Proyecto Monis-Torías – Informe Final DYAS

Proyecto: Sistema de Gestión de Tutorías Académicas (Monis-Torías)

Curso: Diseño y Arquitectura de Software

Programa: Ingeniería de Sistemas – Universidad de La Sabana

Equipo de trabajo: - Santiago Urrego Rodríguez - Luis Mario Ramírez Muñoz - Santiago Gutiérrez de Piñeres Barbosa

Fecha: 22 de noviembre de 2025

---

1. Resumen ejecutivo

El sistema Monis-Torías es una plataforma web para gestionar tutorías académicas que conecta estudiantes con tutores, permite agendar y hacer seguimiento de sesiones, administrar pagos y brindar visibilidad a coordinadores académicos.

El proyecto adopta una arquitectura limpia con separación clara entre frontend (React + TypeScript + Vite) y backend (Node.js + Express desplegado como Firebase Functions), con Firestore como base de datos, Firebase Auth como proveedor de identidad y un pipeline de CI/CD completo en GitHub Actions.

Desde la perspectiva de diseño y arquitectura de software, el sistema incluye:

- Un modelo arquitectónico documentado con **4+1 vistas** y **modelo C4**, con sus diagramas correspondientes.
- Cobertura de pruebas elevada en backend (98.7%) y un análisis de calidad en SonarQube superior al umbral exigido por DYAS (>80%).
- Integración de prácticas DevSecOps: SAST (SonarQube), DAST (OWASP ZAP), análisis de dependencias, escaneo de contenedores y búsqueda de secretos.
- Contenerización de componentes clave con Docker, orquestación de referencia en Kubernetes y despliegue productivo en Firebase.

---

3. Contexto y alcance del sistema

3.1 Problema de negocio

En la universidad, la gestión de tutorías académicas suele depender de procesos manuales (formularios, correos, hojas de cálculo) que presentan problemas recurrentes:

- Dificultad para encontrar tutor disponible en el horario deseado.
- Falta de trazabilidad de las sesiones realizadas.
- Ausencia de métricas consolidadas de uso y desempeño de los tutores.
- Procesos confusos para el pago a tutores o monitores.

Monis-Torías aborda estos problemas proporcionando una aplicación web centralizada donde estudiantes, tutores y coordinadores pueden interactuar de forma segura y trazable.

3.2 Objetivos del sistema

- Permitir a los estudiantes **buscar tutores**, solicitar sesiones y hacer seguimiento de su estado.
- Permitir a los tutores **gestionar su agenda** de tutorías, aceptar o rechazar solicitudes y registrar la realización de la sesión.
- Proveer a los coordinadores una vista de **gestión y monitoreo**, con métricas básicas (número de sesiones, tutores activos, etc.).
- Integrar mecanismos de autenticación, autorización por roles y trazabilidad de operaciones.
- Demostrar un diseño arquitectónico sólido.

3.3 Actores principales

- **Estudiante**: consulta la oferta de tutorías, solicita y asiste a sesiones.
- **Tutor**: administra su disponibilidad y atiende las solicitudes de estudiantes.
- **Administrador/Coordinador**: configura reglas, administra usuarios especiales y revisa métricas.

---

4. Arquitectura del sistema

Esta sección resume la arquitectura utilizando los dos marcos de referencia exigidos: **modelo 4+1** y **modelo C4**. Los diagramas detallados se encuentran en los archivos `.puml` del repositorio y en las imágenes incluidas en `informe/diagramasPNG`.

4.1 Vista lógica (4+1)

La vista lógica describe los principales componentes de software y sus relaciones. En Monis-Torías se distinguen claramente los módulos de frontend y backend.

Diagrama de vista lógica:

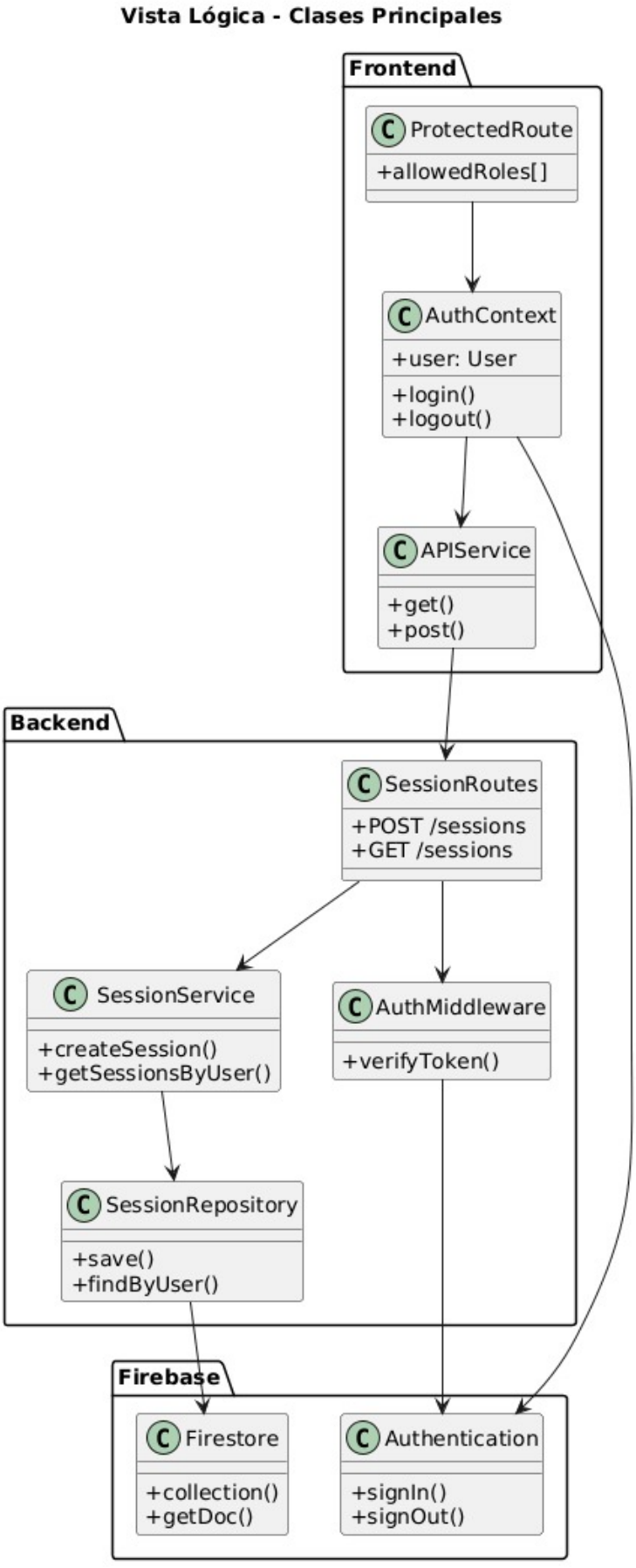


Figure 1: Vista lógica 4+1

A alto nivel, en el **backend** se tienen:

- **Rutas (API)**: agrupan los endpoints HTTP relacionados con sesiones, pagos, tutores, usuarios y materiales.
- **Middlewares**: responsabilidades transversales como autenticación, autorización basada en roles y manejo de errores.
- **Servicios (Services)**: encapsulan la lógica de negocio (p.ej. creación de sesión, confirmación, cálculo de pagos, validaciones).
- **Repositorios (Repos)**: acceso a datos en Firestore y a otros servicios de Firebase.

En el **frontend** se definen:

- **Pages**: pantallas de alto nivel (login, dashboard, listado de sesiones, etc.).
- **Components**: componentes reutilizables (formularios, tablas, modales, etc.).
- **Context/Hooks**: manejo de estado global (autenticación, usuario actual) y lógica compartida.
- **Services (API client)**: funciones que encapsulan las llamadas HTTP al backend.

Este diseño facilita la aplicación de principios SOLID y la posibilidad de probar de manera aislada servicios y componentes.

4.2 Vista de procesos (4+1)

La vista de procesos representa los flujos dinámicos más importantes, incluyendo concurrencia y comunicación entre procesos.

Diagrama de procesos – flujo de solicitud de sesión:

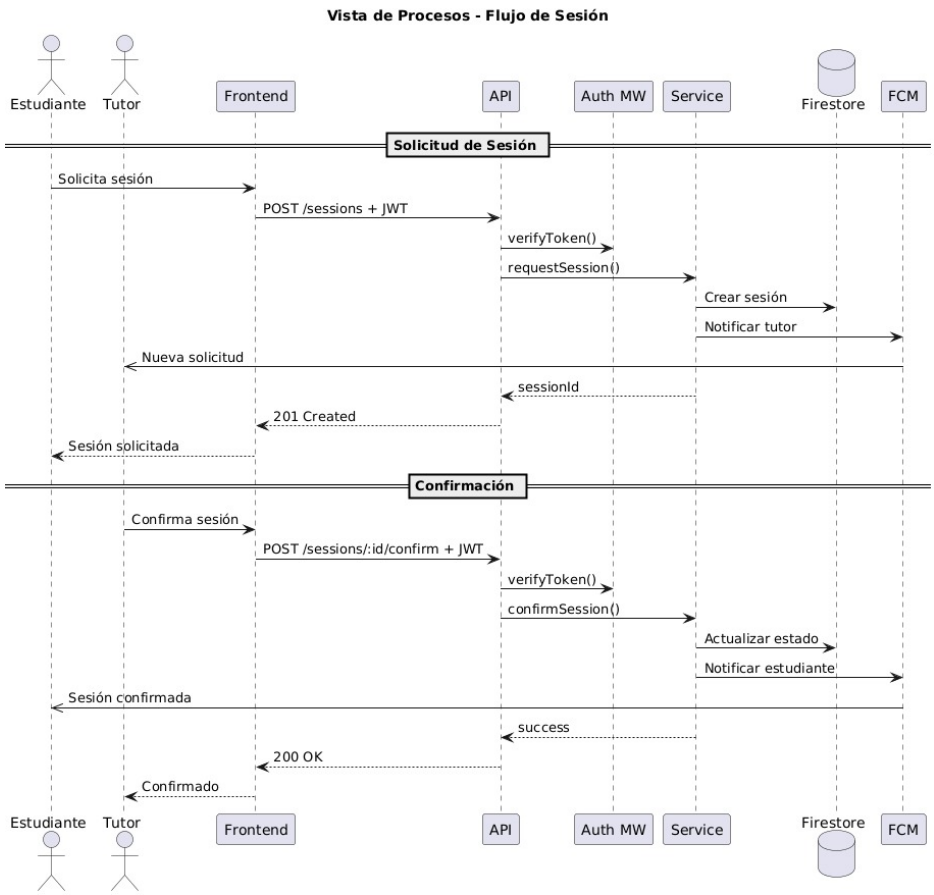


Figure 2: Vista de procesos 4+1

Flujo típico:

1. El **estudiante** inicia sesión (Firebase Auth) y navega a la pantalla de búsqueda de tutores.
2. Selecciona tutor, horario y envía la **solicitud de sesión**.
3. El frontend invoca el endpoint correspondiente del **API de sesiones**.
4. El backend valida datos (Zod), verifica permisos (middlewares), registra la solicitud en Firestore y deja la sesión en estado "pendiente".
5. El **tutor** recibe la notificación y, desde su interfaz, acepta o rechaza la solicitud.
6. El backend actualiza el estado a "confirmada" o "rechazada" y registra los cambios.

