Diagnóstico e implementación de pruebas unitarias en módulos críticos para el sistema de Invessoft

Joan Manuel Jaramillo Avila

Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Tuluá
2026

Diagnóstico e implementación de pruebas unitarias en módulos críticos para el sistema de Invessoft

Joan Manuel Jaramillo Avila
Código 202159930
jaramillo.joan@correounivalle.edu.co

Director
Ms.C Royer David Estrada Esponda. Ing
Profesor de la Escuela de Ingeniería de Sistemas y Computación
royer.estrada@correounivalle.edu.co

Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Tuluá
2026

Resumen

Este trabajo presenta el diagnóstico y la implementación de pruebas unitarias en módulos críticos de Invessoft, una plataforma web utilizada para apoyar la gestión académica, administrativa y operativa de procesos como eventos, perfiles, certificados, agenda, trabajos, reportes y nuevos servicios funcionales. El proyecto se desarrolló en el contexto de la pasantía en Webcloster y partió de la necesidad de fortalecer la confiabilidad del sistema frente a cambios frecuentes y riesgos de regresión.

El diagnóstico inicial permitió identificar una situación desigual en la cobertura de pruebas: mientras en backend existía una protección limitada sobre servicios y controladores con lógica sensible, en frontend la cobertura se concentraba principalmente en servicios y no reflejaba con suficiente fidelidad el comportamiento de componentes, formularios y flujos de interacción. A partir de este análisis se priorizaron módulos con alta criticidad funcional, reglas de negocio condicionales, integraciones entre frontend y backend, validaciones complejas y alta exposición al cambio.

La intervención se realizó sobre el stack tecnológico real de la aplicación, empleando NestJS y Jest en backend, y Angular, Jasmine y Karma en frontend. Además de incrementar la cobertura, el trabajo permitió documentar criterios de priorización, aislar dependencias mediante *mocks*, validar formularios, servicios y componentes críticos, y relacionar la evidencia de pruebas con cambios funcionales concretos en módulos como espacios, reservas, notificaciones, certificados, CRM, agenda, landings y percepción de usabilidad. Como resultado, se consolidó una base técnica y metodológica para ampliar el aseguramiento de calidad en Invessoft con un enfoque orientado al riesgo y al valor funcional.

Palabras clave: pruebas unitarias, aseguramiento de la calidad, cobertura de pruebas, frontend, backend.

Abstract

This work presents the diagnosis and implementation of unit tests in critical modules of Invessoft, a web platform used to support academic, administrative, and operational processes such as events, profiles, certificates, scheduling, papers, reports, and new functional services. The project was carried out during an internship at Webcloster and was motivated by the need to improve system reliability in the presence of frequent changes and regression risks.

The initial diagnosis revealed an uneven testing scenario: backend coverage offered limited protection for services and controllers with sensitive business logic, while frontend coverage was largely concentrated on services and did not accurately represent the behavior of components, forms, and interaction flows. Based on this analysis, the work prioritized modules with high functional criticality, conditional business rules, frontend-backend integrations, complex validations, and high exposure to change.

The intervention was performed on the actual application stack, using NestJS and Jest in the backend, and Angular, Jasmine, and Karma in the frontend. Beyond increasing coverage, the project documented prioritization criteria, isolated dependencies through *mocks*, validated forms, services, and critical components, and linked test evidence with concrete functional changes in modules such as spaces, reservations, notifications, certificates, CRM, scheduling, landing pages, and usability perception. The main result is a technical and methodological baseline for expanding quality assurance in Invessoft through a risk-based and function-oriented testing strategy.

Keywords: unit testing, quality assurance, test coverage, frontend, backend.

Tabla de Contenido

Lista de tablas

Lista de Figuras

1. Dedicatoria

Dedico este trabajo, en primer lugar, a mis padres y a mi hermano, por ser el soporte constante de mi proceso personal y profesional. Su confianza, esfuerzo, paciencia y acompañamiento han sido fundamentales en cada etapa de este camino. A mis amigos y a las personas cercanas que me han brindado apoyo, escucha y ánimo en los momentos de mayor exigencia, les dedico también este logro. Su presencia hizo más llevadero el proceso y me recordó la importancia de avanzar con constancia, tranquilidad y sentido. Finalmente, dedico este trabajo a quienes, de una u otra manera, creyeron en mí y aportaron con sus palabras, su tiempo o su compañía a que este proyecto llegara a buen término.

2. Agradecimientos

Expreso mi agradecimiento a Webcloster por brindarme el espacio para desarrollar esta pasantía y por permitirme participar en procesos reales de evolución de Invessoft. Agradezco de manera especial a mi director, Royer, por su orientación, acompañamiento y criterio durante el desarrollo de este trabajo. También agradezco a mis amigos, a mis mentores y a las personas cercanas que acompañaron este proceso, entre ellas JM, por su apoyo, confianza y aporte al crecimiento personal y profesional que hizo posible la culminación de este proyecto.

3. Introducción

El software a lo largo de la historia ha desempeñado un papel que busca solucionar o facilitar actividades en la vida del ser humano. Entre las más comunes se encuentran mejorar la interacción entre diferentes grupos de la sociedad e incluso ser un eje de apoyo para la transferencia del conocimiento.

La relación ciencia, tecnología y sociedad se ha constituido como un campo interdisciplinar y de investigación, alrededor del cual se mueven conceptos como el de comunicación pública, popularización, apropiación, interacción, ciencia - público, entre otros, que intentan darle forma a tan complejo espacio que, durante las últimas décadas, ha tomado un papel fundamental a la hora de diseñar estrategias, políticas y mecanismos de socialización del conocimiento científico, en dónde la participación, la reflexión y el análisis por parte de los públicos es esencial para superar las barreras de la comunicación unidireccional de la ciencia y la tecnología [1].

Actualmente, en Colombia se desarrollan congresos, simposios y eventos que promueven estas relaciones; con el fin de llevar satisfactoriamente estas reuniones surgen tecnologías de tipo software que pretenden conectar a las personas para lograr la transferencia del conocimiento. Estos softwares son conocidos como sistemas de gestión de conferencias, los cuales deben permitir la planificación de reuniones, el registro de los asistentes, la información detallada de los trabajos a exponer con el fin de facilitar la interactividad ciencia - público.

Invessoft es una herramienta utilizada para apoyar la promoción e intercambio de información educativa a fin de fomentar el desarrollo de la investigación uniendo las diferentes instituciones educativas en un solo lugar para el intercambio de saberes de las diferentes regiones. Este software verá una variación en la implementación del módulo que verá muchas otras funcionalidades unidas en el producto. Por lo tanto, un análisis costo-beneficio determinará cuál es el mejor curso de acción para el futuro de esta aplicación. Esto incluye todas las otras funcionalidades que están en espera, lo que le dará una mayor funcionalidad.

4. Descripción de la empresa

Webcloster es una empresa del sector de tecnologías de la información y la comunicación que, en respuesta a las necesidades del entorno, innova en la forma de trabajar y administrar recursos mediante la creación de plataformas tecnológicas de servicios. A través de sus propios sistemas de información, la compañía apoya a organizaciones de todos los tamaños para que puedan alcanzar su máximo potencial mediante las soluciones tecnológicas que ofrece.

4.1. Misión

Empresa dedicada a la creación y desarrollo de Software que facilita la gestión y administración efectiva de las operaciones empresariales y personales, brindando soluciones innovadoras que se adaptan a las necesidades de nuestros clientes buscando siempre incrementar la productividad y competitividad con servicios de alto valor agregado, y confiabilidad.

4.2. Visión

Establecer un posicionamiento en el sector de las TIC; implementado los cambios que la globalización demande con respecto al desarrollo de software y aplicativos móviles, mejorando continuamente cada uno de los procesos implementados, cumpliendo requerimientos de nuestros clientes y comprometidos de forma transparente con sus necesidades.

4.3. Valores organizacionales

- Honestidad: es vital la transparencia y sinceridad.
- Pasión: "Disfrutar lo que hacemos"
- Puntualidad: tener especial consideración con el tiempo de clientes, proveedores y jefes.
- Calidad: garantizar los mejores productos y servicios ante el mundo. Trabajo en equipo: tolerancia, respeto, admiración, lealtad y solidaridad.
- Orientación al cliente: adecuarnos a las necesidades de nuestros clientes.
- Mejora continua: mejora y revolución de procesos, productos y servicios para aumentar la calidad y satisfacción de resultados de forma sostenible.
- Resolución de problemas: incentivamos el pensamiento de nuestros colabora- dores orientado a la búsqueda de soluciones culpables óptimas para mejorar y no buscar
- Responsabilidad social empresarial: causar un impacto amplio y positivo ante la sociedad que nos rodea.

4.4. Razón social

Webcloster S.A.S.

4.5. Tipo de empresa

Microempresa - Sociedad por Acciones Simplificada - Privada - Sector Tecnología

4.6. Número de empleados

Trece (13) empleados

4.7. Organigrama de la empresa

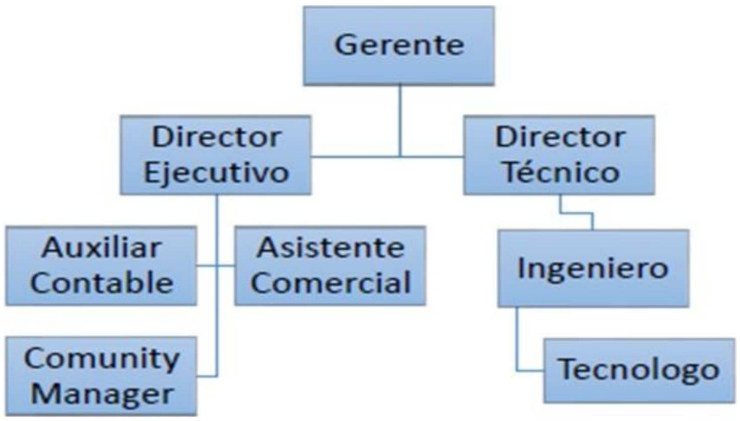


Figura 1: Organigrama Webcloster

5. Planteamiento del Problema

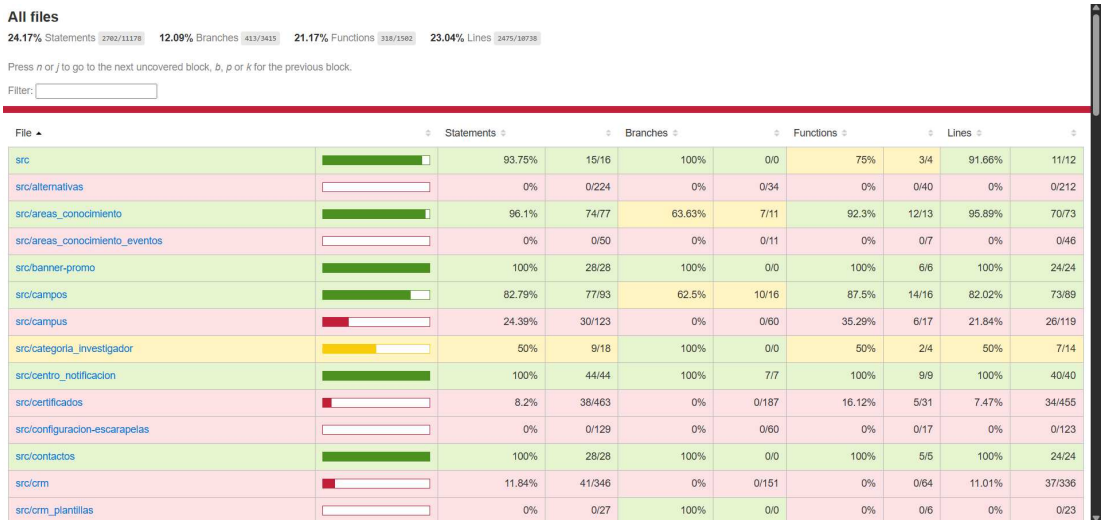
5.1. Descripción del problema

La aplicación Invessoft en la actualidad se define como un producto complejo y robusto que ha logrado evolucionar a través del tiempo, en la que se han implementado múltiples módulos funcionales en la parte del frontend y backend. No obstante, como es habitual en aplicaciones de gran tamaño, la cobertura de pruebas unitarias no es homogénea y completa, lo que puede complicar la pronta detección de errores y el mantenimiento de la aplicación.

La falta de documentación clara del estado actual de la actividad de las pruebas, así como la ausencia de una estrategia sistemática para abordar las pruebas faltantes, es un gran riesgo para la calidad del software. Este riesgo va aumentando en los proyectos que se encuentran en desarrollo activo, ya que la inclusión de última hora de nuevos requerimientos puede generar pequeñas modificaciones a partes del sistema que no están adecuadamente protegidas por pruebas automatizadas.

Dado que abarcar todo el sistema sería una opción poco realista en el contexto de la pasantía, por lo que se delimitará el ámbito de la práctica a módulos concretos, teniendo en cuenta la relevancia o criticidad de los mismos dentro de la aplicación.

Actualmente, la cobertura de pruebas unitarias con Jest en el backend de Invessoft es de un 24.17 % aproximadamente; si bien esta cobertura abarca tanto servicios como controladores, el porcentaje es bajo si tenemos en cuenta que es un sistema en producción, lo que significaría una amenaza cuando se trata de la confiabilidad y mantenibilidad del backend.



All files

24.17% Statements 2702/11178 **12.09%** Branches 413/3415 **21.17%** Functions 318/1502 **23.04%** Lines 2475/10738

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

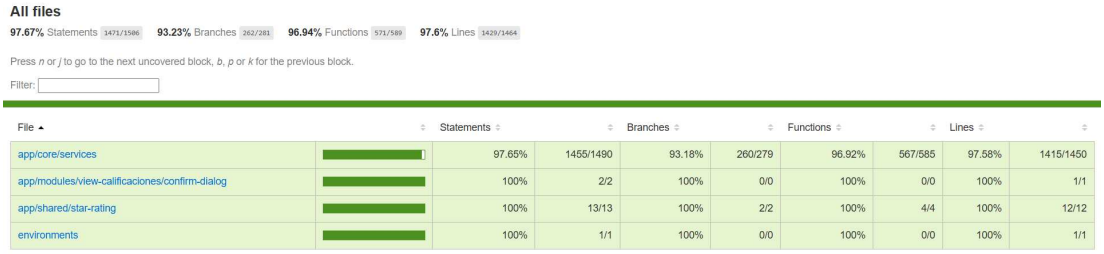
Filter:

File	Statements		Branches		Functions		Lines	
src	93.75%	15/16	100%	0/0	75%	3/4	91.66%	11/12
src/alternativas	0%	0/224	0%	0/34	0%	0/40	0%	0/212
src/areas_conocimiento	96.1%	74/77	63.63%	7/11	92.3%	12/13	95.89%	70/73
src/areas_conocimiento_eventos	0%	0/50	0%	0/11	0%	0/7	0%	0/46
src/banner-promo	100%	28/28	100%	0/0	100%	6/6	100%	24/24
src/campos	82.79%	77/93	62.5%	10/16	87.5%	14/16	82.02%	73/89
src/campus	24.39%	30/123	0%	0/60	35.29%	6/17	21.84%	26/119
src/categoria_investigador	50%	9/18	100%	0/0	50%	2/4	50%	7/14
src/centro_notificacion	100%	44/44	100%	7/7	100%	9/9	100%	40/40
src/certificados	8.2%	38/463	0%	0/187	16.12%	5/31	7.47%	34/455
src/configuracion-escarapelas	0%	0/129	0%	0/60	0%	0/17	0%	0/123
src/contactos	100%	28/28	100%	0/0	100%	5/5	100%	24/24
src/crm	11.84%	41/346	0%	0/151	0%	0/64	11.01%	37/336
src/crm_plantillas	0%	0/27	100%	0/0	0%	0/6	0%	0/23

Figura 2: Reporte de cobertura de pruebas del backend

Por el contrario, el frontend por medio de Jasmine y Karma supone una cobertura alcanzada del 97.67 %. No obstante, esta cifra se refiere únicamente a servicios, dado que no se encuentran siendo evaluados mediante pruebas unitarias los componentes visuales, formu-

larios, validaciones y flujos de navegación, lo que presenta una percepción errónea de alta cobertura, dejando fuera partes que tienen un impacto directo en la experiencia del usuario.



All files

97.67% Statements 1471/1506 **93.23%** Branches 262/281 **96.94%** Functions 571/589 **97.6%** Lines 1429/1464

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
app/core/services		97.65%	1455/1490	93.18%	260/279	96.92%	567/585	97.58%	1415/1450
app/modules/view-calificaciones/confirm-dialog		100%	2/2	100%	0/0	100%	0/0	100%	1/1
app/shared/star-rating		100%	13/13	100%	2/2	100%	4/4	100%	12/12
environments		100%	1/1	100%	0/0	100%	0/0	100%	1/1

Figura 3: Reporte de cobertura de pruebas del frontend (servicios)

5.2. Hipótesis

Para garantizar la calidad, mantenibilidad y detección temprana de errores en la aplicación Invessoft, es fundamental contar con una cobertura robusta y documentada de pruebas unitarias en sus módulos críticos.

5.2.1. Gestión del Estado Actual de Pruebas

Actualmente, no existe un informe consolidado que detalle el estado de las pruebas unitarias implementadas en los módulos de Invessoft, lo que dificulta identificar brechas de cobertura, priorizar áreas de mejora y asignar recursos eficientemente para su implementación.

5.2.2. Implementación de Pruebas Faltantes

La ausencia de una estrategia sistemática para desarrollar pruebas unitarias en módulos clave (especialmente aquellos con alta criticidad o frecuentes cambios) incrementa el riesgo de regresiones, errores no detectados y dificultades en el mantenimiento futuro.

5.2.3. Formulación del problema

¿Cómo diagnosticar el estado actual de las pruebas unitarias en Invessoft y, con base en este análisis, diseñar e implementar las pruebas faltantes en módulos críticos, para incrementar la confiabilidad y mantenibilidad del sistema?

