DESENVOLVIMENTO DE SOFTWARE SEGURO

Project – Phase 2: Sprint 1

M1B

António Fernandes 1190402

Carla Barbosa 1200928

Carlos Rodrigues 1230172

Jorge Almeida 1222598

Nuno Figueiredo 1230202

pbs@isep.ipp.pt

25/05/2025

Index

Index 1
Table of Figures 1
Table index 1
Introduction 0
Development 1
Pipeline 1
Running pipeline 1
SNYK 12
SCA – Software Composition Analysis 14
Development Best Practices 15
Main Branches 15
Supporting Branches 15
Merge & Pull Request Policy 16
CI/CD Integration 16
Security Tests 16
Security Tests – Entity Property Restrictions 18
ASVS 18

Table of Figures

Figure 1 - Pipeline Example 2
Figure 2 - Pipeline IAST 5
Figure 3 - Pipeline DAST 8
Figure 4 - Pipeline SAST 10
Figure 5 - Sonar Tool Web Interface 11
Figure 6 - Snyk Tool Web Interface 13
Figure 7 - Snyk Tool Web Interface Vulnerabilities 13
Figure 8 - Authentication between Github and Snyk 14
Figure 9 - SCA- snyk-code.sarif report 14

Table index

Table 1 - Table Comparative 11
Table 2 - Entities and their restrictions 16

Introduction

This phase emphasizes integrating security early and continuously in the development lifecycle, leveraging automated code reviews, Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST), and Software Composition Analysis (SCA) to detect and remediate vulnerabilities effectively.

During this phase, the planned architecture and security requirements are translated into code using secure coding best practices. This includes the integration of authentication and authorization controls, secure handling of user data, and enforcement of input validation to prevent common vulnerabilities.

Security testing was performed iteratively within the DevSecOps pipeline, ensuring that every build undergoes thorough security checks before deployment.

Code reviews complement automated tools to maintain code quality and enforce secure coding standards.

Throughout this phase, the project follows the OWASP Application Security Verification Standard (ASVS) Level 2 guidelines, aiming to meet industry best practices for secure application development.

The activities carried out in Phase 2 set the foundation for a secure, resilient application ready for further deployment and operational stages.

Development

Solution was divided into 2 projects: one of them is API and other is unit tests.

API follows the DDD approach and Entity Framework based on Domain Model created last phase.

The API project was divided by application layer. Controller, Domain and Repository. The Controller layer receives HTTP requests and forwards them to the domain, returning responses. The Domain layer contains business logic and application rules, remaining independent of technical details. Repository takes care of communication with the database, being responsible for saving, seeking or updating the data.

The Unit tests project was made to test the Domain in order to validate the business logic/rules.

Pipeline

They are a set of automated processes, executed sequentially to create, test, and deploy source code, they help ensure that software is developed, tested, and released consistently and efficiently. They are a key component in the practice of DevOps, which aim to improve collaboration between development and operations teams by automating and monitoring all stages of software development, from integration to delivery continuously.

Pipelines ensure that every change to the code is automatically verified and validated, providing immediate feedback to developers and making it easier to identify and fix issues quickly.

Running pipeline

The execution of this pipeline was completed successfully, indicating that all steps were performed as expected, as shown in the following figure.