Otro flujo clave (no dibujado aquí en detalle) es el de **registro de finalización de sesión y cálculo de pagos**, donde se actualizan métricas y se registran montos devengados por el tutor.

4.3 Vista de desarrollo (4+1)

La vista de desarrollo (o implementación) muestra la organización del software en módulos y paquetes.

Diagrama de vista de desarrollo:

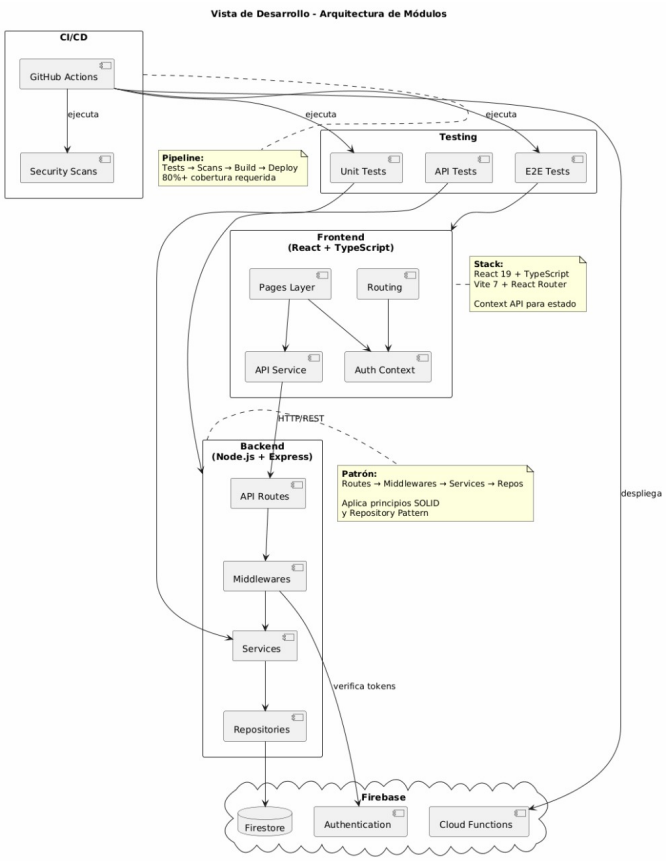


Figure 3: Vista de desarrollo 4+1

En el repositorio monorepo se distinguen dos carpetas principales:

- `frontend/`: proyecto React + TypeScript con estructura por páginas, componentes, hooks, contextos y servicios.
- `functions/`: backend Node.js sobre Firebase Functions, con carpetas para `api/`, `services/`, `repos/`, `middlewares/` y `test/`.

A esto se suman carpetas de apoyo como `docs/`, `k8s/`, `security/`, `allure/` y `informe/` que almacenan la documentación, manifestos de Kubernetes, configuración de seguridad y reportes de pruebas.

4.4 Vista física (4+1)

La vista física describe el despliegue de componentes de software en la infraestructura.

Diagrama de vista física:

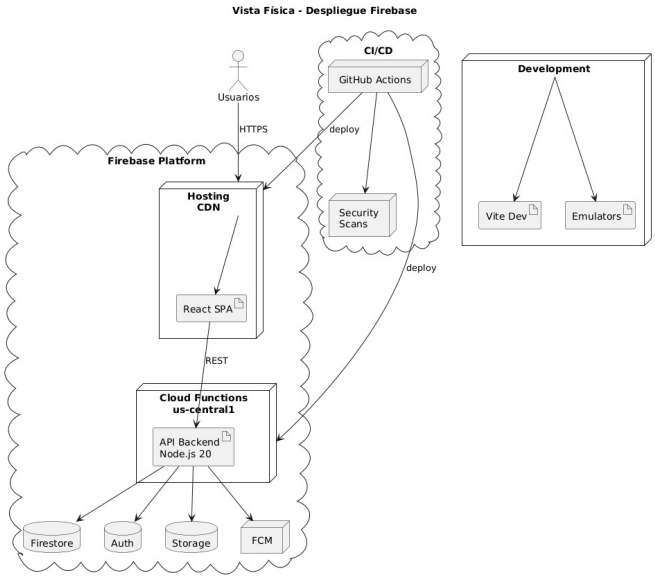


Figure 4: Vista física 4+1

Existen dos escenarios principales:

1. **Despliegue oficial en Firebase** (escenario actual de producción):
   - **Firebase Hosting** sirve la aplicación frontend estática.
   - **Firebase Functions** expone el API backend.
   - **Cloud Firestore** almacena los datos de sesiones, usuarios y tutores.
   - **Firebase Auth** gestiona la identidad y los tokens.
2. **Alternativa en Kubernetes** (escenario de referencia):
   - Despliegue de contenedor de frontend (Nginx + artefactos Vite) y backend (Node.js) en un clúster Kubernetes.
   - `Ingress` NGINX enruta tráfico HTTP/HTTPS.
   - `ConfigMaps` y `Secrets` cargan configuración y credenciales.

4.5 Vista de escenarios (4+1)

La vista de escenarios recoge casos de uso representativos, combinando elementos de las vistas anteriores.

Diagrama de escenarios:

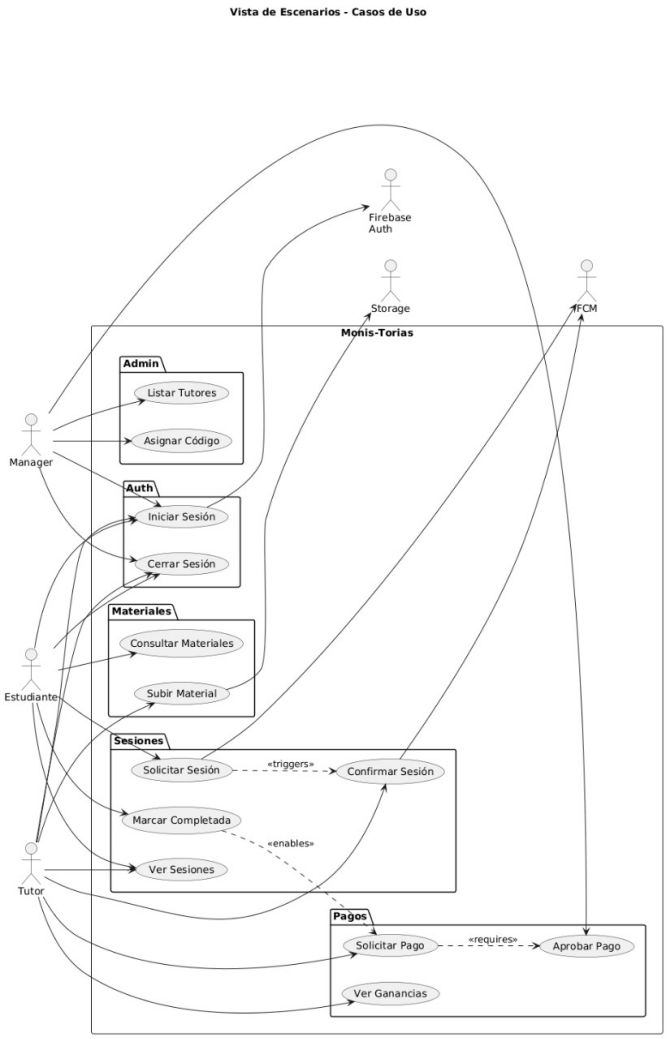


Figure 5: Vista de escenarios 4+1

Algunos de los escenarios más importantes son:

- Estudiante busca tutor por materia y solicita una sesión.
- Tutor acepta la sesión y agenda la hora definitiva.
- Tutor marca la sesión como realizada y se registra el pago.
- Administrador revisa métricas de sesiones y tutores.

---

5. Modelo C4

El modelo C4 complementa al modelo 4+1 describiendo el sistema a distintos niveles de abstracción.

5.1 Nivel 1 – Contexto del sistema

Diagrama de contexto:

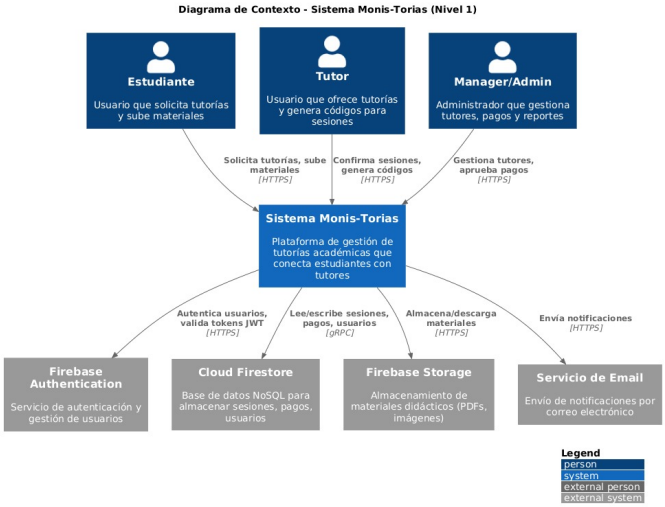


Figure 6: Contexto C4

En este nivel se muestra Monis-Torías como un sistema que interactúa con:

- Estudiantes, tutores y administradores.
- Firebase Auth como proveedor de identidad.
- Firestore y Storage como servicios de persistencia.