6. Marco Referencial

6.1. Antecedentes

6.1.1. Open Conference System

Este software para la administración de ponencias permite la creación de eventos, redactar y enviar convocatorias de ponencias, permite a los usuarios editar sus trabajos, publicar las ponencias, formatearlas e indexarlas y registrar participantes. Actualmente, es un software que ya no recibe mantenimiento y es completamente legacy.

Figura 4: Open Conference System - Vista 1

Figura 5: Open Conference System - Vista 2

6.1.2. Programa interinstitucional para la investigación y posgrado del pacífico-Delfín.

El Programa Delfín, se creó en 1995 con el objetivo fortalecer la cultura de colaboración entre las Instituciones de Educación Superior y Centros de Investigación integrantes del Programa, a través de la movilidad de profesores-investigadores, estudiantes y de la divulgación de productos científicos y tecnológicos. En particular para fortalecer el desarrollo de la investigación y el posgrado nacional.[2].

Figura 6: Programa interinstitucional para la investigación y posgrado del pacífico - Delfín - Vista 1

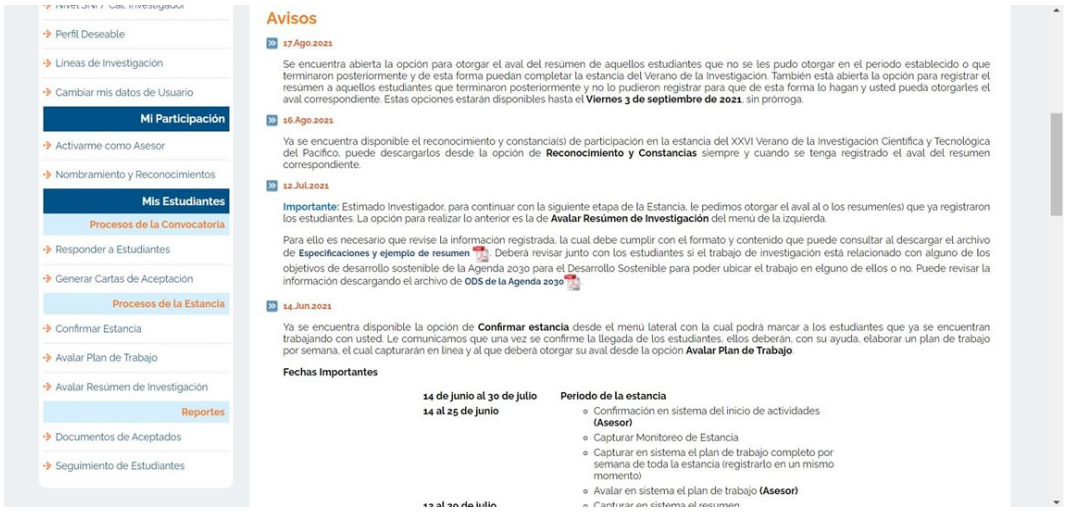


Figura 7: Programa interinstitucional para la investigación y posgrado del pacífico - Delfín - Vista 2

6.1.3. OnSite

OnSite, es una herramienta para la creación de webs para congresos, sociedades científicas, eventos corporativos y empresas. Soporta inscripciones, certificados, se puede acceder vía web o desde la aplicación móvil, también control de accesos e integra su propio sistema audiovisual.

6.1.4. Whova

Es un software de gestión de eventos que provee funcionalidades como la compra de entradas, informes, encuestas, gestión de exposiciones y proveedores, perfil de usuario, perfil de ponente, programación de eventos, streaming en vivo, entre otras características.

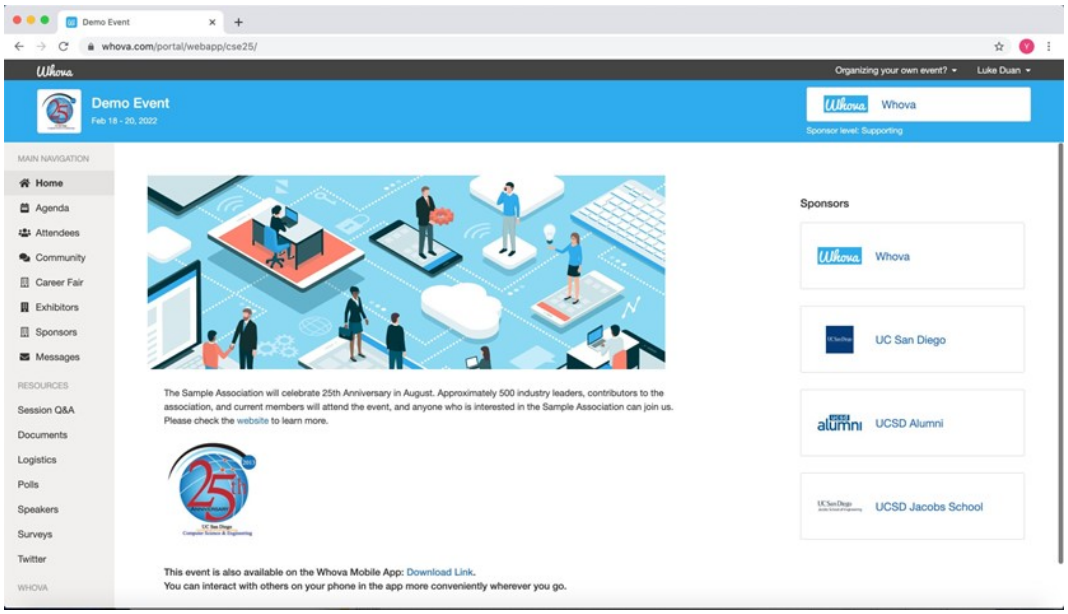


Figura 8: Whova

6.1.5. Microsoft Conference Management Toolkit (CMT)

Microsoft Conference Management Toolkit (CMT) es una plataforma en la nube respaldada por Microsoft Research, diseñada para la gestión eficiente de conferencias académicas. Gracias a su arquitectura escalable y su interfaz moderna, facilita la organización de procesos complejos en eventos de este tipo. [3]

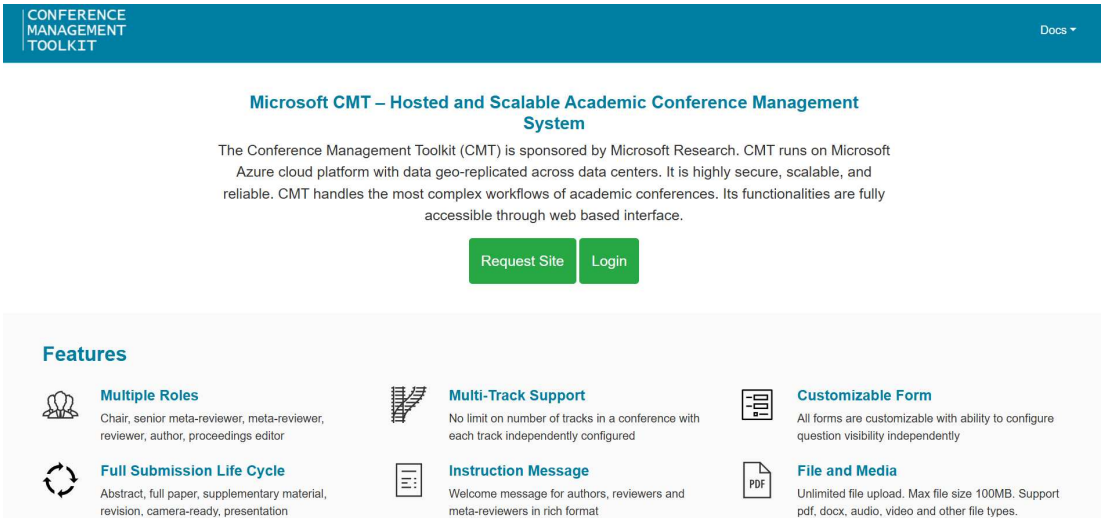


Figura 9: Microsoft CMT

6.1.6. Oxford Abstracts

Oxford Abstracts, por su parte, es una solución especializada en la gestión de congresos académicos, que agiliza el manejo de propuestas, revisiones y decisiones. Además, incluye funciones integradas para la venta de entradas y otras herramientas clave para optimizar la logística de eventos. Actualmente, es utilizada en más de 100 países. [4]

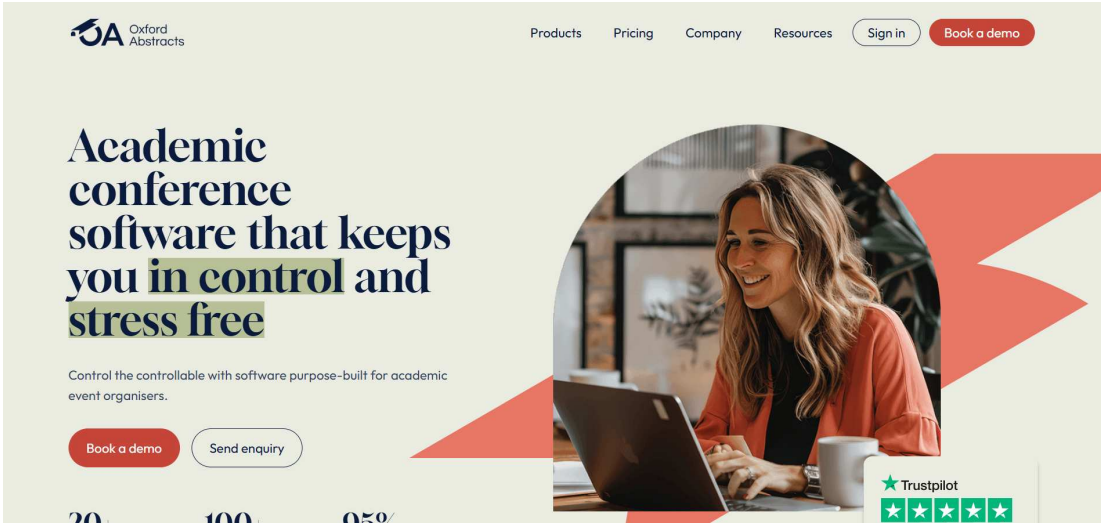


Figura 10: Oxford Abstracts

6.2. Tabla comparativa de funcionalidades de sistemas relacionados

La escala de valoración será atómica, es decir, posee, o no posee dicha característica.

Funcionalidades	**Programa Delfín**	**Open Conference System**	**Onsite**	**Whova**	**Microsoft CMT**	**Oxford Abstracts**
Gestión de ponencias		X	X	X	X	X
Inscripciones		X	X	X		X
Autenticación	X	X			X	X
Caducidad de sesiones	X				X	
Gestor de usuarios	X	X	X		X	X
Gestor de eventos		X				X
Control de accesos			X	X	X	X
Certificados	X	X				X
Audiovisual integrado			X	X		
Procesos de convocatoria	X				X	X
Procesos de estancia	X					
Reportes	X	X			X	X

Tabla 1: Comparativa de funcionalidades de sistemas relacionados

6.3. Marco teórico

6.3.1. Sistemas de Información

En el ámbito de los sistemas de información, R. Andreu, J. Ricart y J. Valor (en su libro "Estrategia y sistema de información") [5] definen estos sistemas como: "Un conjunto formal de procesos que, operando sobre una colección de datos estructurada de acuerdo a las necesidades de la empresa, recopila, elabora y distribuye selectivamente la información necesaria para la operación de dicha empresa y para las actividades de dirección y control correspondientes, apoyando, al menos en parte, los procesos de toma de decisiones necesarios para desempeñar funciones de negocio de la empresa de acuerdo con su estrategia".

En el contexto de este estudio, se propone la actualización de la arquitectura y tecnología de los sistemas de información, específicamente mediante la migración del backend a Nestjs y el frontend a Angular. Estas decisiones de migración se basan en la construcción de una arquitectura orientada a componentes, que ofrece numerosas ventajas en términos de modularidad, reutilización de código y escalabilidad. [5]

6.3.2. Sistemas de información Gerencial

Los Sistemas de Información Gerencial (SIG) pueden definirse como un conjunto integrado de componentes, que tiene el objetivo de recolectar, almacenar, procesar y proporcionar datos y cualquier otro tipo de producto digital [6].

6.3.3. Aplicaciones Web

Una web app (aplicación web en español) se basa en HTML, JavaScript y CSS. Dado que se aloja en el servidor web y se ejecuta en un navegador web, no es necesario instalar una instalación previamente. Además, se puede usar un marcador para acceder directamente a ella en el escritorio del ordenador o en la pantalla de inicio de los terminales móviles.

Las aplicaciones web ofrecen una amplia gama de posibilidades y se adaptan a diferentes necesidades y sectores. Desde pequeñas herramientas hasta software de gráficos y juegos de navegador; estas aplicaciones han demostrado su versatilidad y han sido utilizadas en servicios de mensajería instantánea, paquetes de Office y otras aplicaciones conocidas. A menudo, se presentan en dos modalidades: Web Apps y Native Apps, cada una con sus propias características y consideraciones. [7]

6.3.4. Test-Driven Development (TDD)

El Desarrollo Guiado por Pruebas (TDD, por sus siglas en inglés) es una metodología de desarrollo ágil que se basa en escribir primero pruebas automatizadas antes de implementar el código que las hace pasar. Esta técnica promueve la simplicidad del diseño, reduce errores y mejora la mantenibilidad del software. Consiste en un ciclo repetitivo de tres pasos: escribir una prueba que falle, escribir el código mínimo necesario para que pase, y luego refactorizar ese código manteniendo las pruebas en verde. [8]

Kent Beck, uno de los pioneros de las metodologías ágiles, formalizó el proceso de TDD como una práctica fundamental para lograr código de alta calidad desde el inicio del desarrollo. En su enfoque, las pruebas no solo verifican que el sistema funcione, sino que también guían su diseño y evolución. [8]

6.4. Marco conceptual

6.4.1. Experiencia de usuario (UX)

Norma ISO 9241-210:2010(E) La experiencia de usuario se refiere al diseño en un sistema que permite ser usable y útil, centrándose en las necesidades y requisitos, mientras conserva factores ergonómicos del usuario.

La experiencia de usuario en el ámbito del software consiste en su mayoría a interacción del usuario con la interfaz gráfica de la aplicación, valorando el tiempo de respuesta, funcionalidades interactivas, versatilidad funcional, entre otras características que buscan aumentar la satisfacción del usuario al navegar por la aplicación. [9]

6.4.2. Refactorización

La refactorización es una técnica de ingeniería de software que consiste en reestructurar el código fuente para mejorar su claridad, consistencia y mantenibilidad, sin alterar su comportamiento externo calidad-software. Se realiza frecuentemente como parte del mantenimiento y desarrollo iterativo, empleando buenas prácticas como renombrar variables, extraer funciones o eliminar redundancias, todo ello respaldado por pruebas que garantizan que no se introducen errores. [10]

6.4.3. Pruebas Unitarias

Las pruebas unitarias son un método de evaluación en el desarrollo de software que se enfoca en verificar el comportamiento de cada parte individual de un programa por separado. Estas partes suelen ser los elementos más básicos del código, como funciones o procedimientos, con el objetivo de asegurar que operen correctamente de manera independiente. [11]

6.5. Pruebas de concepto

Para comprobar la viabilidad técnica del proyecto y validar que se cuenta con los conocimientos necesarios para su desarrollo, durante la etapa de prácticas profesionales en Webcloster SAS se realizaron diversas tareas en el sistema Invessoft, utilizando los frameworks NestJS para el backend y Angular para el frontend. Estas actividades no solo sirvieron como aporte al desarrollo y mantenimiento de la plataforma, sino también como base práctica para emprender el proceso de implementación de pruebas unitarias como parte del presente proyecto.

A continuación, se describen algunos de los módulos y funcionalidades abordadas como parte de estas pruebas de concepto:

6.5.1. Reestructuración e implementación de endpoints

Durante las prácticas, se trabajó activamente en la reestructuración y creación de endpoints en el backend, cuidando siempre el cumplimiento de buenas prácticas y la correcta documentación mediante Swagger, garantizando así una comunicación clara y mantenible con el frontend y otros servicios.

6.5.2. Modificaciones en base de datos

Se realizaron tareas de creación y modificación de entidades en la base de datos, ajustando estructuras existentes o generando nuevas tablas para soportar funcionalidades requeridas. Esto incluyó el diseño de relaciones, validaciones y adaptaciones necesarias para que los datos fluyeran correctamente entre los distintos módulos del sistema.

6.5.3. Ajustes en módulos funcionales

Se colaboró en la corrección de errores en diferentes módulos como los KPIs, la tabla de auditoría para registrar retiros de eventos y problemas relacionados con fechas y horas debido a diferencias horarias en el servidor. Estos ajustes ayudaron a estabilizar el sistema y evitar inconsistencias en la visualización de información.

6.5.4. Mejora en experiencia de usuario

Desde el lado del frontend, se llevaron a cabo reestructuraciones en selectores, implementación de filtros de búsqueda y paginación para mejorar la experiencia de navegación y el rendimiento en el consumo de datos. También se realizaron mejoras funcionales como la posibilidad de agregar trabajos de un evento a favoritos o preinscribirse a talleres, respondiendo a necesidades específicas del sistema.

6.5.5. Pruebas unitarias

Como parte del fortalecimiento de la calidad del software, se iniciaron procesos de implementación de pruebas unitarias sobre algunas de las funcionalidades nuevas o modificadas. Esto sirvió como experiencia directa para comprender la estructura del código, la manera en que se organizan los tests en NestJS y Angular, y los retos particulares que enfrenta el sistema.

7. Declaración del Alcance

Este proyecto se enfoca en evaluar y mejorar las pruebas unitarias de la aplicación Invessoft, desarrollada por Webcloster SAS, en los siguientes aspectos:

7.1. Diagnóstico del estado actual de pruebas

- Realizar un análisis de la cobertura de pruebas existentes en el backend (NestJS + Jest) y frontend (Angular + Jasmine + Karma).
- Identificar módulos con baja cobertura pero alta criticidad.
- Documentar brechas y riesgos asociados a la falta de pruebas.

