Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Ingeniería de Sistemas y Computación

Prototipo de un Sistema de Consolidación de Información para Programas de Posgrado en Medicina

Carlos Andrés Camacho Peña
Laura Isabella Rojas Cardenas

Director: Ms. Juan Carlos Martínez Arias

18 de Julio de 2025

Resumen

El presente proyecto desarrolló un prototipo de sistema que consolida, en un único repositorio consultable, la información de los programas de especialización clínica en medicina ofrecidos por universidades colombianas con Acreditación de Alta Calidad. La iniciativa surge ante la dispersión y heterogeneidad de datos publicados en los portales institucionales, factores que dificultan a los médicos recién graduados comparar opciones académicas y cumplir con plazos de admisión. El prototipo aborda la problemática mediante scrapers modulares implementados en Python y Selenium, parametrizados con un archivo JSON que describe los selectores y rutas específicas de cada portal. Los datos extraídos se normalizaron y almacenaron en una base relacional PostgreSQL, lo que garantizó consistencia y posibilitó consultas complejas. Sobre esta base se construyó una aplicación web con Django que permite filtrar programas por universidad o nombre de especialización, presentar tarjetas con información clave y desplegar fichas detalladas que incluyen duración, costos, fechas y contactos. El sistema fue validado mediante pruebas unitarias, de integración y de usabilidad; estas últimas, realizadas con perfiles representativos del público objetivo, mostraron una aceptación superior al 90 % en claridad, relevancia y velocidad, sin reportes de fallos ni enlaces rotos. Los resultados confirman la viabilidad técnica de automatizar la obtención y estandarización de datos académicos y evidencian el valor añadido de una interfaz simple para facilitar la toma de decisiones educativas. Se concluye que el prototipo cumple los objetivos planteados y puede escalarse a más instituciones y programas, incorporando filtros avanzados y notificaciones automatizadas para convertirse en un buscador académico integral orientado al sector salud.

Palabras Clave: Web scraping, programas de posgrado, medicina, consolidación de información, Django, Selenium.

Abstract

The present project developed a prototype system that consolidates information on clinical medical specialization programs offered by Colombian universities with High Quality Accreditation into a single searchable repository. This initiative emerged in response to the dispersion and heterogeneity of data published on institutional websites, making it challenging for newly graduated students to compare academic opportunities and apply on time. The prototype addresses this issue through modular scrapers implemented in Python and Selenium, parameterized using a JSON file that specifies the selectors and specific paths for each portal. The extracted data was normalized and stored in a PostgreSQL relational database, ensuring consistency and enabling complex queries. Based on this database, a web application was built using Django, allowing users to filter programs by university or specialization name, display cards with key information, and access detailed profiles that include duration, costs, deadlines, and contact details. The system was validated through unit, integration, and usability testing; the latter, conducted with representative users from the target audience, showed over 90 % approval in terms of clarity, relevance, and speed, with no reports of errors or broken links. The results confirm the technical feasibility of automating the retrieval and standardization of academic data, highlighting the added value of a simple interface in supporting educational decision-making. It is concluded that the prototype meets the stated objectives and can be scaled to include more institutions and programs by incorporating advanced filters and automated notifications, aiming to become a comprehensive academic search engine for the healthcare sector.

Keywords: Web scraping, postgraduate programs, medicine, information consolidation, Django, Selenium.

Índice general

Capítulo 1

Introducción

Cada año miles de médicos recién graduados deben navegar por decenas de portales universitarios para encontrar información fiable sobre especializaciones clínicas; la dispersión de datos, la falta de estandarización y la actualización irregular de los sitios web convierten esa búsqueda en un proceso lento y propenso a errores. Con el fin de mitigar ese problema, este proyecto desarrolló un prototipo de sistema que consolida automáticamente la oferta de posgrados médicos publicados por universidades colombianas con Acreditación de Alta Calidad.

El trabajo partió de una pregunta guía: ¿Cómo desarrollar el prototipo de un sistema que consolide la información de los programas académicos de posgrado en medicina que ofrecen las diversas universidades, mediante técnicas de obtención de información en la web de forma automática?

Se seleccionaron cuatro instituciones (Universidad Nacional, Universidad El Bosque, Universidad del Valle y Pontificia Universidad Javeriana Cali) con estructuras web heterogéneas, lo que permitió probar la robustez del enfoque.

Se implementaron scrapers modulares en Python/Selenium, parametrizados a través de un archivo JSON, que recorren los portales, extraen los campos definidos y normalizan los valores antes de volcarlos en una base PostgreSQL diseñada para garantizar integridad referencial. Sobre esa base se construyó una aplicación Django que ofrece filtros por universidad y nombre de programa, tarjetas con información clave y vistas detalladas, articulada según el modelo C4 (niveles 1-3). Las pruebas unitarias y de usabilidad mostraron la estabilidad del sistema: el 100 % de los casos de prueba superaron los criterios establecidos y los usuarios calificaron con 4 o 5 la claridad, relevancia y velocidad de la búsqueda, así como la legibilidad de la ficha de detalle.

En conjunto, el documento describe la ruta seguida, desde la contextualización del problema hasta la validación del prototipo, y sienta las bases para futuras extensiones que incluyan más universidades, filtros avanzados y notificaciones proactivas.

Capítulo 2

Contextualización del Proyecto

2.1. Definición del Problema

2.1.1. Planteamiento del Problema

Cada año se gradúan una gran cantidad de personas de pregrado, en el 2023 se graduaron 266.548 estudiantes de Pregrado en Colombia, según el Ministerio de Educación [1], y muchos de ellos desean continuar su formación académica realizando una especialización o posgrado. Uno de los primeros pasos en la búsqueda de este objetivo es investigar en cada universidad los programas de posgrados ofrecidos por estas entidades, recurriendo a medios digitales, principalmente los sitios web oficiales de cada universidad, para poder obtener la información necesaria para tomar una decisión. Sin embargo, las ofertas tanto de universidades como de los propios programas de estudio son muy extensas, sumado a información confusa, desactualizada o ausente sobre aspectos importantes para aplicar a unos de los programas ofrecidos; provocando una mayor dificultad en el acceso a la información, generando que los aspirantes no se presenten a posibles oportunidades deseadas o se presenten con requisitos faltantes, y que algunos programas no logren llenar las plazas destinadas.

La ausencia de un sistema que consolide la información de manera automática de los programas académicos en posgrado de Medicina ofrecidos por diversas universidades, crea un desafío para los estudiantes recién graduados de pregrado en Medicina, que en el año 2023 fueron 7.672 según el Ministerio de Educación [1], quienes buscan inscribirse en dichos programas. Actualmente, el proceso de búsqueda es ineficiente, disperso y manual, lo que puede resultar en una pérdida de oportunidades. Así mismo, muchos de estos estudiantes no solo exploran oportunidades en su ciudad de origen, sino que también buscan opciones en otras ciudades que se ajusten a sus necesidades y aspiraciones.

Los problemas existentes se centran en la heterogeneidad y la dispersión de los datos disponibles en los sitios web de las universidades. Los datos relacionados con los programas de posgrado suelen estar distribuidos en múltiples plataformas, cada una con su propio formato, estructura, y nivel de actualización, lo que complica su acceso y procesamiento. La ausencia de estandarización en la presentación de la información provoca dificultades para extraer, integrar y comparar los datos de manera eficiente.

2.1.2. Formulación

La pregunta ahora es: **¿Cómo desarrollar el prototipo de un sistema que consolide la información de los programas académicos de posgrado en medicina que ofrecen las diversas universidades, mediante técnicas de obtención de información en la web de forma automática?**

2.1.3. Sistematización

Para resolver esta pregunta se deben tener en cuenta algunas subpreguntas:

- ¿Cómo identificar las estructuras y datos que tiene las universidades en sus páginas web?
- ¿Qué estrategia de extracción, estandarización y actualización automática de datos debe implementarse para que sean almacenados en una base de datos escalable con el objetivo de asegurar la consistencia, uniformidad y actualidad de la información sobre los programas académicos?
- ¿Cómo desarrollar una interfaz de usuario intuitiva que permita a los usuarios acceder y consultar fácilmente la información consolidada de los programas académicos?
- ¿Cómo evaluar la funcionalidad del sistema con el fin que cumpla con los requisitos establecidos?

2.2. Objetivos

2.2.1. Objetivo General

Desarrollar un prototipo de un sistema que consolide la información de los programas académicos de posgrado en medicina ofrecidos por las universidades nacionales con Acreditación de Alta Calidad, mediante técnicas de obtención de información en la web de forma automática.

2.2.2. Objetivos Específicos

1. Identificar las estructuras y datos que manejan las páginas web de las universidades que ofrecen programas de especialización en medicina.
2. Extraer la información de la web de manera estructurada, aplicando normas y formatos para su estandarización, asegurando su almacenamiento en una base de datos escalable.
3. Desarrollar una interfaz de usuario intuitiva que permita a los usuarios acceder y consultar fácilmente la información consolidada de los programas académicos.
4. Realizar pruebas del funcionamiento del sistema con datos reales con el fin que cumpla con los requisitos establecidos.