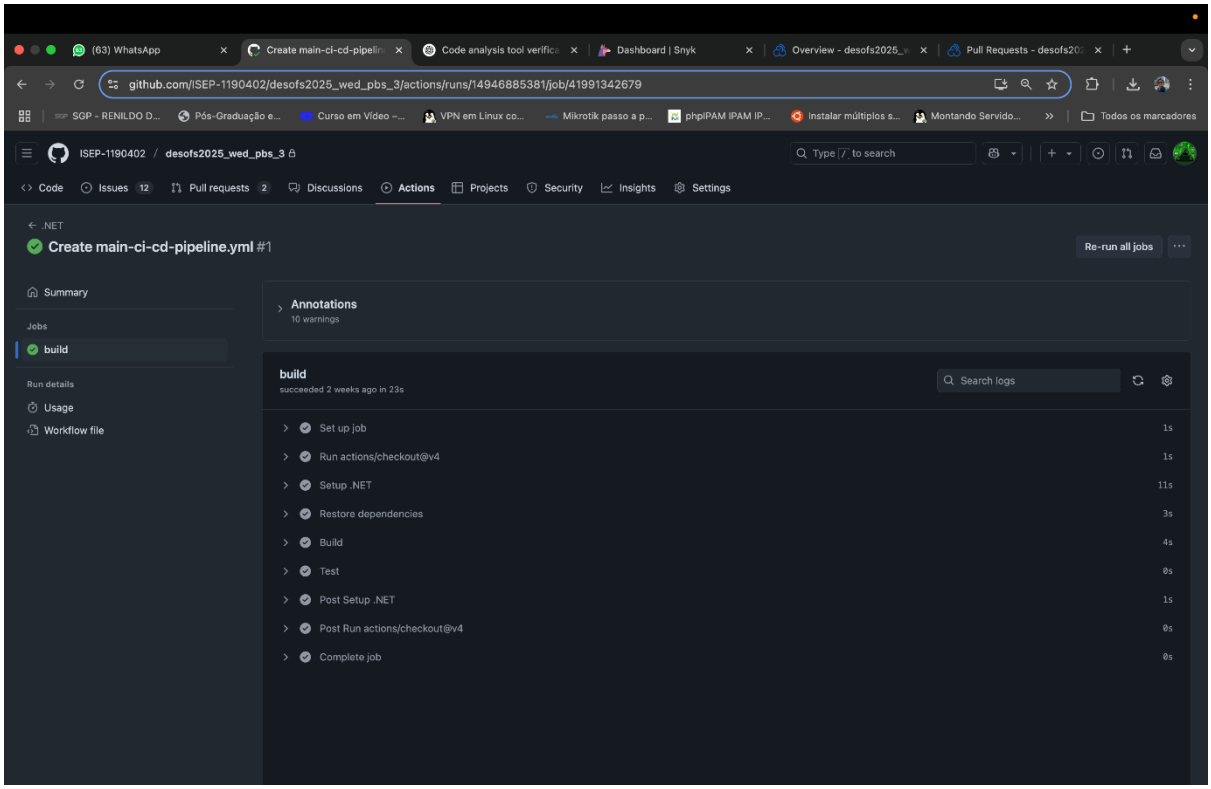

Figure 1 - Pipeline Example

IAST - Interactive Application Security Testing

It is a technology that analyzes the application while it is running (runtime), but with access to the internal code. It works interactively, monitors the behavior of the application during functional, automatic or manual tests. Combining static (code analysis) and dynamic (running tests) to offer a more accurate analysis with fewer false positives. A very important aspect is that the tools that help in this task are all paid. Below we will demonstrate an IAST made with the combination of SAST and DAST, to obtain the closest possible result.

```
name: Full Security (.NET + SAST + DAST + Simulated IAST)

on:
push:
branches: ["**"]
pull_request:
branches: ["**"]
workflow_dispatch:

env:
DOCKER_IMAGE_NAME: library-api
API_PORT: 8080

jobs:
sast-dast-iast-security:
```

```yaml
runs-on: ubuntu-latest
timeout-minutes: 25

steps:
# 1. Checkout código
- name: Checkout repository
uses: actions/checkout@v4

# 2. Setup .NET
- name: Setup .NET
uses: actions/setup-dotnet@v3
with:
dotnet-version: '8.0.x'

# 3. Restaurar dependências
- name: Restore dependencies
run: dotnet restore ./LibraryOnlineRentalSystem/LibraryOnlineRentalSystem.csproj

# 4. Instalar Snyk CLI
- name: Install Snyk CLI
run: npm install -g snyk@latest

# 5. Rodar Snyk (SAST + Deps) e exportar SARIF
- name: Run Snyk Scans
working-directory: ./LibraryOnlineRentalSystem
env:
SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
run: |
mkdir -p ./snyk_results
snyk test --file=LibraryOnlineRentalSystem.csproj --all-projects --sarif-file-output=./snyk_results/snyk-deps.sarif ||
true
snyk code test --severity-threshold=high --sarif-file-output=./snyk_results/snyk-code.sarif || true
ls -la ./snyk_results/

- name: Upload Snyk SARIF
uses: actions/upload-artifact@v4
with:
name: snyk-sarif-results
path: |
./LibraryOnlineRentalSystem/snyk_results/snyk-deps.sarif
./LibraryOnlineRentalSystem/snyk_results/snyk-code.sarif

# 5. Build imagem Docker
- name: Build Docker Image
working-directory: ./LibraryOnlineRentalSystem
run: docker build -t $DOCKER_IMAGE_NAME .

- name: Create Docker Network
run: docker network create zap-net
```

```
# 6. Prepara API no container
- name: Run API Container
run: |
docker run -d \
--name api \
--network zap-net \
-p $API_PORT:$API_PORT \
$DOCKER_IMAGE_NAME
sleep 10

# 7. ZAP Scan
- name: Run OWASP ZAP Scan
run: |
API_IP=$(docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' api)
echo "API container IP: $API_IP"
mkdir -p $GITHUB_WORKSPACE/zap-work
chmod -R 777 $GITHUB_WORKSPACE/zap-work
docker run --rm \
--network zap-net \
-v $GITHUB_WORKSPACE/zap-work:/zap/wrk:rw \
ghcr.io/zaproxy/zaproxy:stable \
zap-baseline.py \
-t http://$API_IP:${{ env.API_PORT }} \
-J report_json.json \
-w report_md.md \
-r report_html.html \
-I

# 8. Upload dos arquivos gerados Owasp
- name: Upload ZAP Reports
uses: actions/upload-artifact@v4
with:
name: zap-reports_iast
path: zap-work/
# 9. Stop Container
- name: Stop API Container
if: always()
run: docker stop api && docker rm api
```