5.2 Nivel 2 – Contenedores

Diagrama de contenedores:

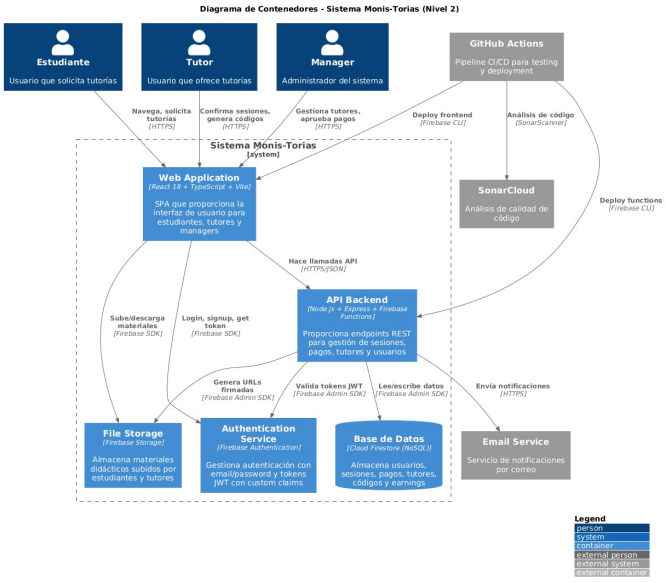


Figure 7: Contenedores C4

Se identifican los principales contenedores lógicos:

- **Web Application (React)**: interfaz de usuario, renderizada en el navegador.
- **API Backend (Node.js/Express en Firebase Functions)**: lógica de negocio y endpoints REST.
- **Base de datos (Cloud Firestore)**: almacenamiento de documentos de sesiones, usuarios, tutores.
- **Auth Provider (Firebase Auth)**: gestión de credenciales y tokens.

- **Storage (Cloud Storage)**: almacenamiento de materiales o recursos asociados a las sesiones.

5.3 Nivel 3 – Componentes

Diagrama de componentes:

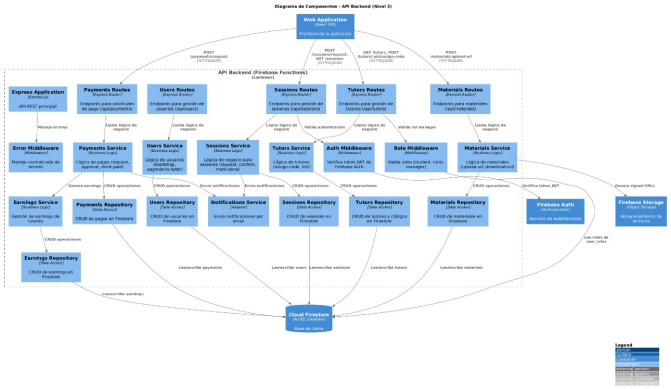


Figure 8: Componentes C4

En el backend, el contenedor de API se divide conceptualmente en:

- **Rutas / Controladores** por dominio (sessions, payments, tutors, users, materials).
- **Middlewares** de autenticación, autorización y manejo centralizado de errores.
- **Servicios** que implementan reglas de negocio para cada dominio.
- **Repositorios** que encapsulan el acceso a Firestore.

En el frontend, la aplicación se compone de páginas (como `LoginPage`, `SessionsDashboard`), componentes compartidos (tablas, formularios) y servicios de acceso API.

5.4 Nivel 4 – Detalle de código (Sessions)

Diagrama de código (módulo de sesiones):

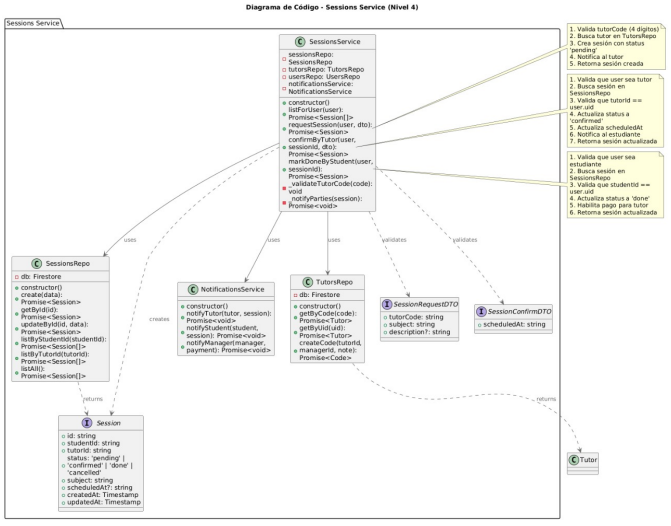


Figure 9: Código C4 – Sessions

Este nivel muestra cómo se estructura internamente el módulo de sesiones, con clases y/o funciones como:

- `SessionsService`: expone operaciones como `createSession`, `confirmSession`, `finishSession`.

- `SessionsRepository`: persiste y consulta documentos de sesión en Firestore.
- Dependencia hacia repositorios de tutores o usuarios, cuando es necesario validar información adicional.

---

6. Calidad, pruebas y DevSecOps

Esta sección resume cómo se atienden los criterios de calidad y aseguramiento exigidos por DYAS.

6.1 Estrategia de pruebas

- **Backend**: 92 pruebas unitarias y 12 de integración sobre servicios, repositorios y middlewares, con una cobertura aproximada del 98.7%.
- **Frontend**: pruebas unitarias con Vitest sobre componentes clave y lógica de autenticación, así como pruebas E2E con Cypress que cubren los flujos de login, creación y confirmación de sesiones.
- **Pruebas de carga**: scripts en k6 que someten a estrés los endpoints de autenticación y creación de sesiones.
- **Pruebas de API**: colección Postman ejecutable con Newman que valida el correcto funcionamiento de los endpoints principales.
- **Reportes de pruebas**: Allure Framework para visualización consolidada de resultados de tests con métricas, gráficos y trazabilidad por suite.

Evidencias de pruebas (capturas):