7.2. Implementación de pruebas unitarias en módulos priorizados

7.2.1. Backend (NestJS/Jest)

- Implementar pruebas para los servicios y lógica de negocio más relevantes del sistema.
- Asegurar el correcto funcionamiento de las APIs principales y los flujos críticos de datos.

7.2.2. Frontend (Angular/Jasmine/Karma)

- Evaluar y ampliar las pruebas de componentes y servicios clave del frontend.
- Asegurar que las funcionalidades principales tengan una cobertura de pruebas adecuada.

7.2.3. Documentación y métricas de mejora

- Generar un informe con el estado inicial y final de la cobertura de pruebas.
- Entregar recomendaciones para mantener y escalar las pruebas en el futuro.

7.3. Restricciones

7.3.1. Dependencias técnicas

- Las pruebas se adaptarán al código existente sin modificar su funcionalidad.
- Se desarrollarán bajo los estándares de Webcloster SAS y las directrices del líder técnico de Invessoft.
- En módulos con deuda técnica alta, se documentarán las limitaciones para pruebas exhaustivas.

7.3.2. Compatibilidad tecnológica

- Las pruebas deberán ajustarse a las tecnologías y herramientas ya implementadas (Jest, Karma, etc.).

7.3.3. Cobertura y tiempo

- El proyecto se limitará al período de la pasantía, priorizando módulos críticos.

8. Objetivos

8.1. Objetivo General

Implementar pruebas unitarias en la plataforma invessoft en sus componentes backend y frontend.

8.2. Objetivos específicos

1. Documentar el estado actual de las pruebas unitarias existentes en la aplicación Invessoft, tanto en el backend como en el frontend.

2. Determinar, en conjunto con el equipo de desarrollo, los módulos prioritarios en los que se enfocará el trabajo.

3. Codificar pruebas unitarias para los módulos seleccionados, siguiendo buenas prácticas de desarrollo y asegurando una cobertura adecuada.

8.3. Relación entre los objetivos específicos y los resultados.

Tabla 2: Relación entre los objetivos específicos y los resultados

Objetivo específico	Resultados
Documentar el estado actual de las pruebas unitarias existentes en la aplicación Invessoft, tanto en el backend como en el frontend.	▪ Informe técnico detallado con el análisis de cobertura de pruebas. ▪ Identificación de módulos críticos con baja cobertura. ▪ Registro de riesgos asociados a la falta de pruebas.
Determinar, en conjunto con el equipo de desarrollo, los módulos prioritarios en los que se enfocará el trabajo.	▪ Listado consensuado de módulos priorizados para la implementación de pruebas. ▪ Justificación técnica de la priorización basada en criticidad y riesgo.
Codificar pruebas unitarias para los módulos seleccionados, siguiendo buenas prácticas de desarrollo y asegurando una cobertura adecuada.	▪ Casos de prueba implementados y validados en backend (NestJS/Jest). ▪ Casos de prueba implementados y validados en frontend (Angular/Jasmine/Karma). ▪ Comparativa entre cobertura inicial y final de pruebas. ▪ Código más confiable y mantenible.

9. Metodología

9.1. Metodología de Investigación

La propuesta desarrollada en el presente trabajo se enmarca en una metodología de investigación aplicada, debido a que parte de una necesidad real identificada en un entorno empresarial concreto y se orienta a la construcción de una solución técnica con impacto directo sobre la calidad del software. En este caso, el problema abordado no se limitó a una formulación conceptual, sino que exigió intervenir un producto en uso, revisar su estado actual, identificar debilidades en el aseguramiento de calidad y plantear acciones de mejora viables dentro del contexto operativo de Invessoft.

Desde esta perspectiva, la pasantía constituyó el escenario principal para observar el comportamiento del sistema, comprender la dinámica del equipo de desarrollo y recoger evidencia sobre los módulos que presentaban mayor exposición al cambio. La experiencia diaria en el proyecto permitió contrastar los objetivos del trabajo de grado con la realidad del mantenimiento evolutivo del software, de manera que la investigación no se desarrolló al margen de la práctica profesional, sino integrada a ella. Este enfoque hizo posible formular el problema, delimitar los módulos críticos y definir una estrategia de intervención sustentada en situaciones verificables del entorno de trabajo.

La figura que se presenta a continuación muestra los niveles de madurez tecnológica (TLR) según Minciencias. Estos niveles representan una referencia útil para ubicar el tipo de desarrollo realizado, ya que permiten relacionar investigación, desarrollo tecnológico e implementación en contextos reales. Con base en esta escala, puede afirmarse que Invessoft se sitúa en una etapa de madurez suficiente para ser intervenida, evaluada y mejorada dentro de un entorno de desarrollo activo, condición que resulta coherente con la naturaleza aplicada del presente trabajo.

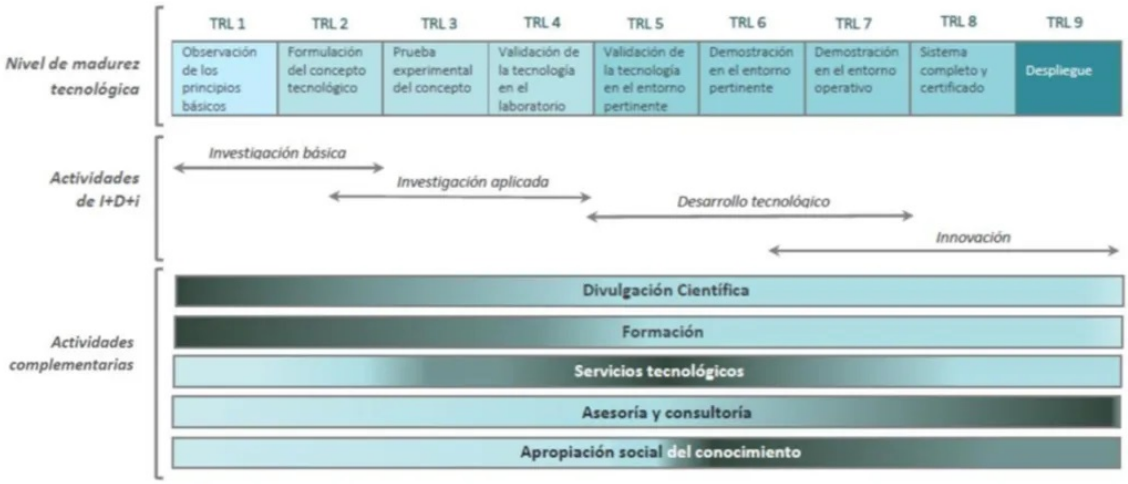


Figura 11: Niveles de maduración tecnológica — Fuente: Minciencias 2021

9.2. Metodología de trabajo aplicada durante la pasantía

La organización del trabajo durante la pasantía no respondió a una metodología rígida aplicada de manera puramente normativa, sino a una dinámica de trabajo ágil ajustada a las necesidades reales del equipo de desarrollo. En la práctica, el proceso combinó elementos de planeación iterativa, seguimiento visual de tareas, revisión frecuente de avances y adaptación continua a cambios de prioridad. Esta forma de trabajo permitió atender tanto actividades nuevas de desarrollo como correcciones, ajustes funcionales, revisiones de calidad y requerimientos emergentes del sistema.

La lógica de trabajo adoptada por el equipo puede entenderse como una aplicación práctica de ScrumBan. Por una parte, se mantuvo una organización por ciclos cortos de trabajo, con reuniones periódicas para acordar prioridades, revisar avances y detectar bloqueos. Por otra, el seguimiento del proyecto se apoyó en un tablero visual que permitía observar el estado de cada tarea, identificar cuellos de botella y mantener trazabilidad sobre el paso de las actividades por diferentes etapas. Esta combinación resultó adecuada para el contexto de la pasantía, ya que Invessoft se encontraba en evolución constante y requería una metodología suficientemente ordenada para coordinar al equipo, pero también lo bastante flexible para responder a cambios durante la semana.

9.2.1. Dinámica semanal de trabajo

El trabajo se estructuró en ciclos semanales que funcionaban como unidades breves de planificación, ejecución y revisión. Al inicio de cada semana se realizaban reuniones de planificación con el equipo de desarrollo, en las cuales se revisaban las prioridades vigentes del producto, se asignaban tareas, se estimaba el esfuerzo requerido y se definían los objetivos

inmediatos del periodo. Estas reuniones no se limitaban a distribuir actividades, sino que también permitían aclarar requerimientos, discutir implicaciones técnicas y anticipar posibles dependencias entre módulos.

Durante la ejecución semanal, el desarrollo se complementaba con espacios de seguimiento en los que se informaban avances, se exponían dificultades y se reajustaban prioridades cuando surgían nuevas necesidades del proyecto. Al cierre del ciclo se realizaba una revisión de lo ejecutado, orientada a valorar qué tareas habían sido completadas, cuáles quedaban en proceso y qué aspectos debían corregirse o replantearse para la siguiente semana. En términos prácticos, esta dinámica favoreció una gestión continua del trabajo, más cercana al comportamiento real del proyecto que a una planificación cerrada de largo plazo.

Las reuniones también cumplieron una función importante de alineación técnica, ya que permitían compartir contexto sobre módulos intervenidos, dependencias con otros desarrolladores y decisiones que podían afectar el comportamiento del sistema en etapas posteriores. La Figura 12 corresponde a un espacio de planificación del trabajo semanal, en el que se discutían tareas, prioridades y consideraciones técnicas antes de iniciar la ejecución. De forma complementaria, la Figura 13 muestra un escenario de seguimiento y revisión, utilizado para validar el estado de una tarea y comentar avances sobre funcionalidades en desarrollo.

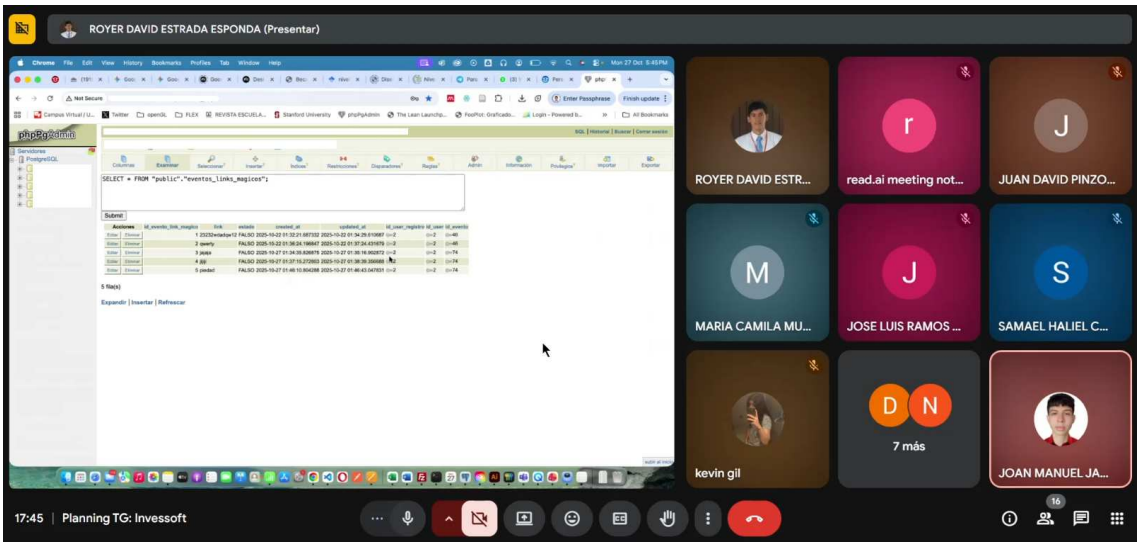


Figura 12: Reunión de planificación del trabajo semanal en la que se revisaban prioridades y tareas a desarrollar durante el ciclo de trabajo

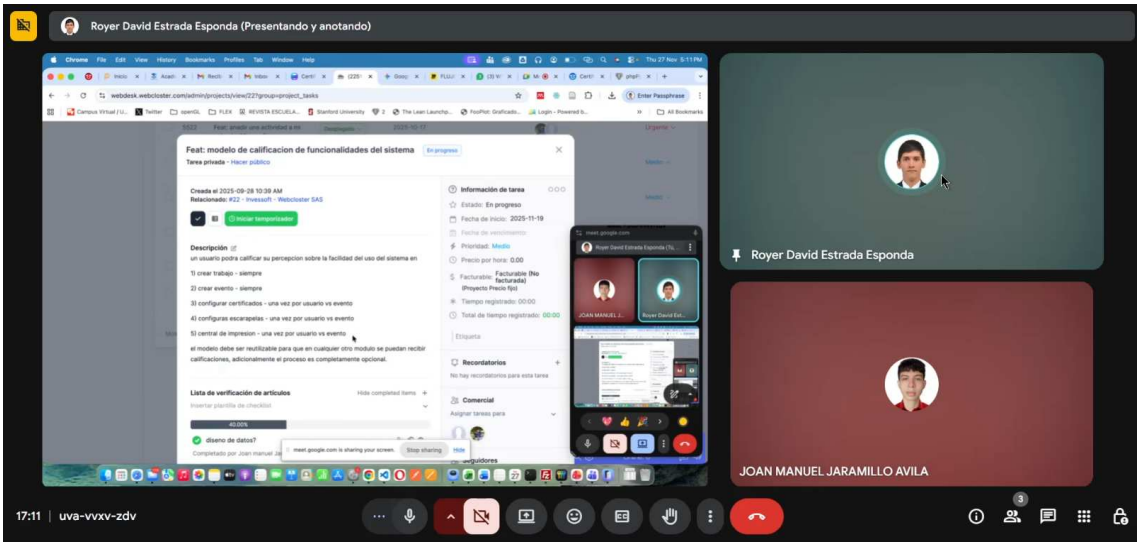


Figura 13: Espacio de seguimiento y revisión utilizado para validar avances, aclarar requerimientos y ajustar tareas en curso

9.2.2. Seguimiento del trabajo mediante Webdesk

La gestión operativa de las actividades se apoyó en Webdesk, herramienta empleada por el equipo para registrar, organizar y hacer seguimiento a las tareas del proyecto. El uso de este tablero permitió mantener una visión compartida del estado del trabajo, asociando cada requerimiento con una tarjeta específica que contenía información sobre prioridad, responsables, comentarios, avances y validaciones. Esta trazabilidad fue especialmente útil en un contexto como el de la pasantía, donde varias tareas podían estar relacionadas entre sí o requerir ajustes posteriores a una primera entrega.

El tablero funcionaba como un mecanismo de coordinación visible entre desarrollo, revisión y despliegue. A través de sus columnas era posible reconocer qué tareas estaban pendientes de iniciar, cuáles habían sido priorizadas, cuáles se encontraban en desarrollo y cuáles estaban listas para ser validadas o incorporadas a ambientes posteriores. Este modelo reducía la dispersión del trabajo y ayudaba a conservar foco sobre los requerimientos más urgentes. De igual forma, facilitaba la comunicación entre integrantes del equipo, pues el estado de cada actividad podía consultarse sin depender exclusivamente de reportes verbales.

La Figura 14 muestra una vista del tablero empleada para la organización de tareas en etapas tempranas del flujo, con columnas asociadas a actividades por iniciar, tareas priorizadas, elementos pausados y desarrollos en progreso. Por su parte, la Figura 15 presenta una vista orientada a etapas posteriores del proceso, donde se observa el tránsito hacia estados vinculados con pruebas, finalización y despliegue. En conjunto, estas imágenes permiten apreciar que la metodología aplicada durante la pasantía no solo dependía de reuniones periódicas, sino también de un soporte visual permanente para administrar el avance real del proyecto.