2.3. Marco de Referencia

2.3.1. Marco Teórico

En el contexto actual de la educación superior, la demanda de programas de posgrado, especialmente en áreas como la medicina, ha aumentado significativamente. Esta tendencia responde a la necesidad de los profesionales de continuar su formación académica para mejorar sus competencias y oportunidades laborales. Sin embargo, la búsqueda y consolidación de información relevante sobre los programas de posgrado disponibles presenta desafíos considerables debido a la dispersión de los datos en los sitios web de las universidades. Ante esta problemática, el uso de tecnologías de obtención de información en la web, como el web scraping y web crawling, combinadas con sistemas de gestión de bases de datos (SQL y NoSQL) para la estandarización y consolidación de los datos, se presenta como una solución para centralizar la información, garantizando su integridad y permitiendo un acceso eficiente a los datos actualizados.

2.3.1.1. Web Scraping

Es un proceso automatizado que permite extraer datos de sitios web. Este método implica varias etapas: primero, se debe crear una plantilla de scraping que defina qué datos se necesitan y cómo se encuentran en las páginas web. A continuación, se explora el sitio para identificar las estructuras de las páginas que contienen la información deseada. Luego, se automatiza la extracción de datos mediante herramientas o scripts, y finalmente, se almacenan los datos extraídos en una base de datos para su análisis posterior [2]. El web scraping es una técnica eficaz para obtener grandes volúmenes de datos desde diversas fuentes web, evitando las limitaciones de la recolección manual. Herramientas como la librería Selenium en Python son comúnmente utilizadas para automatizar la extracción de datos de sitios que no proporcionan una API [3].

Además de su utilidad general para la recolección automatizada de datos, el web scraping ha demostrado ser una herramienta eficaz en sistemas de búsqueda especializados, como los motores de búsqueda de ofertas laborales. Por ejemplo, la combinación de técnicas de scraping con algoritmos de clasificación como Naïve Bayes permite no solo extraer grandes volúmenes de datos desde múltiples portales, sino también organizarlos y categorizarlos automáticamente según atributos relevantes, como el área profesional o los requisitos del cargo. Este tipo de integración mejora significativamente la eficiencia en la recolección y análisis de información, facilitando la toma de decisiones en contextos donde los datos están distribuidos en diversas fuentes y formatos [2].

2.3.1.2. Web Crawler

También conocido como rastreo web, se refiere al uso de scripts automatizados para navegar por la web y recolectar datos. Los webs crawlers siguen un conjunto de reglas o patrones predefinidos para visitar diferentes páginas y recoger enlaces y datos. Su propósito es clasificar el contenido web y extraer información de múltiples páginas de manera sistemática [3]. Aunque los crawlers y las herramientas como Selenium comparten la capacidad de automatizar la navegación, los crawlers suelen operar a una escala más amplia, enfocándose en recolectar grandes volúmenes de datos de

diversas fuentes web. En aplicaciones recientes, como la preservación digital del patrimonio cultural, los web crawlers han demostrado su eficacia para recolectar grandes volúmenes de datos desde fuentes distribuidas, como redes sociales, plataformas multimedia y sitios web especializados. Estos sistemas automatizados no solo recorren páginas siguiendo enlaces, sino que también permiten aplicar filtros semánticos y temporales para identificar información relevante en contextos complejos. Por ejemplo, en el caso del patrimonio de la región de Nínive, se utilizó un enfoque basado en rastreo y scraping para extraer datos textuales, imágenes y videos de plataformas como Twitter, Instagram y YouTube, lo que permitió construir un repositorio estructurado en la nube para su análisis, visualización y conservación futura [3].

2.3.1.3. Bases de Datos

Es un sistema que permite almacenar, organizar y gestionar información de forma estructurada, de tal manera que facilite su acceso, consulta y manipulación. Las bases de datos son fundamentales para garantizar la integridad y coherencia de los datos. Existen dos tipos principales de bases de datos [4]:

- **Bases de Datos Relacionales (SQL):** Son sistemas de gestión que organizan la información en tablas estructuradas en filas y columnas. Estas tablas representan entidades, y sus atributos permiten la organización de los datos de manera coherente y sistemática. Estas bases de datos se destacan por garantizar la integridad de los datos mediante reglas de normalización [4].

- **Bases de Datos No Relacionales (NoSQL):** Estas bases de datos están diseñadas para gestionar grandes volúmenes de datos que no necesariamente siguen un esquema estructurado, permitiendo una mayor flexibilidad en la organización de la información. A diferencia de las bases de datos relacionales, NoSQL no requiere que los datos estén organizados en tablas, sino que puede almacenar información en diferentes formatos, como documentos, grafos, columnas o pares claves-valor [4].

2.3.1.4. Metodología SCRUM

Es una metodología ágil que permite gestionar proyectos de manera flexible, facilitando la colaboración y la adaptación a cambios. Se basa en un enfoque iterativo e incremental, organizando el trabajo en ciclos cortos llamados "sprints", que generalmente duran de dos a cuatro semanas. Cada sprint busca entregar una parte funcional del producto, permitiendo recibir retroalimentación continua y realizar ajustes según las necesidades del proyecto. SCRUM está compuesto por roles específicos, como el Product Owner, el SCRUM Master y el equipo de desarrollo, además de eventos clave como las reuniones diarias (Daily Scrum), la planificación del sprint (Sprint Planning) y la retrospectiva del sprint (Sprint Retrospective), que aseguran la mejora constante en el proceso de desarrollo y maximizar el valor entregado en cada etapa del proyecto [5].

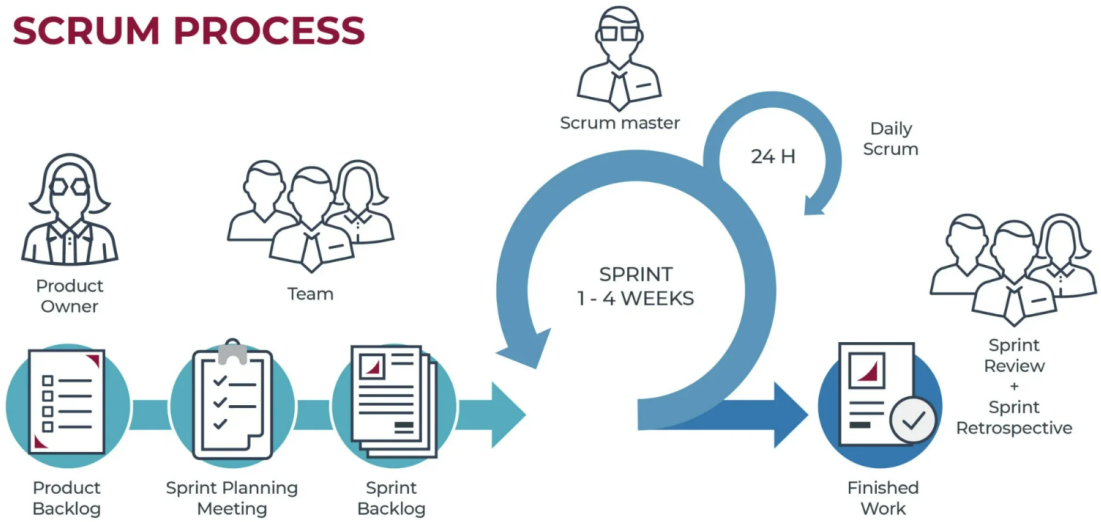


Figura 2.1: Representación del proceso SCRUM. Fuente: [6]

La figura 2.1, es una representación visual del proceso SCRUM, la cual permite entender cómo interactúan los elementos principales a lo largo de un sprint. La figura ilustra la secuencia de eventos, los roles involucrados y cómo el backlog, al final del proceso, es finalizado y se da mediante iteraciones cortas, promoviendo la entrega continua de valor.

2.3.2. Antecedentes

2.3.2.1. Web Scraping and Naïve Bayes Classification for Job Search Engine [2]

Este estudio explora cómo la técnica de web scraping combinada con el algoritmo de clasificación Naïve Bayes puede simplificar la búsqueda de vacantes de empleo a través de un motor de búsqueda específico para trabajos en Indonesia. Este enfoque automatiza la recolección de información de múltiples sitios web y clasifica la información de las vacantes de empleo para facilitar el acceso de los usuarios a oportunidades que se ajusten a sus intereses y habilidades. Este caso resalta la efectividad del uso de técnicas de obtención automática de datos para consolidar información dispersa en una única plataforma accesible y de fácil consulta

2.3.2.2. Extracting and Archiving Data from Social Media to Support Cultural Heritage Preservation in Nineveh [3]

Este estudio implementa técnicas de web scraping para la recolección de datos desde redes sociales con el fin de preservar el patrimonio cultural de Nínive. Utiliza herramientas como Python y Selenium para extraer datos de sitios como Twitter, Instagram y YouTube, los cuales luego son limpiados, organizados y almacenados en una base de datos NoSQL y en Amazon S3. Este proceso

permite manejar grandes volúmenes de datos no estructurados y almacenarlos de forma eficiente para usos futuros como análisis, visualización y preservación digital del patrimonio.