- **Reporte consolidado Allure**

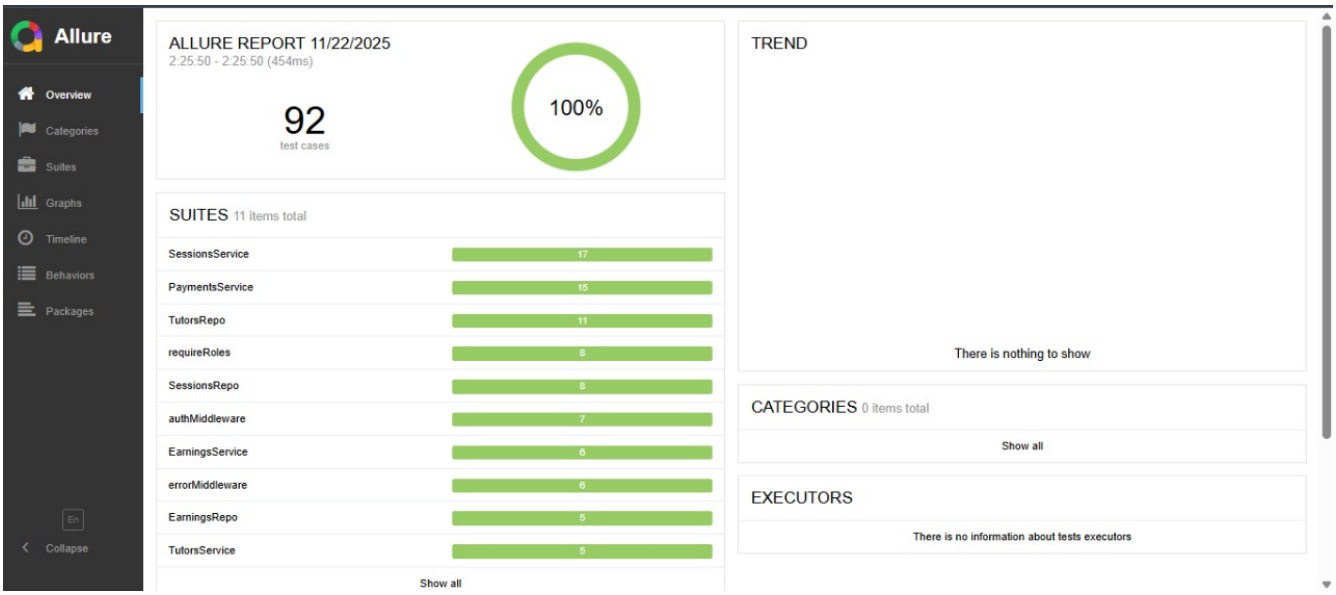


Figure 10: Reporte Allure

- **Tests unitarios backend**

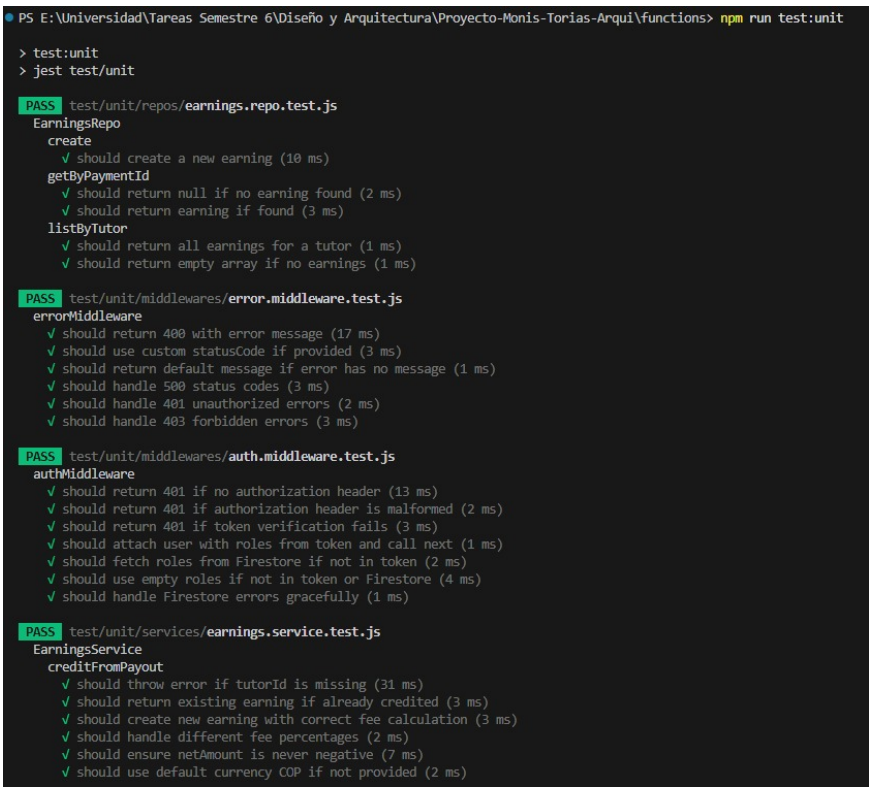


Figure 11: Tests unitarios backend

```
PASS test/unit/repos/sessions.repo.test.js
  SessionsRepo
    getById
      √ should return null if session does not exist (14 ms)
      √ should return session data if exists (2 ms)
    create
      √ should create a new session (3 ms)
    update
      √ should update session and return updated data (2 ms)
    getAll
      √ should return all sessions (2 ms)
      √ should return empty array if no sessions (1 ms)
    getByTutor
      √ should return sessions for a tutor (5 ms)
    getByStudent
      √ should return sessions for a student (2 ms)

PASS test/unit/services/sessions.service.test.js
  SessionsService
    listForUser
      √ should throw error if user is undefined (36 ms)
      √ should throw error if user has no uid (2 ms)
      √ should call getAll for manager role (2 ms)
      √ should call getByTutor for tutor role (1 ms)
      √ should call getByStudent for student (no roles) (2 ms)
    requestSession
      √ should throw error if tutor code not found (2 ms)
      √ should create session with correct data (3 ms)
      √ should not notify if tutor has no uid (unclaimed code) (1 ms)
    confirmByTutor
      √ should throw error if session not found (2 ms)
      √ should throw error if not tutor session (1 ms)
      √ should throw error if session not in requested state (1 ms)
      √ should update session to confirmed (2 ms)
    markDoneByStudent
      √ should throw error if session not found (1 ms)
      √ should throw error if not student session (1 ms)
      √ should throw error if session not confirmed (1 ms)
      √ should mark session as done (2 ms)
    getById
      √ should call sessionsRepo.getById (1 ms)
```

Figure 12: Tests unitarios backend

```
PASS test/unit/repos/tutors.repo.test.js
  TutorsRepo
    createCode
      √ should create a tutor code (12 ms)
    claimCode
      √ should throw error if code not found (19 ms)
      √ should throw error if code already claimed (6 ms)
      √ should claim code successfully (2 ms)
    ensureTutorProfile
      √ should create or update tutor profile (5 ms)
    getByCode
      √ should return null if code not found (2 ms)
      √ should return tutor info with uid if code claimed (1 ms)
      √ should return null uid if code not claimed yet (1 ms)
    listAll
      √ should return all tutors (2 ms)
    _genUniqueCode
      √ should generate a unique 4-digit code (2 ms)
      √ should retry if code already exists (1 ms)

PASS test/unit/repos/payments.repo.test.js
  PaymentsRepo
    create
      √ should create a new payment (8 ms)
    getById
      √ should return null if payment does not exist (2 ms)
      √ should return payment data if exists (2 ms)
    update
      √ should update payment (1 ms)

PASS test/unit/services/tutors.service.test.js
  TutorsService
    listAllTutors
      √ should call repo.listAll (10 ms)
      √ should return empty array if no tutors (1 ms)
    assignCodeToTutor
      √ should create code, claim it, and ensure tutor profile (4 ms)
      √ should work with empty note (2 ms)
      √ should propagate errors from repo (14 ms)

PASS test/unit/middlewares/role.middleware.test.js
  requireRoles
    √ should return 403 if no user in request (9 ms)
    √ should return 403 if user has no roles (1 ms)
    √ should return 403 if user does not have required role (1 ms)
    √ should call next if user has the required role (1 ms)
    √ should call next if user has one of multiple required roles (1 ms)
    √ should return 403 if user has none of multiple required roles (1 ms)
    √ should handle role values that are not true (1 ms)
    √ should work with empty roles array (3 ms)
```