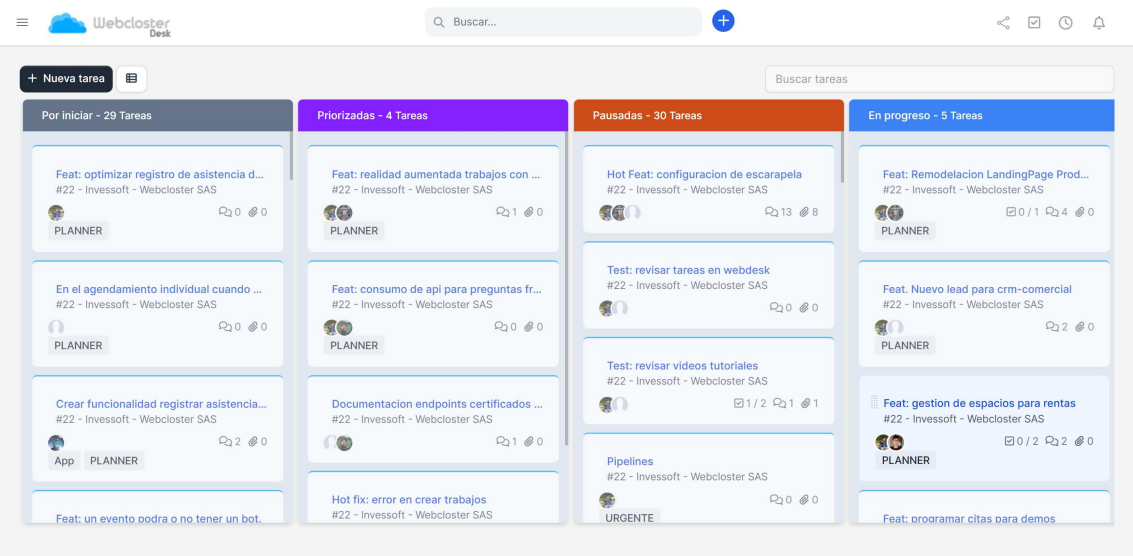


Figura 14: Tablero Webdesk utilizado para la priorización y seguimiento inicial de tareas, con énfasis en estados de preparación y ejecución

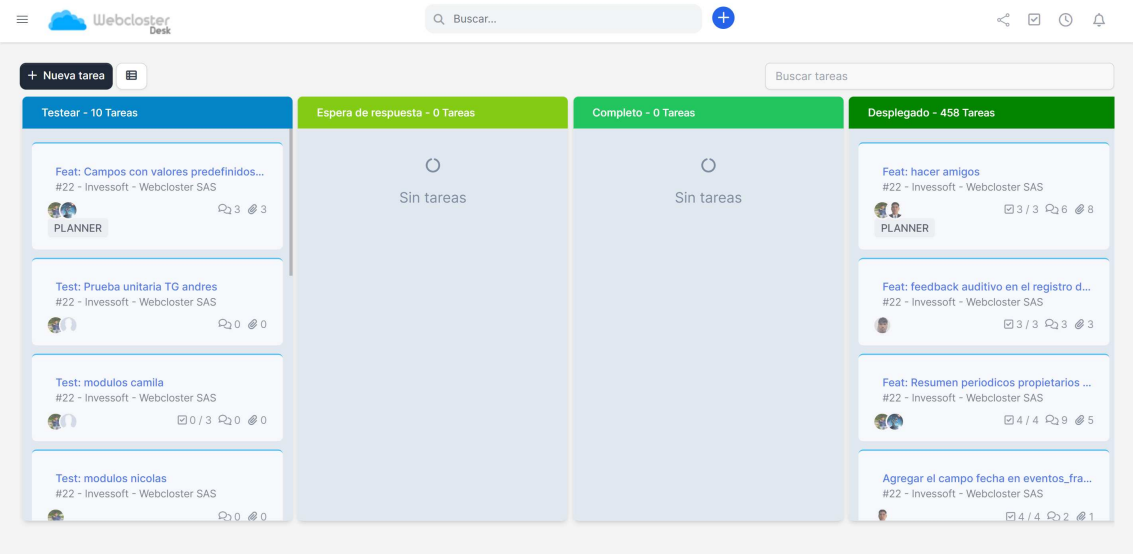


Figura 15: Tablero Webdesk en una etapa posterior del flujo de trabajo, con tareas en validación, cierre y despliegue

9.2.3. Ciclo de desarrollo, validación y ajuste

La metodología aplicada durante la pasantía no terminaba con la implementación de una tarea. Una vez desarrollada una funcionalidad o una corrección, el resultado pasaba por una etapa de revisión funcional en la que se verificaba su comportamiento frente a los criterios esperados. Este proceso permitía detectar inconsistencias tempranas, registrar observaciones y devolver ajustes cuando era necesario. En consecuencia, el ciclo de trabajo se entendía como una secuencia continua de desarrollo, validación y refinamiento, más que como una

entrega lineal cerrada en el momento de programar.

Esta dinámica fue especialmente relevante para el enfoque del trabajo de grado, ya que hizo visible la necesidad de fortalecer mecanismos de aseguramiento de calidad en módulos que cambiaban con frecuencia. La revisión funcional del equipo, los comentarios sobre hallazgos y la necesidad de corregir comportamientos no esperados mostraron que el proceso de desarrollo dependía en buena medida de la capacidad para verificar cada cambio antes de considerarlo estable. Por ello, la experiencia metodológica de la pasantía se relaciona directamente con el objetivo del proyecto: comprender cómo se trabaja en un entorno real para proponer mejoras concretas en pruebas unitarias, cobertura y mantenibilidad.

9.2.4. Documentación, trazabilidad y aprendizaje continuo

Otro componente importante de la metodología aplicada fue la documentación del trabajo. Cada tarea conservaba información sobre su descripción, prioridad, estado, observaciones y antecedentes de revisión, lo que facilitaba reconstruir el contexto de implementación incluso después de varios ciclos de trabajo. Esta trazabilidad fue útil no solo para la coordinación del equipo, sino también para el desarrollo del presente documento, ya que permitió identificar módulos intervenidos, reconocer patrones de cambio y relacionar decisiones técnicas con necesidades reales del sistema.

Asimismo, la metodología de trabajo favoreció un aprendizaje continuo durante la pasantía. La participación en reuniones, la revisión constante de tareas, la corrección de hallazgos y la interacción con módulos de distinta naturaleza contribuyeron a construir una comprensión más amplia de Invessoft como producto de software. En ese sentido, la metodología no fue únicamente una forma de organizar actividades, sino también el marco operativo que permitió pasar de la observación del problema a una intervención técnica sustentada en evidencia del trabajo cotidiano.

10. Desarrollo del proyecto

El desarrollo del proyecto se orientó a diagnosticar el estado real de las pruebas unitarias en Invessoft y, a partir de dicho diagnóstico, intervenir módulos con alta exposición funcional para fortalecer su confiabilidad y mantenibilidad. No obstante, el proceso no se limitó a los módulos donde finalmente se concentró la implementación más intensa de pruebas. La pasantía se desarrolló en paralelo con actividades reales de mantenimiento evolutivo y correctivo sobre distintos frentes del sistema, tales como CRM, agenda académica, perfiles, certificados, landings, trabajos, reportes y nuevos módulos funcionales. Por esta razón, la presente sección integra esas intervenciones dentro de una lectura técnica unificada: las tareas ejecutadas no se presentan como un inventario aislado, sino como evidencia del comportamiento dinámico del sistema y del tipo de problemas que justifican una estrategia de aseguramiento más robusta.

Desde esta perspectiva, las actividades realizadas durante la pasantía cumplieron un doble papel. Por un lado, aportaron valor directo al producto mediante correcciones, refactorizaciones, mejoras de experiencia de usuario e incorporación de funcionalidades. Por otro, sirvieron como base empírica para reconocer patrones de fragilidad técnica, dependencias críticas y flujos con alta probabilidad de regresión. Así, el desarrollo del proyecto se entiende como la articulación entre la evolución funcional de Invessoft y la necesidad de construir criterios de calidad verificables mediante pruebas unitarias.

10.1. Diagnóstico inicial del sistema

El punto de partida del proyecto fue el reconocimiento de una situación desigual en materia de pruebas automatizadas. En el backend, la cobertura inicial reportada mediante Jest era de aproximadamente 24.17 %, lo que evidenciaba un bajo nivel de protección frente a regresiones en servicios y controladores que soportan lógica de negocio sensible. En contraste, el frontend presentaba un reporte cercano al 97.67 %; sin embargo, dicha cifra correspondía principalmente a servicios, por lo que no representaba con fidelidad el comportamiento de componentes visuales, formularios, validaciones ni flujos de interacción. En consecuencia, el problema no era únicamente cuantitativo, sino también cualitativo: existía una percepción de cobertura aceptable en capas que no necesariamente estaban siendo verificadas en los puntos donde el usuario final y la lógica de negocio se acoplan con mayor intensidad.

El diagnóstico se apoyó en tres fuentes complementarias. En primer lugar, se revisó la estructura del código y la distribución de responsabilidades entre módulos de NestJS y Angular, con el fin de identificar unidades con alta complejidad condicional, dependencia de servicios externos o manejo intensivo de estados de interfaz. En segundo lugar, se analizaron los módulos y flujos funcionales intervenidos durante la pasantía, ya que estos mostraban con claridad las áreas que estaban siendo modificadas con mayor frecuencia y, por tanto, los sectores donde el riesgo de regresión era más alto. En tercer lugar, se contrastó este panorama con la experiencia obtenida en las pruebas de concepto descritas en la Sección 4.5, las cuales permitieron comprender el dominio funcional del sistema.

A partir de este análisis se determinó que los mayores riesgos se concentraban en módulos que reunían alguna de las siguientes características: formularios con validación condicional, flujos de confirmación de múltiples pasos, reglas de negocio asociadas a créditos o cupos, operaciones con notificación por correo, integración frontend-backend basada en DTOs, renderizado dinámico de información en landings y componentes con interacción intensiva del usuario. La diversidad de tareas ejecutadas durante la pasantía confirmó que Invessoft no presentaba un único foco de fragilidad, sino varios subsistemas en evolución continua: gestión de eventos, trabajos académicos, agenda, CRM, perfiles de usuario, certificación, encuestas de percepción y nuevos servicios como espacios arrendables. Bajo estos criterios, el proyecto no se enfocó en aumentar cobertura de forma indiscriminada, sino en intervenir áreas donde una falla pequeña podía propagarse a funcionalidades de alto impacto operativo y, al mismo tiempo, tomar el resto de módulos trabajados como sustento para caracterizar la deuda técnica del sistema.

La evidencia del diagnóstico inicial se apoya en los reportes de cobertura presentados previamente en la Figura 2 para el backend y en la Figura 3 para el frontend, junto con una tabla comparativa de módulos con mayor exposición al cambio.

10.2. Componentes o módulos priorizados

La priorización de módulos respondió a una lógica de criticidad funcional y de oportunidad técnica, pero también al comportamiento real de la cobertura observada durante la pasantía. En el backend, se tomaron como referentes prioritarios los módulos que destacan en el reporte final de cobertura, entre ellos `banner-promo`, `categoria_investigador`, `centro_notificacion`, `contactos`, `estadisticas_landing`, `modalidades_eventos`, `personas_eventos_fungibles`, `video-url`, `areas_conocimiento`, `perfil-usuario`, `encuestas`, `eventos_campos`, `campos` y `espacios`. La selección de estos módulos permitió concentrar el análisis en unidades donde ya existía un esfuerzo visible de aseguramiento y donde resultaba posible documentar con mayor claridad qué tipo de reglas de negocio, validaciones y flujos estaban siendo protegidos por pruebas automatizadas.

En el frontend, la estrategia de priorización se desarrolló en dos momentos. Inicialmente, se tomó la decisión de implementar pruebas unitarias sobre la totalidad de los servicios, debido a que en su mayoría encapsulan funciones de consumo de API y, por tanto, ofrecían un punto de partida más estable y rápido para incrementar cobertura base. Posteriormente, una vez consolidado ese frente, se definió avanzar hacia componentes con mayor criticidad funcional y mayor probabilidad de regresión, entre ellos `dashboard`, `eventos`, `evento-pagos`, `editar-perfil`, `espacios-gestion`, `mis-eventos`, `reservas-espacios`, `salas-remotas`, `centro-notificaciones`, `facturacion-eventos`, `perfil-evento`, `perfil-publico`, `respuesta-epayco` y `respuesta-wompi`. Esta transición fue importante porque permitió pasar de una cobertura inicialmente concentrada en servicios a una cobertura más representativa del comportamiento real de la interfaz.

Sin embargo, la priorización técnica no excluyó otros módulos trabajados durante la pasantía. También se consideraron como líneas funcionales relevantes la gestión de eventos

y landings, los flujos de agenda y talleres, la administración de trabajos académicos, la gestión de personas y perfiles, los mecanismos de certificación e insignias, y los componentes orientados a analítica y percepción de usabilidad. Aunque no todos estos frentes recibieron el mismo nivel de profundización en la implementación de pruebas, sí fueron incorporados al análisis porque aportan evidencia sobre el tipo de cambios que experimenta Invessoft y sobre la necesidad de extender progresivamente la estrategia de testing hacia módulos donde hoy predominan ajustes visuales, filtros, estados derivados y validaciones contextuales.

En términos funcionales, la priorización se agrupó en cinco frentes. El primero correspondió al fortalecimiento de cobertura en servicios y módulos backend con mayor avance observable en el reporte final. El segundo se orientó a componentes frontend críticos para la operación diaria del sistema, especialmente aquellos asociados a paneles, gestión de eventos, pagos, perfiles, reservas y notificaciones. El tercero abarcó agenda, landings y trabajos académicos, por tratarse de un conjunto de vistas públicas y privadas con alta interacción de usuario y frecuentes ajustes de negocio. El cuarto incluyó perfiles, personas e instituciones, donde pequeñas inconsistencias afectan la calidad de los datos que alimentan otros módulos. Finalmente, el quinto frente reunió certificados, insignias, reportes y encuestas de percepción, debido a su impacto en la trazabilidad, experiencia de usuario y explotación de información del sistema.

10.3. Stack técnico

El proyecto se desarrolló sobre el mismo stack tecnológico de Invessoft, lo que permitió que la estrategia de pruebas se integrara al flujo normal de evolución del sistema. En el backend se trabajó con NestJS como framework principal, apoyado en Jest para la ejecución de pruebas unitarias, generación de cobertura y validación de comportamiento en servicios, controladores y capas auxiliares. La organización por módulos, controladores, servicios, DTOs y entidades facilitó el aislamiento de dependencias mediante *mocks* y la verificación de reglas de negocio sin necesidad de ejecutar integraciones completas.

En el frontend se utilizó Angular como base de construcción de la interfaz, junto con Jasmine y Karma para la definición y ejecución de pruebas unitarias orientadas a componentes y servicios. Este entorno permitió validar formularios reactivos, comportamiento condicional de plantillas, apertura de diálogos, control de navegación, estados de autenticación y respuestas ante errores. De manera complementaria, se aprovechó la estructura de `spec.ts` por componente para expandir la cobertura sobre unidades que previamente no estaban suficientemente protegidas.

Como soporte transversal se emplearon Git como mecanismo de trazabilidad técnica, Swagger como referencia para contratos de API, reportes de cobertura para medir impacto y los propios comentarios de revisión de código como insumo para identificar puntos frágiles o comportamientos que requerían aseguramiento adicional. Aunque estas herramientas no sustituyen el diseño de pruebas, sí aportaron un marco de validación continua que hizo posible relacionar cambios funcionales con evidencia objetiva de calidad.

10.4. Arquitectura y organización técnica

La arquitectura de Invessoft condicionó directamente la estrategia de pruebas. En el backend, la solución sigue una organización modular en la que cada dominio funcional encapsula controlador, servicio, DTOs y entidades. Esta separación favorece el diseño de pruebas unitarias porque permite verificar, por un lado, la lógica de negocio contenida en los servicios y, por otro, la consistencia de entradas y salidas modeladas mediante DTOs y respuestas estructuradas. En los módulos intervenidos, especialmente aquellos con validaciones, relaciones entre entidades y efectos secundarios, fue necesario aislar repositorios, servicios de correo, configuración por entorno y dependencias de autenticación para concentrar la prueba en la conducta esperada de la unidad.

En el frontend, la organización por módulos, componentes, servicios y diálogos reutilizables implicó que una parte importante del esfuerzo de pruebas se orientara a la interacción entre vista y lógica de presentación. No se trataba solamente de comprobar que un servicio devolviera datos, sino de verificar que formularios, botones, mensajes, filtros, selectores y ventanas modales respondieran correctamente a combinaciones diversas de estados. Esto fue particularmente relevante en módulos como agenda de eventos, preinscripción a talleres, consulta de certificados, landings de eventos, trabajos, reservas y perfiles, donde la experiencia del usuario depende de información parametrizada, permisos, horarios, estados de publicación y reglas de visibilidad.

Desde la perspectiva del testing, esta arquitectura llevó a adoptar dos principios de organización. El primero consistió en probar unidades donde la lógica de decisión es dominante, incluso si el efecto visible termina manifestándose en la interfaz. El segundo fue mantener alineación entre contratos backend y modelos frontend, de modo que cada cambio en entidades, DTOs o modelos de consumo pudiera rastrearse también en los componentes y servicios involucrados. Este principio resultó relevante tanto en módulos con reglas específicas de negocio como en vistas con alto nivel de interacción, formularios y estados derivados.

La Figura 16 resume esta organización técnica mediante una vista por capas, en la que se distingue la presentación en Angular, la comunicación mediante API REST, la lógica de aplicación implementada en NestJS, la persistencia de datos y la relación de estas capas con los módulos funcionales más representativos del sistema. Aunque se trata de una síntesis visual, el esquema resulta útil para mostrar cómo la separación entre componentes, servicios, controladores, DTOs, entidades y persistencia favorece tanto la mantenibilidad como la definición de pruebas unitarias sobre responsabilidades bien delimitadas.