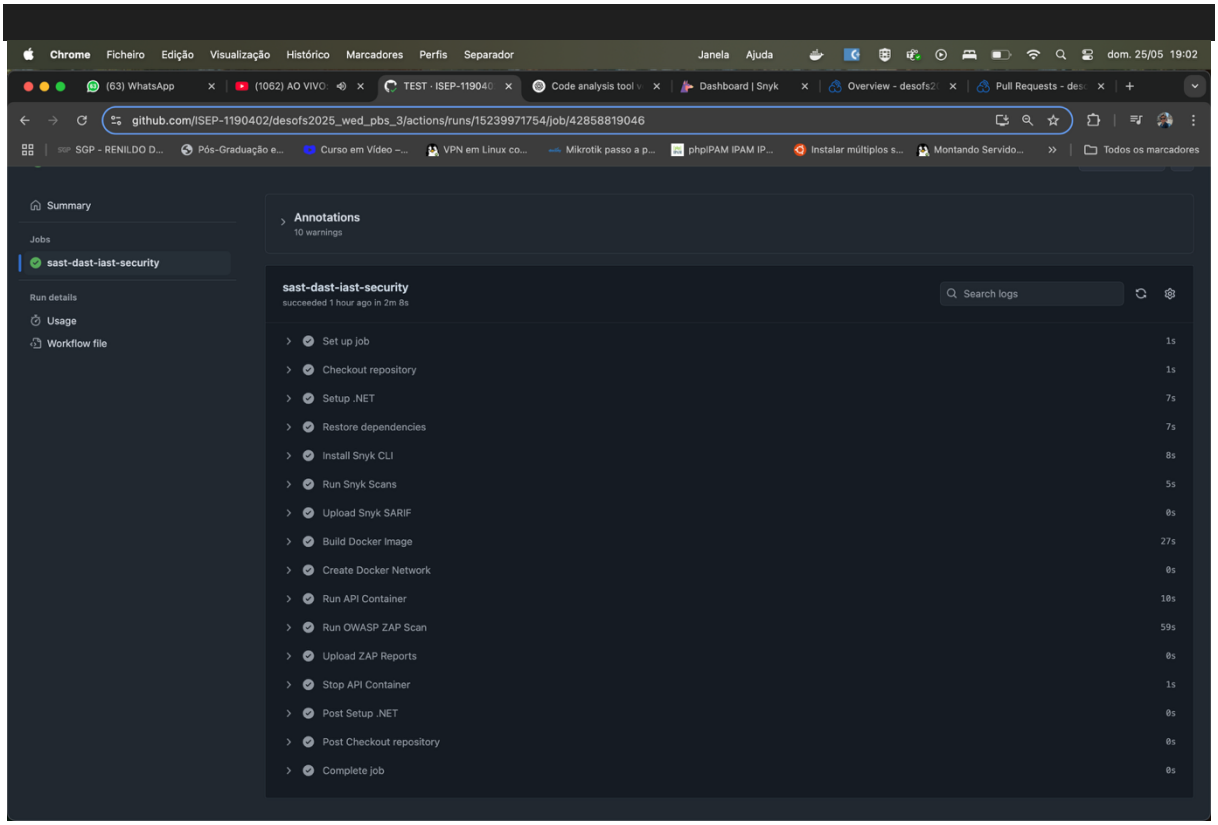

Figure 2 - Pipeline IAST

DAST - Dynamic Application Security Testing

It is a way to test the security of a running application, as if a hacker were interacting with it from the outside – without access to the source code. It is known with a black box approach. The DAST tools send http requests, in this code we use Sonar to help us in this context, since it is a free service, different from the tools that help in IAST.

DAST detects the following threats:

- SQL Injection
- XSS
- Open Redirects
- Authentication failures.

```
name: .NET API Security Scan

on:
push:
branches: ["**"] # Executa em todas as branches
pull_request:
branches: ["**"] # Executa em todas as branches
workflow_dispatch:
```

```
env:
DOCKER_IMAGE_NAME: library-api
API_PORT: 8080

jobs:
security-scans_DAST:
runs-on: ubuntu-latest
timeout-minutes: 20

steps:
# 1. Checkout (usando versão explícita)
- name: Checkout
uses: actions/checkout@v4.1.1 # Versão fixa

# 2. Setup .NET
- name: Setup .NET
uses: actions/setup-dotnet@v3 # Versão fixa
with:
dotnet-version: 8.0.x

# 3. Debug: Verificação completa
- name: Verify files
run: |
echo "Conteúdo do diretório raiz:"
ls -la
echo "Conteúdo do diretório do projeto:"
ls -la ${{ env.PROJECT_DIR }}
echo "Verificando Dockerfile:"
ls -la ${{ env.PROJECT_DIR }}/Dockerfile || echo "Dockerfile não encontrado"
cat ${{ env.PROJECT_DIR }}/Dockerfile || true
# 4. Build Docker (solução definitiva)
- name: Build Docker Image
working-directory: ./LibraryOnlineRentalSystem
run: |
echo "Building Docker image from $(pwd)"
docker build -t $DOCKER_IMAGE_NAME .

- name: Create Docker Network
run: docker network create zap-net

# 5. Prepara API no container
- name: Run API Container
run: |
docker run -d \
--name api \
--network zap-net \
-p $API_PORT:$API_PORT \
$DOCKER_IMAGE_NAME
```

```
sleep 10
# 6. Preparacao Sonar
- name: Install SonarScanner for .NET
run: dotnet tool install --global dotnet-sonarscanner

- name: Start SonarCloud Analysis
run: |
dotnet sonarscanner begin \
/k:"jorgealmeidadeveloper_LibraryOnlineRentalSystem" \
/o:"jorgealmeidadeveloper" \
/d:sonar.login="${{ secrets.SONAR_TOKEN }}" \
/d:sonar.host.url="https://sonarcloud.io" \
/d:sonar.dotnet.buildConfiguration="Release"
# 6. OWASP ZAP Scan
- name: Run OWASP ZAP Scan manually
run: |
# Get API container IP address in zap-net
API_IP=$(docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' api)
echo "API container IP: $API_IP"

# Prepare writable directory for ZAP reports
mkdir -p $GITHUB_WORKSPACE/zap-work
chmod -R 777 $GITHUB_WORKSPACE/zap-work

# Run ZAP docker container with scan targeting API IP, saving reports to current directory
docker run --rm \
--network zap-net \
-v $GITHUB_WORKSPACE/zap-work:/zap/wrk:rw \
ghcr.io/zaproxy/zaproxy:stable \
zap-baseline.py \
-t http://$API_IP:${{ env.API_PORT }} \
-J report_json.json \
-w report_md.md \
-r report_html.html \
-I

- name: List ZAP report files
run: ls -l $GITHUB_WORKSPACE/zap-work



# 7. Upload do relatório (usando v4, a mais recente)
- name: Upload ZAP reports
uses: actions/upload-artifact@v4
with:
name: zap-reports
path: zap-work/

# 8. Parar container
- name: Stop Container
```

```
if: always()
run: docker stop api && docker rm api
```