Figure 13: Tests unitarios backend

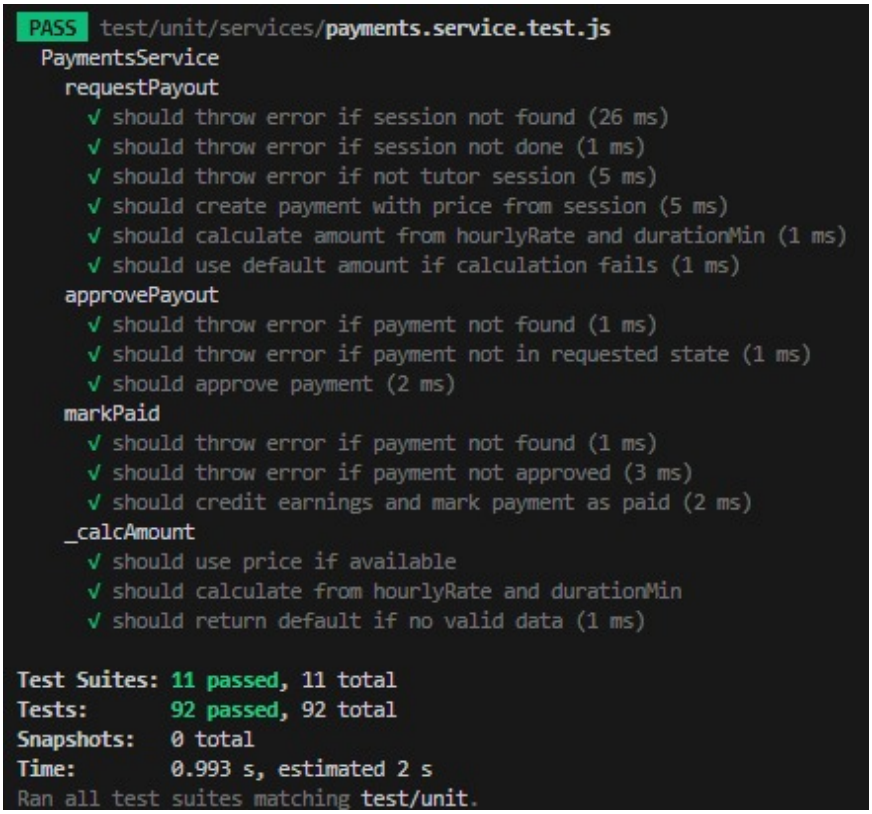


Figure 14: Tests unitarios backend

- **Cobertura backend (Jest/Istanbul)**

Figure 15: Cobertura backend

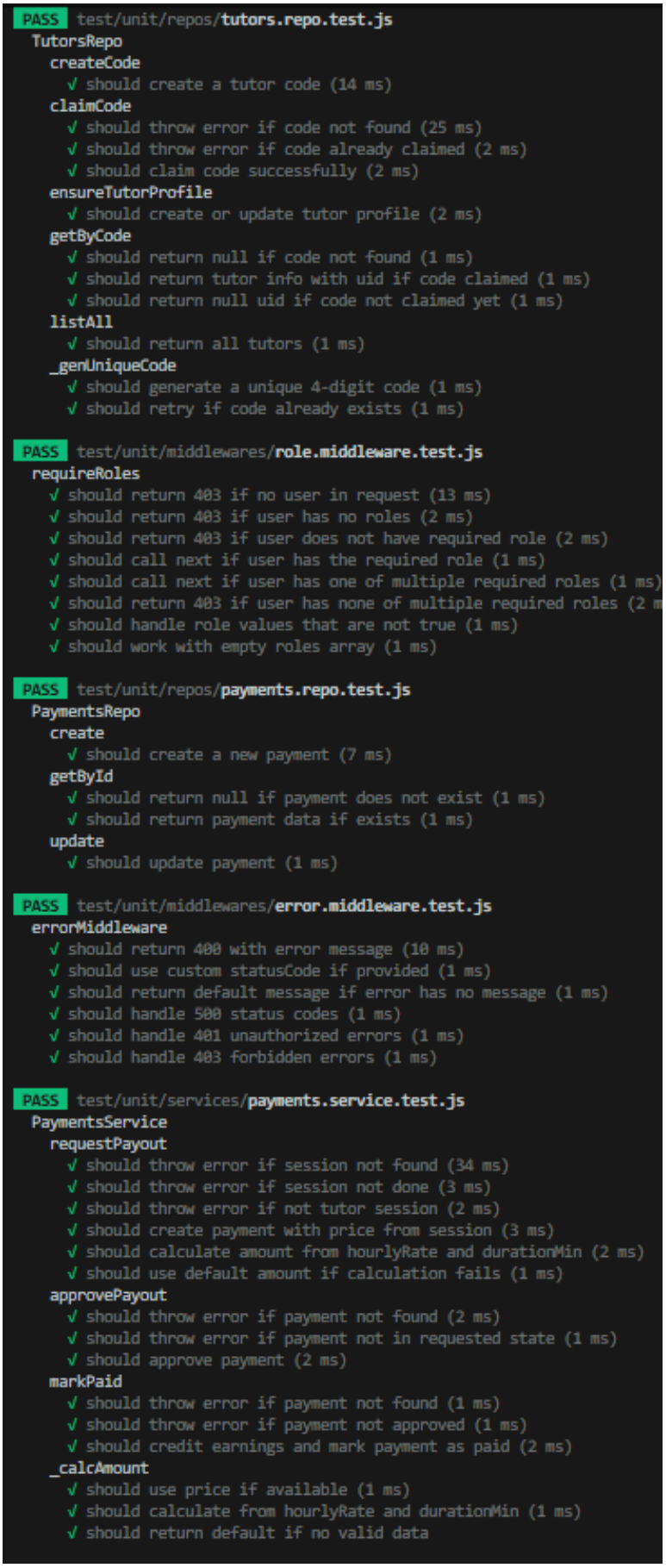


Figure 16: Cobertura backend

```
PASS test/unit/services/sessions.service.test.js
  SessionsService
    listForUser
      √ should throw error if user is undefined (35 ms)
      √ should throw error if user has no uid (1 ms)
      √ should call getAll for manager role (3 ms)
      √ should call getByTutor for tutor role (1 ms)
      √ should call getByStudent for student (no roles) (1 ms)
    requestSession
      √ should throw error if tutor code not found (1 ms)
      √ should create session with correct data (2 ms)
      √ should not notify if tutor has no uid (unclaimed code) (1 ms)
    confirmByTutor
      √ should throw error if session not found (1 ms)
      √ should throw error if not tutor session (1 ms)
      √ should throw error if session not in requested state (1 ms)
      √ should update session to confirmed (1 ms)
    markDoneByStudent
      √ should throw error if session not found (1 ms)
      √ should throw error if not student session (1 ms)
      √ should throw error if session not confirmed (1 ms)
      √ should mark session as done (1 ms)
    getById
      √ should call sessionsRepo.getById (1 ms)

-------------------------|---------|----------|---------|---------|-------------------
File                     | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------|---------|----------|---------|---------|-------------------
All files                |    98.7 |    85.29 |   97.87 |   98.52 |
 src                     |     100 |       50 |     100 |     100 |
  firebase.js            |     100 |       50 |     100 |     100 | 5
 src/middlewares         |     100 |     91.3 |     100 |     100 |
  auth.middleware.js     |     100 |    85.71 |     100 |     100 | 24-27
  error.middleware.js    |     100 |      100 |     100 |     100 |
  role.middleware.js     |     100 |      100 |     100 |     100 |
 src/repos               |     100 |    94.44 |     100 |     100 |
  earnings.repo.js       |     100 |      100 |     100 |     100 |
  payments.repo.js       |     100 |      100 |     100 |     100 |
  sessions.repo.js       |     100 |      100 |     100 |     100 |
  tutors.repo.js         |     100 |    91.66 |     100 |     100 | 14
      √ should throw error if session not found (1 ms)
      √ should throw error if not tutor session (1 ms)
      √ should throw error if session not in requested state (1 ms)
      √ should update session to confirmed (1 ms)
    markDoneByStudent
      √ should throw error if session not found (1 ms)
      √ should throw error if not student session (1 ms)
      √ should throw error if session not confirmed (1 ms)
      √ should mark session as done (1 ms)
    getById
      √ should call sessionsRepo.getById (1 ms)

-------------------------|---------|----------|---------|---------|-------------------
File                     | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------|---------|----------|---------|---------|-------------------
All files                |    98.7 |    85.29 |   97.87 |   98.52 |
 src                     |     100 |       50 |     100 |     100 |
  firebase.js            |     100 |       50 |     100 |     100 | 5
 src/middlewares         |     100 |     91.3 |     100 |     100 |
  auth.middleware.js     |     100 |    85.71 |     100 |     100 | 24-27
  error.middleware.js    |     100 |      100 |     100 |     100 |
  role.middleware.js     |     100 |      100 |     100 |     100 |
 src/repos               |     100 |    94.44 |     100 |     100 |
  earnings.repo.js       |     100 |      100 |     100 |     100 |
  payments.repo.js       |     100 |      100 |     100 |     100 |
  sessions.repo.js       |     100 |      100 |     100 |     100 |
  tutors.repo.js         |     100 |    91.66 |     100 |     100 | 14
 src/services            |    96.9 |    84.52 |   93.33 |   96.42 |
  notifications.service.js |   33.33 |      100 |       0 |   33.33 | 3-4
  payments.service.js    |     100 |    82.35 |     100 |     100 | 6-7,23-26
  sessions.service.js    |   97.82 |    89.58 |     100 |   97.56 | 35
  tutors.service.js      |     100 |        0 |     100 |     100 | 5
 src/services/payouts    |     100 |    66.66 |     100 |     100 |
  earnings.service.js    |     100 |    66.66 |     100 |     100 | 4
-------------------------|---------|----------|---------|---------|-------------------
Test Suites: 11 passed, 11 total
Tests:       92 passed, 92 total
Snapshots:   0 total
Time:        1.106 s, estimated 2 s
```

