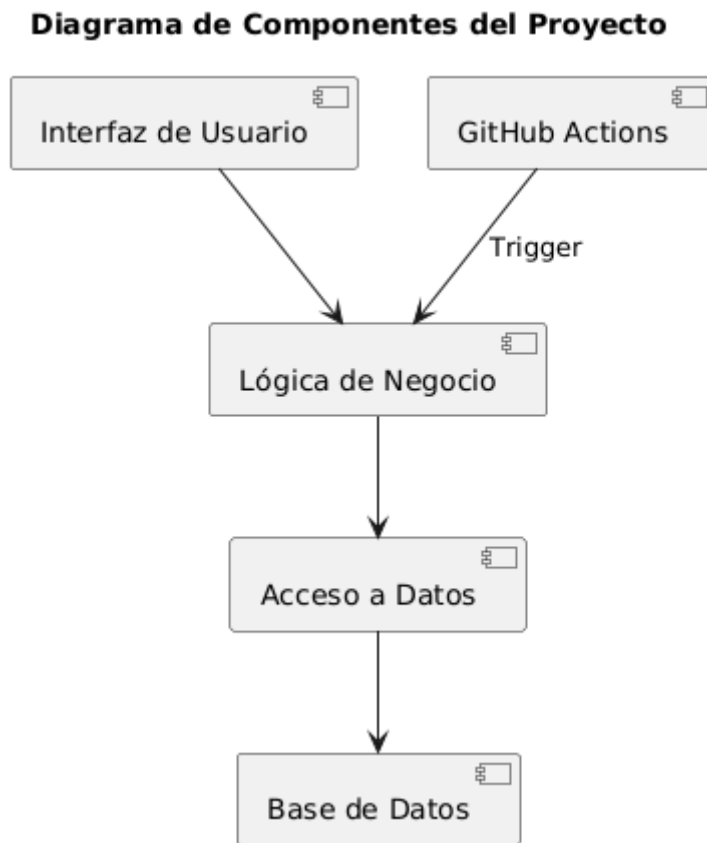


## 2.6 Diagrama de clases de frontera

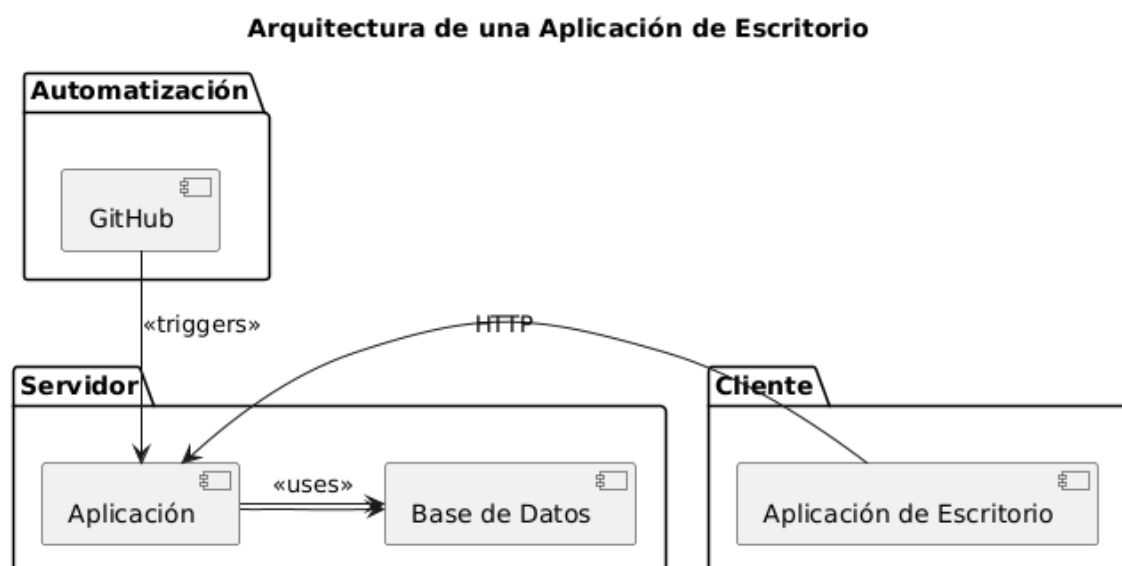




## 2.7 Diagrama de Componentes



## 2.8 Diagrama de Arquitectura



## 3. Análisis de la Situación actual

### 3.1 Planteamiento del problema

El planteamiento del problema en el contexto del "Sistema Administrativo SoftVet" se fundamenta en los desafíos y limitaciones enfrentados por las clínicas veterinarias en la actualidad. Las prácticas tradicionales de gestión, que involucran hojas de cálculo y métodos manuales, han resultado insuficientes para abordar la creciente complejidad de las operaciones diarias en el ámbito veterinario. A medida que la demanda de servicios veterinarios aumenta, se evidencian problemas en la programación de citas, la gestión de historias clínicas en formato de papel y la eficiencia general en la coordinación de las diferentes áreas de la clínica.

El uso de métodos convencionales, se vuelve cada vez más ineficiente a medida que el volumen de pacientes crece, generando redundancias, inconsistencias y tiempos de espera prolongados. La falta de una solución tecnológica integral dificulta la comprensión de datos, limita las capacidades de búsqueda y extracción de información relevante, y afecta la coordinación eficiente entre las diversas áreas de la clínica veterinaria. SoftVet aborda estos problemas al ofrecer una solución tecnológica completa, aprovechando herramientas como Visual Studio, C# y SQL Server Management Studio.

### 3.2 Consideraciones de hardware y software

Consideraciones de Hardware:

- Procesador i5-7th o equivalente para garantizar un rendimiento adecuado durante la ejecución del sistema.
- Windows 8 o versiones superiores, para soportar las tecnologías utilizadas en el desarrollo del software.
- 8 GB de memoria RAM tipo DDR4 para garantizar una ejecución fluida y eficiente del sistema
- Un monitor básico, junto con un mouse y teclado estándar, son suficientes para las operaciones del sistema.

## Consideraciones de Software:

- Visual Studio: Plataforma de desarrollo integrado (IDE) que se utiliza como entorno principal para la creación del software.
- C#: Herramienta de desarrollo con Programación Orientada a Objetos (POO)
- SQL Server Management Studio (SSMS): Entorno integrado para la administración, configuración, y desarrollo de SQL Server.

## 4. Metodologías de análisis

### 4.1 Sonarcloud

SonarCloud es una plataforma de análisis de código que permite a los desarrolladores mejorar la calidad y la seguridad de sus proyectos de software, incluyendo aquellos construidos sobre .NET Framework. Al proporcionar análisis en tiempo real, SonarCloud ayuda a identificar problemas de calidad del código, vulnerabilidades de seguridad y deudas técnicas. Con su integración en flujos de trabajo de CI/CD, SonarCloud permite a los equipos recibir retroalimentación continua sobre su código, lo que facilita la identificación temprana de errores y la implementación de mejores prácticas de programación. Además, SonarCloud ofrece un panel de control intuitivo y visualizaciones claras que permiten a los desarrolladores monitorear el progreso de la calidad del código a lo largo del tiempo, asegurando que las aplicaciones no solo sean funcionales, sino también mantenibles y seguras.