2.3.2.3. Browserless Web Data Extraction: Challenges and Opportunities [7]

Este estudio discute los desafíos y oportunidades de la extracción de datos web sin el uso de navegadores, una técnica que puede ser más eficiente en términos de recursos y tiempo en comparación con los métodos que utilizan navegadores embebidos para simular acciones de usuario. Propone un sistema para la conversión automática de "wrappers"basados en navegador a "wrappers"sin navegador, lo cual puede tener aplicaciones significativas en la recolección eficiente de datos desde la web.

2.3.2.4. Ingredient Recipe Algorithm Using Web Mining and Web Scraping for Smart Chef [8]

Este estudio propone un algoritmo para extraer detalles de recetas utilizando técnicas de web scraping con el objetivo de facilitar la búsqueda de ingredientes específicos en recetas, lo cual es útil para seguir un plan de dieta personalizado. Este algoritmo emplea Python y MongoDB para extraer contenido de sitios web de recetas y almacenarlo en una base de datos, permitiendo una búsqueda eficiente y automatizada de recetas basadas en ingredientes seleccionados por el usuario. Esta técnica demuestra el potencial del scraping para extraer y organizar grandes volúmenes de datos específicos de la web para su uso práctico.

2.3.2.5. Data Mining for Identifying Trends in Markets [9]

Este estudio presenta el uso de técnicas de data mining para identificar tendencias en el mercado de ventas de automóviles en Rumanía, utilizando un web scraper para recopilar datos públicamente disponibles de una plataforma de ventas de automóviles en línea. La solución implementada utiliza un backend desarrollado en C# y ASP.NET, junto con un frontend en ReactJS, para procesar, analizar y presentar la información de manera amigable al usuario. Además, se utiliza una base de datos SQL creada mediante el enfoque Çode-First"de Entity Framework (EF) para almacenar y gestionar la información extraída. Esta solución permite calcular el precio promedio de un automóvil basado en las especificaciones seleccionadas por el usuario y facilita la identificación de tendencias en el mercado para mejorar la toma de decisiones comerciales.

Aunque los estudios citados comparten el uso de técnicas de web scraping y otras tecnologías de extracción de datos, el enfoque de este trabajo de grado difiere significativamente en cuanto a su aplicación y alcance. A diferencia de los proyectos que se centran en la clasificación de ofertas de empleo, la preservación de datos culturales o la extracción de recetas e información de mercado, el presente proyecto tiene como objetivo desarrollar un sistema que consolide la información académica de programas de posgrado en medicina ofrecidos por diversas universidades. Además, este trabajo incluye el reto de estandarizar y actualizar de manera automatizada datos provenientes de

múltiples fuentes institucionales, integrándolos en un sistema de consulta accesible y eficiente para los estudiantes.

Capítulo 3

Identificación y análisis inicial de información

3.1. Investigación y selección de universidades colombianas para el prototipo

Se llevó a cabo una investigación para identificar las universidades colombianas con Acreditación de Alta Calidad, donde se encontró un total de 106 instituciones, incluyendo tanto sedes principales como seccionales [1].

1	NOMBRE_INSTITUCIÓN	PRINCIPAL_S	SECTOR	DEPARTAMENT(	MUNICIPIO_DOMICILIO
29	UNIVERSIDAD DE PAMPLONA	Principal	Oficial	Norte de Santar	Pamplona
30	UNIVERSIDAD DEL MAGDALENA - UNIMAGDALENA	Principal	Oficial	Magdalena	Santa Marta
31	UNIVERSIDAD DE SUCRE	Principal	Oficial	Sucre	Sincelejo
32	UNIVERSIDAD DE LA GUAJIRA	Principal	Oficial	La Guajira	Riohacha
33	UNIVERSIDAD DE ANTIOQUIA	Seccional	Oficial	Antioquia	El Carmen de Viboral
34	UNIVERSIDAD DE ANTIOQUIA	Seccional	Oficial	Antioquia	Andes
35	UNIVERSIDAD DE ANTIOQUIA	Seccional	Oficial	Antioquia	Caucasia
36	UNIVERSIDAD DE ANTIOQUIA	Seccional	Oficial	Antioquia	Puerto Berrío
37	UNIVERSIDAD DE ANTIOQUIA	Seccional	Oficial	Antioquia	Turbo
38	UNIVERSIDAD DISTRITAL-FRANCISCO JOSE DE CALDAS	Principal	Oficial	Bogotá, D.C.	Bogotá, D.C.
39	PONTIFICIA UNIVERSIDAD JAVERIANA	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
40	PONTIFICIA UNIVERSIDAD JAVERIANA	Seccional	Privado	Valle del Cauca	Santiago de Cali
41	UNIVERSIDAD SANTO TOMAS	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
42	UNIVERSIDAD SANTO TOMAS	Seccional	Privado	Santander	Bucaramanga
43	UNIVERSIDAD EXTERNADO DE COLOMBIA	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
44	FUNDACION UNIVERSIDAD DE BOGOTA - JORGE TADEO LOZANO	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
45	FUNDACION UNIVERSIDAD DE BOGOTA - JORGE TADEO LOZANO	Seccional	Privado	Bolívar	Cartagena de Indias
46	UNIVERSIDAD CENTRAL	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
47	UNIVERSIDAD PONTIFICIA BOLIVARIANA	Principal	Privado	Antioquia	Medellín
48	UNIVERSIDAD DE LA SABANA	Principal	Privado	Cundinamarca	Chía
49	UNIVERSIDAD EAFIT-	Principal	Privado	Antioquia	Medellín
50	UNIVERSIDAD DEL NORTE	Principal	Privado	Atlántico	Barranquilla
51	COLEGIO MAYOR DE NUESTRA SEÑORA DEL ROSARIO	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
52	UNIVERSIDAD DE SAN BUENAVENTURA	Seccional	Privado	Valle del Cauca	Santiago de Cali
53	UNIVERSIDAD DE SAN BUENAVENTURA	Seccional	Privado	Antioquia	Medellín
54	UNIVERSIDAD DE SAN BUENAVENTURA	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
55	UNIVERSIDAD CATOLICA DE COLOMBIA	Principal	Privado	Bogotá, D.C.	Bogotá, D.C.
56	UNIVERSIDAD MARIANA	Principal	Privado	Nariño	Pasto
57	UNIVERSIDAD DE MANIZALES	Principal	Privado	Caldas	Manizales
58	UNIVERSIDAD PONTIFICIA BOLIVARIANA	Seccional	Privado	Santander	Bucaramanga
59	UNIVERSIDAD DE SAN BUENAVENTURA	Seccional	Privado	Bolívar	Cartagena de Indias

Figura 3.1: Fragmento del listado de universidades colombianas con Acreditación de Alta Calidad. Fuente: [1]

En la 3.1, se observa una parte del archivo con la recopilación de estas instituciones, entre las cuales se encuentran tanto universidades públicas como privadas distribuidas en diferentes regiones del país. Dentro de este conjunto, se identificaron específicamente 39 universidades que ofrecen programas de especialización clínica en medicina.

Desde un comienzo, se determinó que serían seleccionadas cuatro universidades para el prototipo, escogiendo dos instituciones en Bogotá por ser la capital del país, y dos en Cali por ser la ciudad donde se desarrolla el proyecto. Además, se decidió que en cada ciudad se seleccionaría una universidad pública y una privada para obtener una mayor diversidad de contextos académicos y administrativos. En Cali, se optó por incluir a la Pontificia Universidad Javeriana Cali por ser la institución en la cual se lleva a cabo este proyecto de grado.

Las universidades seleccionadas fueron:

- Universidad Nacional de Colombia (Bogotá, pública)
- Universidad El Bosque (Bogotá, privada)
- Universidad del Valle (Cali, pública)
- Pontificia Universidad Javeriana Cali (Cali, privada)

Esta selección diversa permite capturar distintos enfoques institucionales frente a la oferta de especializaciones médicas, facilitando así que el prototipo responda a una variedad de necesidades informativas por parte de usuarios con distintos intereses y contextos.

3.2. Análisis de la estructura web y definición de información

La selección de las universidades no solo se dio por su reconocida calidad académica y oferta amplia en especializaciones clínicas médicas, sino especialmente por la variedad en la organización y estructura de sus sitios web. Este aspecto representó uno de los desafíos iniciales más importantes para la extracción automatizada de datos, ya que cada universidad presenta su información utilizando estructuras HTML diversas.

En el caso de la Universidad El Bosque [10], esta organiza la información mediante múltiples elementos , donde cada especialización se presenta en tarjetas con enlaces directos hacia páginas individuales de los programas. Además, esta universidad emplea mecanismos de paginación para mostrar todas las especializaciones disponibles, requiriendo interacción adicional del scraper para navegar a través de distintas páginas. Parte de la información adicional, como fechas específicas de exámenes y requisitos particulares, se encuentra en secciones desplegables o acordeones, que no siempre están disponibles debido a la periodicidad de admisiones.