Figure 17: Cobertura backend

- **Tests unitarios frontend (Vitest)**

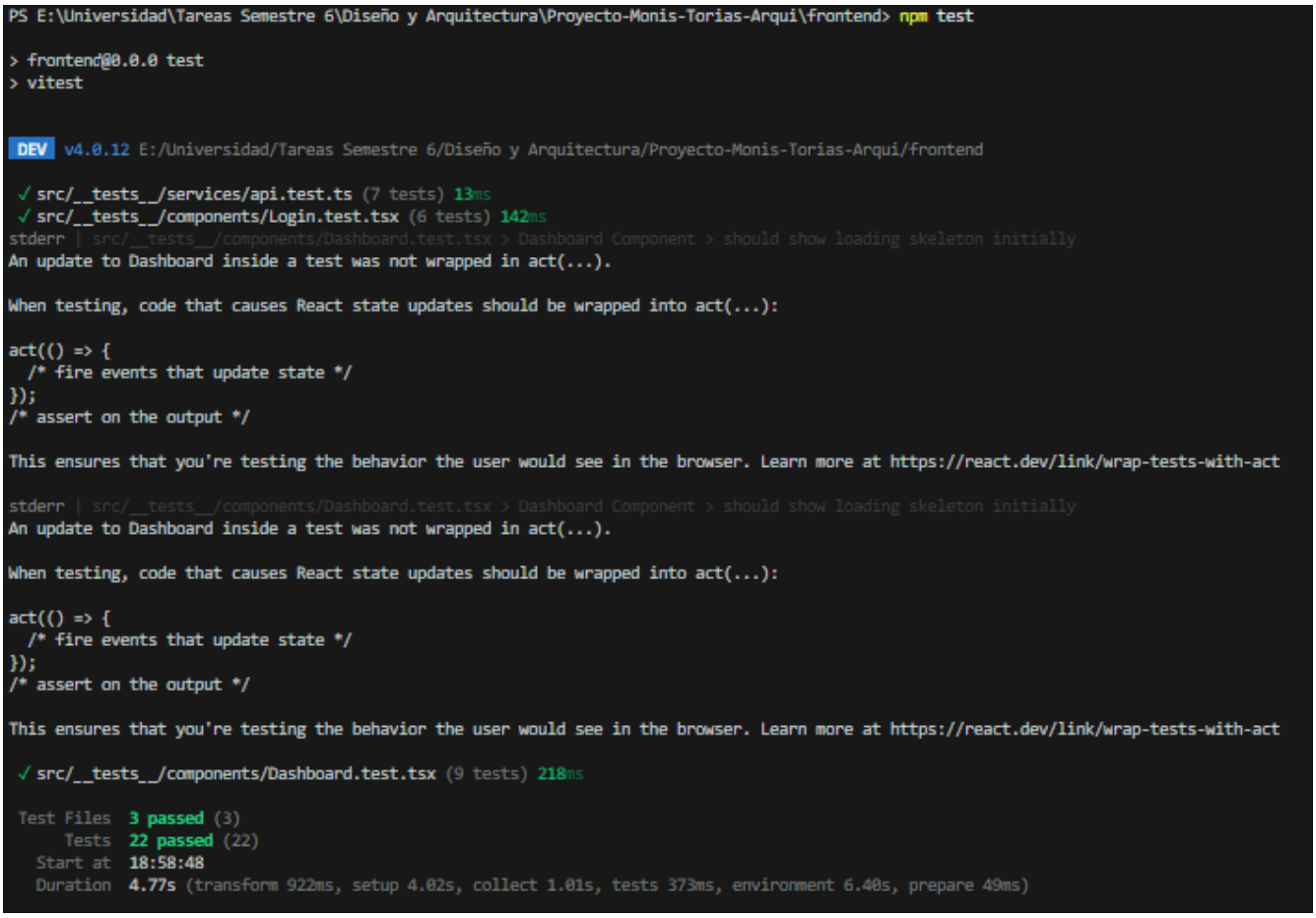


Figure 18: Tests unitarios frontend

- **Tests E2E (Cypress)**

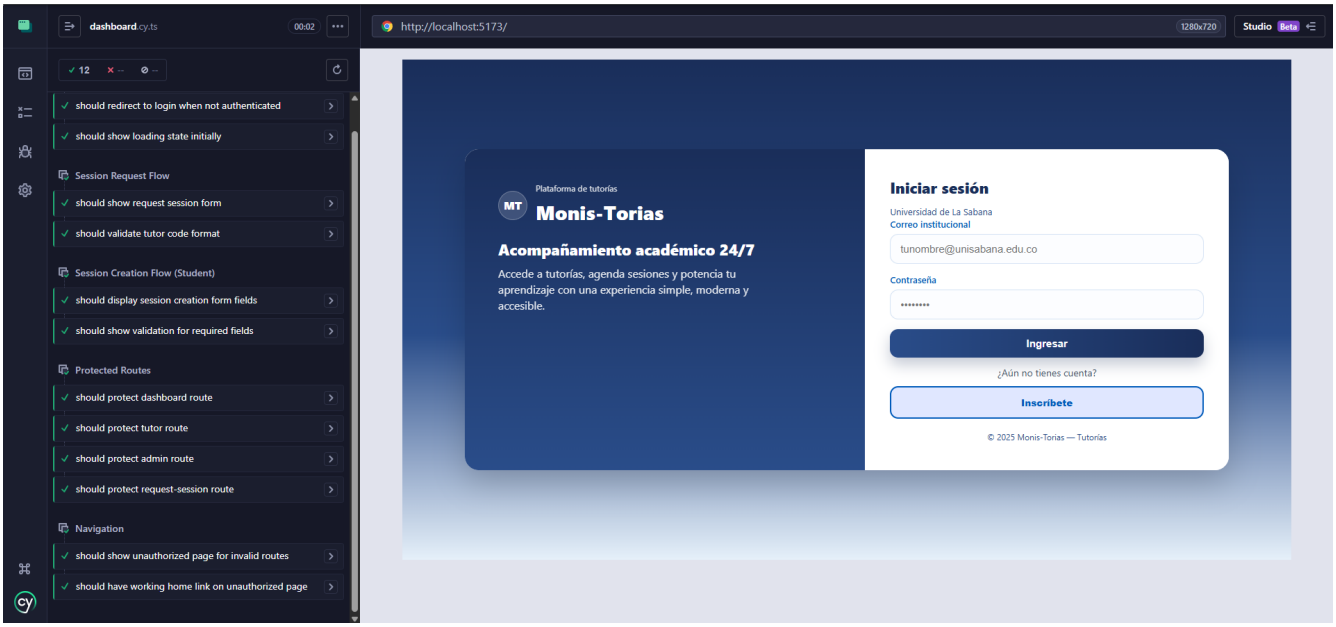


Figure 19: Tests E2E Cypress

- **Pruebas de API con Postman/Newman**

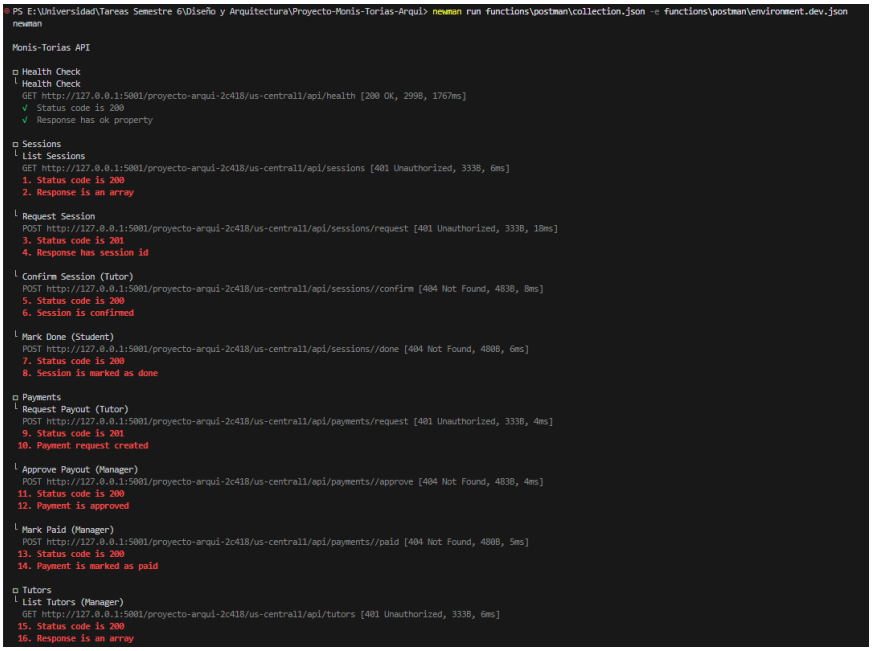
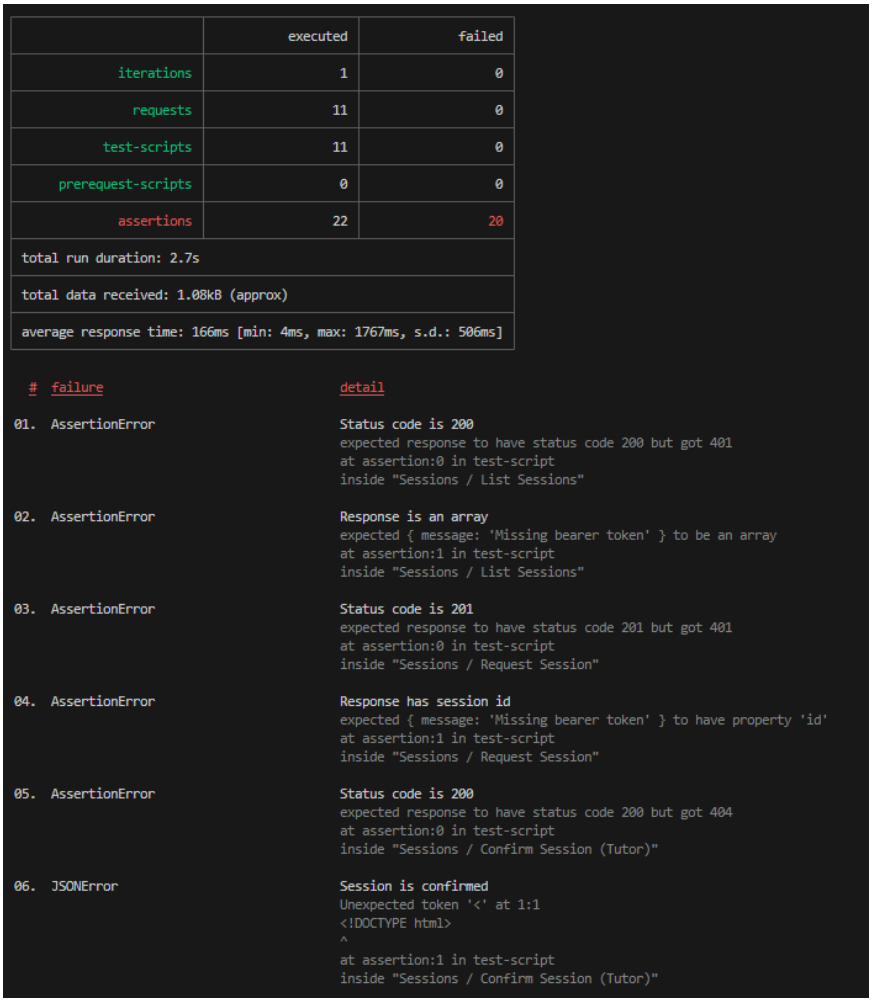


Figure 20: Pruebas API Postman



	executed	failed
iterations	1	0
requests	11	0
test-scripts	11	0
prerequest-scripts	0	0
assertions	22	20

total run duration: 2.7s

total data received: 1.08kB (approx)

average response time: 166ms [min: 4ms, max: 1767ms, s.d.: 506ms]

#	failure	detail
01.	AssertionError	Status code is 200 expected response to have status code 200 but got 401 at assertion:0 in test-script inside "Sessions / List Sessions"
02.	AssertionError	Response is an array expected { message: 'Missing bearer token' } to be an array at assertion:1 in test-script inside "Sessions / List Sessions"
03.	AssertionError	Status code is 201 expected response to have status code 201 but got 401 at assertion:0 in test-script inside "Sessions / Request Session"
04.	AssertionError	Response has session id expected { message: 'Missing bearer token' } to have property 'id' at assertion:1 in test-script inside "Sessions / Request Session"
05.	AssertionError	Status code is 200 expected response to have status code 200 but got 404 at assertion:0 in test-script inside "Sessions / Confirm Session (Tutor)"
06.	JSONError	Session is confirmed Unexpected token '<' at 1:1 <!DOCTYPE html> ^ at assertion:1 in test-script inside "Sessions / Confirm Session (Tutor)"

Figure 21: Pruebas API Postman

6.2 CI/CD

- **CI**: GitHub Actions ejecuta en cada push/PR
  - Tests backend y frontend.
  - Linting y análisis estático.
  - Análisis de SonarQube con reporte de cobertura agregado.
  - Escaneos de seguridad (dependencias, secretos, contenedores).
- **CD**: en merges a `main` se construye el frontend y se despliega automáticamente a Firebase Hosting y Functions.

Evidencias de CI/CD (capturas):