#### 4.1.1 Workflow

Para la creación del nuevo flujo de trabajo se creó el archivo sonar.yml. Para ello, se consideró la siguiente estructura

```
sonar.yml
1 name: Sonar Continuous Integration
2 env:
3   DOTNET_VERSION: '8.x' # la versión de .NET
4   SONAR_ORG: 'jaimeflores' # Nombre de la organización de SonarCloud
5   SONAR_PROJECT: 'jaimeflores_softvet' # Key ID del proyecto de Sonar
6
7 on:
8   push:
9     branches: [ "main" ]
10  workflow_dispatch:
11
12 jobs:
13   sonarqube:
14     name: Sonarqube Analysis
15     runs-on: windows-latest
16     steps:
17       - uses: actions/checkout@v4
18
19       - name: Instalar Herramientas de Construcción de .NET Framework
20         uses: actions/setup-dotnet@v4
21         with:
22           dotnet-version: '8.0.x'
23
24       - name: Configurar MSBuild
25         uses: microsoft/setup-msbuild@v2
```

```
27 - name: Restaurar la Aplicación
28   run: nuget restore ProyectoFinal.sln
29
30 - name: Instalar Scanner
31   run: dotnet tool install -g dotnet-sonarscanner
32
33 - name: Ejecutar escaneo
34   run: |
35     cd ProyectoFinal
36     dotnet-sonarscanner begin /k:"${{ env.SONAR_PROJECT }}" /o:"${{ env.SONAR_ORG }}" /d:sonar.login="${{ secrets.SONAR_TOKEN }}" /d:sonar.host.url="https://sonarcloud.io"
37     msbuild ProyectoFinal.csproj /p:Configuration=Debug
38     dotnet-sonarscanner end /d:sonar.login="${{ secrets.SONAR_TOKEN }}"
39
```

Este workflow de GitHub Actions está diseñado para realizar un análisis de calidad de código utilizando SonarCloud en A continuación se presenta un resumen de las tareas que ejecuta:

**Definición de Variables de Entorno:** Se establecen variables como la versión de .NET, la organización en SonarCloud y el ID del proyecto.

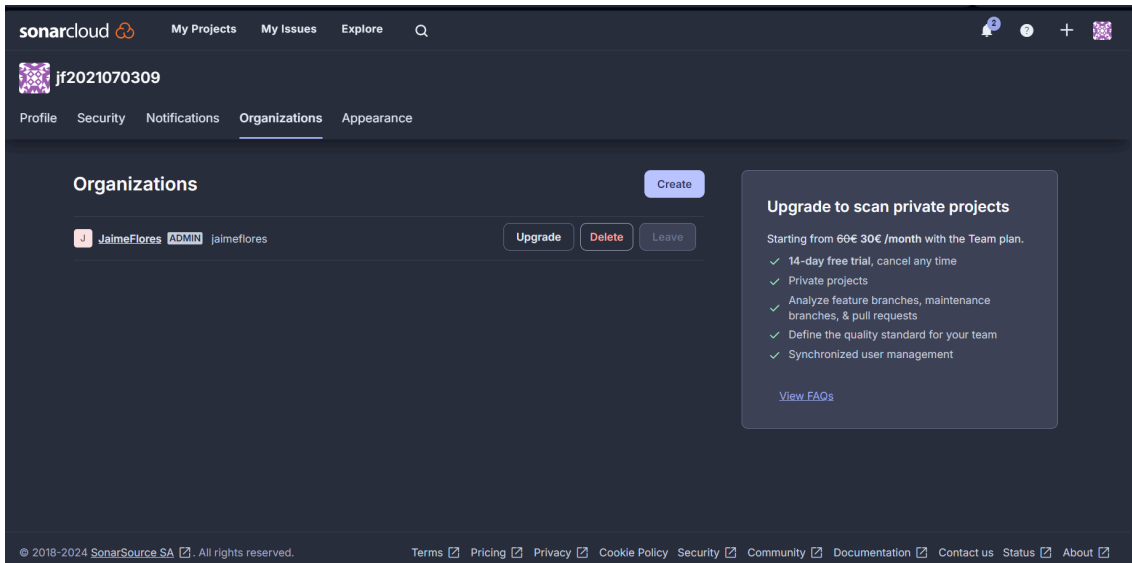
**Activación del Workflow:** Se ejecuta automáticamente cuando hay un push a la rama main.

**Análisis de Sonarqube:**

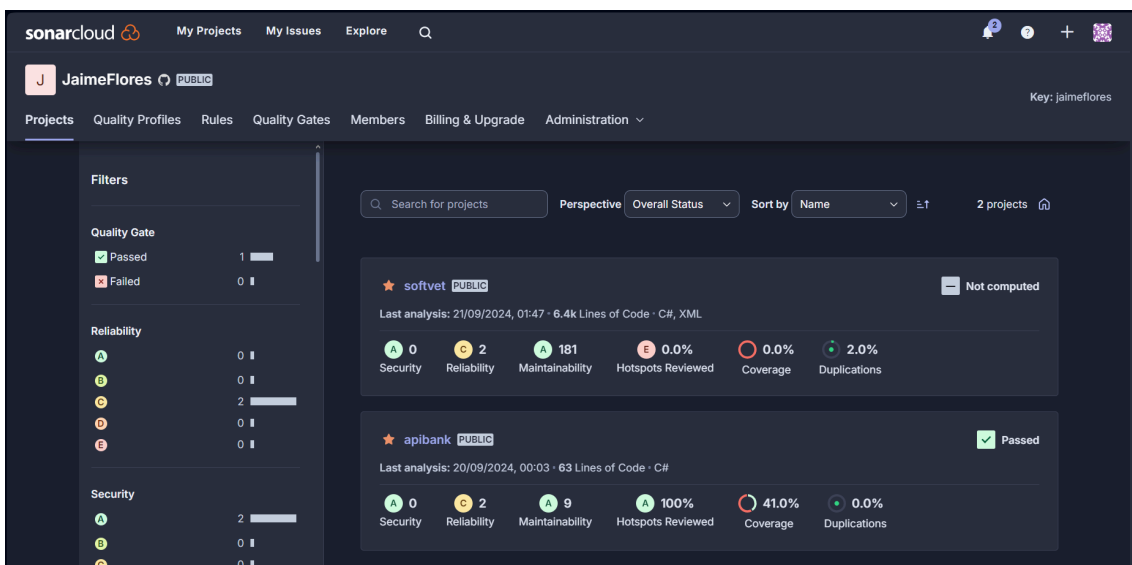
- **Checkout del Repositorio:** Se clona el repositorio actual para acceder al código fuente.
- **Instalación de Herramientas de Construcción:** Se configura el entorno de .NET Framework para asegurar que se utilice la versión adecuada.
- **Configuración de MSBuild:** Se prepara el entorno para compilar proyectos de .NET.
- **Restauración de Paquetes:** Se restauran las dependencias del proyecto utilizando nuget restore.
- **Instalación del Scanner de SonarCloud:** Se instala la herramienta dotnet-sonarscanner necesaria para el análisis.
- **Ejecución del Escaneo:** Se inicia el escaneo de SonarCloud con las credenciales necesarias, se compila el proyecto con msbuild y se finaliza el escaneo, enviando los resultados a SonarCloud.