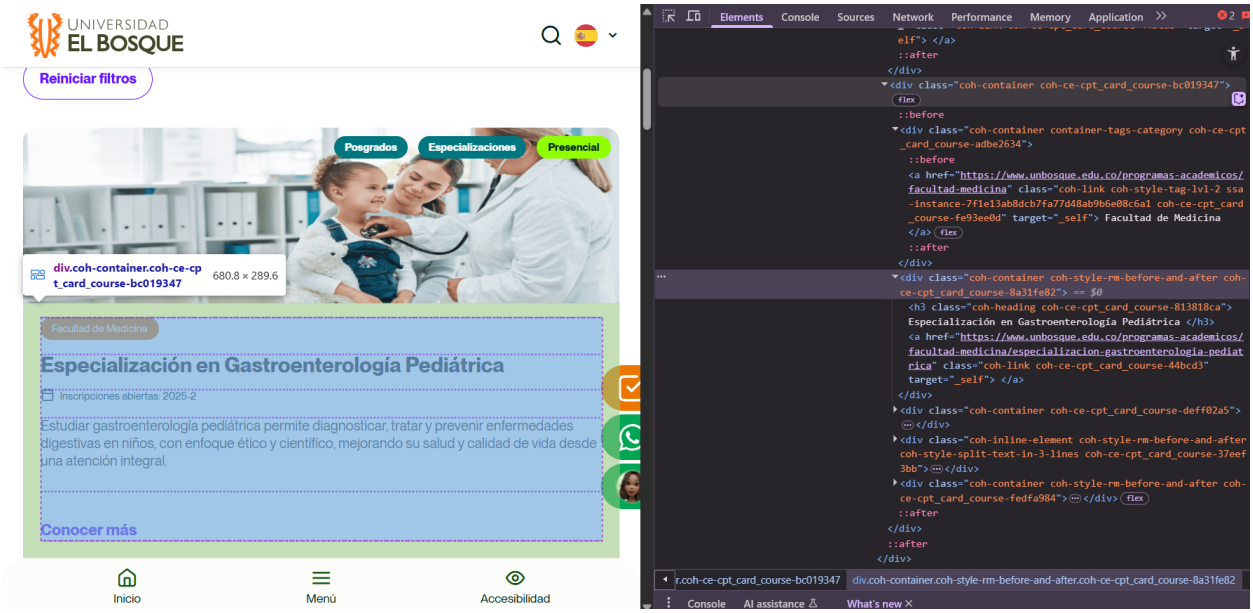


Figura 3.2: Ejemplo de la estructura de una especialización médica en la página web de la Universidad El Bosque

La Figura 3.2 muestra la estructura interna de una tarjeta utilizada para presentar un programa de especialización médica en la página de la Universidad El Bosque. Cada tarjeta está construida a partir de múltiples elementos , sin emplear listas ni estructuras jerárquicas uniformes. Aunque esta presentación resulta clara y atractiva para los usuarios, representó un reto para el proceso de automatización, ya que dificulta la identificación consistente de la información mediante técnicas de scraping.

Por su parte, la Universidad Nacional [11] presenta un formato basado principalmente en listas estructuradas para las especializaciones, desde donde se debe navegar a través de varios enlaces para acceder a detalles específicos del programa académico y procedimientos de inscripción. Esta información suele estar organizada en tablas o modales interactivos que contienen descripciones más detalladas y guías paso a paso, lo cual representó un reto adicional debido a la necesidad de automatizar la navegación entre múltiples páginas y ventanas emergentes.

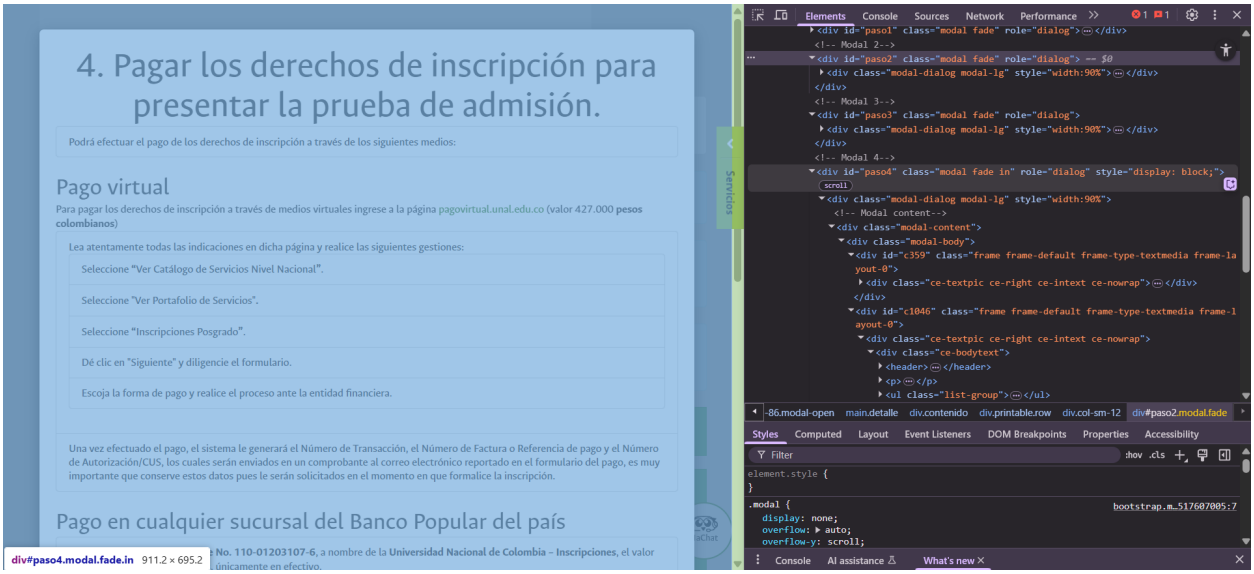


Figura 3.3: Ejemplo de modal interactivo utilizado por la Universidad Nacional para mostrar detalles sobre el proceso de inscripción.

La Figura 3.3 muestra un ejemplo de los modales emergentes que utiliza la Universidad Nacional para desplegar información clave, en este caso para mostrar el valor de la inscripción. Este tipo de elementos dinámicos, aunque mejora la experiencia del usuario, dificulta el proceso de extracción automatizada de datos, ya que requiere que el scraper interactúe con componentes que no están visibles de forma directa en el DOM al cargar la página inicial.

En el caso de la Pontificia Universidad Javeriana Cali [12], la información específica por especialización se encuentra inicialmente en listas filtradas por facultad, especialmente la facultad de salud. Desde estas listas, cada especialización tiene enlaces hacia páginas individuales donde los detalles están organizados principalmente en listas y cuadros informativos, mostrando datos relevantes como título otorgado, duración, modalidad y costos. Información adicional, como contactos y procesos específicos de inscripción, se encuentra en contenedores desplegables o cartas interactivas, requiriendo interacción adicional para visualizar los contenidos detallados.

Finalmente, la Universidad del Valle [13] presenta una estructura similar a la Universidad Nacional en términos de navegación entre múltiples enlaces para obtener información completa. Inicialmente, ofrece una lista con enlaces a cada especialización, que dirige a páginas individuales que contienen tablas con información básica como código SNIES, duración, créditos y costo. Sin embargo, gran parte de la información adicional sobre inscripciones, fechas y número de cupos se encuentra en paneles expandibles o tablas interactivas, lo que implica desafíos en términos de automatización debido a la necesidad de esperar la carga dinámica del contenido.

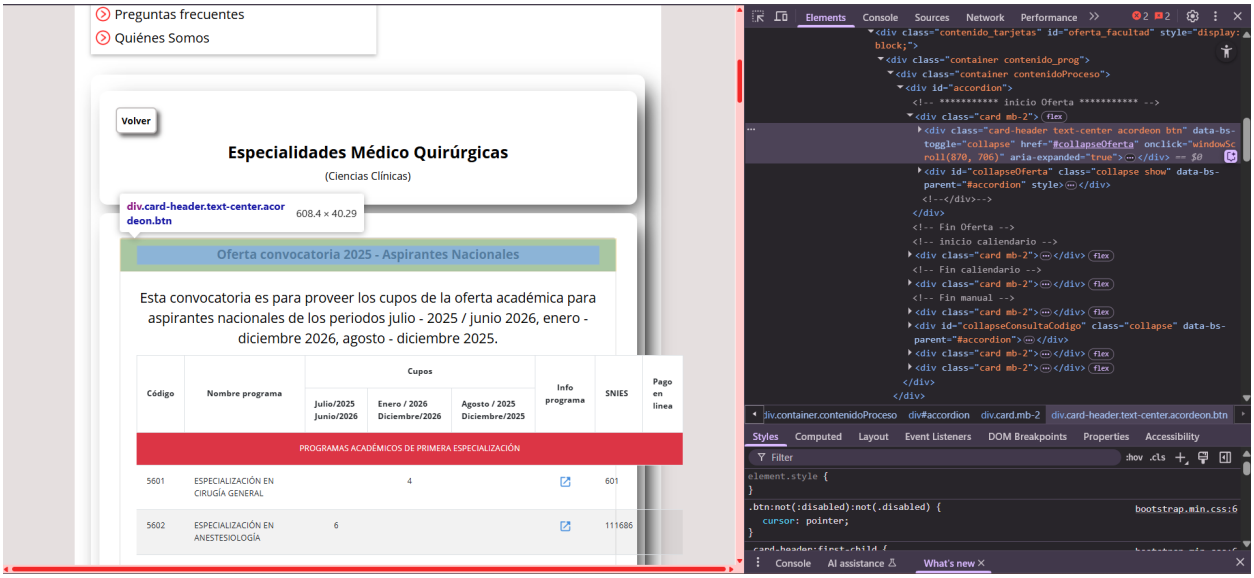


Figura 3.4: Ejemplo de los paneles desplegables utilizados por la Universidad del Valle

La Figura 3.4 muestra uno de los paneles expandibles empleados por la Universidad del Valle para organizar información detallada sobre sus programas de especialización médica. Esta estrategia de diseño, basada en componentes interactivos como acordeones y tablas dinámicas, implica que el sistema automatizado debe esperar a que los elementos se carguen completamente y realizar acciones específicas para acceder a los datos relevantes, lo cual incrementa la complejidad del proceso de extracción.