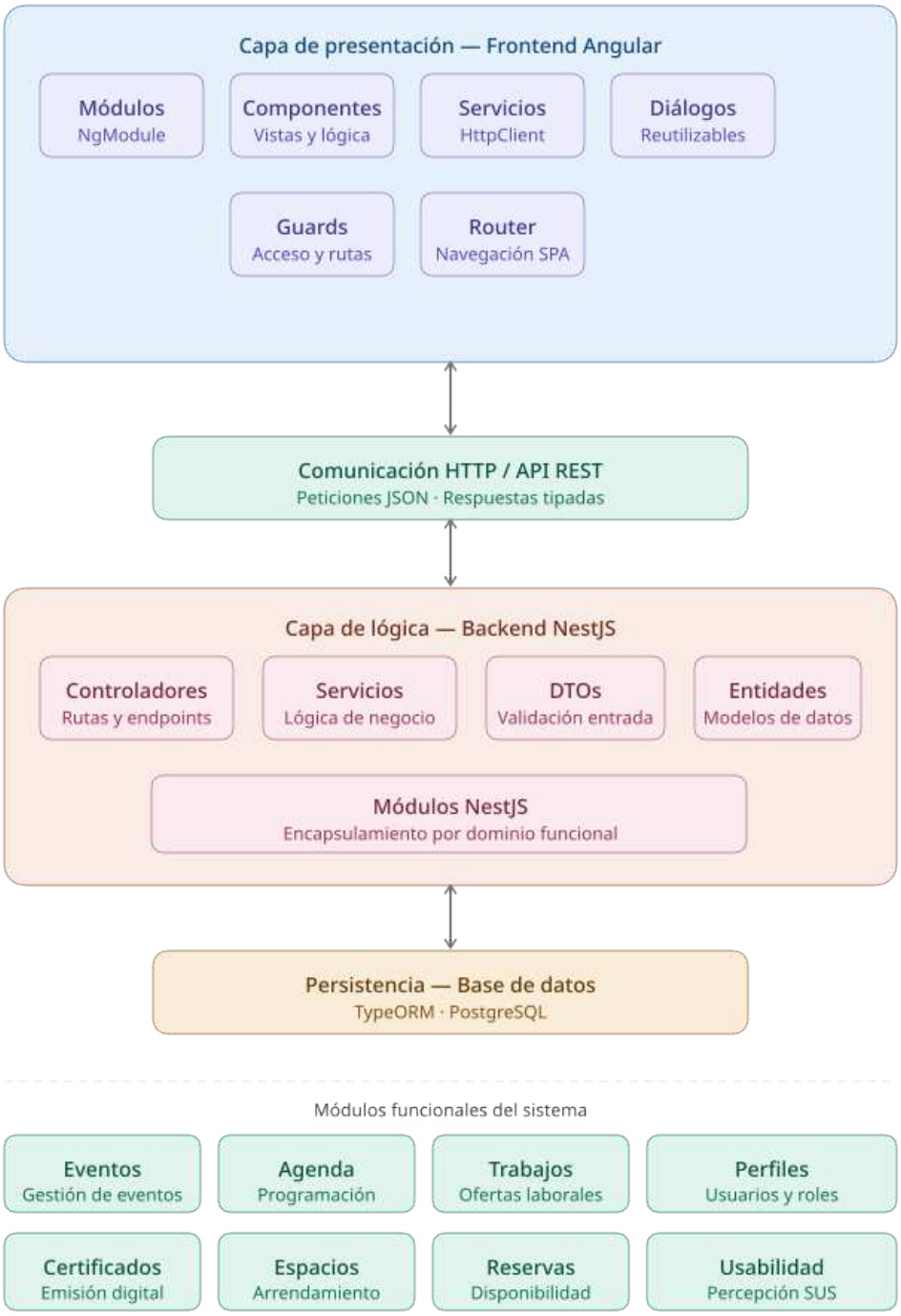


Figura 16: Diagrama general de la arquitectura modular de Invessoft, con separación entre frontend Angular, backend NestJS, persistencia y módulos funcionales del sistema

10.5. Estrategia por fases

La estrategia de desarrollo se organizó en fases sucesivas, aunque parcialmente solapadas, debido a la naturaleza evolutiva del proyecto y al contexto real de mantenimiento del sistema. La primera fase correspondió al levantamiento del estado actual, en la que se revisaron reportes de cobertura, estructura del repositorio y antecedentes funcionales de los módulos. Esta etapa permitió distinguir entre cobertura nominal y cobertura efectiva, y establecer que el principal reto no era solo aumentar el porcentaje global, sino cubrir comportamientos realmente críticos.

La segunda fase se orientó a la selección y priorización de módulos. En este punto se analizaron las intervenciones más representativas del sistema, identificando componentes donde una modificación reciente introducía nuevas condiciones de negocio o alteraba flujos sensibles. La frecuencia de cambio y la criticidad funcional fueron los criterios centrales para definir el alcance de intervención. De esta forma, el trabajo reconoció tanto módulos centrales para la estrategia de pruebas como módulos complementarios cuya evolución sirvió para mapear riesgos técnicos del sistema en su conjunto.

La tercera fase consistió en el diseño de escenarios de prueba. En lugar de construir pruebas genéricas, se definieron casos alineados con el comportamiento del negocio: validaciones condicionales en formularios, visibilidad de acciones según configuración, tratamiento de errores, filtros dependientes del contexto, navegación entre vistas y consistencia de estados después de una operación. Esta fase fue determinante para evitar que la implementación de pruebas se redujera a verificaciones superficiales.

La cuarta fase se concentró en la implementación de pruebas unitarias en frontend, primero sobre servicios y después sobre componentes críticos donde se corrigieron y actualizaron pruebas existentes, además de añadir nuevos casos para cubrir umbrales de cobertura en módulos que habían quedado sin aseguramiento suficiente. La quinta fase se orientó al backend, reforzando la validación de reglas de negocio en módulos priorizados y tomando los cambios funcionales implementados como referencia del tipo de lógica que debía protegerse. Finalmente, la sexta fase correspondió a la validación de resultados y a la consolidación documental, integrando cobertura, ejecución de pruebas, evidencia visual y trazabilidad por módulo.

10.6. Desarrollo por módulo

10.6.1. Gestión de eventos, landings y agenda académica

Una parte importante de la pasantía se concentró en módulos vinculados a la navegación pública y privada de eventos, particularmente en landings, agenda académica y flujos de inscripción. En este frente se abordaron tareas relacionadas con la landing del evento, la mejora del flujo de inscripción desde la landing, la agenda pública, la agenda específica de un evento, la visualización de la agenda de trabajos y talleres, la preinscripción a talleres, la creación de agenda privada y la integración con calendarios externos. A esto se sumaron

correcciones sobre filtros de agenda, estados por defecto y manejo de autenticación al redirigir desde la agenda hacia el landing de un trabajo.

Desde el punto de vista técnico, este conjunto de intervenciones puso en evidencia que los módulos de consulta e interacción pública suelen concentrar alta variabilidad de estados y reglas de visualización. Un mismo componente puede modificar su comportamiento según el evento seleccionado, la disponibilidad de cupos, la autenticación del usuario, la fecha del evento o la presencia de configuraciones específicas. Por esta razón, aunque muchas de estas tareas surgieron como mejoras funcionales o correctivas, su análisis fue relevante para el proyecto porque mostró la necesidad de ampliar el testing hacia componentes donde predominan filtros, redirecciones, carga condicional de datos y comportamiento dependiente del contexto.

Asimismo, tareas como compartir eventos y trabajos en redes sociales, mostrar trabajos activos en landings y mejorar la visualización de un trabajo individual evidenciaron la importancia de verificar no solo la exactitud de los datos, sino también su coherencia de presentación. En estos módulos, una prueba unitaria bien diseñada puede detectar regresiones asociadas a parámetros de ruta, inicialización del componente, selección de evento, renderizado de secciones y tratamiento de estados vacíos, todos ellos aspectos que influyen directamente en la experiencia del usuario y en la mantenibilidad del frontend.

La Figura 17 ilustra uno de los resultados de este frente: una vista de agenda con filtros por evaluador, actividad, área de conocimiento, espacio, fecha y texto libre, además de acciones complementarias como favoritos y tutorial. Esta interfaz evidencia la cantidad de estados que deben mantenerse coherentes cuando el usuario explora programación académica y valida por qué estos componentes requieren aseguramiento sobre filtros, conteo de resultados y representación de tarjetas.

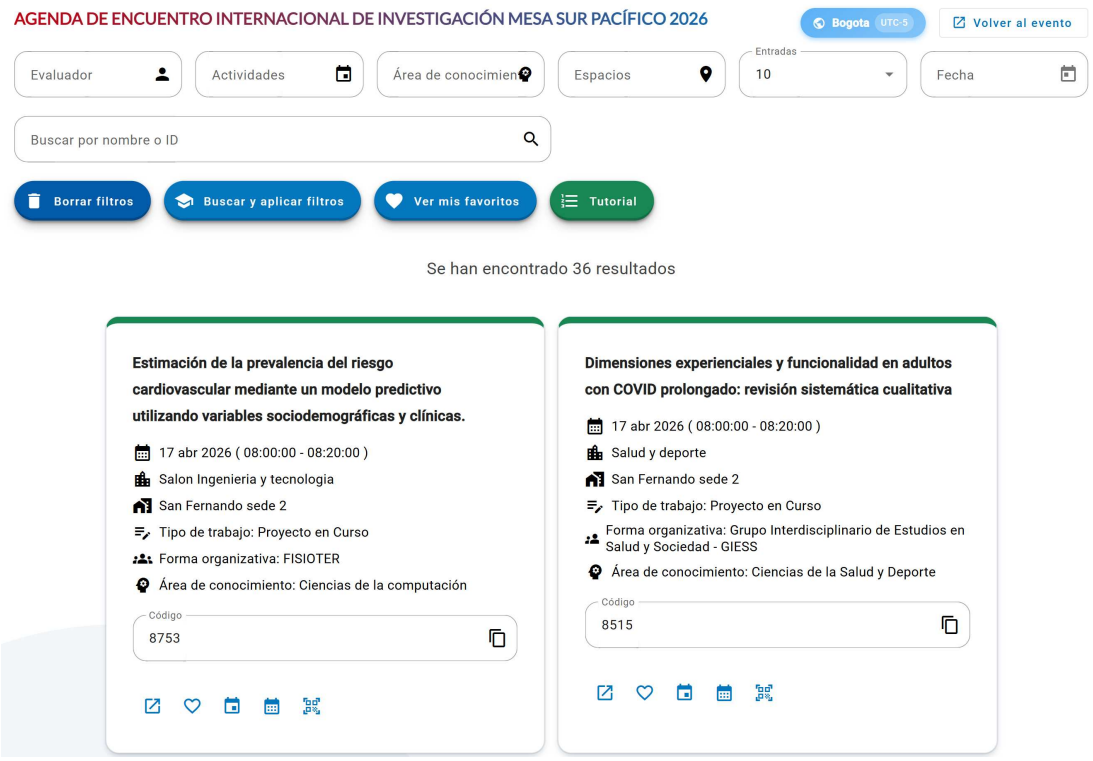


Figura 17: Vista de agenda académica con filtros y listado de resultados por evento

10.6.2. Gestión de trabajos académicos y procesos editoriales

Otro frente de intervención correspondió a los módulos relacionados con trabajos académicos y sus procesos asociados. Durante la pasantía se participó en tareas como la generación de QR para trabajos, el control de trabajos activos en vistas públicas, el orden de metadatos en la visualización, la carta de asignación como evaluador, el informe de trabajos no evaluados, el ranking de resultados y la paginación de listados especializados. De igual forma, se trabajó en la incorporación de links mágicos para registro de trabajos y en su posterior administración mediante un CRUD específico, ampliando así los mecanismos de acceso y gestión del proceso académico.

Estas intervenciones permitieron observar que el dominio de trabajos combina requisitos de trazabilidad documental, restricciones temporales, estados de publicación y flujos diferenciados según el rol del usuario. En consecuencia, representan un escenario particularmente sensible para pruebas unitarias, ya que pequeñas alteraciones en la forma de filtrar, ordenar o mostrar información pueden afectar procesos de evaluación, registro o consulta. Desde la perspectiva metodológica del proyecto, este conjunto de tareas sirvió para reconocer que el aseguramiento no debe restringirse a módulos novedosos o aislados, sino extenderse también a funcionalidades consolidadas cuyo valor depende de la consistencia de sus reglas y de la precisión en el tratamiento de los datos.

La incorporación de links mágicos merece una mención especial porque introdujo un flujo

con validaciones excepcionales sobre fechas límite, emisión de enlaces y uso controlado por usuario. Este tipo de funcionalidad refuerza la necesidad de diseñar pruebas sobre casos de autorización, vigencia, consumo del enlace y coherencia entre backend y frontend, ya que una falla en estos puntos compromete directamente la seguridad lógica del sistema.

Las Figuras 18 y 19 muestran, respectivamente, la administración de enlaces mágicos desde el panel del evento y el consumo efectivo del enlace para redirigir al formulario de registro de un nuevo trabajo. Estas evidencias permiten describir el flujo completo: creación del enlace, control de estado y uso posterior en una ruta parametrizada que simplifica el acceso del usuario sin sacrificar trazabilidad.

Figura 18: Administración de links mágicos para el registro de trabajos

Figura 19: Uso de link mágico para acceder al formulario de creación de un trabajo

10.6.3. Gestión de personas, perfiles e instituciones

La pasantía también incluyó intervenciones en módulos de personas y perfiles, particularmente en la posibilidad de indicar "otras instituciones" al crear o editar registros, la corrección de casos específicos de edición de perfil, el ajuste de selectores y la incorporación de filtros por evento en consultas de personas. Aunque estas tareas pueden parecer de menor criticidad frente a flujos como pagos, reservas o CRM, en realidad ocupan un lugar estratégico dentro del sistema, ya que la información de personas e instituciones alimenta múltiples procesos posteriores, entre ellos inscripciones, certificación, evaluación y visualización pública de datos.

Desde el punto de vista técnico, estos trabajos revelaron la importancia de validar formularios dinámicos, dependencias entre campos y consistencia en selectores reutilizados por distintos módulos. La refactorización de selectores, por ejemplo, no solo buscó mejorar la mantenibilidad del código de interfaz, sino también reducir errores derivados de componentes duplicados o comportamientos inconsistentes. En el marco del proyecto, este frente mostró que las pruebas unitarias pueden aportar un valor significativo al asegurar que cambios en formularios base o en componentes compartidos no introduzcan defectos en cascada sobre otros módulos consumidores.

La Figura 20 evidencia la incorporación de la opción de registrar una institución no existente dentro del flujo de información personal, habilitando campos adicionales como

nombre de institución y dominio. A su vez, la Figura 21 muestra la refactorización del selector con autocompletado para búsquedas más eficientes y reutilizables. Ambas capturas sintetizan cómo pequeñas decisiones de interfaz repercuten en validaciones condicionales y en la necesidad de proteger componentes compartidos.

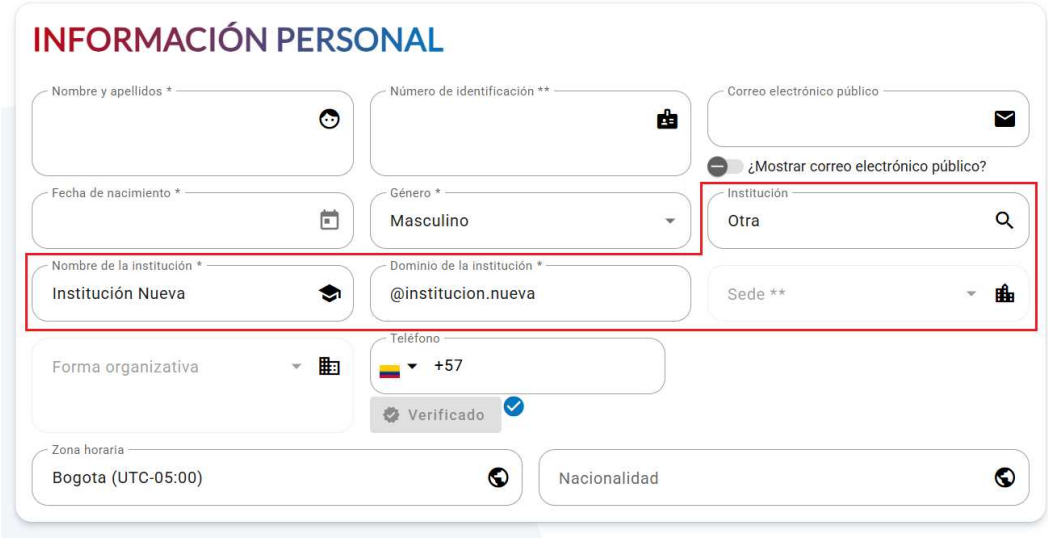


Figura 20: Formulario de información personal con soporte para registrar otras instituciones

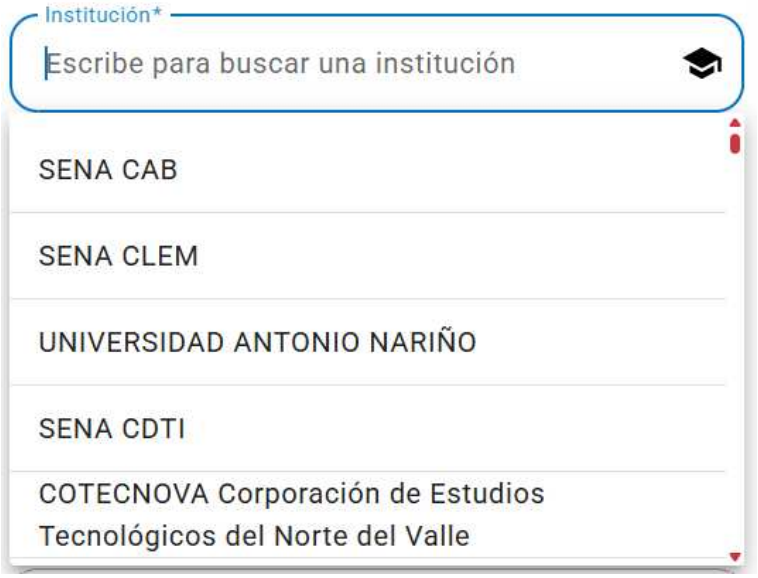


Figura 21: Selector con autocompletado para la búsqueda y selección de instituciones

10.6.4. Certificados, insignias y percepción de usabilidad

Un cuarto frente de desarrollo correspondió a los módulos de certificación, insignias y percepción de usabilidad. En este ámbito se abordaron tareas como la encuesta obligatoria

para descarga de certificados, la consulta de certificados desde el módulo de personas, la configuración de insignias, la corrección del valor por defecto de puntajes mínimos y un ajuste correctivo sobre estados de carga en certificados. A ello se sumó el desarrollo del sistema de encuestas de percepción de usabilidad, concebido para capturar valoraciones de los usuarios sobre distintos módulos del sistema y posteriormente visualizar sus resultados.

Estas intervenciones tienen un valor especial para el trabajo de grado porque vinculan dos dimensiones fundamentales de la calidad del software: el correcto funcionamiento del sistema y la percepción de su uso por parte del usuario final. En el caso de certificados e insignias, el reto técnico se centró en asegurar consistencia de reglas, disponibilidad de información y comportamiento correcto de formularios y estados visuales. En el caso de las encuestas de percepción, el aporte fue más amplio, ya que involucró nuevas estructuras de datos, formularios, lógica de presentación de resultados y trazabilidad del uso del sistema. En ambos casos, la experiencia adquirida permitió identificar unidades funcionales donde el testing futuro debe cubrir no solo respuestas exitosas, sino también estados de carga, validaciones previas, dependencias entre configuraciones y consistencia de la interfaz frente a datos incompletos o cambios de contexto.