#### 4.1.2 Resultado de análisis

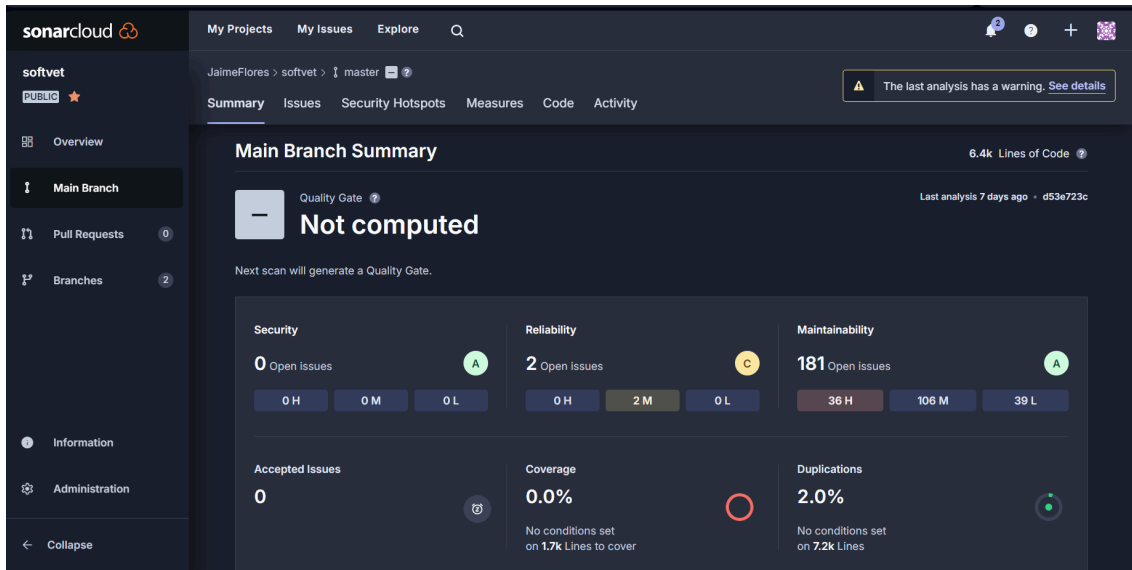
Para revisar el análisis que se revisó desde github actions, nos dirigimos a la plataforma de sonarcloud y nos autenticamos. Luego, nos dirigimos a la sección de “Organizations” para ingresar a nuestra organización personal el cual está vinculada con nuestro proyecto e ingresamos.



Una vez dentro de la plataforma, visualizaremos que tenemos dos análisis de dos proyectos, pero en este caso ingresaremos a nuestro proyecto softvet. Previamente, nos da algunos resultados en porcentajes o por puntos según las categorías existentes.



Al haber ingresado, notaremos que nos muestra a más detalle los indicadores del análisis realizado en nuestro proyecto, como que en rentabilidad solo tenemos 2 puntos, en mantenibilidad 181 y en el área de duplicación tendremos 2%, lo cual son aspectos que debemos corregir a futuro para tener un mejor resultado. Respecto al porcentaje de cobertura, notamos que esta un 0.0%, esto es porque cómo es un proyecto de escritorio en .NET Framework, con el que se gestiona el registro de pacientes y la generación de reportes para una clínica veterinaria, no se mantiene registro de pruebas realizadas dentro del proyecto, dado que este tipo de aplicaciones tiene el enfoque específico y limitado en sus funcionalidades.



## 4.2 Snyk

Snyk es una herramienta de análisis de seguridad diseñada para identificar y remediar vulnerabilidades en las dependencias de proyectos, incluida la plataforma .NET Framework. Al integrarse en el ciclo de vida del desarrollo de software, Snyk permite a los desarrolladores detectar de manera proactiva problemas de seguridad en sus bibliotecas y componentes de terceros. A través de análisis automatizados, Snyk proporciona informes detallados sobre vulnerabilidades conocidas y sugiere soluciones efectivas, ayudando a los equipos a mantener la seguridad de sus aplicaciones y a cumplir con las mejores prácticas de desarrollo seguro. Su capacidad para integrarse con flujos de trabajo existentes, como CI/CD, lo convierte en una opción valiosa para proyectos .NET Framework que buscan mejorar su postura de seguridad sin interrumpir el proceso de desarrollo.

### 4.2.1 Workflow:

Para la creación del nuevo flujo de trabajo se creó el archivo snyk.yml. Para ello, se consideró la siguiente estructura

```
snyk.yml
.github > workflows > snyk.yml
1  name: Snyk Security
2
3  on:
4    push:
5      branches: ["main"]
6
7  permissions:
8    contents: read
9    security-events: write
10   actions: read
11
12  jobs:
13    snyk:
14      runs-on: ubuntu-latest
15
16      steps:
17        - uses: actions/checkout@v4
18
19        - name: Set up .NET
20          uses: actions/setup-dotnet@v3
21          with:
22            dotnet-version: '8.x'
23
24        - name: Set up Snyk CLI
25          uses: snyk/actions/setup@806182742461562b67788a64410098c9d9b96adb
26
```

```

27        - name: Authenticate with Snyk
28          env:
29            SNYK_TOKEN: ${ secrets.SNYK_TOKEN }
30          run: snyk auth $SNYK_TOKEN
31
32        - name: Change to Project Directory
33          run: cd ProyectoFinal
34
35        - name: Snyk Code test
36          env:
37            SNYK_TOKEN: ${ secrets.SNYK_TOKEN }
38          run: |
39            cd ProyectoFinal
40            snyk code test --sarif --all-projects > snyk-code.sarif || echo "Snyk code test failed with exit"
41
42        - name: Snyk Open Source monitor
43          env:
44            SNYK_TOKEN: ${ secrets.SNYK_TOKEN }
45          run: |
46            cd ProyectoFinal
47            snyk monitor
48
49        - name: Upload result to GitHub Code Scanning
50          uses: github/codeql-action/upload-sarif@v3
51          with:
52            sarif_file: ProyectoFinal/snyk-code.sarif
53
```

Este workflow de GitHub Actions, llamado "Snyk Security", tiene como objetivo realizar un análisis de seguridad en un proyecto cada vez que se realiza un push a la rama main. Entre las tareas que realiza este workflow están:

Activación del workflow: Se activa automáticamente al hacer un push en la rama main.

Configuración de permisos: Se establecen permisos para leer el contenido y escribir eventos de seguridad.

Definición del entorno de ejecución: El job snyk se ejecuta en un entorno Ubuntu.

Checkout del repositorio: Se utiliza la acción actions/checkout para clonar el repositorio en el entorno de ejecución.

Configuración de .NET: Se configura la versión de .NET requerida (en este caso, la versión 8.x) utilizando la acción actions/setup-dotnet.

Configuración de Snyk CLI: Se configura la interfaz de línea de comandos (CLI) de Snyk para permitir el análisis del proyecto.

Autenticación con Snyk: Se autentica en Snyk utilizando un token de acceso almacenado en los secretos del repositorio (SNYK\_TOKEN).

Cambio al directorio del proyecto: Se cambia al directorio del proyecto (ProyectoFinal) para realizar los análisis.

Análisis de código con Snyk: Se ejecuta el comando snyk code test, que realiza un análisis de seguridad del código y genera un informe en formato SARIF (snyk-code.sarif).

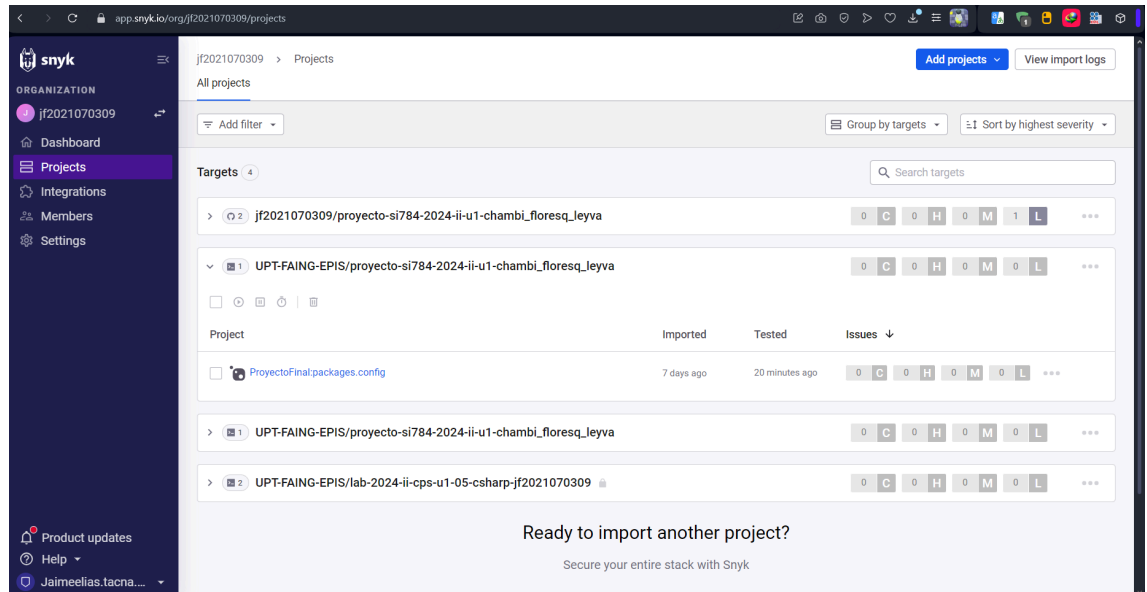
Monitoreo de código abierto con Snyk: Se utiliza el comando snyk monitor para monitorear las dependencias de código abierto del proyecto en busca de vulnerabilidades.