En conjunto, estos elementos evidencian que uno de los principales desafíos técnicos del prototipo fue la necesidad de adaptar el proceso de scraping a estructuras HTML dinámicas y no uniformes, lo cual implicó el diseño de estrategias específicas por universidad.

En la siguiente tabla se presenta el resultado inicial de la información disponible en cada una de estas universidades, recopilada directamente desde sus sitios web oficiales:

	Nacional Bogotá	Bosque	Valle	Javeriana Cali
Precio inscripción	X	X	X	
Precio semestre	X	X	X	X
codigo SNIES	X	X	X	X
Paso a paso del proceso de admisión	X	X	X	X
Perfil del aspirante	X	X		X
Perfil egresado	X	X	X	X
Duración del programa	X	X	X	X
Número de cupos			X	
Periodicidad de Admisiones		X	X	
Plan de estudios	X	X	X	X
Objetivos de la formación	X	X	X	X
Fecha de inicio de inscripciones	X	X	X	x
Fecha de cierre de inscripciones	X	X	X	x
Papeles para la inscripción	X	X		X
Fecha de examen de admisión		X	X	Se les envía un email a los que pasan
Fecha de la prueba Psicotécnicas		X		N/A
Fecha de la entrevista		X		Se les envía un email a los que pasan
Fecha de publicación de admitidos		X	X	X
Director(a) especialización	X	X	X	
Número de contacto para más información	X	X	X	X
Profesores del programa		X (En el plan de estudios, algunas materias indican quién es el profesor)	X	
Semilleros de investigación			X	
Apoyo financiero				X
Posibilidad de internacionalización			X	

Figura 3.5: Información disponible inicialmente en los sitios web de las universidades seleccionadas

Tras este análisis inicial, se llevó a cabo una revisión detallada para identificar los elementos de información más relevantes y comunes en todas las instituciones. Este proceso permitió definir claramente un conjunto básico de información que se integraría en el prototipo, facilitando la estandarización y posterior comparación objetiva entre los programas académicos.

Adicionalmente, se optó por incluir ciertos datos adicionales que, aunque no estuvieran disponibles en todos los casos, aportan un valor significativo para el usuario final en términos de toma de decisiones. Estos datos adicionales se extrajeron únicamente cuando se encontraban disponibles en las fuentes oficiales.

A continuación, se presenta la tabla final con el conjunto de campos seleccionados, indicando cuáles son comunes a todas las universidades y cuáles se presentan únicamente en algunas:

	Nacional Bogotá	Bosque	Valle	Javeriana Cali
Precio inscripción	X	X	X	
Precio semestre	X	X	X	X
codigo SNIES	X	X	X	X
Número de cupos			X	
Periodicidad de Admisiones		X	X	
Duración del programa	X	X	X	X
Título Otorgado	X	X	X	X
Fecha de inicio de inscripciones	X	X	X	X
Fecha de cierre de inscripciones	X	X	X	X
Fecha de examen de admisión		X	X	X
Fecha de publicación de admitidos		X	X	X
Número telefónico de contacto para más información	X	X	X	X
Nombre del contacto para más información	X		X	X
Correo electrónico de contacto para más información	X	X	X	X
URL del programa académico	X	X	X	X
Perfil del aspirante	X	X		X
Perfil egresado	X	X	X	X
Información del programa	X	X	X	X
Número de Creditos	X	X	X	X

Figura 3.6: Campos finales seleccionados para extracción de datos

La figura 3.6 resume los campos que fueron seleccionados como resultado final del proceso de depuración y análisis de la información disponible en los sitios web de las universidades. Si bien inicialmente se identificó una mayor cantidad de datos potenciales, se optó por conservar aquellos que resultan más relevantes para el usuario final. En consecuencia, estos son los campos que se extrajeron mediante el web de scraping implementado en el prototipo.

3.3. Historias de Usuario y Diagrama de Casos de Uso

Para el desarrollo de este proyecto se optó por adoptar la metodología ágil Scrum, ajustándola a las condiciones particulares del trabajo de grado. Aunque no se implementó una distribución estricta de roles como Scrum Master o Product Owner, sí se mantuvo la lógica de trabajo en sprints

semanales, donde se hicieron encuentros periódicos con el director de proyecto para revisar avances, establecer prioridades y redefinir tareas.

Como parte del enfoque ágil adoptado, se construyeron historias de usuario a partir de entrevistas con estudiantes de medicina y de lo planteado en el anteproyecto, con el fin de identificar los requerimientos funcionales desde la perspectiva de los usuarios potenciales del sistema. Estas historias permitieron comprender las necesidades concretas de los distintos actores involucrados, tales como médicos recién graduados, padres de familia o personas interesadas en conocer la oferta educativa en especializaciones médicas.

Cada historia fue construida siguiendo el formato estándar de Scrum:

Como [tipo de usuario], quiero [necesidad o acción], para [objetivo o beneficio].

Adicionalmente, a cada historia se le asignaron dos métricas clave:

- **Valor:** estimación del beneficio que representa para el usuario y el proyecto. Se asignó de manera consensuada entre los autores del proyecto, basándose en el impacto esperado en la experiencia del usuario final.

- **Estimación:** medida de la complejidad y el esfuerzo necesario para implementarla. Se utilizó una escala de puntos de historia (story points), adoptando un enfoque similar al Planning Poker [14], donde se valoran las tareas con base en el esfuerzo relativo, considerando factores como la dificultad técnica, la incertidumbre y el tiempo requerido.

Este enfoque permite priorizar el desarrollo de funcionalidades de acuerdo con su valor para el usuario y la viabilidad dentro de los tiempos del proyecto.

A continuación, se presenta el conjunto de historias de usuario recopiladas:

1	Mostrar mínimo 4 universidades	
Como usuario final, quiero que el prototipo muestre al menos 4 universidades con sus programas para tener un catálogo mínimo y lograr comparar opciones.		
Estimación: 3	**Valor:** 80	**Dependencias:**
Condiciones de Satisfacción:		
* Al cargar la vista principal, aparecen las tarjetas con el nombre de los programas que ofrecen las 4 diferentes universidades.		
* Si hay menos de 4 universidades, se muestra mensaje “No hay suficientes datos”.		

Figura 3.7: HU 01 - Mostrar mínimo 4 universidades

2	Búsqueda rápida desde BD	
Como usuario final, quiero que la búsqueda devuelva resultados rápidos sin ejecutar los scrapers en vivo para no esperar cada vez que consulto datos.		
Estimación: 5	**Valor:** 90	**Dependencias:** 1
Condiciones de Satisfacción:		
* Al hacer una búsqueda, el sistema consulta sólo la base de datos y responde de forma agíl.		
* No se dispara ningún proceso de scraping en tiempo real.		
*Si la BD está vacía, devuelve mensaje "Datos no disponibles"; no se intenta scraping.		

Figura 3.8: HU 02 - Búsqueda rápida desde BD

3	Scrapers programados	
Como desarrollador del backend, quiero que los scrapers se ejecuten de forma programada (p. ej. a las 2 am) para mantener la información actualizada sin afectar el rendimiento en búsqueda.		
Estimación: 5	**Valor:** 70	**Dependencias:** 2
Condiciones de Satisfacción:		
* Existe tarea programada que lanza scrapers a la hora configurada.		
* Tras cada ejecución, los datos de la BD se actualizan correctamente y se pueden consultar.		
* En caso de fallo, se registra un log de error.		

Figura 3.9: HU 03 - Scrapers programados

4	Extracción de datos completos	
Como usuario, quiero que el sistema extraiga los campos título, costos, fechas clave, código SNIES, contacto y enlace web de cada especialización para tener información homogénea.		
Estimación: 8	**Valor:** 60	**Dependencias:**
Condiciones de Satisfacción:		
* El scraper recorre cada universidad configurada y extrae los campos definidos.		
* Si un campo no está presente en una página, deja el valor vacío.		