Las Figuras 22, 23 y 24 permiten observar este frente desde tres perspectivas complementarias: la creación de la encuesta de percepción, la configuración del certificado con la bandera de encuesta obligatoria y la experiencia del usuario final al intentar descargar el certificado sin haber respondido primero la encuesta. De forma complementaria, las Figuras 25 y 26 muestran la gestión de percepción por módulos y la ventana modal de calificación desplegada sobre un flujo real de uso.

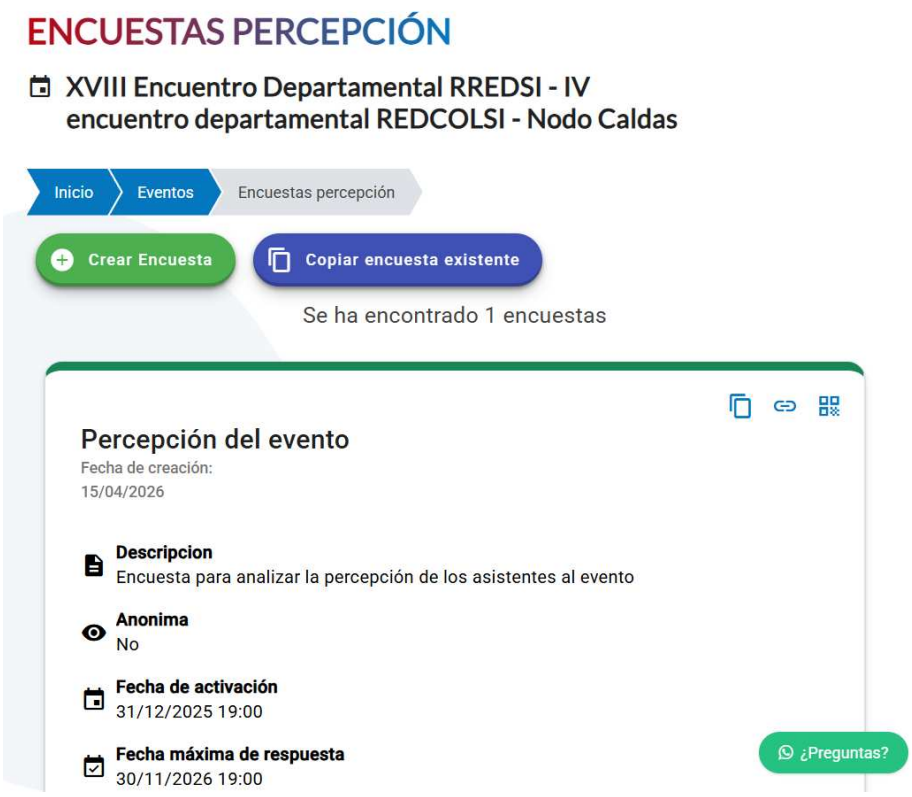


Figura 22: Creación y administración de encuesta de percepción asociada a un evento

Figura 23: Configuración de certificado con encuesta obligatoria previa a la descarga

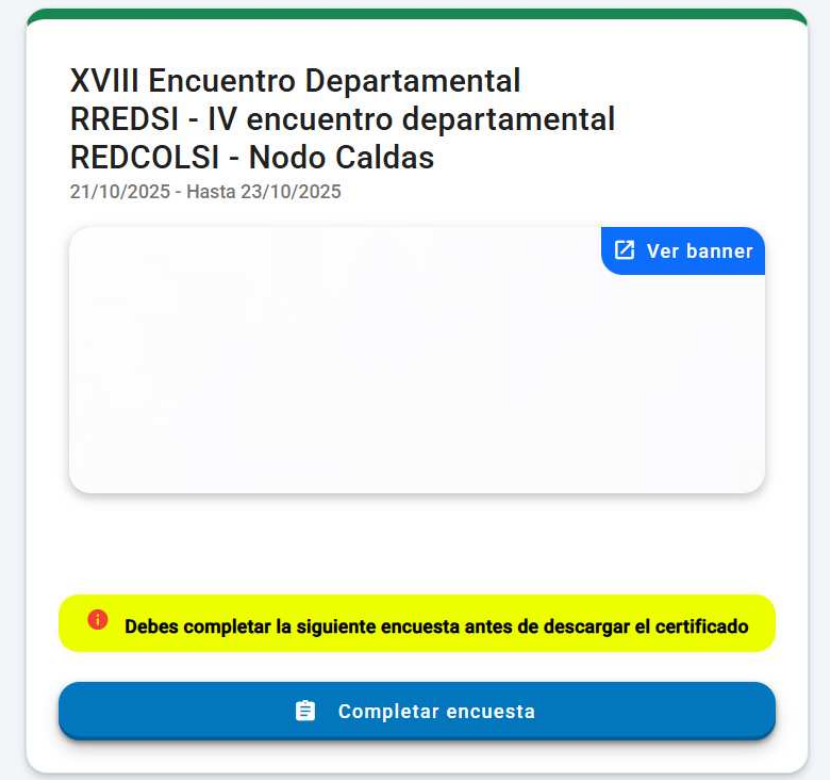


Figura 24: Bloqueo de descarga de certificado hasta completar la encuesta requerida

Figura 25: Gestión de percepción por módulos con frecuencia, estado y promedio de valoración

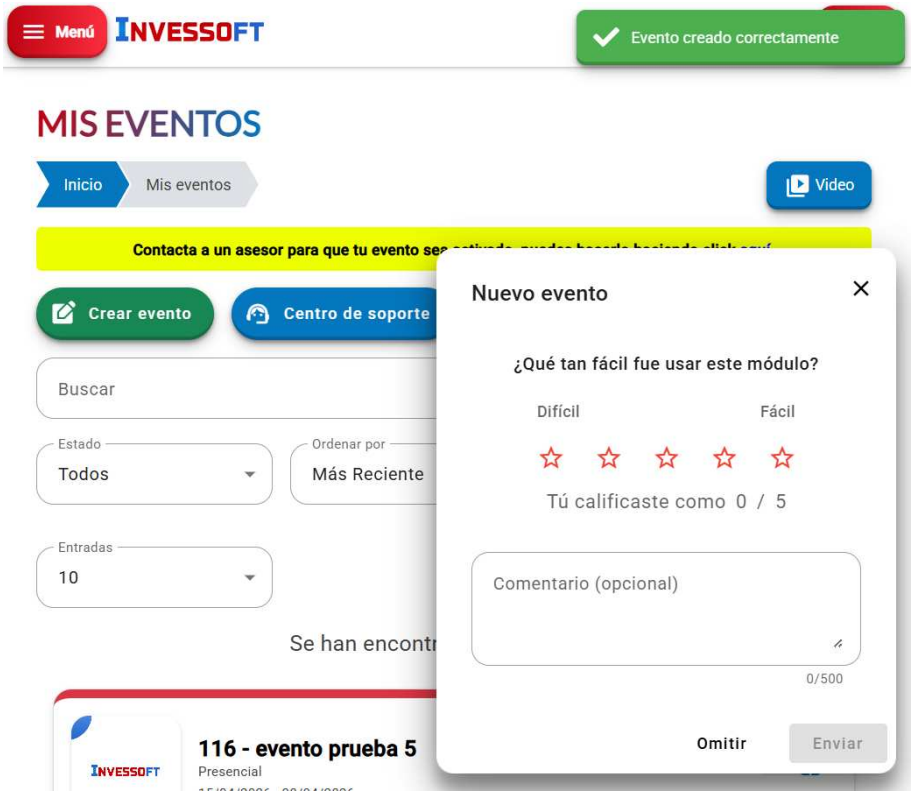


Figura 26: Modal de calificación de percepción desplegado durante el uso de un módulo

10.6.5. Analítica, reportes y mejoras transversales

También se realizaron intervenciones en módulos de analítica y soporte transversal, tales como la corrección de visualización de KPIs, la elaboración de reportes de votos, la auditoría de retiros mediante exportación a Excel y la incorporación de voz a texto para formatos de evaluación en versión web. Aunque estas tareas no siempre derivaron en una implementación directa de pruebas unitarias dentro del alcance principal del proyecto, sí resultaron importantes para comprender la heterogeneidad técnica de Invessoft. En particular, mostraron que el sistema combina operaciones CRUD convencionales con procesamiento de datos, exportaciones, visualizaciones derivadas y componentes de apoyo al trabajo académico.

Desde la perspectiva de mantenibilidad, estos módulos son relevantes porque suelen depender de transformaciones de datos, agregaciones y reglas de interpretación que pueden degradarse silenciosamente cuando se introduce una modificación menor. Por ello, su inclusión en la presente sección busca dejar constancia de que la estrategia de calidad no debe agotarse en los módulos más visibles, sino proyectarse también hacia capas analíticas y utilitarias cuya estabilidad influye en la confianza general sobre el sistema.

La Figura 27 muestra un ejemplo de visualización corregida de indicadores, en este caso una tabla de instituciones participantes con soporte para paginación y descarga. La Figura 28 complementa este frente al evidenciar la incorporación de entrada por voz dentro de

un formato de evaluación web, lo que amplía la complejidad de interacción y refuerza la necesidad de verificar componentes utilitarios más allá de los CRUD tradicionales.

Figura 27: Visualización de KPIs asociados a instituciones participantes en un evento

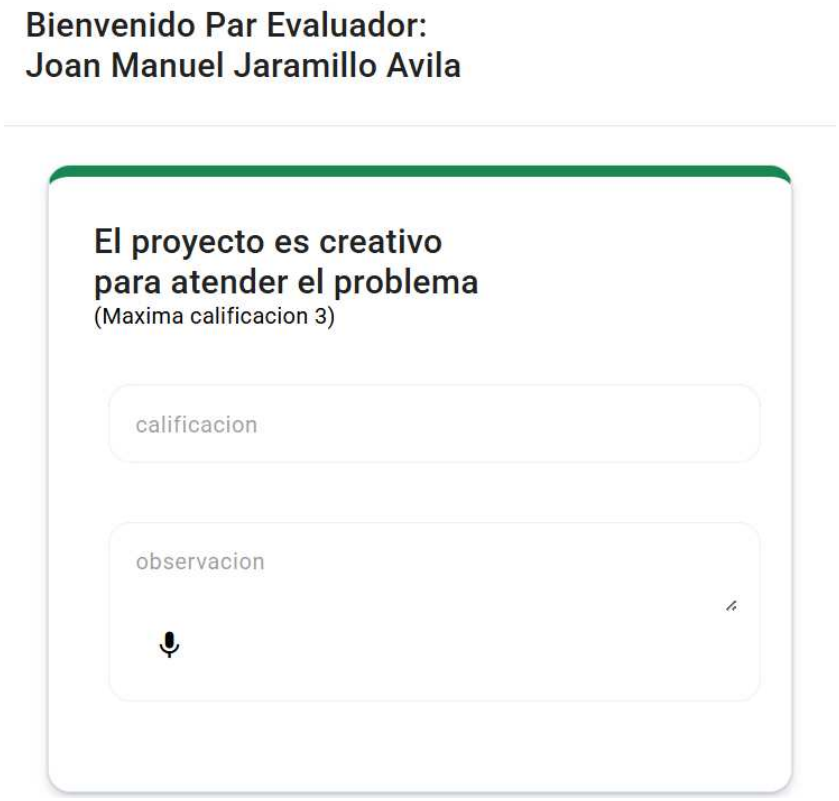


Figura 28: Incorporación de voz a texto en formularios web de evaluación académica

10.6.6. Módulo de espacios arrendables y reservas

El módulo de espacios arrendables constituyó uno de los casos más representativos para el desarrollo del proyecto, ya que reúne operaciones CRUD, filtros de búsqueda, manejo de disponibilidad, validaciones condicionales, comunicación entre usuarios y, posteriormente, un subflujo de reservas. Desde el punto de vista de calidad, este tipo de módulo presenta un riesgo elevado porque combina lógica de negocio en backend con una interacción de usuario rica en frontend, lo que multiplica la cantidad de escenarios posibles y hace insuficiente una cobertura centrada únicamente en servicios.

En backend, la construcción del módulo implicó modelar entidades, relaciones y criterios de búsqueda para soportar la publicación y consulta de espacios. Posteriormente, se realizaron ajustes funcionales orientados a mejorar la búsqueda por ciudad, incorporando un comportamiento más tolerante frente a tildes y diacríticos. En frontend, este módulo fue uno de los principales focos de implementación de pruebas unitarias, con actualización y ampliación de pruebas para componentes como formularios, listados, filtros, tarjetas y diálogos de contacto con propietarios. En estas unidades se validaron escenarios asociados a formularios reactivos, visibilidad de controles, restricciones del mensaje, comportamiento frente a autenticación y activación condicional de medios de contacto por correo o WhatsApp.

La ampliación posterior hacia reservas agregó estados transaccionales, validaciones tem-

porales, consultas por rol y notificaciones por correo. Esto reforzó el valor del módulo como caso de estudio para el trabajo de grado, ya que permitió articular decisiones de diseño de pruebas sobre reglas de negocio, efectos secundarios y configuración por entorno. En este frente, el aporte del proyecto no consistió solamente en incrementar cobertura, sino en mostrar cómo un módulo nuevo puede diseñarse con mayor disciplina de aseguramiento desde sus primeras iteraciones.

Las Figuras 29, 30 y 31 muestran la evolución funcional de este módulo desde la gestión y filtrado de espacios hasta la solicitud y seguimiento de reservas.

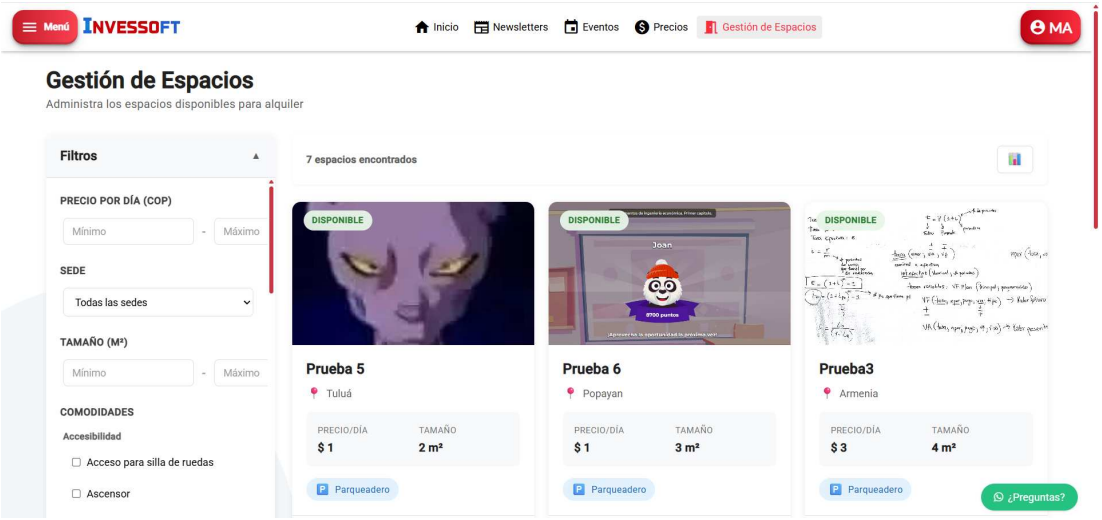


Figura 29: Módulo de gestión de espacios arrendables con filtros y tarjetas de consulta

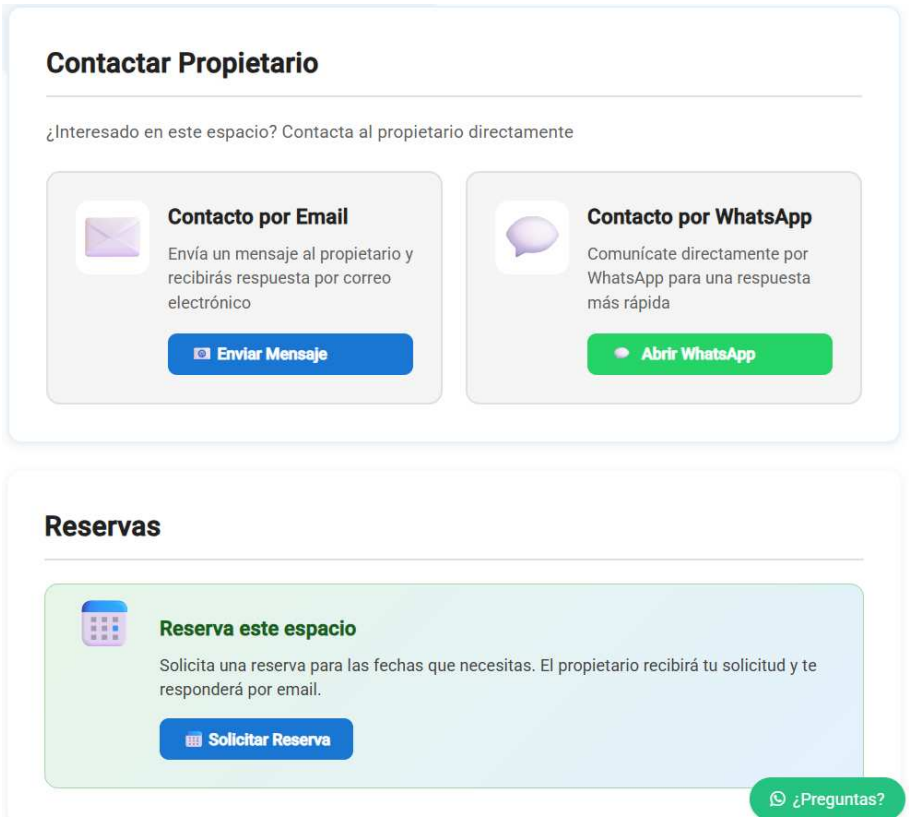


Figura 30: Flujo de contacto con el propietario y solicitud de reserva de un espacio