Carga de resultados a GitHub Code Scanning: Se suben los resultados del análisis a GitHub utilizando la acción github/codeql-action/upload-sarif, lo que permite a los desarrolladores ver los problemas de seguridad directamente en la interfaz de GitHub.

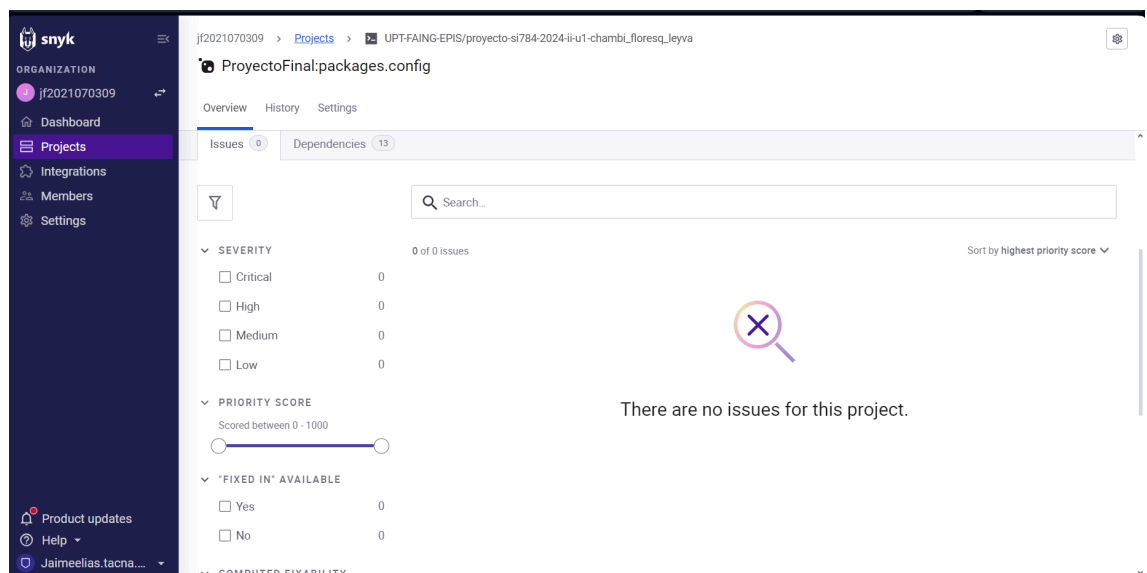
#### 4.2.2 Resultado

Para revisar el análisis que se revisó desde github actions, nos dirigimos a la plataforma de snyk y nos autenticamos. Luego, nos dirigimos a la sección de Projects para visualizar nuestro repositorio.



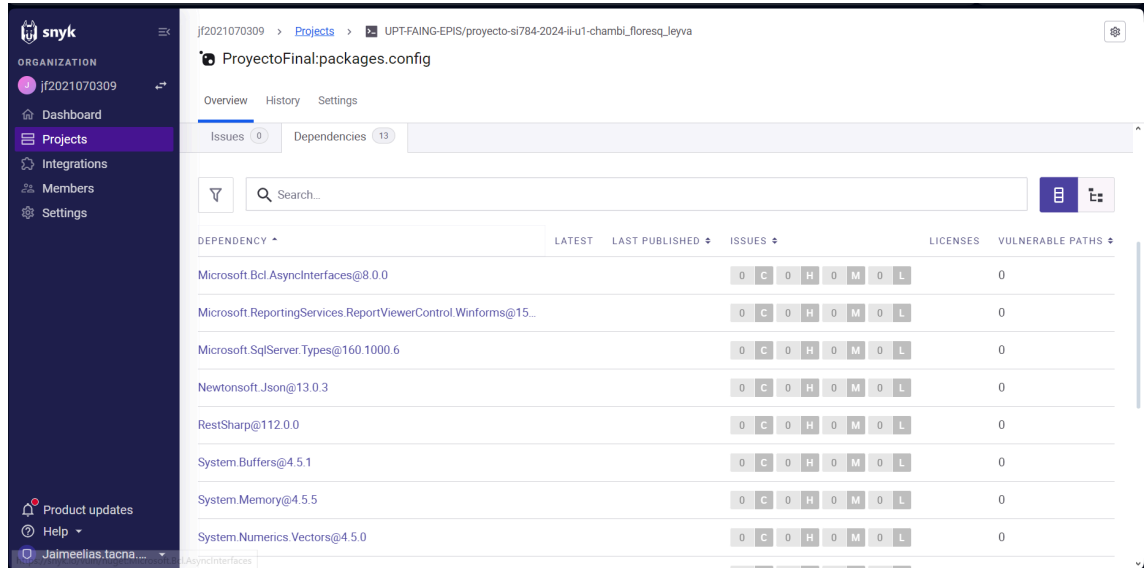


Ahora, ingresamos dentro del proyecto para revisar el resultado del análisis



Dando a entender que, Snyk no ha encontrado vulnerabilidades críticas, altas, medias o bajas. Por lo que, cómo es un proyecto de escritorio en .NET Framework, con el que se gestiona el registro de pacientes y la generación de reportes para una clínica veterinaria, no se muestra problemas en Snyk, dado que este tipo de aplicaciones tiene el enfoque específico y limitado en sus funcionalidades.

Pero si nos vamos a la sección de dependencias notaremos que tenemos enlazadas varias referencias como Microsoft.Bcl.AsyncInterfaces, ReportingServices.ReportViewerControl y SqlServer.Types, entre otras.



DEPENDENCY	LATEST	LAST PUBLISHED	ISSUES	LICENSES	VULNERABLE PATHS
Microsoft.Bcl.AsyncInterfaces@8.0.0			0 C 0 H 0 M 0 L	0	
Microsoft.ReportingServices.ReportViewerControl.Winforms@15...			0 C 0 H 0 M 0 L	0	
Microsoft.SqlServer.Types@160.1000.6			0 C 0 H 0 M 0 L	0	
Newtonsoft.Json@13.0.3			0 C 0 H 0 M 0 L	0	
RestSharp@112.0.0			0 C 0 H 0 M 0 L	0	
System.Buffers@4.5.1			0 C 0 H 0 M 0 L	0	
System.Memory@4.5.5			0 C 0 H 0 M 0 L	0	
System.Numerics.Vectors@4.5.0			0 C 0 H 0 M 0 L	0	

## 4.3 Semgrep

Semgrep es una herramienta de análisis estático de código que permite a los desarrolladores identificar vulnerabilidades de seguridad y problemas de calidad en múltiples lenguajes de programación mediante la búsqueda de patrones de código. Su integración sencilla en flujos de trabajo de CI/CD y su capacidad para generar informes detallados hacen que sea una opción efectiva para mejorar la seguridad y la calidad del software en tiempo real. Además, su enfoque flexible permite a los usuarios crear y personalizar sus propios patrones, adaptándose a las necesidades específicas de sus proyectos.