Figura 3.10: HU 04 - Extracción de datos completos

5	Formato uniforme de datos	
Como usuario, quiero que la información (precios, fechas) se muestre siempre en un formato consistente para facilitar la comparación y evitar errores de interpretación.		
Estimación: 3	**Valor:** 70	**Dependencias:** 4
Condiciones de Satisfacción:		
* Todos los precios aparecen en formato "$ XXX.XXX COP".		
* Todas las fechas usan el estándar ISO YYYY-MM-DD.		

Figura 3.11: HU 05 - Formato uniforme de datos

6	Búsqueda por nombre de especialización	
Como médico interesado en especializarme, quiero buscar programas por nombre de especialización para conocer las universidades que ofrecen esa especialidad.		
Estimación: 5	**Valor:** 100	**Dependencias:**
Condiciones de Satisfacción:		
* Al introducir el texto de la especialidad, se muestran todas las coincidencias con información actualizada		
* Cada resultado indica nombre de programa y nombre de universidad.		
* Si no hay resultados, aparece un mensaje “No se encontraron coincidencias”		

Figura 3.12: HU 06 - Búsqueda por nombre de especialización

7	Filtrado por universidad	
Como médico recién graduado, quiero filtrar los resultados por universidad para ver únicamente las especializaciones que ofrece esa institución.		
Estimación: 3	**Valor:** 80	**Dependencias:** 06
Condiciones de Satisfacción:		
* Se muestra una lista desplegable de las universidades.		
* Al seleccionar una universidad, sólo aparecen los programas de esa universidad.		

Figura 3.13: HU 07 - Filtrado por universidad

8	Búsqueda por nombre de especialización	
Como usuario interesado, quiero combinar filtros por programa y universidad para acotar los resultados a mis preferencias específicas.		
Estimación: 8	**Valor:** 60	**Dependencias:** 06, 07
Condiciones de Satisfacción:		
* Se pueden aplicar simultáneamente texto libre y selección de universidad.		
* Los resultados reflejan ambas condiciones (AND).		
* La interfaz muestra claramente los filtros activos.		

Figura 3.14: HU 08 - Búsqueda por nombre de especialización

9	Vista de detalles de programa	
Como persona que desea informarse, quiero ver información relevante de cada programa para comparar y tomar una decisión informada.		
Estimación: 5	**Valor:** 70	**Dependencias:** 05, 07
Condiciones de Satisfacción:		
* Al hacer clic en un programa, se muestran los datos como nombre, costo, SNIES, fechas clave,etc.		
* Para regresar a la pagina anterior, se devuelve con un botón “Volver”.		
* Los datos muestran un formato consistente (moneda, fechas).		

Figura 3.15: HU 09 - Vista de detalles de programa

10	Consulta de costos	
Como persona que apoyará económicamente al futuro estudiante, quiero ver claramente los costos de inscripción y matrícula para planificar el presupuesto		
Estimación: 5	**Valor:** 50	**Dependencias:** 09
Condiciones de Satisfacción:		
* En la vista inicial se debe visualizar los costos de inscripción y matrícula.		
* Se debe muestra en el mismo formato y moneda.		

Figura 3.16: HU 10 - Consulta de costos

11	Interfaz intuitiva de búsqueda	
Como usuario del sistema, quiero que el buscador sea sencillo e intuitivo para encontrar programas sin conocimientos técnicos.		
Estimación: 3	**Valor:** 90	**Dependencias:**
Condiciones de Satisfacción:		
* La interfaz muestra placeholder claro ("Buscar especialidad...").		
* El buscador responde con agilidad a las busquedas.		

Figura 3.17: HU 11 - Interfaz intuitiva de búsqueda

12	Resultados ordenados y legibles	
Como usuario del sistema, quiero que los resultados estén ordenados y sean visualmente claros para facilitar la lectura y comparación		
Estimación: 3	**Valor:** 40	**Dependencias:** 06
Condiciones de Satisfacción:		
* Los resultados se muestran en tarjetas con letra clara y de forma organizada.		
* Cada resultado indica nombre del programa, nombre de la universidad, costo de inscripción, costo de la matricula, y fecha de cierre de inscripciones.		

Figura 3.18: HU 12 - Resultados ordenados y legibles

Estas Historias de Usuario serán realizadas en versiones futuras del sistema:

13	Guardar búsquedas favoritas (Versión futura)	
Como usuario registrado, quiero guardar búsquedas favoritas para revisarlas posteriormente sin repetir filtros.		
Estimación: 5	**Valor:** 5	**Dependencias:**
Condiciones de Satisfacción:		
*Existe botón "Guardar en favoritos".		
* Desde el perfil del usuario registrado se puedo ver todos lo programas que él ha guardado como favoritos.		

Figura 3.19: HU 13 - Guardar búsquedas favoritas (Versión futura)

14	Alertas de cierre de fecha de inscripción (Versión futura)	
Como usuario registrado, quiero recibir alertas de inscripciones para no perder fechas importantes.		
Estimación: 3	**Valor:** 5	**Dependencias:** 13
Condiciones de Satisfacción:		
* Al guardar un programa como favorito, se agenda notificación por correo 7 días antes de cierre de inscripciones.		
* El sistema envía un mail con nombre del programa, fecha límite y enlace.		

Figura 3.20: HU 14 - Alertas de cierre de fecha de inscripción (Versión futura)

15	Registro para usuarios frecuentes (Versión futura)	
Como usuario frecuente, quiero registrarme e iniciar sesión en la plataforma para guardar mis programas favoritos y recibir notificaciones de cierre de inscripciones de esos programas.		
Estimación: 5	**Valor:** 5	**Dependencias:** 13, 14
Condiciones de Satisfacción:		
* Existe formulario de registro donde el usuario crea cuenta con correo y contraseña.		
* Tras registrarse, el usuario inicia sesión y accede a su perfil.		
* Desde su perfil puede marcar programas como “favoritos”.		
* Los favoritos aparecen en la sección “Mis programas”.		
* El sistema envía automáticamente notificaciones antes del cierre de inscripción de cada favorito.		

Figura 3.21: HU 15 - Registro para usuarios frecuentes (Versión futura)

Con base en estas historias, se diseñó un diagrama de casos de uso que resume las interacciones esperadas entre los distintos usuarios y el sistema. Este diagrama permite visualizar de forma estructurada las funcionalidades principales a implementar y cómo estas se relacionan con los actores definidos.