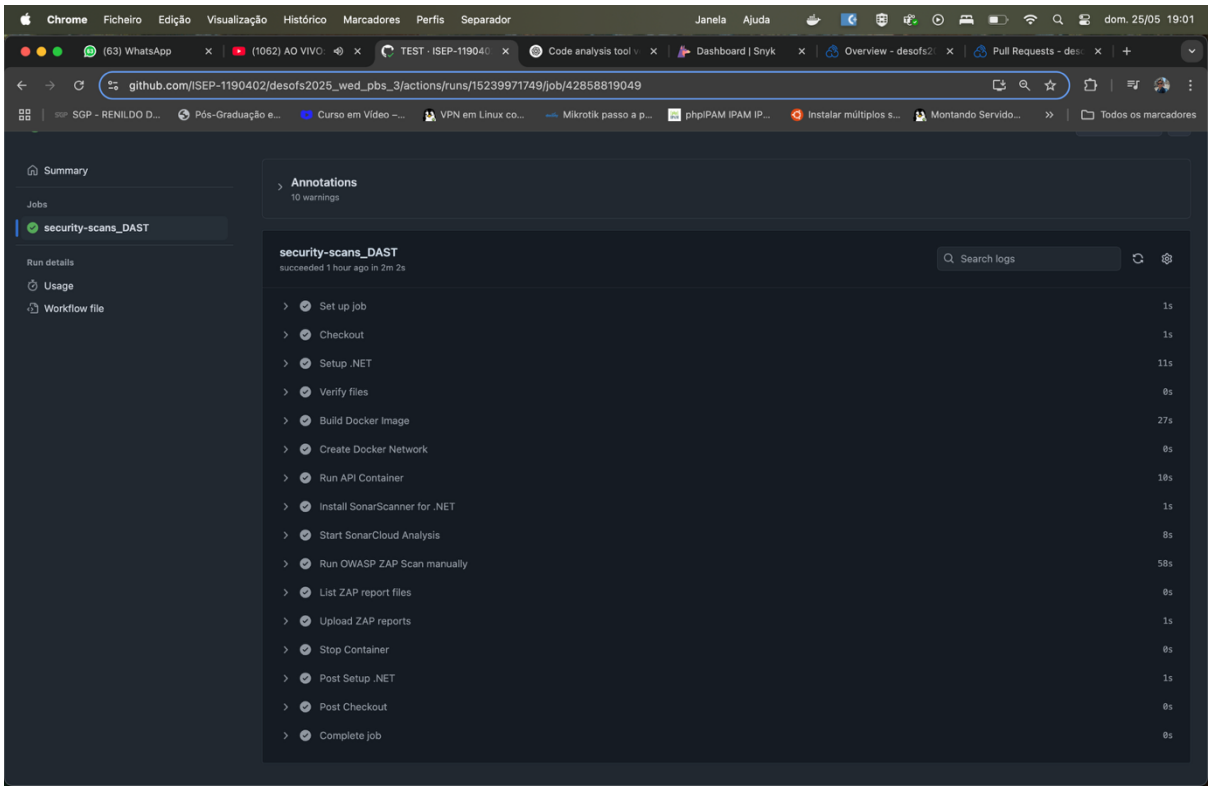

Figure 3 - Pipeline DAST

SAST – Static Application Security Testing

It is the static analysis of the source code, without running the application. It is known as the White Box. Because the tool reads the application's source code, bytecode or binaries. In addition to looking for known but practical vulnerabilities, logic flaws and security breaches before the code runs.

It detects:

- Flow control or logic errors
- Sql Injection
- Hardcoded Secrets
- Violation of good practices

```
name: Snyk Security Scan
```

```yaml
on:
push:
branches: ["**"]
pull_request:
branches: ["**"]

jobs:
security-scan:
name: Run Snyk Security Scan
runs-on: ubuntu-latest
permissions:
contents: read
# Se não for mais fazer upload para Code Scanning, pode remover esta linha:

# 1. Checkout código
steps:
- name: Checkout repository
uses: actions/checkout@v4
with:
path: ./src
# 2. Verificar Estrutura do Projeto
- name: Verify project structure
working-directory: ./src
run: |
echo "Verifying project structure"
ls -la
test -f "LibraryOnlineRentalSystem/LibraryOnlineRentalSystem.csproj" || {
echo "::error::Project file not found"
exit 1
}
# 3. Setup .NET
- name: Setup .NET
uses: actions/setup-dotnet@v3
with:
dotnet-version: '8.0.x'
# 3. Restaurar dependencias
- name: Restore dependencies
working-directory: ./src/LibraryOnlineRentalSystem
run: dotnet restore

# 4. Instalação Snuk
- name: Install Snyk CLI
run: npm install -g snyk@latest

# 5. Scanear o Snyk
- name: Run Snyk Scans
working-directory: ./src/LibraryOnlineRentalSystem
env:
SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
run: |
```

```
mkdir -p ./snyk_results
snyk test --file=LibraryOnlineRentalSystem.csproj --all-projects --sarif-file-output=./snyk_results/snyk-deps.sarif ||
true
snyk code test --severity-threshold=high --sarif-file-output=./snyk_results/snyk-code.sarif || true
ls -la ./snyk_results/

# 6. Salvar Relatorio
- name: Save SARIF results as artifact
uses: actions/upload-artifact@v4
with:
name: snyk-sarif-results
path: |
./src/LibraryOnlineRentalSystem/snyk_results/snyk-deps.sarif
./src/LibraryOnlineRentalSystem/snyk_results/snyk-code.sarif

# 7. Sarif dos arquivos
- name: Debug SARIF files
run: |
echo "Checking SARIF files..."
find ./src -name "*.sarif" -exec ls -la {} \;
find ./src -name "*.sarif" -exec head -n 5 {} \;
if: always()
```