### 4.3.1 Workflow

Para la creación del nuevo flujo de trabajo se creó el archivo semgrep.yml. Para ello, se consideró la siguiente estructura

```
semgrep.yml M X
.github > workflows > semgrep.yml
1  name: Semgrep Analysis
2
3  on:
4    push:
5      branches:
6        - main
7
8  jobs:
9    security:
10     runs-on: ubuntu-latest
11     container:
12       image: semgrep/semgrep
13     steps:
14       - name: Checkout Repository
15         uses: actions/checkout@v4
16
17       - name: Setup .NET Framework
18         run: |
19           echo "Configurar .NET Framework aquí si es necesario."
20
21       - name: Semgrep scan
22         run: semgrep scan --config="p/default" --sarif --output=report.sarif --metrics=off
23
24       - name: Upload result to GitHub Code Scanning
25         uses: github/codeql-action/upload-sarif@v3
26         with:
27           sarif_file: report.sarif
28
```

Este workflow de GitHub Actions llamado Semgrep Analysis se ejecuta en la rama main al realizar un push. A continuación, se detallan las tareas que se llevan a cabo:

Checkout Repository: Se clona el repositorio en el entorno de ejecución.

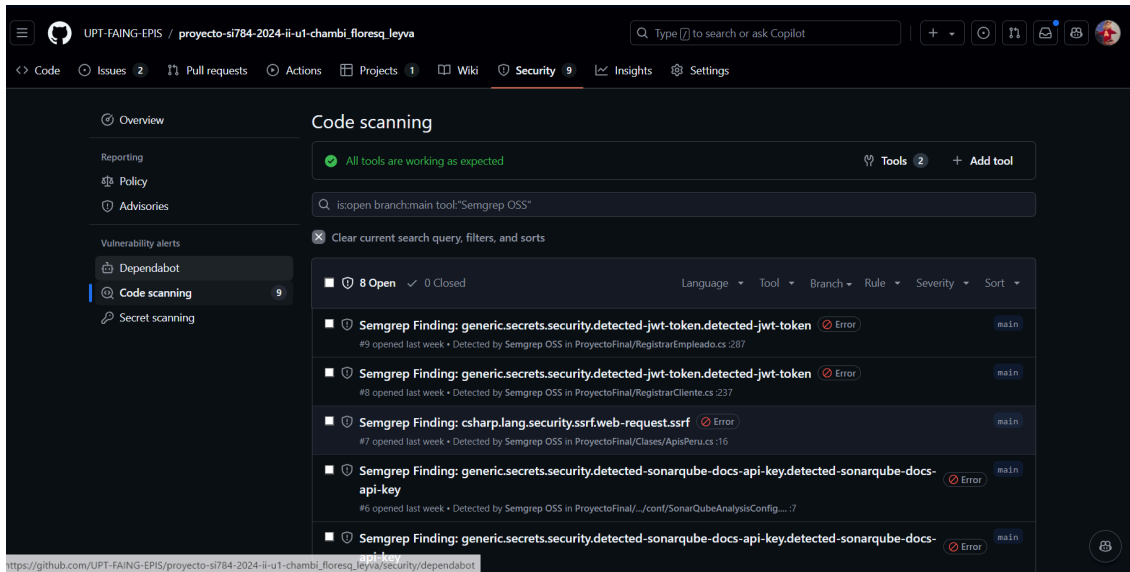
Setup .NET Framework: Se configura el entorno de .NET Framework si es necesario (se incluye un mensaje de ejemplo).

Semgrep Scan: Se ejecuta un análisis de código utilizando Semgrep, con una configuración predeterminada (p/default), generando un informe en formato SARIF (report.sarif).

Upload Result to GitHub Code Scanning: Se sube el informe SARIF generado al sistema de análisis de código de GitHub para su visualización y revisión.

#### 4.3.2 Resultado de análisis

En este caso, hemos generado el análisis y lo hemos subido a code scanning el cual se encuentra dentro de nuestro repositorio de github. Nos aparecerán los siguientes resultados:



El análisis de seguridad realizado mediante GitHub Code Scanning y Semgrep ha identificado varias vulnerabilidades en el código del proyecto. Se han detectado múltiples incidencias relacionadas con la exposición de tokens JWT, específicamente en los archivos RegistrarEmpleado.cs y RegistrarCliente.cs. Además, se ha encontrado una vulnerabilidad SSRF en ApisPeru.cs, lo que podría permitir ataques de tipo Server-Side Request Forgery. Por último, también se ha identificado la exposición de claves de API de SonarQube en la configuración correspondiente. Es fundamental abordar estos hallazgos para mejorar la seguridad general de la aplicación.

#### 4.4 Release

Un **release** es una versión específica de un software que se marca para distribución. Generalmente incluye una combinación de nuevas características, correcciones de errores y mejoras. En GitHub, un release se asocia a un **tag** (como **v1.0.0**) y puede contener binarios o artefactos compilados listos para descarga, junto con una descripción de los cambios realizados en esa versión.

#### 4.4.1 Workflow

```
on:
  push:
    branches:
      - main
    tags:
      - 'v*' # Esto detecta cualquier tag que empiece con 'v' (por ejemplo, v1.0.0, v2.1.0)
```

```
release:
  needs: nuget-package
  runs-on: ubuntu-latest

  steps:
    - name: Checkout Code
      uses: actions/checkout@v4

    - name: Create GitHub Release
      uses: softprops/action-gh-release@v1
      with:
        tag_name: ${{ github.ref }}
        body: |
          ### Cambios en esta versión:
          - Mejora 1
          - Corrección de errores
          - Actualización de dependencias
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Este bloque de código representa un **job** llamado **release** dentro del workflow de GitHub Actions, que se encarga de crear una nueva versión del proyecto, o **release**, en GitHub. Este job tiene las siguientes características:

- **release::** Define el trabajo que realizará la tarea de release.
- **needs: nuget-package:** Indica que este job depende de que el job **nuget-package** haya finalizado exitosamente. Es decir, el release solo se ejecutará si el paquete NuGet ya ha sido creado y publicado correctamente.
- **runs-on: ubuntu-latest:** Especifica que este job se ejecutará en un entorno con el sistema operativo **Ubuntu**, en su versión más reciente.

Pasos dentro del job release:

- **Checkout del código:**
  - El paso **Checkout Code** utiliza la acción **actions/checkout@v4** para descargar el código fuente del repositorio en la máquina virtual de GitHub Actions. Esto es necesario porque el release se basa en el código actual del proyecto.

- **Crear el release en GitHub:**
  - El paso **Create GitHub Release** utiliza la acción `softprops/action-gh-release@v1` para crear un **release** en GitHub.
  - **tag\_name: \${{ github.ref }}**: Se utiliza la referencia de Git que activó el workflow (almacenada en `{{ github.ref }}`), generalmente un tag de versión (como `v1.0.0`), para asociar el release a ese tag en GitHub.
  - **body::** Aquí se define el cuerpo del release, que describe los cambios realizados en esta versión. En este caso, se mencionan mejoras, correcciones de errores y actualizaciones de dependencias, aunque estos detalles pueden personalizarse según los cambios reales de cada versión.
- **Autenticación:**
  - En el bloque **env::**, se pasa el token secreto `GITHUB_TOKEN` a través de la variable de entorno, lo que autoriza el proceso de creación del release. Este token es automáticamente proporcionado por GitHub para permitir que los workflows realicen acciones seguras, como la creación de un release en el repositorio.

#### 4.4.2 Resultado



The screenshot displays the GitHub releases page for the 'github-actions' repository. It shows two releases:

- Release v3.0.3 (2 days ago):**
  - Changes: Mejora 1, Corrección de errores, Actualización de dependencias.
  - Assets: Source code (zip) and Source code (tar.gz), both uploaded 2 days ago.
- Release v0.3.1 (3 days ago):**
  - Changes: Mejora 1, Corrección de errores, Actualización de dependencias.

El resultado de la ejecución del código demuestra la creación automatizada de **releases** mediante el uso de tags cuyo nombre comienza con "v". Esto garantiza que, cada vez que se realice un **push** en el proyecto acompañado de un tag que cumpla con las condiciones definidas en el workflow, se activará el proceso, generando un nuevo **release** asociado a la versión indicada por el tag. Además, se utilizará la descripción predeterminada establecida en el workflow para ese release.

Este enfoque automatiza el proceso de publicación de nuevas versiones, asegurando que cualquier nueva versión del proyecto (identificada por un tag) sea lanzada automáticamente como un **release** en GitHub, simplificando la gestión de versiones y liberaciones.

#### 4.5 Package Nuget

NuGet es el administrador de paquetes para la plataforma de desarrollo .NET, que permite a los desarrolladores compartir, usar, y administrar librerías o herramientas de terceros en sus proyectos de forma fácil. Un paquete NuGet es un archivo .nupkg que contiene el código compilado (normalmente en forma de ensamblado .DLL), junto con metadatos como su nombre, versión, dependencias, y otra información relevante.

- Creación: Un desarrollador crea un proyecto .NET y, una vez listo, genera un paquete NuGet a partir del código fuente. Esto se hace empaquetando los binarios junto con los metadatos en un archivo .nupkg.
- Publicación: Una vez creado el paquete, puede ser subido a repositorios públicos o privados, como NuGet.org, GitHub Packages o cualquier otro servidor NuGet.
- Uso: Otros desarrolladores pueden buscar, descargar, e instalar estos paquetes directamente desde sus IDEs (como Visual Studio) o desde la línea de comandos usando el comando `nuget` o `dotnet add package`. Esto les permite integrar fácilmente la funcionalidad del paquete en sus propios proyectos.



##### 4.5.1 Workflow