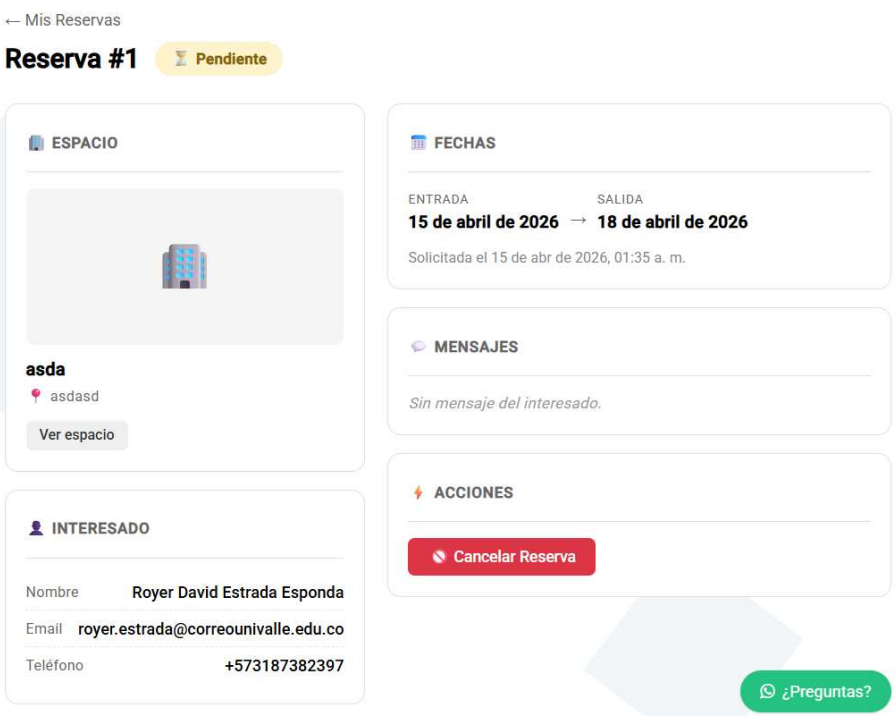


Figura 31: Detalle de una reserva con estado, fechas, datos del interesado y acciones disponibles

10.6.7. Otros módulos funcionales con reglas de negocio específicas

Además de los módulos priorizados de manera explícita para la estrategia de pruebas, durante la pasantía también se intervinieron otros frentes funcionales con reglas de negocio específicas. Entre ellos se encontró el CRM de eventos y el control de créditos de WhatsApp, cuya evolución exigió incorporar nuevos campos, propagar cambios sobre entidades y DTOs, y reforzar validaciones previas a acciones sensibles. Aunque estos desarrollos fueron relevantes dentro del trabajo realizado, en esta sección se presentan como parte del conjunto de intervenciones funcionales que ayudaron a comprender la deuda técnica del sistema, y no como el único eje de priorización del proyecto.

En backend, la incorporación de campos como `cantidad_maxima_whatsapp` y `total_whatsapp_enviados` obligó a modificar entidades, DTOs y servicios responsables de actualizar configuraciones y contabilizar consumos. En frontend, los componentes asociados a este flujo debieron adaptarse para representar estas reglas de forma comprensible al usuario, incluyendo validaciones de visibilidad, agrupación de acciones y variantes de confirmación. Este tipo de intervención resultó valiosa porque mostró que muchos cambios funcionales del sistema requieren trazabilidad entre ajuste de negocio, adaptación visual y posterior diseño de casos de prueba.

Las Figuras 32, 33 y 34 muestran la trazabilidad de este cambio en tres niveles: persis-

tencia, contrato de entrada y representación visual en la configuración del CRM. En conjunto, estas capturas permiten demostrar que la incorporación del campo `cantidad_maxima_whatsapp` no fue un ajuste superficial, sino un cambio propagado de forma consistente entre backend y frontend.

```
@Entity({ name: 'eventos_conf_crm' })
export class EventosConfCrm {
    @Column({ nullable: true, default: 0 })
    cantidad_maxima_whatsapp: number;
```

Figura 32: Incorporación del campo `cantidad_maxima_whatsapp` en la entidad de configuración CRM

```
export class CreateEventosConfCrmDto {
    @IsOptional()
    @IsNumber()
    @ApiProperty({ default: 0 })
    cantidad_maxima_whatsapp?: number;
```

Figura 33: Definición del campo `cantidad_maxima_whatsapp` en el DTO de creación de configuración CRM

Figura 34: Visualización del límite de mensajes de WhatsApp dentro de la configuración de mensajes del CRM

Las mejoras posteriores de interfaz, orientadas a ocultar acciones cuando no existe configuración suficiente y a agrupar opciones en flujos más compactos, también se integran de

manera natural al enfoque de pruebas. Aunque en apariencia pertenecen al plano visual, estas decisiones dependen de estados y datos concretos, de modo que deben verificarse como parte de la lógica del componente y no solo como ajustes cosméticos. El resultado es una interfaz más predecible, con menor probabilidad de exponer acciones inválidas y con mayor coherencia entre las reglas del negocio y la experiencia ofrecida al usuario.

10.7. Diseño de pruebas

El diseño de pruebas se basó en el principio de que una unidad debe validarse por su comportamiento observable y por las reglas que encapsula, no únicamente por la ejecución lineal de sus métodos. En consecuencia, los casos se construyeron a partir de escenarios exitosos, escenarios de error y condiciones de frontera. Para los módulos intervenidos esto implicó definir pruebas que verificaran tanto la respuesta correcta en condiciones esperadas como la negativa controlada ante entradas inválidas, ausencia de permisos, falta de créditos, errores de autenticación o configuraciones incompletas.

En frontend, la estrategia se apoyó en la simulación de dependencias y en la activación controlada de eventos del usuario para observar cambios de estado en componentes, formularios y diálogos. En una primera etapa se fortaleció la cobertura de servicios; en una segunda, se diseñaron pruebas para componentes con mayor impacto funcional, verificando requisitos de obligatoriedad, longitudes mínimas y máximas, validaciones condicionales, navegación, estados de carga, visibilidad de acciones y comportamiento frente a distintos contextos de uso. En backend, por su parte, las pruebas debían validar reglas de negocio desacopladas de infraestructura, por lo que resultó esencial el uso de *mocks* para repositorios, servicios de correo y proveedores de configuración.

Otro criterio de diseño relevante fue la trazabilidad entre el cambio funcional y el caso de prueba. Cada vez que una solicitud de integración introducía un nuevo campo, una regla de visibilidad, un flujo excepcional o una bifurcación adicional del proceso, se procuró que existiera una prueba equivalente capaz de detectar regresiones. Este principio es el que convierte la suite de pruebas en un activo de mantenibilidad: no solo verifica el presente del sistema, sino que documenta el comportamiento que no debería perderse cuando el módulo sea refactorizado o ampliado.

Las Figuras 35, 36, 37, 38 y 39 evidencian esta trazabilidad en el módulo de reservas. Los *mocks* de espacio, reserva y usuario permiten aislar dependencias y construir escenarios controlados, mientras que las pruebas verifican comportamientos concretos como impedir que el propietario reserve su propio espacio y filtrar correctamente fechas pasadas, nulas u ocupadas en el formulario de reserva.

```
const mockSpace: Espacio = {
    id_espacio: 5,
    id_sede: 1,
    nombre_sede: 'Yumbo',
    nombre: 'Auditorio Test',
    descripcion: 'Un auditorio',
    direccion: 'Calle 100',
    precio_por_dia: 500000,
    precio_por_hora: 100000,
    moneda: 'COP',
    tamano_metros_cuadrados: 200,
    capacidad_maxima: 150,
    comodidades: [],
    sw_tiene_parqueadero: false,
    espacios_parqueadero: 0,
    disponibilidad: {
        rangos_fecha: [],
        estado: true,
        fechas_bloqueadas: [],
    },
    restricciones: [],
    imagen_principal: 'img.jpg',
    urls_imagenes: [],
    createdAt: new Date(),
    updatedAt: new Date(),
    estado: true,
    id_usuario_registro: 99,
    sw_contacto_email: true,
    sw_contacto_whatsapp: false,
};
```

Figura 35: Mock de espacio rentable utilizado para aislar escenarios de prueba en frontend

```
const mockReservaResponse: ReservaResponse = {
    id_reserva: 10,
    espacio: {
        id_espacio: 5,
        nombre: 'Auditorio Test',
        direccion: 'Calle 100',
        imagen_principal: null,
    },
    usuario_interesado: {
        id: 42,
        email: 'user@test.com',
        nombre_persona: null,
        telefono: null,
    },
    fecha_inicio: '2026-04-01',
    fecha_fin: '2026-04-05',
    estado_reserva: EstadoReserva.PENDIENTE,
    mensaje_interesado: null,
    mensaje_respuesta: null,
    created_at: '2026-03-01T00:00:00Z',
    updated_at: '2026-03-01T00:00:00Z',
};
```

Figura 36: Mock de respuesta de reserva empleado en pruebas del flujo de reservas

```
const mockUser: usuarioDto = {
    id: 42,
    email: 'user@test.com',
    username: 'user',
    estado: true,
    createdAt: '2024-01-01T00:00:00Z',
    updatedAt: '2024-01-01T00:00:00Z',
    sw_admin: false,
    email_verified_at: '2024-01-01T00:00:00Z',
    persona: {
        id_persona: 1,
        nombre_persona: 'Juan Pérez',
        email: 'user@test.com',
        telefono: '3001234567',
        fecha_nacimiento: '1990-01-01',
        genero: 'M',
        nro_identificacion: '12345678',
        estado: true,
        usuario_registro: null,
        createdAt: '2024-01-01T00:00:00Z',
        updateAt: '2024-01-01T00:00:00Z',
        sw_telefono_verificado: true,
    } as PersonaDto,
};
```

Figura 37: Mock de usuario utilizado para simular distintos roles y condiciones de sesión

```
it('should redirect when owner tries to book their own space', () => {
    const ownerUser = { ...mockUser, id: 99 };
    const ownerSessionSpy = jasmine.createSpyObj('SessionService', [], {
        currentUser$: of(ownerUser),
    });
    component['sessionService'] = ownerSessionSpy;
    component.currentUser = ownerUser;
    fixture.detectChanges();
    // Re-trigger loadSpace manually
    component['loadSpace']();
    expect(toastrSpy.warning).toHaveBeenCalledWith(
        'No puedes reservar tu propio espacio'
    );
});
```

Figura 38: Prueba unitaria que impide reservar el propio espacio del usuario propietario

```
describe('ReservaFormComponent', () => {
    describe('dateFilter', () => {
        beforeEach(() => fixture.detectChanges());

        it('should return false for null date', () => {
            expect(component.dateFilter(null)).toBe(false);
        });

        it('should return false for past date', () => {
            const past = new Date('2020-01-01');
            expect(component.dateFilter(past)).toBe(false);
        });

        it('should return true for future date not occupied', () => {
            const future = new Date(Date.now() + 30 * 24 * 60 * 60 * 1000);
            expect(component.dateFilter(future)).toBe(true);
        });

        it('should return false for date inside occupied range', () => {
            const future = new Date(Date.now() + 10 * 24 * 60 * 60 * 1000);
            // Use local date parts to match the component's toYMD logic
            const y = future.getFullYear();
            const m = String(future.getMonth() + 1).padStart(2, '0');
            const d = String(future.getDate()).padStart(2, '0');
            const futureStr = `${y}-${m}-${d}`;
            component.fechasOcupadas = [
                { fecha_inicio: futureStr, fecha_fin: futureStr },
            ];
            expect(component.dateFilter(future)).toBe(false);
        });
    });
```

Figura 39: Pruebas unitarias del filtro de fechas en el formulario de reserva

10.8. Validación y métricas

La validación del proyecto se abordó desde una combinación de métricas cuantitativas y cualitativas. En el plano cuantitativo, se consideraron los reportes de cobertura inicial y final, el número de archivos de prueba creados o actualizados en los módulos intervenidos, y la ampliación efectiva de pruebas sobre componentes del frontend que anteriormente no

contribuían de forma significativa a la cobertura real. En el plano cualitativo, se evaluó la capacidad de las pruebas para detectar errores en flujos sensibles, preservar comportamiento esperado durante refactorizaciones y reducir la incertidumbre al integrar cambios sobre módulos activos.

Dado que el objetivo del trabajo no fue únicamente elevar un porcentaje global, la interpretación de las métricas se realizó con cautela. En backend, el reporte final mostró un incremento sustancial frente al diagnóstico inicial, alcanzando 64.35 % en *statements*, 45.4 % en *branches*, 63.37 % en *functions* y 64.17 % en *lines*. En frontend, la cobertura final consolidada alcanzó 95.52 % en *statements*, 86.86 % en *branches*, 95.7 % en *functions* y 95.77 % en *lines*. Estos resultados muestran que el avance no solo se reflejó en la base de servicios, sino también en una expansión progresiva hacia componentes de mayor criticidad funcional.

Junto con el incremento de cobertura, se establecieron decisiones de proceso para sostener el crecimiento del aseguramiento en el tiempo. En particular, se definió que todo cambio propuesto hacia la rama `develop` debía ejecutar las pruebas unitarias correspondientes antes de ser integrado. Bajo este criterio, si un módulo no cuenta con pruebas unitarias o si estas fallan, la *pipeline* no debe permitir la mezcla de cambios. Además, cuando una modificación impacta de forma directa a otros módulos, se planteó como criterio de calidad que dichos módulos relacionados también deben alcanzar una cobertura mínima del 80 % en pruebas unitarias. Aunque esta política puede incrementar los tiempos de entrega en ciertas tareas, se consideró necesario contemplarla dentro de la planeación del trabajo para no distorsionar las métricas de rendimiento de los desarrolladores y, al mismo tiempo, aumentar de forma sostenida la cobertura del frontend a largo plazo.

La Tabla 3 sintetiza los principales indicadores cuantitativos y cualitativos utilizados para valorar el avance del proyecto. En los casos donde no se consolidó un conteo histórico exacto de pruebas añadidas o actualizadas por iteración, se optó por registrar la evidencia disponible a partir de reportes de cobertura, ejecución de la suite y módulos efectivamente intervenidos.

Tabla 3: Métricas comparativas de validación del proyecto

Indicador	**Backend**	**Frontend**
Cobertura inicial global	24.17 %	97.67 % concentrado principalmente en servicios
Cobertura final global	64.35 % *statements*; 45.4 % *branches*; 63.37 % *functions*; 64.17 % *lines*	95.52 % *statements*; 86.86 % *branches*; 95.7 % *functions*; 95.77 % *lines*
Unidades con evidencia destacada	Módulos priorizados con avance visible en cobertura, especialmente perfiles, encuestas, eventos, campos y espacios	Componentes intervenidos de espacios, reservas, links mágicos, certificados, agenda y percepción
Ejecución de pruebas	1106 pruebas exitosas	11681 pruebas exitosas
Cobertura funcional protegida	Reglas de negocio, validaciones, reservas, notificaciones y configuración por dominio	Componentes críticos, formularios, navegación, reservas y estados de interfaz
Ampliación futura	Agenda, landings, certificación, perfiles, trabajos y reportes	Agenda, landings, certificación, perfiles, trabajos y reportes
Decisión de aseguramiento	Pruebas obligatorias antes de integrar cambios a `develop`	Pruebas obligatorias y cobertura mínima del 80 % en módulos impactados

También es pertinente incorporar capturas de ejecución satisfactoria de la suite de pruebas.