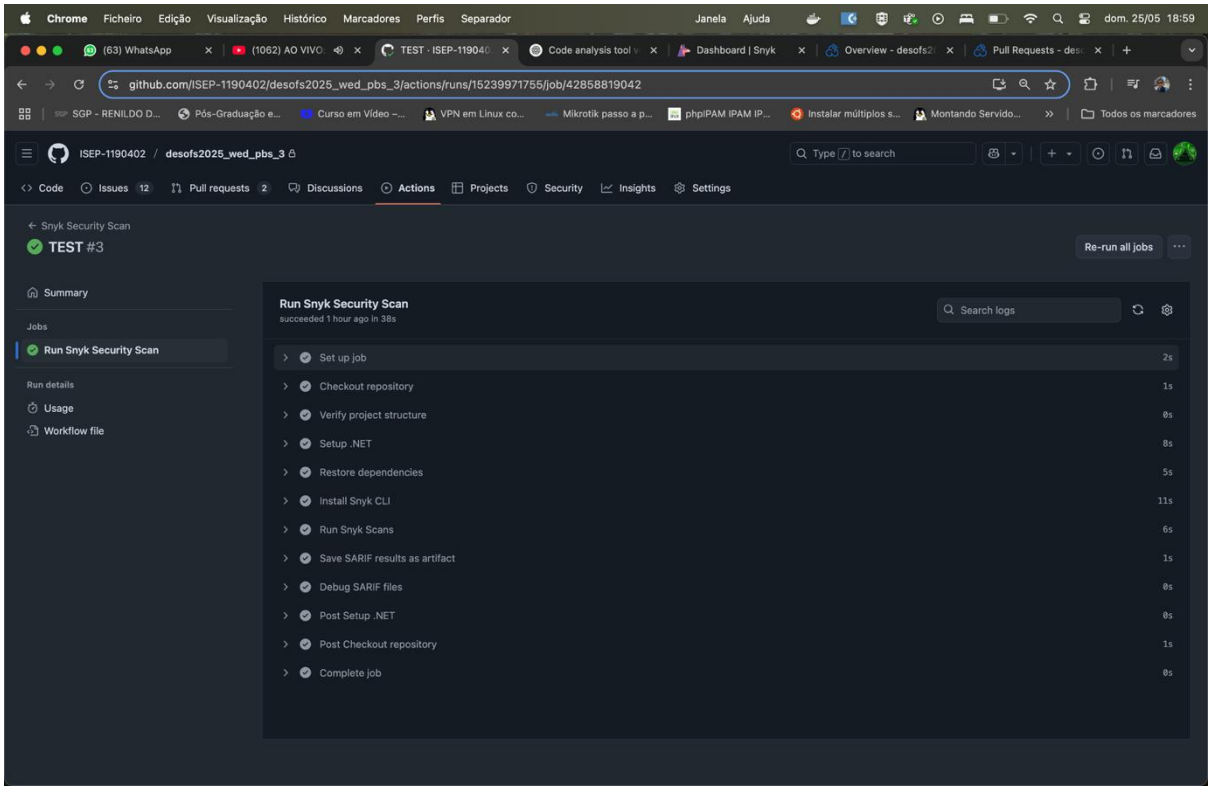

Figure 4 - Pipeline SAST

Table Comparative

Table 1 - Table Comparative

Technique	Source code	App execution	Best for detecting
SAST	✓	✗	Vulnerabilities in code, prior to deployment
DAST	✗	✓	Runtime failures, externally visible
IAST	✓	✓	Vulnerabilities while the app runs with access to code

SonarQube

It is a form of continuous code quality analysis that helps development teams detect, monitor, and fix problems in the source code. Therefore, it is widely used for:

- Identify security vulnerabilities (SAST)
- Detect bugs and code smells (bad code practices)
- Ensure quality standards and testing coverage.
- Monitor the health of the code over time.