```
! ci.yml X
C: > Users > ELVIS > 3D Objects > proyecto-si784-2024-ii-u1-chambi_floresq_leyva > .github > workflows > ! ci.yml
1  name: Build, Release and NuGet Publish
2
3  on:
4    push:
5      branches:
6        - nugget
7      tags:
8        - 'v*' # Esto detecta cualquier tag que empiece con 'v' (por ejemplo, v1.0.0, v2.1.0)
9
10   workflow_run:
11     workflows:
12       - "Sonar Continuous Integration"
13       - "Snyk Security"
14       - "Sengrep Analysis"
15     types:
16       - completed
17
18   jobs:
19     build:
20       runs-on: windows-latest
21
22       steps:
23         - name: Checkout Code
24           uses: actions/checkout@v4
25
26         - name: Setup .NET Framework
27           uses: actions/setup-dotnet@v4
28           with:
29             dotnet-version: '8.0.x'
30
31         - name: Setup MSBuild
32           uses: microsoft/setup-msbuild@v2
33
34         - name: Restore NuGet Packages
35           run: nuget restore ProyectoFinal.sln
36
37         - name: Clean Build Directories
38           run: |
39             if (Test-Path "${{ github.workspace }}/ProyectoFinal/bin") {
40               Remove-Item -Recurse -Force "${{ github.workspace }}/ProyectoFinal/bin"
41             }
42             if (Test-Path "${{ github.workspace }}/ProyectoFinal/obj") {
43               Remove-Item -Recurse -Force "${{ github.workspace }}/ProyectoFinal/obj"
44             }
45
46         - name: Build Project
47           run: msbuild ProyectoFinal.sln /p:Configuration=Release /p:Platform="Any CPU"
48
49         - name: Verify Build Artifacts
50           run: |
51             dir "${{ github.workspace }}/ProyectoFinal/bin/Release/"
52
```



```

53   - name: Upload Artifacts
54     uses: actions/upload-artifact@v3
55     with:
56       name: build-artifacts
57       path: ${{ github.workspace }}/ProyectoFinal/bin/Release/
58
59   nuget-package:
60     needs: build
61     runs-on: windows-latest
62
63     steps:
64       - name: Checkout Code
65         uses: actions/checkout@v4
66
67       - name: Setup .NET Framework
68         uses: actions/setup-dotnet@v4
69         with:
70           dotnet-version: '8.0.x'
71
72       - name: Add NuGet Source
73         run: |
74           dotnet nuget add source --username jf2021070309 --password ${{ secrets.NEW_TOKEN_NUGET }} --store-password-in-clear-text --name github "https://nuget.pkg.github.com/ProyectoFinal/vs2022/nuget/index.json"
75
76       - name: Copy Template nuspec
77         run: |
78           cd ProyectoFinal
79           copy ProyectoFinal.template.nuspec ProyectoFinal.nuspec
80
81       - name: Determine Next Version
82         id: determine_version
83         run: |
84           # Define la versión base
85           $baseVersion = "1.0.4"
86           # Incrementa la parte menor de la versión
87           $versionParts = $baseVersion -split '\.'
88           $versionParts[2] = [int]$versionParts[2] + 1
89           $nextVersion = "${versionParts[0]}.${versionParts[1]}.${versionParts[2]}"
90           echo "::set-output name=next_version::$nextVersion"
91
92       - name: Replace Version in nuspec
93         run: |
94           cd ProyectoFinal
95           # Reemplazar la versión en el archivo .nuspec con la nueva versión
96           (Get-Content ProyectoFinal.nuspec) -replace '{VERSION}', '${{ steps.determine_version.outputs.next_version }}' | Set-Content ProyectoFinal.nuspec
97
98       - name: Pack NuGet Package from nuspec
99         run: |
100           cd ProyectoFinal

```