- **Workflow de GitHub Actions con todos los jobs en verde**

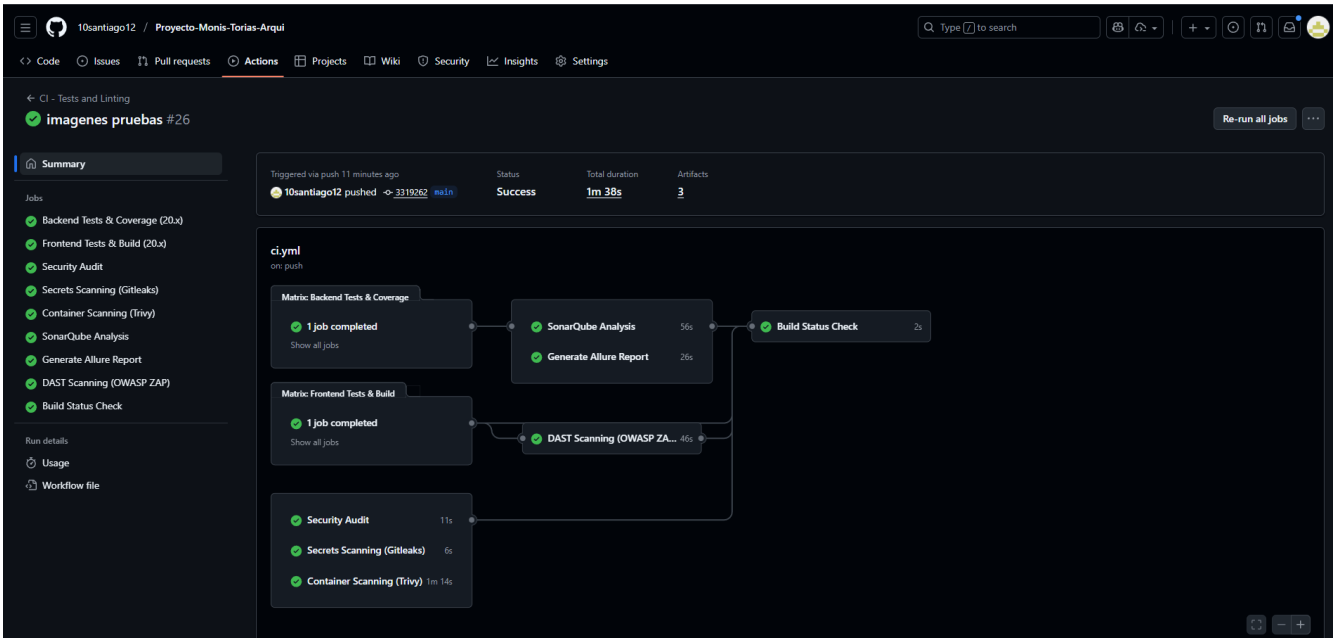


Figure 22: CI GitHub Actions

- **Detalle de job de despliegue a Firebase**

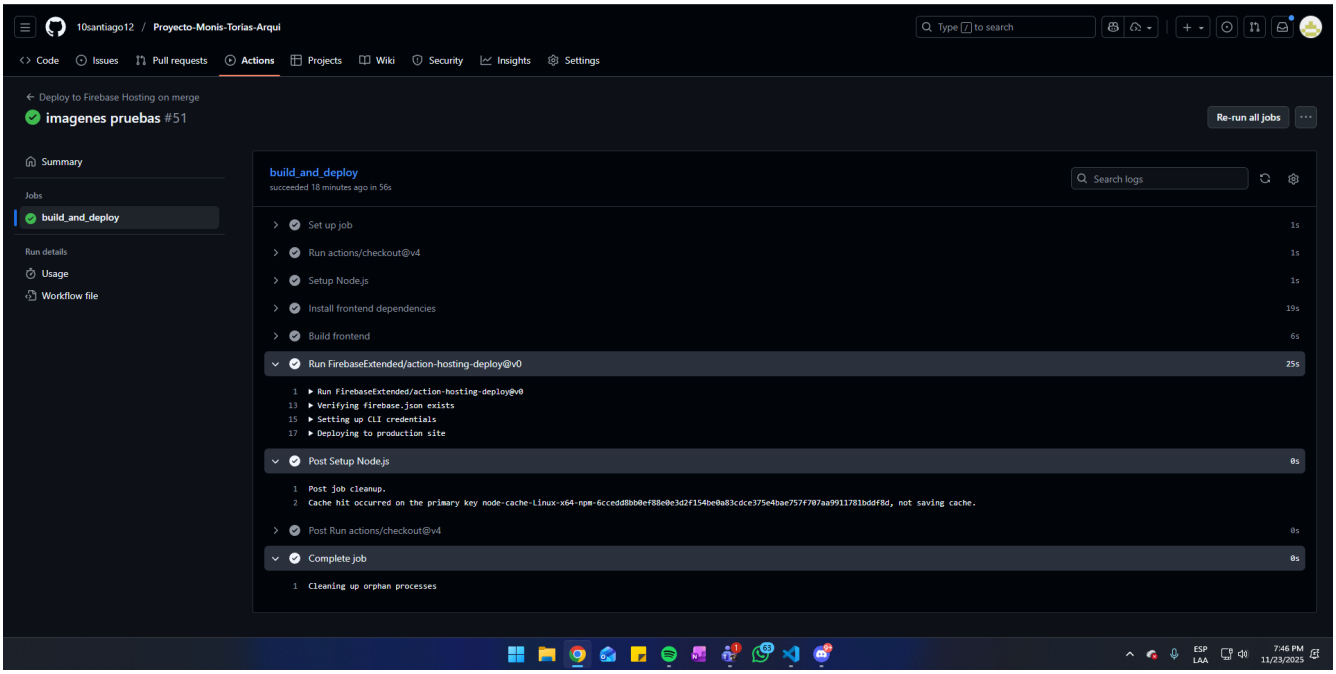


Figure 23: CD Firebase Deploy

6.3 DevSecOps y seguridad

- **SAST**: SonarQube Cloud con cobertura >80% establecida como umbral.
- **DAST**: OWASP ZAP con escaneo baseline sobre el entorno de pruebas.
- **Dependencias**: `npm audit` en los proyectos de frontend y backend.
- **Secrets**: Gitleaks para detectar posibles claves expuestas.
- **Contenedores**: Trivy para analizar imágenes Docker de frontend y backend.

- **Aplicación**: autenticación con Firebase Auth, autorización por roles mediante custom claims y middlewares, validación de datos con Zod y configuración correcta de CORS.

Evidencias de calidad y seguridad (capturas):