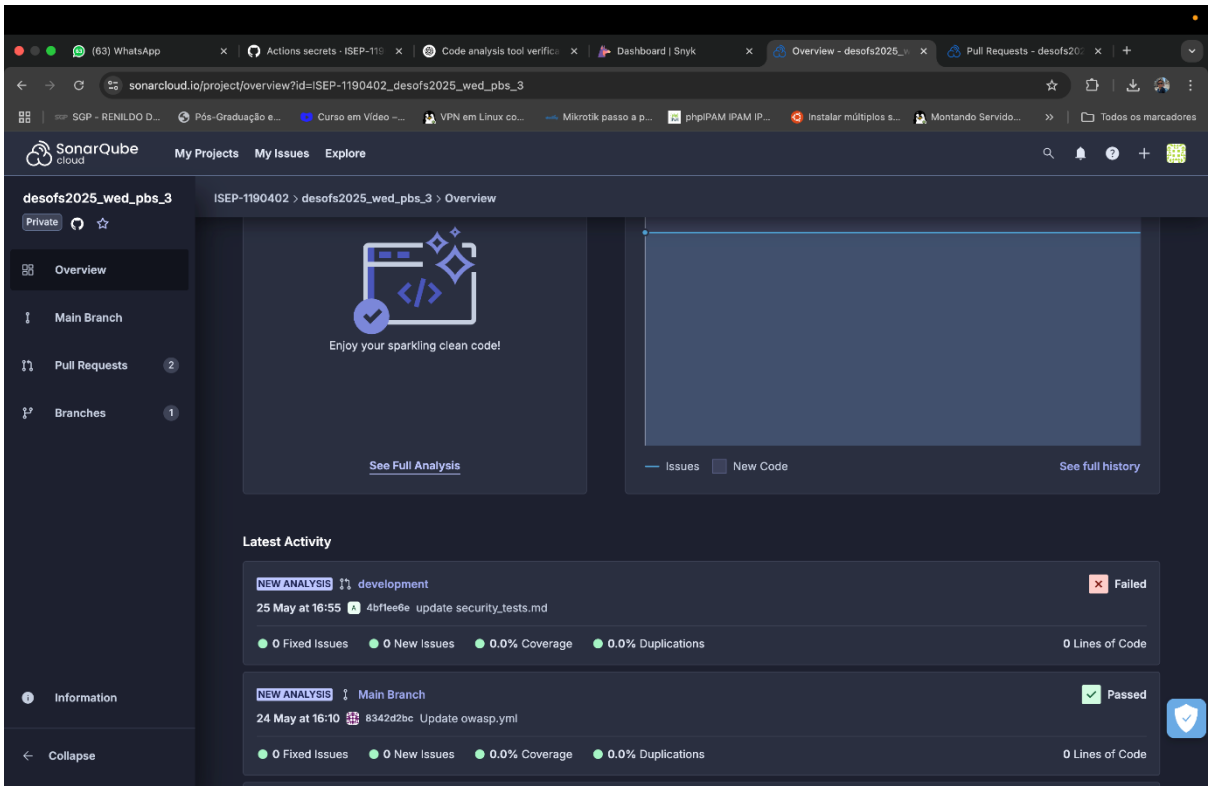

Figure 5 - Sonar Tool Web Interface

SNYK

Snyk is a widely used security tool to identify and fix vulnerabilities in dependencies, source code, containers, and infrastructure-as-code. It is integrated into our pipelines for automated, continuous security analysis.

- Snyk Features: Vulnerability Identification: Analyzes code and dependencies for known vulnerabilities.
- Remediation Suggestions: Provides recommendations and patches to address vulnerabilities found.
- Continuous Monitoring: Continuously monitors projects for new vulnerabilities that may be introduced.
- Detailed Reports: Generates detailed reports on the security status of the project.

Snyk also provides a web interface, accessible through its website, where users can view the security status of their projects, consult vulnerability reports, manage the tool's configurations and integrations, and access documentation and technical support.

This interface facilitates the management and monitoring of project security, providing a clearer and more detailed view of vulnerabilities and the actions needed to mitigate them.