```

101       nuget pack ProyectoFinal.nuspec
102
103     - name: Push NuGet Package to GitHub Packages
104       run: |
105         cd ProyectoFinal
106         nuget push ProyectoFinal.${{ steps.determine_version.outputs.next_version }}.nupkg -Source github -ApiKey ${{ secrets.NEW_TOKEN_NUGET }} -SkipDuplicate
107
108   release:
109     needs: nuget-package
110     runs-on: ubuntu-latest
111
112     steps:
113       - name: Checkout Code
114         uses: actions/checkout@v4
115
116       - name: Create GitHub Release
117         uses: softprops/action-gh-release@v1
118         with:
119           tag_name: ${{ github.ref }}
120           body: |
121             ### Cambios en esta versión:
122             - Mejora 1
123             - Corrección de errores
124             - Actualización de dependencias
125         env:
126           GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

```

## Disparadores:

- El workflow se ejecuta cuando se realiza un push a la rama nugget o cuando se crea un tag que comienza con 'v' (como v1.0.0).
- También se ejecuta automáticamente tras la finalización de otros workflows como "Sonar Continuous Integration", "Snyk Security", y "Semgrep Analysis".

## Build:

- Este trabajo se ejecuta en Windows y realiza la construcción del proyecto.
- Clona el código fuente.
- Configura el entorno para usar la versión 8.0 de .NET y MSBuild.
- Restaura los paquetes NuGet necesarios para el proyecto.

- Limpia los directorios de compilación (bin y obj).
- Compila el proyecto en modo Release utilizando MSBuild.
- Verifica los artefactos generados y los sube como artefactos de GitHub.

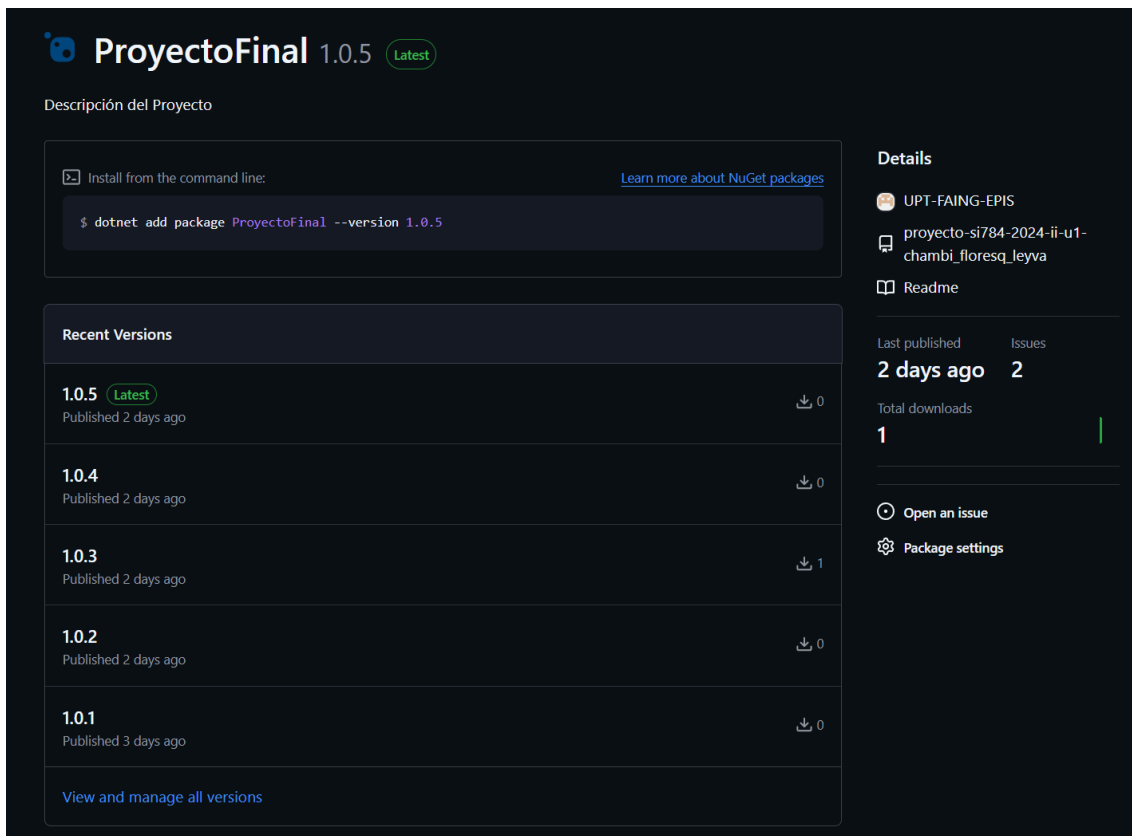
### NuGet Package:

- Se ejecuta después de que el trabajo de build haya sido exitoso.
- Clona nuevamente el código fuente.
- Configura el entorno para utilizar .NET.
- Agrega una fuente de NuGet de GitHub usando las credenciales almacenadas en los secretos (NEW\_TOKEN\_NUGET).
- Copia el archivo .nuspec de plantilla para crear el paquete NuGet.
- Calcula la siguiente versión del paquete incrementando el número de versión.
- Reemplaza la versión en el archivo .nuspec.
- Empaqueta el proyecto en un archivo .nupkg y lo publica en GitHub Packages utilizando el token de autenticación.

### Release:

- Este trabajo depende del empaquetado y publicación exitosa en NuGet.
- Se ejecuta en Ubuntu.
- Clona el código y crea una nueva release en GitHub, incluyendo una lista de cambios, basada en el tag generado.

### 4.5.2 Resultados



**ProyectoFinal** 1.0.5 Latest

Descripción del Proyecto

Install from the command line: [Learn more about NuGet packages](#)

```
$ dotnet add package ProyectoFinal --version 1.0.5
```

**Recent Versions**

Version	Published	Downloads
1.0.5 <span>Latest</span>	Published 2 days ago	0
1.0.4	Published 2 days ago	0
1.0.3	Published 2 days ago	1
1.0.2	Published 2 days ago	0
1.0.1	Published 3 days ago	0

[View and manage all versions](#)

**Details**

UPT-FAING-EPIS

projecto-si784-2024-ii-u1-chambi\_floresq\_leyva

[Readme](#)

Last published: **2 days ago** | Issues: **2**

Total downloads: **1**

[Open an issue](#)

[Package settings](#)

- El proceso de publicación ha permitido identificar y corregir problemas clave, como se refleja en los dos issues que se han abordado. El sistema de control de versiones, junto con la integración continua, ha facilitado la rápida iteración sobre el código, garantizando que cada nueva versión ofrezca mejoras tangibles tanto en rendimiento como en usabilidad.
- Además, la publicación del paquete en NuGet ha permitido su accesibilidad global, lo que asegura que otros desarrolladores puedan integrar el paquete en sus propios proyectos. A pesar de estar en las primeras etapas de difusión, el paquete ya ha sido descargado, lo que muestra que está en uso activo por parte de la comunidad o por equipos interesados en las funcionalidades que ofrece.
- A nivel de funcionalidades, se han incluido mejoras que responden a las necesidades del usuario final, tales como la optimización de la experiencia de usuario, mejoras en la gestión de dependencias y en la compatibilidad con otros paquetes. Asimismo, se ha planificado la inclusión de futuras características que potenciarán aún más el uso del paquete en entornos productivos.

## 4.6 Artifacts

### 4.6.1 Workflow

Este flujo de trabajo de GitHub Actions compila el proyecto ProyectoFinal.sln en modo Release utilizando msbuild, verifica que los artefactos de construcción se generen correctamente al listar los archivos en el directorio de salida, y finalmente, utiliza la acción upload-artifact para subir estos artefactos a GitHub, permitiendo su almacenamiento y acceso en etapas posteriores del proceso de desarrollo.

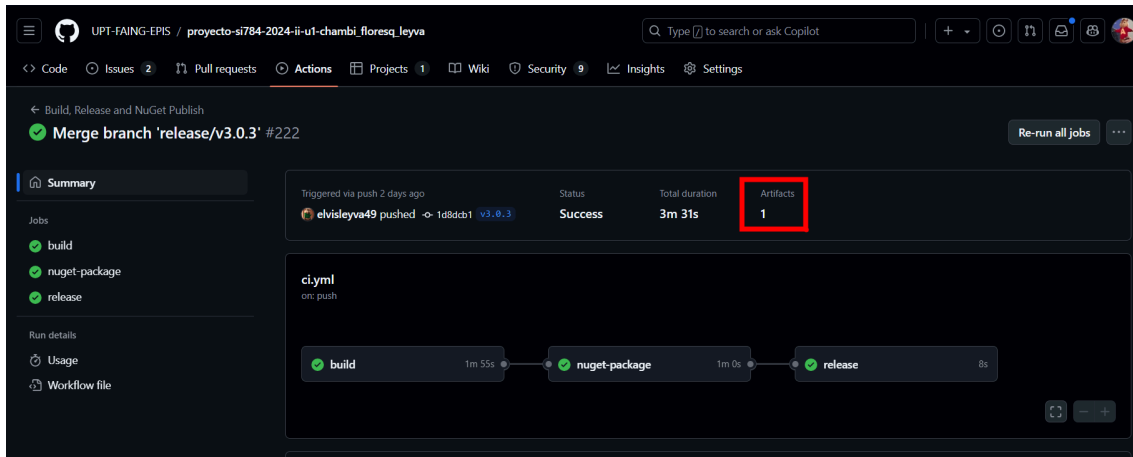
```
- name: Build Project
  run: msbuild ProyectoFinal.sln /p:Configuration=Release /p:Platform="Any CPU"

- name: Verify Build Artifacts
  run: |
    dir ${GITHUB_WORKSPACE}/ProyectoFinal/bin/Release/