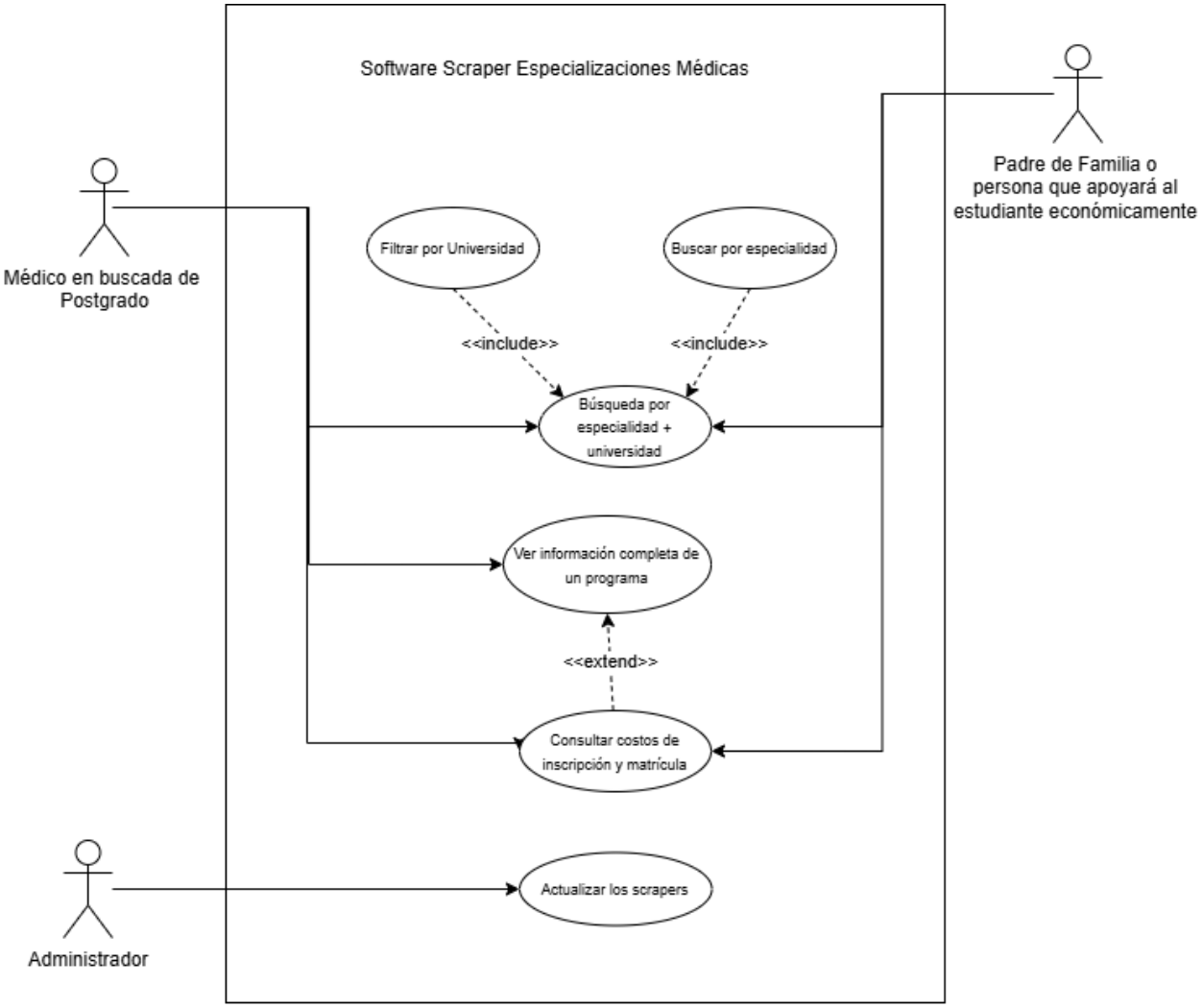


Figura 3.22: Diagrama de Casos de Uso

Capítulo 4

Extracción, estandarización y almacenamiento de información

4.1. Investigación, pruebas y selección de la herramienta y lenguaje para la extracción

Como parte del proceso de investigación del proyecto, se llevó a cabo una revisión comparativa de diversas herramientas de extracción automática de información (web scraping), con el objetivo de identificar las más adecuadas para interactuar con las páginas web de las universidades seleccionadas. Esta revisión consideró herramientas desarrolladas en los dos lenguajes de programación más utilizados en esta área: Python y JavaScript.

En el caso de Python, se identificaron tres herramientas principales:

- **Beautiful Soup** es una librería de Python para analizar HTML/XML con una API intuitiva basada en DOM. Según Zheng [15], reporta una tasa de precisión superior al 95 % en tareas de extracción, ideal para prototipos y páginas estáticas, aunque muestra limitaciones al manejar contenido dinámico generado por JavaScript o AJAX, lo cual puede dificultar su uso en algunas de las universidades objetivo del proyecto.

- **Scrapy** es un framework de scraping en Python basado en Twisted que sobresale en la extracción concurrente y eficiente de grandes volúmenes de datos. Según Dikilitaş [16], Scrapy ofrece un excelente rendimiento en tiempo de ejecución y uso de memoria comparado con herramientas como BeautifulSoup. Su arquitectura modular (spiders, pipelines, middlewares) permite estructurar proyectos robustos, aunque requiere dominio de conceptos asincrónicos, lo que implica una curva de aprendizaje más elevada.

- **Selenium** utiliza WebDriver para automatizar navegadores reales, lo que le permite interactuar con páginas web dinámicas y ejecutar JavaScript, un punto importante para este proyecto. Aunque más lento y consumidor de recursos que las bibliotecas "headless", estudios como el de ScrapingBee [17] indican que su rendimiento puede optimizarse mediante modo headless y combinaciones con técnicas como espera explícita; además, un estudio muestra que puede reducir hasta un 70 % el procesamiento de datos dinámicos versus Beautiful Soup [18].

Además de las herramientas disponibles en Python, también se exploraron alternativas desarrolladas en JavaScript, dado que este lenguaje ofrece ventajas específicas para la manipulación directa del DOM y la ejecución de JavaScript en páginas web dinámicas. Se evaluaron las siguientes herramientas:

- **Puppeteer** es una biblioteca de Node.js creada por el equipo de Chrome DevTools (2017), que permite controlar navegadores Chromium desde JavaScript. Es ideal para páginas dinámicas: permite navegar, hacer clic, cargar contenidos con AJAX y generar PDFs o capturas. Según Proxyway [19], Puppeteer es más rápido que Selenium en Chrome, pero requiere más recursos que herramientas de parsing estático como Cheerio [20].

- **Playwright**, desarrollada por Microsoft en 2020, ofrece una API similar a Puppeteer pero compatible con múltiples navegadores (Chromium, Firefox y WebKit). También permite controlar navegadores desde Python y C#. Según Oxylabs [21], destaca por su auto-waiting y soporte multi-navegador, lo que facilita scraping en páginas complejas. A pesar de ser una herramienta más reciente, es robusta, confiable y bien documentada.

- **Cheerio** es un parser HTML/XML para Node.js, inspirado en jQuery. Extrae datos de páginas estáticas de forma rápida y eficiente, sin ejecutar JavaScript, lo que lo hace muy ligero. Como indica ScraperAPI [22], Cheerio es "lightning fast" y fácil de usar para scraping de contenido estático, pero no funciona para páginas dinámicas.

Tras el análisis comparativo de las herramientas disponibles, se decidió realizar pruebas exploratorias con Selenium en Python, por su capacidad de automatizar la interacción con contenido dinámico en navegadores reales, y con Playwright en JavaScript, debido a su eficiencia en la gestión de múltiples navegadores y su compatibilidad con tecnologías actuales de frontend.

Una vez seleccionadas las herramientas y lenguajes de programación, se dio inicio al desarrollo de los scripts, comenzando con implementaciones básicas orientadas a extraer los datos de una única especialización por universidad. Esta fase inicial permitió comprender la estructura y comportamiento de cada página web de las universidades. Posteriormente, los scripts fueron refinados y automatizados para recorrer de forma sistemática todas las especializaciones médicas disponibles en cada sitio web, extrayendo información clave de manera estructurada.

A continuación, se presentan dos fragmentos de código correspondientes a las primeras pruebas realizadas con ambas herramientas. La Figura 4.1 muestra un ejemplo de script desarrollado en JavaScript con Playwright, mientras que la Figura 4.2 presenta una implementación realizada con Python y Selenium. Estos scripts permitieron validar y explorar la técnica del proceso de extracción de información.

```javascript
async function extraerDatosEspecializacion(page) {
    // Esperar a que la lista esté presente
    await page.waitForSelector('ul.row li');

    // Extraer datos usando evaluación en el contexto del navegador
    const datos = await page.$$eval('ul.row li', items => {
        return items.map(item => {
            const labels = item.querySelectorAll('p');
            if (labels.length >= 2) {
                return {
                    titulo: labels[0].textContent.trim(),
                    valor: labels[1].textContent.trim()
                };
            }
            return null;
        }).filter(Boolean);
    });

    datos.forEach(dato => {
        console.log(`${dato.titulo}: ${dato.valor}`);
    });
}
```

Figura 4.1: Script inicial en Playwright (JavaScript) para extracción de especializaciones médicas en la Universidad Javeriana

Este código en JavaScript utiliza Playwright para esperar la carga de una lista de elementos HTML (ul.row li) y luego evalúa el contenido de cada ítem dentro del navegador. Para cada elemento, extrae los valores del primer y segundo párrafo (<p>), asociándolos a campos clave como título y valor. Este enfoque permitió validar la estructura interna del DOM y establecer las bases para un scraper reutilizable.

```python
def extraer_datos_especializacion(driver):

    lista = WebDriverWait(driver, 10).until(
        EC.presence_of_all_elements_located((By.XPATH, "//ul[@class='row']/li"))
    )

    # Extraer los datos de cada elemento
    for item in lista:
        labels = item.find_elements(By.TAG_NAME, "p")
        if len(labels) ⩾ 2:
            titulo = labels[0].text.strip()
            valor = labels[1].text.strip()
            print(f"{titulo}: {valor}")
```

Figura 4.2: Script inicial en Selenium (Python) para extracción de datos en la Universidad Javeriana

Este script es equivalente al mostrado en la Figura 4.1, con el objetivo de reflejar una lógica similar a la de Playwright, pero en el ecosistema Python. Esta implementación emplea WebDriverWait para garantizar que los elementos estén presentes antes de continuar. Luego, se recorren los nodos li, se extraen los elementos <p>asociados y se imprime la información recuperada. Adicionalmente, esta prueba permitió establecer la compatibilidad del enfoque con entornos no basados en JavaScript.

Por otra parte, durante estas primeras pruebas con Playwright, se planteó como objetivo unificar el proceso de extracción en un único script capaz de obtener información desde las páginas web de dos universidades distintas, utilizando los mismos parámetros de entrada. No obstante, esta estrategia resultó inviable debido a las notables diferencias en la estructura y organización del contenido entre los sitios web analizados. Como se evidenció en etapas anteriores, algunas universidades presentan la información en listas estructuradas, mientras que otras utilizan elementos sin identificadores consistentes, dificultando la generalización de una única lógica de scraping.

Ante esta limitación, se optó por diseñar un archivo de configuración en formato JSON que almacena los parámetros específicos requeridos por cada universidad. Esta solución permitió proporcionar al sistema una mayor flexibilidad, ya que el código puede determinar qué método de extracción aplicar según la universidad. Asimismo, esta estrategia aportó a la escalabilidad del proceso, facilitando la incorporación futura de nuevas instituciones sin necesidad de modificar el núcleo del script.

En cuanto a la herramienta seleccionada para el desarrollo final, si bien tanto Playwright como Selenium demostraron capacidades técnicas suficientes para cumplir con los requisitos de extracción, se decidió continuar con Selenium en Python. Esta elección respondió principalmente a criterios de

facilidad de implementación, mantenibilidad del código y experiencia previa del equipo. Python, al ser un lenguaje ampliamente adoptado en el área del scraping, ofrece una sintaxis sencilla y una amplia documentación. Estos factores facilitaron la integración del scraper con otros componentes del sistema y permitieron un desarrollo más eficiente del proyecto.