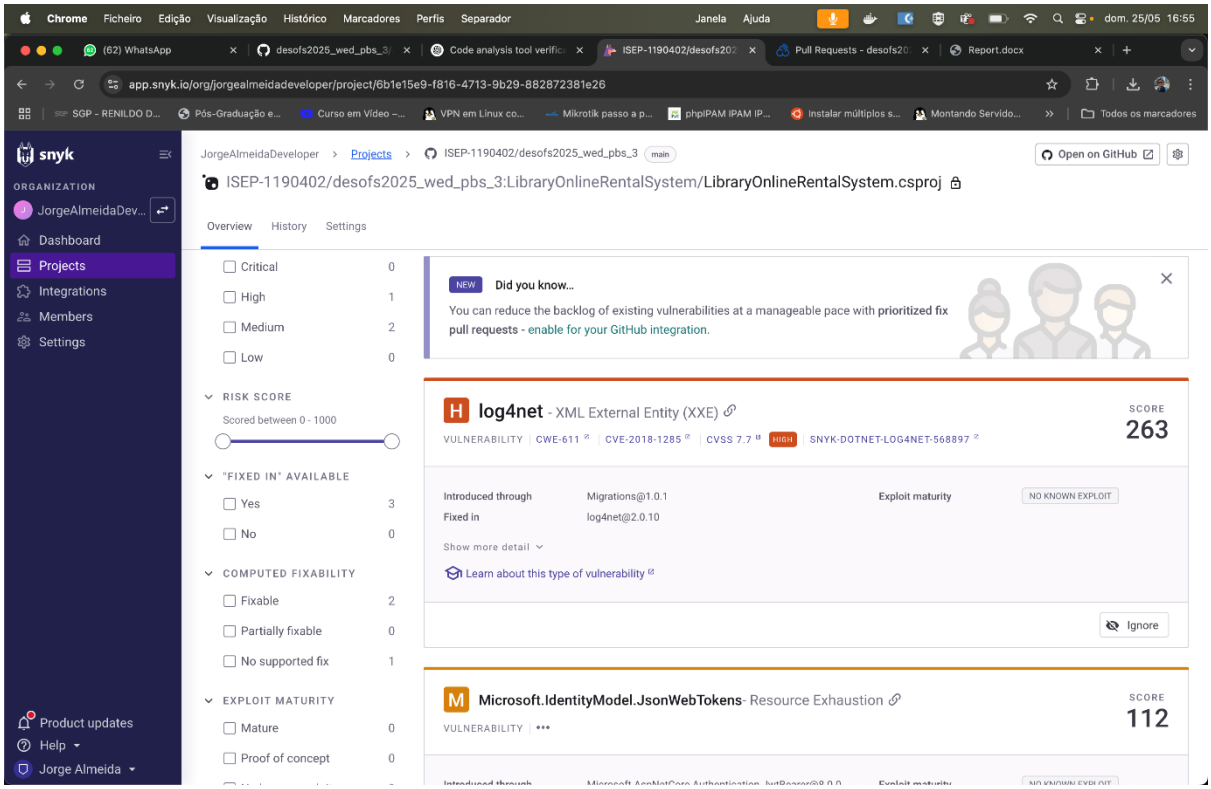

Figure 6 - Snyk Tool Web Interface

Identify vulnerabilities in code, here introduces log4net@2.0.5 .

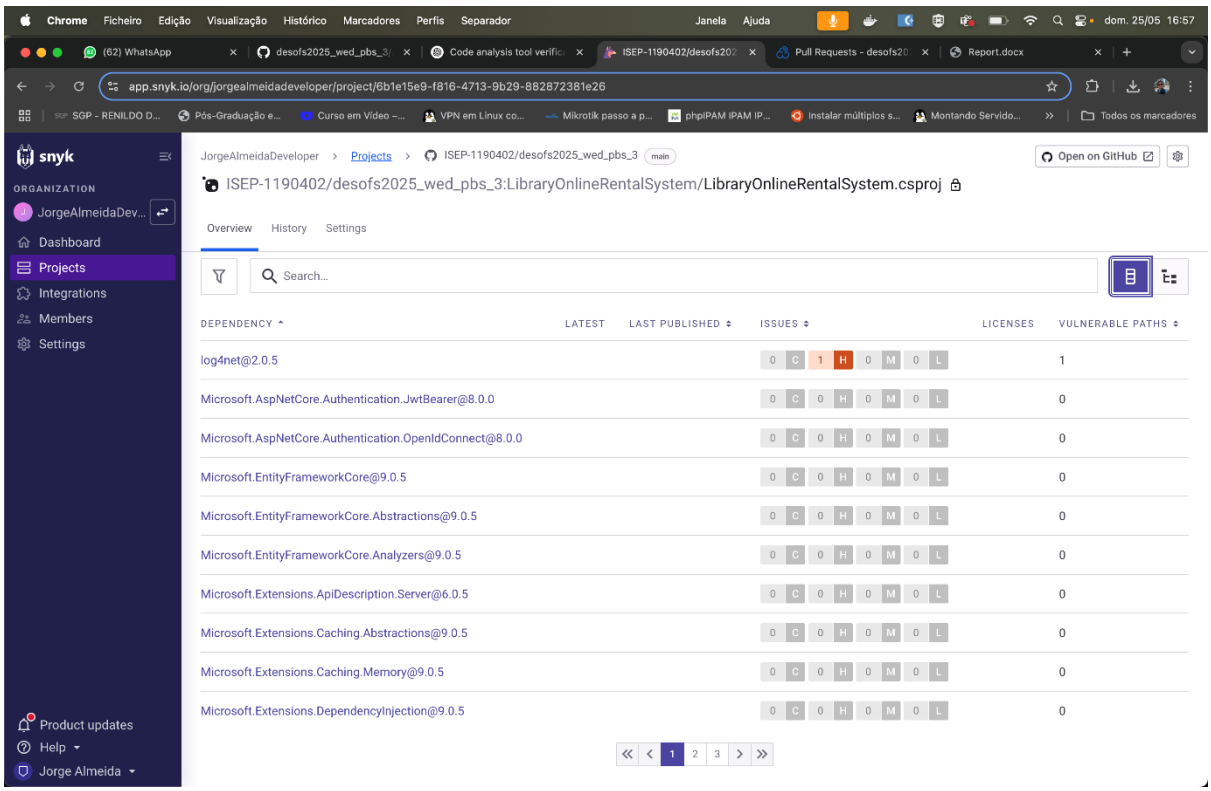

Figure 7 - Snyk Tool Web Interface Vulnerabilities

The way Git Hub and Snyk communicates through the token. We've added this feature inside Settings in git and added the information inside Secrets, with the KEY key when registration is complete. No user and password intervention is required.