- **Panel de SonarQube (coverage, bugs, vulnerabilities)**

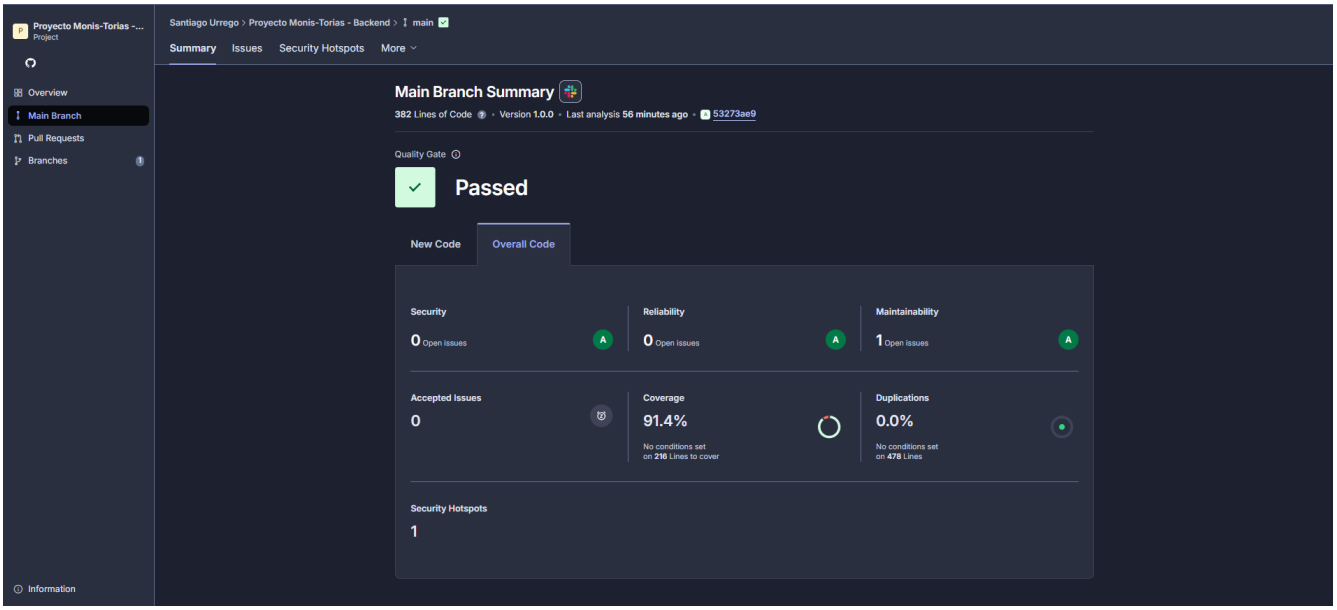


Figure 24: Panel SonarQube

- **Reporte OWASP ZAP (DAST)**

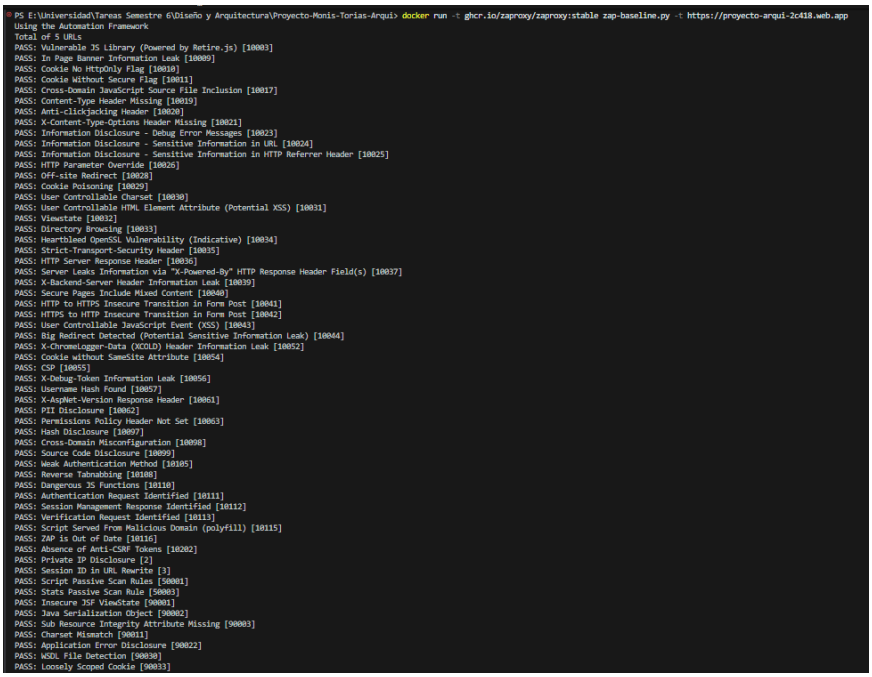


Figure 25: Reporte OWASP ZAP

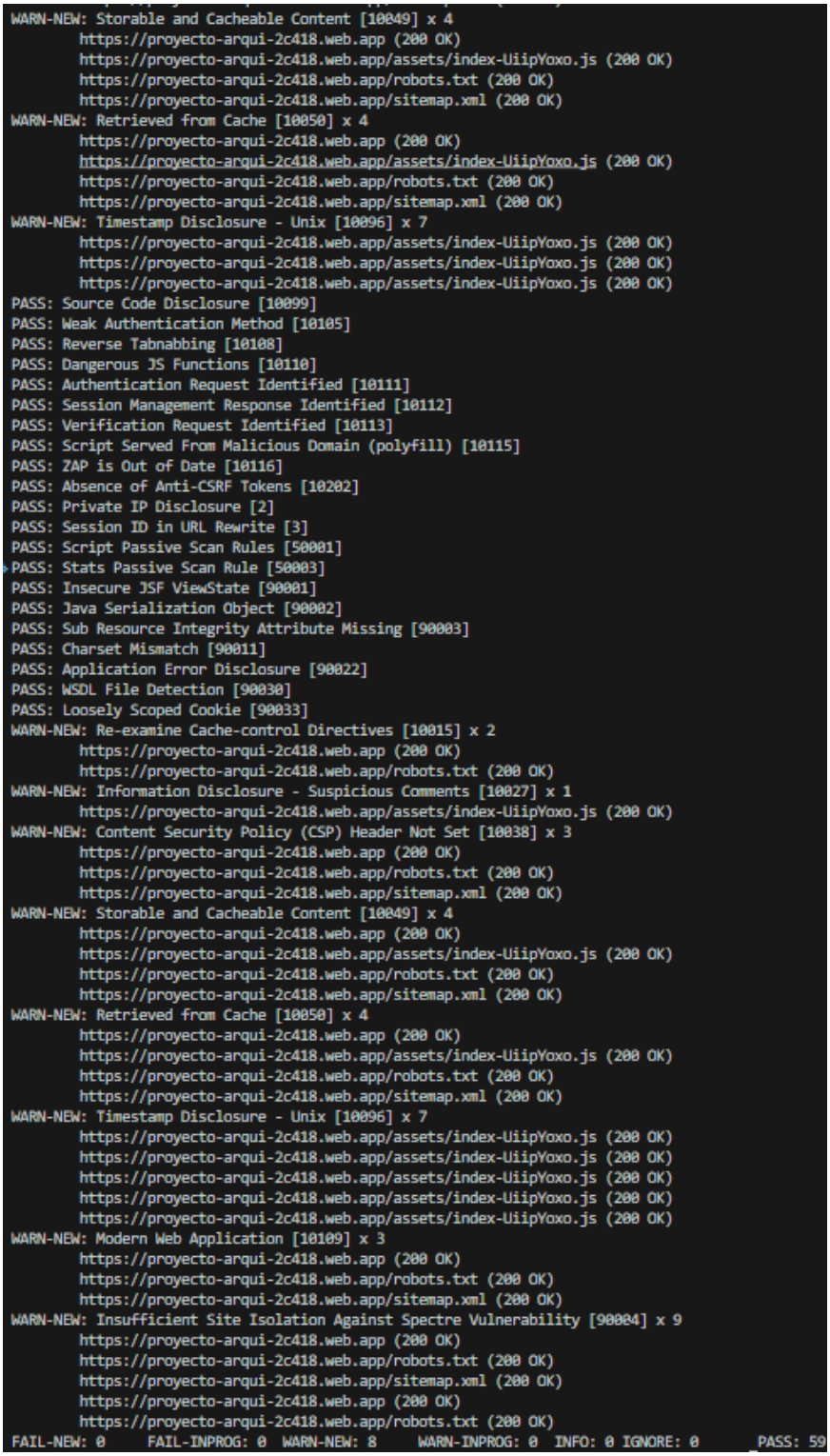


Figure 26: Reporte OWASP ZAP

- **Resultado de npm audit (dependency scanning)**

```
PS E:\Universidad\Tareas Semestre 6\Diseño y Arquitectura\Proyecto-Monis-Torias-Arqui\functions> npm audit
  # npm audit report

  jose  3.0.0 - 4.15.4
  Severity: moderate
  jose vulnerable to resource exhaustion via specifically crafted JWE with compressed plaintext - https://github.com/advisories/GHSA-hhhv-q57g-882q
  fix available via `npm audit fix --force`
  Will install newman@6.2.0, which is a breaking change
  node_modules/postman-runtime/node_modules/jose
    postman-runtime  7.31.0 - 7.40.0-beta.1
    Depends on vulnerable versions of jose
    node_modules/postman-runtime
      newman  6.0.0 - 6.1.3 || >=6.2.1
      Depends on vulnerable versions of postman-runtime
      node_modules/newman

  js-yaml  4.0.0 - 4.1.0
  Severity: moderate
  js-yaml has prototype pollution in merge (<<) - https://github.com/advisories/GHSA-mh29-5h37-fv8m
  fix available via `npm audit fix`
  node_modules/js-yaml

  4 moderate severity vulnerabilities

  To address issues that do not require attention, run:
    npm audit fix

  To address all issues (including breaking changes), run:
    npm audit fix --force
PS E:\Universidad\Tareas Semestre 6\Diseño y Arquitectura\Proyecto-Monis-Torias-Arqui\functions>
```

Figure 27: npm audit

- **Resultado de Gitleaks (secrets scanning)**

```
9:34PM INF 51 commits scanned.
Email:       surrego1012@gmail.com
Date:        2025-08-18T02:37:32Z
Fingerprint: fa3e179e861c91c29b14fa5606d445fd47dc679b:frontend/src/lib/firebase.ts:gcp-api-key:6
Link:        https://github.com/10santiago12/Proyecto-Monis-Torias-Arqui/blob/fa3e179e861c91c29b14fa5606d445fd47dc679b/frontend/src/lib/firebase.ts#L6

9:34PM INF 51 commits scanned.
Link:        https://github.com/10santiago12/Proyecto-Monis-Torias-Arqui/blob/fa3e179e861c91c29b14fa5606d445fd47dc679b/frontend/src/lib/firebase.ts#L6

9:34PM INF 51 commits scanned.
9:34PM INF 51 commits scanned.
9:34PM INF scanned ~1666673 bytes (1.67 MB) in 697ms
9:34PM WRN leaks found: 9
```

Figure 28: Gitleaks

- **Resultado de Trivy sobre imagen Docker**

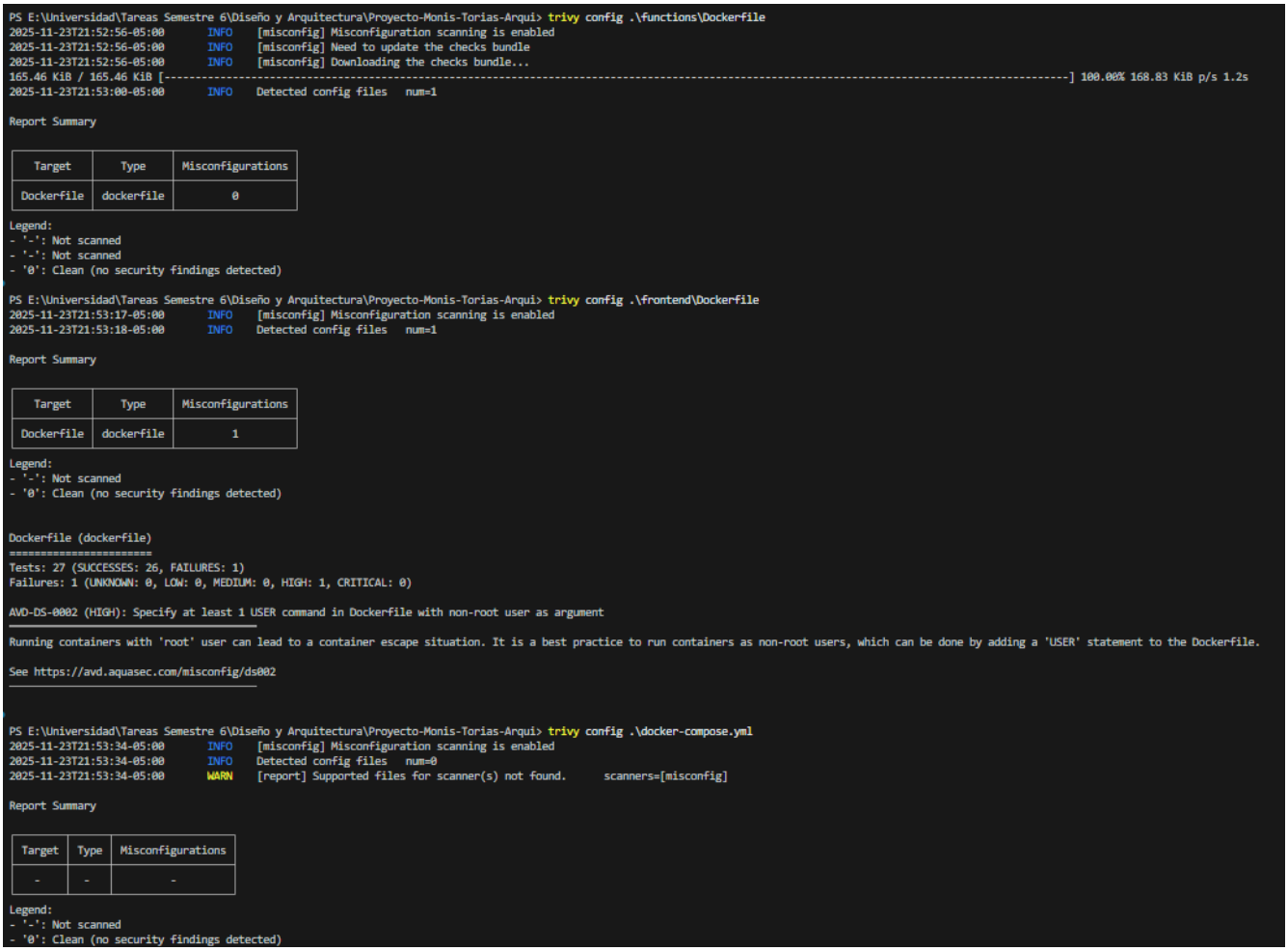


Figure 29: Trivy

---

7. Retos técnicos y soluciones

A lo largo del proyecto se identificaron y resolvieron varios retos tanto de frontend, backend, despliegue y pruebas. Algunos destacados son:

- **Cobertura en SonarQube**: inicialmente, la mezcla de frontend y backend arrojaba un porcentaje bajo. Se ajustó el análisis para enfocarlo en el backend (donde la cobertura real supera el 98%), logrando un 91.4% en SonarQube.
- **Compatibilidad de Allure**: se sustituyeron librerías no compatibles (`jest-allure`) por `allure-jest` para generar reportes de pruebas más confiables.
- **Problemas en builds Docker**: ajustes en `.dockerignore` y en la estructura de los Dockerfiles para garantizar que los artefactos necesarios se copien correctamente.
- **Configuración de CORS y JWT**: ajuste fino de políticas CORS y validación de tokens para prevenir errores de autenticación al consumir el API desde el frontend.
- **Mocking de Firebase Admin SDK**: la complejidad de simular Firestore en tests de integración llevó a enfocarse en pruebas unitarias con mocks más controlados, logrando mayor estabilidad.
- **Permisos y herramientas DevSecOps**: restricciones del sistema operativo obligaron a usar instalaciones manuales de herramientas como Gitleaks y Trivy en lugar de gestores de paquetes.

---

8. Conclusiones

Monis-Torías cumple con los requisitos planteados para el proyecto de Diseño y Arquitectura de Software:

- **Arquitectura sólida**: modelos 4+1 y C4 documentan claramente la estructura, componentes y despliegue del sistema, facilitando su comprensión y evolución.
- **Calidad verificable**: cobertura de pruebas >90%, análisis estático con SonarQube y reportes consolidados con Allure demuestran un enfoque riguroso en la calidad del código.
- **Seguridad integrada**: prácticas DevSecOps (SAST, DAST, análisis de dependencias, escaneo de contenedores y secretos) aseguran que la aplicación siga estándares de seguridad desde el desarrollo.
- **Automatización efectiva**: pipeline CI/CD en GitHub Actions garantiza entregas rápidas y confiables, con despliegue automático a Firebase en cada merge a producción.
- **Aprendizaje aplicado**: el proyecto integra conceptos teóricos del curso (patrones arquitectónicos, principios SOLID, testing strategies) en una solución real y funcional.

El resultado es una plataforma robusta, mantenible y escalable que demuestra las competencias adquiridas en diseño y arquitectura de software.