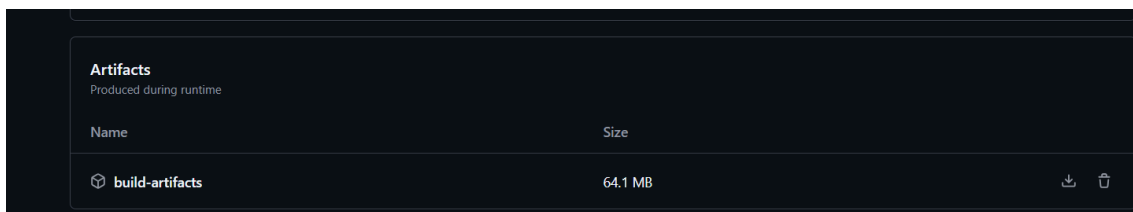
- name: Upload Artifacts
  uses: actions/upload-artifact@v3
  with:
    name: build-artifacts
    path: ${GITHUB_WORKSPACE}/ProyectoFinal/bin/Release/
```

### 4.6.2 Resultados

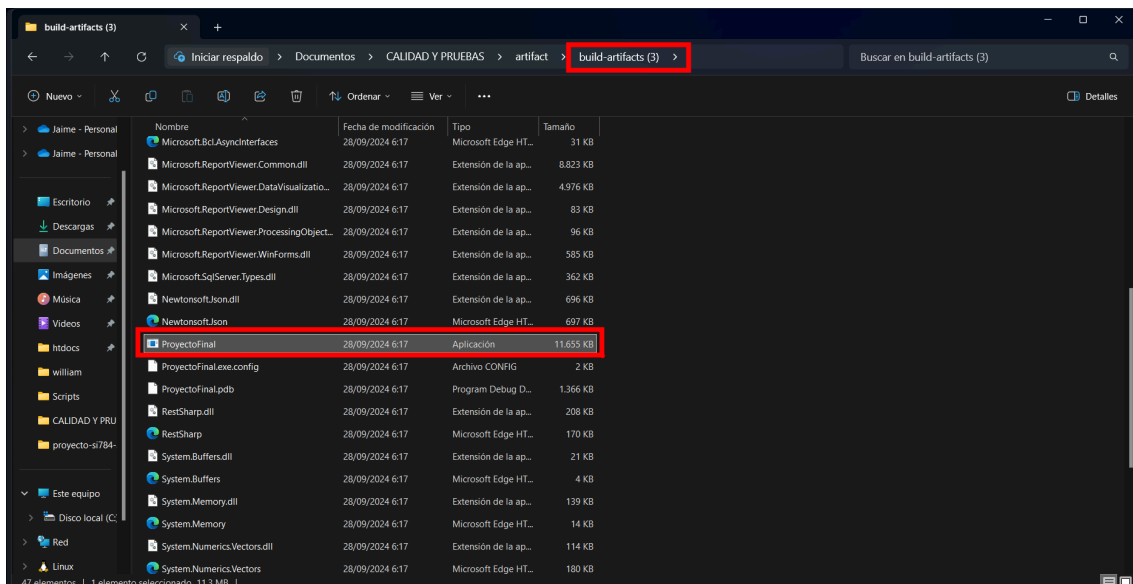
Revisamos el workflow en la sección de Actions, notaremos que aparece “1” en Artifacts, haremos click encima del número.



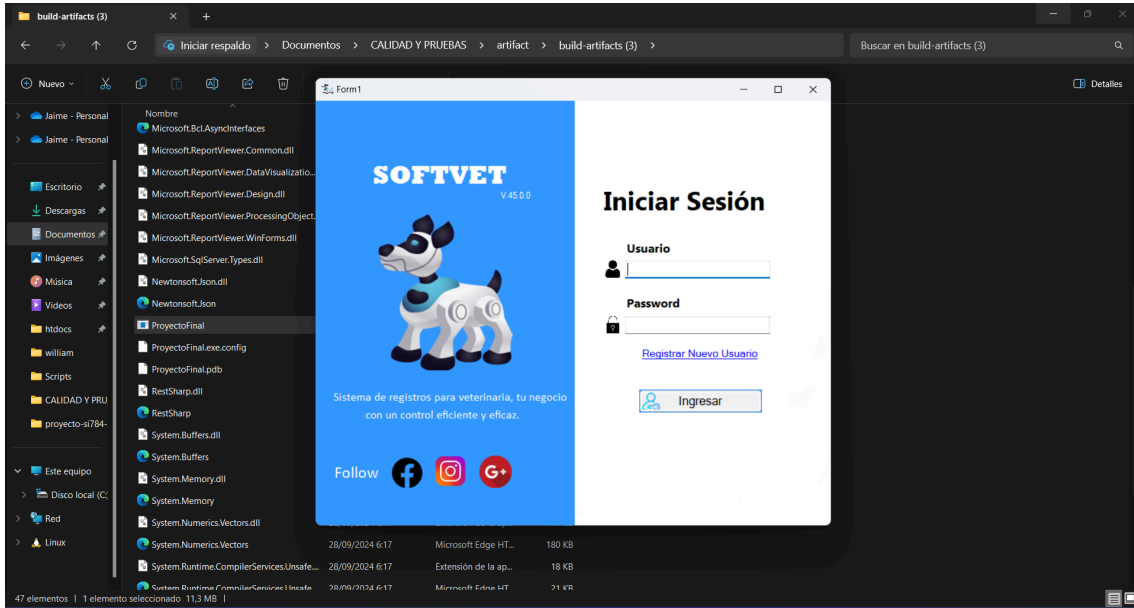
Se desplazará hacia abajo y veremos el apartado de Artifacts donde estará nuestro artefacto generado. Procedemos a descargarlo.



Ahora, en nuestro explorador de archivos local, descomprimiremos el archivo descargado e ingresando a la carpeta veremos el ejecutable de nuestro proyecto.



Abrimos el ejecutable y notaremos que efectivamente el programa está siendo ejecutado.



## 5. Conclusión

En síntesis, tras un análisis exhaustivo de la factibilidad económica, operativa, legal, social y ambiental del proyecto para la implementación del sistema en la clínica veterinaria, se concluye que este demuestra ser viable y sostenible. Desde el punto de vista técnico, se ha identificado la disponibilidad de diversas herramientas y tecnologías que facilitarán el desarrollo eficiente del sistema, adaptándose a las necesidades específicas de la clínica veterinaria.

En cuanto a la viabilidad legal y regulatoria, se garantiza el cumplimiento de normativas para la gestión segura de datos médicos de animales, mitigando posibles riesgos legales. La evaluación ambiental destaca la consideración del impacto ecológico, con prácticas que minimizan el uso de recursos y reducen residuos.

El examen financiero revela que los beneficios obtenidos superan los costos asociados, generando un retorno de inversión positivo en un período de tiempo razonable. Indicadores financieros como el Valor Actual Neto (VAN), la Relación Beneficio-Costo (B/C) y la Tasa Interna de Retorno (TIR) respaldan de manera sólida la viabilidad financiera y económica del proyecto para la clínica veterinaria. En conjunto, estos resultados confirman que el proyecto no solo es técnicamente viable, sino que también presenta un potencial rentable, representando una oportunidad sólida y estratégica de inversión en el ámbito veterinario.

Finalmente, la viabilidad social se respalda en la aceptación y disposición tanto del personal como de los clientes hacia un sistema que promete una atención más personalizada y coordinada. En conjunto, este informe de factibilidad confirma que el "Sistema Integral de Gestión de Servicios Veterinarios" no solo supera los desafíos actuales en la gestión clínica, sino que también establece las bases para adaptarse a las crecientes demandas futuras, contribuyendo significativamente a la mejora continua de las clínicas veterinarias.