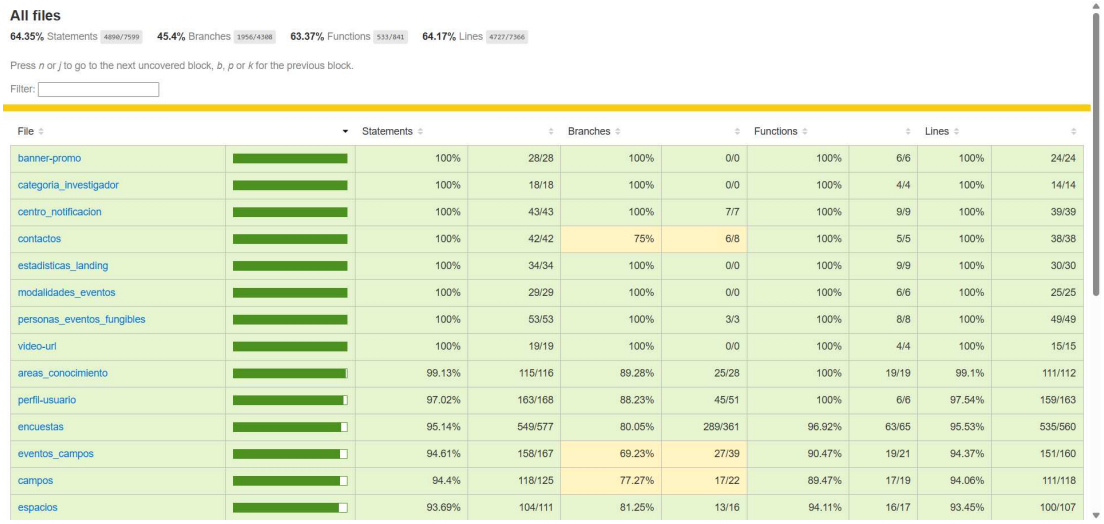
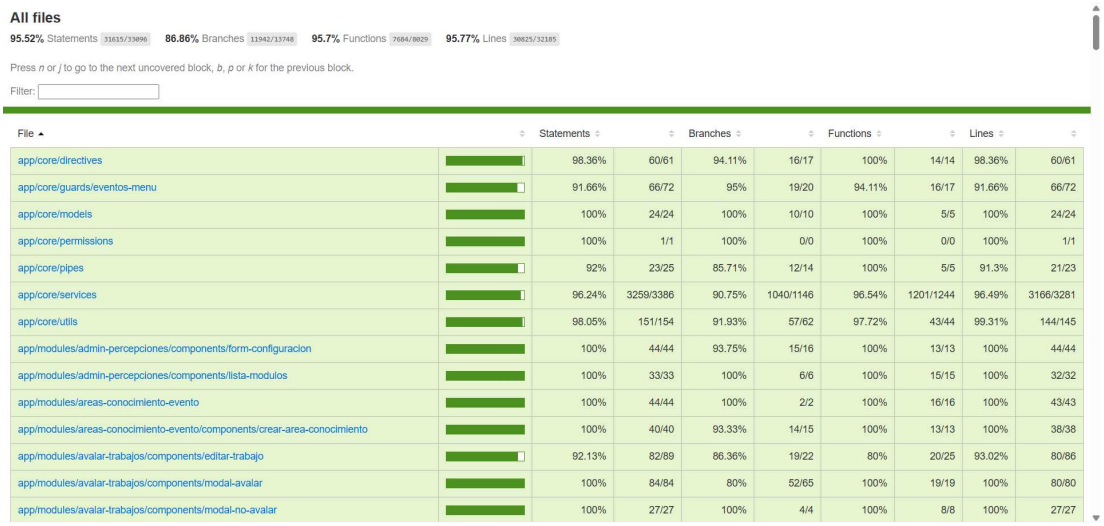


Figura 40: Reporte de cobertura final del backend



All files

95.52% Statements 31615/33096 **86.86%** Branches 11942/13748 **95.7%** Functions 7684/8029 **95.77%** Lines 30825/32185

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
app/core/directives	98.36%	60/61	94.11%	16/17	100%	14/14	98.36%	60/61
app/core/guards/eventos-menu	91.66%	66/72	95%	19/20	94.11%	16/17	91.66%	66/72
app/core/models	100%	24/24	100%	10/10	100%	5/5	100%	24/24
app/core/permissions	100%	1/1	100%	0/0	100%	0/0	100%	1/1
app/core/pipes	92%	23/25	85.71%	12/14	100%	5/5	91.3%	21/23
app/core/services	96.24%	3259/3386	90.75%	1040/1146	96.54%	1201/1244	96.49%	3166/3281
app/core/utils	98.05%	151/154	91.93%	57/62	97.72%	43/44	99.31%	144/145
app/modules/admin-percepciones/components/form-configuracion	100%	44/44	93.75%	15/16	100%	13/13	100%	44/44
app/modules/admin-percepciones/components/lista-modulos	100%	33/33	100%	6/6	100%	15/15	100%	32/32
app/modules/areas-conocimiento-evento	100%	44/44	100%	2/2	100%	16/16	100%	43/43
app/modules/areas-conocimiento-evento/components/crear-area-conocimiento	100%	40/40	93.33%	14/15	100%	13/13	100%	38/38
app/modules/avalar-trabajos/components/editar-trabajo	92.13%	82/89	86.36%	19/22	80%	20/25	93.02%	80/86
app/modules/avalar-trabajos/components/modal-avalar	100%	84/84	80%	52/65	100%	19/19	100%	80/80
app/modules/avalar-trabajos/components/modal-no-avalar	100%	27/27	100%	4/4	100%	8/8	100%	27/27

Figura 41: Reporte de cobertura final del frontend

```
> pnpm test:ci

TOTAL: 11681 SUCCESS

=============================== Coverage summary ===============================
Statements   : 95.52% ( 31615/33096 )
Branches     : 86.86% ( 11942/13748 )
Functions    : 95.7% ( 7684/8029 )
Lines        : 95.77% ( 30825/32185 )
================================================================================
```

Figura 42: Ejecución de pruebas en frontend con 11681 pruebas exitosas

```
> pnpm run test:cov:partial
================================================================================

Test Suites: 58 passed, 58 total
Tests:       1106 passed, 1106 total
Snapshots:   0 total
Time:        130.423 s
```

Figura 43: Ejecución de pruebas en backend con 1106 pruebas exitosas

10.9. Evidencias

Las evidencias del desarrollo deben disponerse de manera que articulen el diagnóstico inicial, la intervención realizada y los resultados obtenidos. En este sentido, los reportes de cobertura inicial y final constituyen el primer nivel de evidencia, ya que permiten comparar el punto de partida del proyecto con el estado alcanzado tras la implementación en los módulos priorizados. Las Figuras 2, 3, 40 y 41 cumplen esta función al hacer visible la evolución del aseguramiento tanto en backend como en frontend.

Un segundo nivel de evidencia corresponde a la ejecución satisfactoria de la suite de pruebas. Estas capturas complementan los porcentajes de cobertura porque demuestran que las pruebas no solo existen como artefacto estadístico, sino que se ejecutan correctamente sobre el sistema en su estado final. En este punto, las Figuras 42 y 43 aportan respaldo empírico a la validación presentada en la sección anterior, al documentar la estabilidad de las suites y su contribución efectiva al proceso de verificación.

Un tercer nivel de evidencia lo constituyen las capturas funcionales y los registros de implementación asociados a los módulos intervenidos. Estas evidencias son relevantes porque permiten mostrar el alcance técnico de los cambios más allá de la métrica de cobertura, particularmente en frentes como *Espacios para rentar*, *Reservas de espacios*, *Links mágicos para trabajos*, *Sistema de percepción de usabilidad*, *Certificados desde personas*, *Agenda y landings de eventos* y *Mejoras en visualización de trabajos*. Su inclusión resulta pertinente porque documenta de manera concreta qué flujos fueron modificados, qué comportamientos se protegieron mediante pruebas y en qué sectores del sistema se concentró el mayor esfuerzo de aseguramiento.

En conjunto, estas evidencias permiten vincular de forma directa los requerimientos intervenidos, los riesgos identificados y los mecanismos de validación aplicados. Por esta razón, la presente sección no debe entenderse como un anexo meramente ilustrativo, sino como el soporte empírico que demuestra la coherencia entre el diagnóstico inicial, la selección de módulos críticos, la implementación realizada durante la pasantía y los resultados obtenidos en términos de confiabilidad y mantenibilidad.

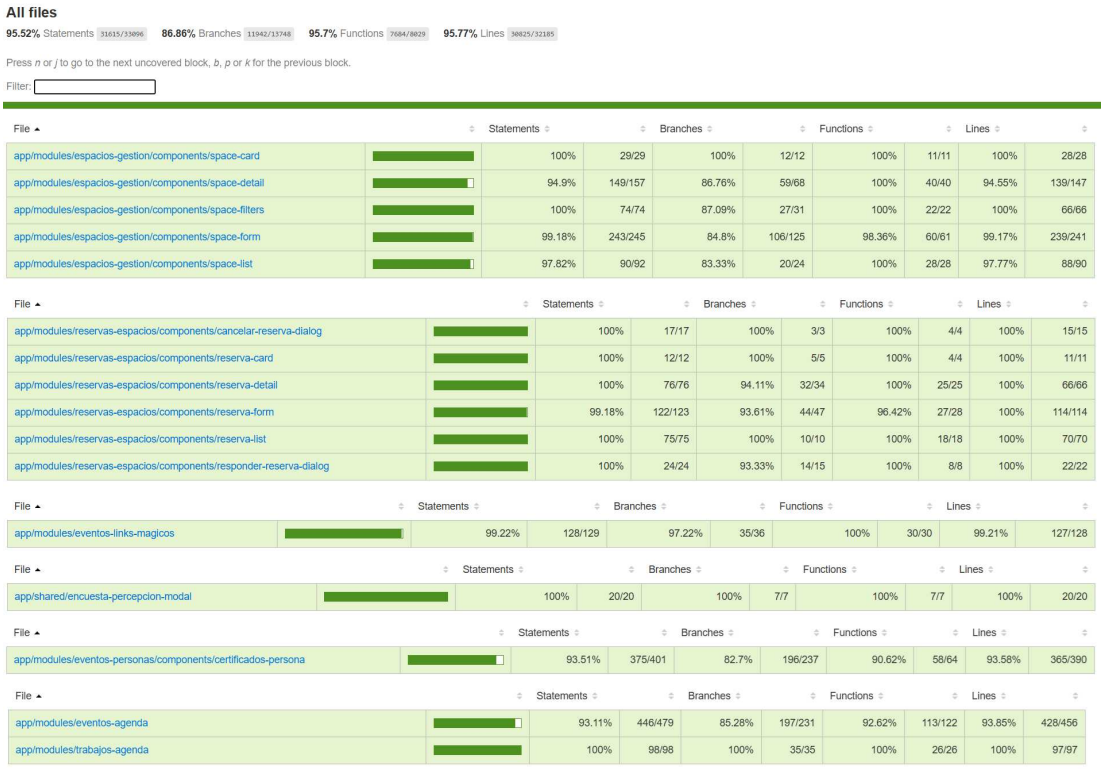


Figura 44: Cobertura destacada en componentes intervenidos de espacios, reservas, links mágicos, percepción y agenda

Tabla 4: Trazabilidad entre requerimientos, riesgos y validación aplicada

Requerimiento intervenido	**Riesgo principal**	**Evidencia o validación**
Espacios arrendables y reservas	Errores en filtros, fechas, contacto y flujo de solicitud	Capturas funcionales del módulo, cobertura destacada y pruebas de componentes de reserva
Links mágicos para trabajos	Acceso inválido, expiración o uso inconsistente del enlace	Capturas del CRUD y del consumo del link en el formulario de registro
Certificados con encuesta obligatoria	Descarga sin cumplimiento del requisito previo	Capturas de configuración del certificado y bloqueo de descarga hasta responder la encuesta
Percepción de usabilidad	Registro incompleto de valoración o inconsistencia en resultados	Gestión de percepción por módulos, modal de calificación y evidencia funcional del flujo
CRM y límites de WhatsApp	Inconsistencias entre entidad, DTO e interfaz de configuración	Evidencia de cambios en entidad, DTO, interfaz y validaciones del flujo asociado

10.10. Resultados

Como resultado del trabajo final, se obtuvo una caracterización precisa del estado de las pruebas unitarias en Invessoft y una intervención concreta sobre módulos críticos donde la cobertura aporta valor directo al mantenimiento del sistema. En términos técnicos, el proyecto consolidó una suite de pruebas más representativa del comportamiento real de la aplicación, especialmente en componentes del frontend con validaciones, formularios y decisiones de interfaz, así como en módulos del backend con lógica de negocio sensible y mayor riesgo de regresión.

De manera complementaria, la documentación generada dejó una base metodológica reutilizable para nuevas iteraciones de aseguramiento. El aporte del proyecto no se limita a los módulos intervenidos, sino que establece criterios de priorización, diseño de casos y validación que pueden extenderse a otros sectores del sistema, como agenda, landings, trabajos académicos, perfiles, certificados, reportes y percepción de usabilidad. En una aplicación en evolución continua como Invessoft, este resultado fortalece la mantenibilidad no solo por el código cubierto con pruebas, sino también por la incorporación de una forma más sistemática de decidir qué probar, cómo probarlo y con qué evidencia sustentar su impacto.

11. Conclusiones

El desarrollo del proyecto permitió demostrar que el fortalecimiento de la calidad en Invessoft no dependía únicamente de aumentar un indicador global de cobertura, sino de intervenir de manera selectiva aquellos módulos donde convergen reglas de negocio, validaciones condicionales, interacción intensiva de usuario y cambios frecuentes sobre el producto. Bajo esta lógica, la pasantía evidenció que una estrategia de pruebas unitarias bien orientada ofrece mayor valor cuando se articula con el comportamiento real del sistema y con los riesgos de regresión identificados durante el mantenimiento evolutivo.

Los resultados obtenidos muestran además que la cobertura debe interpretarse en función de su representatividad funcional. Aunque el frontend partía de un porcentaje elevado, el análisis realizado permitió reconocer que gran parte de esa cobertura se concentraba en servicios, por lo que fue necesario extender el aseguramiento hacia componentes, formularios, diálogos y flujos de interacción donde se manifiestan con mayor claridad los errores percibidos por el usuario. En backend, el incremento alcanzado confirma que es posible avanzar de forma significativa cuando se priorizan servicios y módulos con alto impacto operativo, especialmente aquellos asociados a reservas, validaciones, configuraciones y notificaciones.

De igual forma, las actividades ejecutadas durante la pasantía permitieron consolidar una visión más amplia de la deuda técnica del sistema. La participación en módulos como CRM, agenda, trabajos, certificados, percepción de usabilidad, reportes y espacios arrendables mostró que Invessoft evoluciona sobre múltiples frentes simultáneamente y que, por tanto, la estrategia de aseguramiento debe entenderse como una práctica transversal de desarrollo y no como una tarea aislada al final de cada implementación. Esta conclusión refuerza la pertinencia de establecer criterios de integración, ejecución obligatoria de pruebas y metas mínimas de cobertura en módulos impactados.

En síntesis, el proyecto deja como aporte principal una base técnica y metodológica para continuar ampliando las pruebas unitarias con un criterio de priorización orientado al riesgo y al valor funcional. Más allá de las cifras finales obtenidas, el desarrollo realizado contribuye a que Invessoft disponga de un proceso más confiable para introducir cambios, detectar regresiones tempranamente y sostener la mantenibilidad de una plataforma que se encuentra en crecimiento continuo.

12. Trabajos futuros

Como continuación natural de este trabajo, se propone ampliar la estrategia de pruebas unitarias hacia módulos que aún presentan alta dependencia funcional y exposición al cambio, de modo que el aseguramiento cubra de manera más homogénea tanto el frontend como el backend de Invessoft. Esta ampliación puede incluir nuevos componentes, servicios y flujos de interacción relacionados con agenda, reportes, perfiles, certificados y funcionalidades emergentes incorporadas durante la evolución de la plataforma.

En esa misma línea, un trabajo futuro relevante consiste en definir una hoja de ruta de priorización continua para el aseguramiento, de manera que cada nuevo requerimiento o ajuste funcional pueda evaluarse no solo por su impacto de negocio, sino también por el riesgo técnico que introduce. Esto permitiría institucionalizar criterios para decidir qué módulos deben recibir pruebas unitarias de forma inmediata, cuáles requieren pruebas de integración y en qué casos conviene reforzar validaciones sobre componentes visuales, formularios o servicios con múltiples dependencias. Con una priorización de este tipo, la práctica de testing podría incorporarse de manera más orgánica al ciclo de desarrollo y mantenimiento de Invessoft.

También resulta pertinente complementar el avance alcanzado con prácticas adicionales de calidad que amplíen el alcance del esquema ya existente en los flujos de despliegue, especialmente mediante la extensión de criterios de cobertura hacia otros módulos aún no intervenidos con el mismo nivel de profundidad y la incorporación progresiva de pruebas de integración y validaciones end-to-end en procesos críticos. Con ello, el trabajo realizado podría evolucionar hacia un esquema de aseguramiento más robusto, medible y alineado con el crecimiento funcional del sistema.

Asimismo, sería conveniente fortalecer la trazabilidad entre incidentes reportados, módulos modificados y casos de prueba implementados, con el fin de construir una base histórica que facilite identificar patrones recurrentes de falla y oportunidades de refactorización. Esta proyección no solo ayudaría a mejorar la cobertura de manera cuantitativa, sino también a aumentar su utilidad práctica dentro del proceso de desarrollo. En un sistema con evolución constante, disponer de esa relación entre cambios, riesgos y evidencia de validación permitiría orientar con mayor precisión los esfuerzos de calidad y sostener una estrategia de mejora continua a largo plazo.

Referencias

[1] N. J. R. M., M. L. Borda y O. Maldonado, "Activities for social appropriation of science and technology and meeting spaces with Colombian audiences. A look at the projects supported by COLCIENCIAS 2005-2010: Una mirada a los proyectos apoyados por Colciencias 2005 – 2010," *Folios*, n.º 25, págs. 165-191, nov. de 2011. dirección: https://revistas.udea.edu.co/index.php/folios/article/view/10606

[2] P. Delfín. "Programa Delfín - Agencia de Noticias Univalle. "dirección: https://www.univalle.edu.co/proyeccion-internacional/sede-caicedonia-en-el-%20programa-delfin

[3] M. Corporation, *Microsoft Conference Management Toolkit*. visitado 12 de jun. de 2025. dirección: https://cmt3.research.microsoft.com/About

[4] O. Abstracts, *Oxford Abstracts — The Academic Conference Platform*. visitado 12 de jun. de 2025. dirección: https://oxfordabstracts.com/

[5] R. Andreu, J. Ricart y J. Valor, *Estrategia y sistema de información*. 1991, págs. 187-187.

[6] K. C. Laudon y J. P. Laudon, *Sistemas de Información Gerencial*, 12.ª ed.

[7] IONOS, *Conceptos básicos: definición de web app y ejemplos*, Accedido el 6 de junio de 2025, 2019. dirección: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-una-web-app-y-que-clases-hay/

[8] K. Beck, *Test-driven development: by example* (The Addison-Wesley signature series), eng, 20. printing. Boston: Addison-Wesley, 2015, ISBN: 978-0-321-14653-3.

[9] J. Kiruthika, S. Khaddaj, D. Greenhill y J. Francik, "User Experience Design in Web Applications," en *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, 2016, págs. 642-646. DOI: 10.1109/CSE-EUC-DCABES.2016.253

[10] IONOS, *Refactorización: cómo mejorar el código fuente*, es, sep. de 2020. visitado 12 de jun. de 2025. dirección: https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/que-es-la-refactorizacion/

[11] GeeksforGeeks, *Unit Testing - Software Testing*, en-US, Section: Software Engineering, jun. de 2025. visitado 12 de jun. de 2025. dirección: https://www.geeksforgeeks.org/unit-testing-software-testing/

[12] I. O. T. Gómez, I. Pedro, P. R. López e I. J. S. Bacalla, "Criterios de selección de metodologías de desarrollo de software," *Ind. Data*, vol. 13, n.º 2, págs. –, 2010.

[13] Andesco, "Resumen Ejecutivo de Los 17 Objetivos de Desarrollo Sostenible," vol. 4, n.º 1, págs. 64-75, 2016.