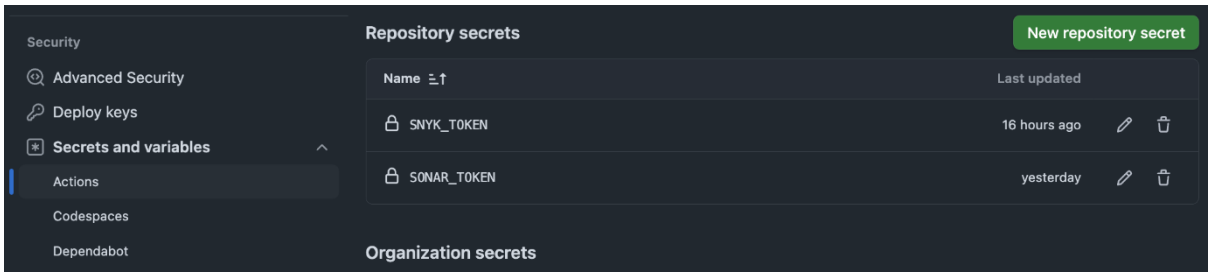

Figure 8 - Authentication between Github and Snyk

SCA – Software Composition Analysis

The Software Composition Analysis (SCA) was performed using the Snyk CLI, which conducted both dependency scanning and static application security testing on the project's codebase.


```
{
  "$schema": "https://raw.githubusercontent.com/oasis-tcs/sarif-spec/master/Schemata/sarif-schema-2.1.0.json",
  "version": "2.1.0",
  "runs": [
    {
      "tool": {
        "driver": {
          "name": "SnykCode",
          "semanticVersion": "1.0.0",
          "version": "1.0.0",
          "rules": []
        }
      },
      "results": [],
      "properties": {
        "coverage": [
          {
            "isSupported": true,
            "lang": "C#",
            "files": 46,
            "type": "SUPPORTED"
          },
          {
            "isSupported": true,
            "lang": "XML",
            "files": 5,
            "type": "SUPPORTED"
          }
        ]
      }
    }
  ]
}
```


Figure 9 - SCA- snyk-code.sarif report

The scan covered all relevant project dependencies and source code files, including 46 C# files and 5 XML files.

Snyk's dependency scanning analyzed third-party libraries for known vulnerabilities, while the static code analysis (SAST) assessed the source code to detect potential security flaws.

According to the SARIF report generated from the static analysis, no issues were detected ("results": []) within the scanned codebase at the time of the scan.

Overall, the findings suggest that the project currently does not contain any known critical security vulnerabilities based on the Snyk scans performed.

The SCA process was automated through GitHub Actions, and the SARIF reports generated were uploaded as artifacts, enabling further inspection and integration with GitHub's code scanning tools.

Secure scanning was ensured using authentication tokens and environment variables configured in the workflow.

In addition to the automated scans performed via GitHub Actions, the project's dependencies and vulnerabilities are continuously monitored on Snyk's platform. Although detailed scan history and vulnerability information are private, ongoing monitoring helps maintain security throughout the project lifecycle.

The pipeline configuration used for this process can be reviewed in the project repository here.

Development Best Practices

To ensure high code quality, team collaboration, and controlled release management, we follow industry-standard Git branching and merge practices:

Main Branches

- ***Main***: Represents the production-ready code. Only thoroughly reviewed, tested, and approved changes are merged here.
- ***development*:** Integrates features before they are released to production. Acts as a staging branch.

Supporting Branches

- **Feature branches** (example: **feature/rent-book**): Used for developing new features. Branches from development. Merged back via Pull Requests (PRs).

- **Bugfix branches** (bugfix/issue-xxxx-fix-auth): Short-lived branches for fixing non-critical issues. Merge via PRs to **development** branch.

Merge & Pull Request Policy

- All merges are done via **Pull Requests** (PRs) with mandatory **code review**.
- PRs must pass CI checks (unit tests, build success, security tests) before merging.

CI/CD Integration

- Branches trigger automatic builds and test runs via CI pipelines
- **main** pushes trigger deployment to production (not yet implemented)
- **development** pushes deploy to staging environments (not yet implemented)

Security Tests

Restrictions made for validation.

Table 2 - Entities and their restrictions

Entity	Property	Restriction
User	Username	Is required, Max Length 30 characters and minimum length 4 characters, Allowed Characters: Must contain at least one letter Can contain letters, numbers, and underscores, Must be unique across all users
	NIF	Is required, Must contain exactly 9 digits, No letters or special characters allowed, Leading/trailing whitespace is trimmed
	Name	Is required, Max Length 40 characters, Cannot be null or empty, Cannot contain digits, Cannot contain special characters or symbols,

		Cannot contain punctuation, Leading/trailing whitespace is automatically trimmed
	PhoneNumber	Is required, Must contain exactly 9 digits, Digits only, no spaces, dashes, or other characters allowed, Leading/trailing whitespace is trimmed
	Biography	Not required: can be empty, Max Length 150 characters, Cannot contain special characters (Allowed Characters: Letters, Numbers, Spaces), Leading/trailing whitespace is trimmed
	UserID	Cannot be null or empty
	Role (by Keycloak)	Is Required, Cannot be null or empty, Automatically given the USER role
	Email	Is required, Must have a valid email address format (@, second-level domain and top-level domain): Must match the pattern: ^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$ Cannot contain "..", Must contain exactly one "@" symbol, Leading/trailing whitespace is trimmed, Must be unique across all users
	Password (by Keycloak)	Is required, Max Length 128 digits and Min Length 12 digits,

		Cannot have emojis and spaces, Must have both upper and lower cases, special characters and numbers
Book	AmountOfCopies	Must be greater than or equal to 0
	Author	Cannot be null or empty
	BookID	Cannot be null or empty
	Category	Cannot be null or empty
	Description	Cannot have more than 1000 characters
	ISBN	Must comply with the rules defined for the ISBN
	Publisher	Cannot have more than 50 characters

Security Tests – Entity Property Restrictions

To view the table that outlines the security validations enforced for each business entity in the Library Online Rental System, access here.

ASVS

This file can be found in the Deliverables folder for this Phase 2 – Sprint 1 in the repository. Able to access here.

We have some screenshots of proof of implementation/configuration in the ASVS folder in Deliverables folder for this Phase 2 – Sprint 1 in the repository (.\desofs2025_wed_pbs_3\Deliverables\Phase 2 - Sprint 1\ASVS proof).