4.2. Desarrollo y modularización de los scrapers

A partir de lo encontrado en las pruebas iniciales, se concluyó que no era viable implementar un único script genérico que funcionara para todas las universidades, dado que cada sitio web presentaba diferencias estructurales significativas tanto en su organización visual como en la distribución de la información. Algunas instituciones concentran los datos en una sola página con listas claramente definidas, mientras que otras los fragmentan en múltiples secciones enlazadas o los encapsulan en elementos sin identificadores estándar, lo cual dificulta aplicar una lógica de extracción uniforme.

Como respuesta a esta diversidad, se diseñó una arquitectura modular basada en un archivo de configuración (*config.json*) que almacena los parámetros técnicos requeridos por cada universidad. Esta solución permitió establecer una capa de abstracción que facilita la selección dinámica de los métodos de navegación y scraping más apropiados en cada caso, al tiempo que favorece la escalabilidad del sistema para incorporar nuevas instituciones sin necesidad de modificar la lógica central del código.

A continuación, se muestra un fragmento del archivo config.json correspondiente a la Universidad del Valle:

```json
{
    "univalle": {
        "url": "https://salud.univalle.edu.co/posgrados/especializaciones-clinicas",
        "url_info_inscripcion": "https://admisiones.univalle.edu.co/new/index.php",
        "info_inscripcion_selector": "div.tipo_programa",
        "info_inscripcion_item_selector": "div.opcion_programa.estudiosFormales.posgrado",
        "card_content_selector": "div.contenido_tarjetas",
        "info_card_selector": "div.card.mb-2",
        "inscripcion_content_selector": "div#accordion",
        "calendario_content_selector": "div#collapseCalendario",
        "header_xpath": "//h2[contains(., 'Escuela de Medicina')]",
        "list_xpath": "./following-sibling::table//ul",
        "item_selector": "li a.h4",
        "details_selector": "table.tabla-info tr",
        "contact_selector": "table[style*='height'] p",
        "specific_contact_selector": "table[style*='height'] p[style*='padding']"
    },
```

Figura 4.3: Fragmento del archivo de configuración config.json para la Universidad del Valle

Este fragmento del archivo de configuración define los parámetros específicos necesarios para automatizar la extracción de datos del sitio web de la Universidad del Valle. Incluye las URLs de

acceso y los selectores CSS que permiten identificar secciones clave como tarjetas de programas, detalles específicos o información de contacto. Esta parametrización posibilita que el sistema adapte su comportamiento dinámicamente a la estructura de cada institución.

De esta manera, el sistema no solo permite manejar las características de cada universidad de forma separada, sino que también facilita la incorporación de nuevas universidades. Basta con añadir en el archivo *config.json* los parámetros de configuración correspondientes a la nueva institución y, posteriormente, incluir en el código principal el llamado a la función de scraping correspondiente.

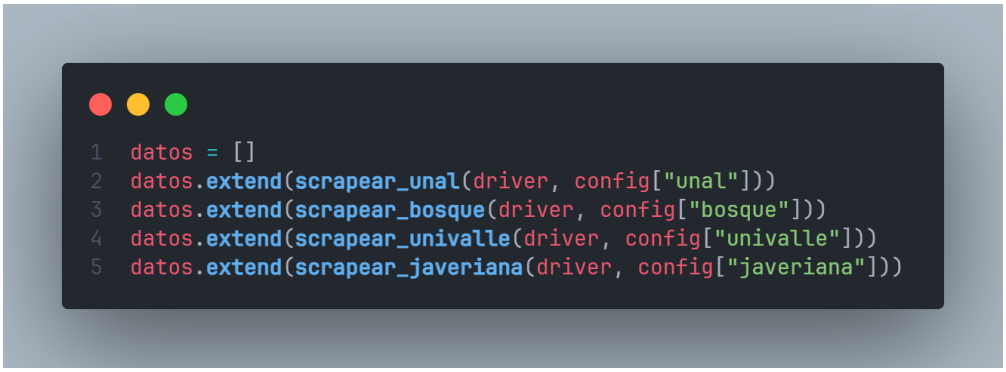


Figura 4.4: Fragmento del código principal donde se realizan las llamadas a los scrapers de cada universidad

En la Figura 4.4 se observa un fragmento del código donde se realizan las llamadas a los distintos scrapers. En este mismo bloque es donde se añadiría el llamado al scraper de cualquier universidad adicional, garantizando así la extensibilidad y escalabilidad del sistema.

A partir de la estructura definida en el archivo de configuración, se desarrollaron scrapers independientes para cada universidad, organizados en módulos separados con funciones adaptadas a las particularidades de cada sitio web. El proceso completo es coordinado desde el archivo principal del sistema, *main.py*, el cual se encarga de cargar la configuración correspondiente, invocar el scraper adecuado según la universidad, aplicar un mapeo estandarizado a los campos extraídos para normalizarlos y almacenar los resultados en archivos CSV o insertarlos en la base de datos. Este enfoque centralizado garantiza la coherencia del flujo de ejecución, y facilita la validación, trazabilidad y mantenimiento del sistema.

Cada scraper individual fue diseñado para encargarse de las necesidades de la página web correspondiente, incluyendo tareas como la navegación entre pestañas, la extracción condicional de datos visibles y la gestión de múltiples ventanas del navegador cuando es necesario. Esta arquitectura modular proporciona al sistema una alta flexibilidad, facilitando la incorporación de nuevas universidades o la modificación de configuraciones sin afectar el funcionamiento general.

Durante el desarrollo de los scrapers se presentaron diversos desafíos, especialmente relacionados con cambios inesperados en las páginas web institucionales. En algunos casos, las universidades modificaron la estructura de sus sitios o incorporaron nuevos campos de información, lo cual exigió ajustes continuos en la lógica de extracción implementada. En otros casos, las actualizaciones intro-

dujeron fallos en funcionalidades clave (como los filtros para buscar exclusivamente especializaciones médicas), lo que derivó en la recopilación no deseada de programas de otras áreas académicas.

Gracias a la organización modular del sistema, estos ajustes pudieron implementarse de manera específica, sin comprometer el resto de la arquitectura o funcionamiento. Esta experiencia resaltó la importancia de contar con una estructura desacoplada y mantenible, especialmente en este proyecto que dependen de sitios web de terceros y que están susceptibles a cambios sin previo aviso.

4.3. Diseño e implementación de la base de datos

Después de completar el desarrollo de los scrapers, se realizó un análisis técnico para seleccionar el sistema de gestión de bases de datos más adecuado para almacenar la información recopilada. Dado que los datos extraídos presentan una estructura homogénea entre universidades —pues cada registro corresponde a un programa de especialización médica con campos comunes como título, código SNIES, duración, costos y contactos—, se optó por implementar una base de datos relacional (SQL) en lugar de una base de datos NoSQL.

Según Camargo y Arciniegas [23], los sistemas relacionales son especialmente adecuados para manejar datos estructurados que requieren integridad referencial, consultas complejas y una normalización rigurosa. En contraste, las bases de datos NoSQL están mejor orientadas a contextos donde la estructura de los datos es altamente variable, no relacional o cuando se requiere escalabilidad horizontal extrema, propiedades que no aportan ventajas significativas en un entorno como este, donde los datos presentan una estructura constante y se benefician de un modelo relacional con claves foráneas y normalización.

Dentro del ecosistema de bases de datos relacionales se evaluaron tres opciones principales: **PostgreSQL**, **MySQL** y **SQLite**, cada una con características particulares en cuanto a rendimiento, escalabilidad y extensibilidad.

- **PostgreSQL** se destaca por su soporte a tipos de datos complejos, funciones definidas por el usuario, integridad referencial avanzada y capacidad para ejecutar consultas SQL complejas con alto rendimiento [24].
- **MySQL** es una alternativa ampliamente utilizada, reconocida por su velocidad y eficiencia en operaciones básicas, aunque con menor soporte para funcionalidades avanzadas [25].
- **SQLite** es una solución ligera y embebida, ideal para prototipos o aplicaciones locales, pero limitada en escenarios que demandan concurrencia elevada o consultas complejas [26].

Teniendo en cuenta las características del proyecto que involucra múltiples entidades relacionadas, además de requerir mantener relaciones consistentes entre tablas (integridad referencial) y realizar consultas cruzadas entre universidades, programas y procesos de admisión, se seleccionó PostgreSQL como sistema gestor. Esta elección ofrece un equilibrio adecuado entre robustez, flexibilidad y escalabilidad, además de contar con amplia documentación y compatibilidad con el entorno tecnológico usado en el sistema.

Una vez seleccionada la base de datos PostgreSQL como el sistema de almacenamiento, se procedió al diseño del modelo relacional, con el fin de estructurar de forma lógica y eficiente la información extraída de los sitios web de las universidades. La figura 4.5 ilustra dicho modelo, el cual está conformado por cuatro entidades principales: *Universidad, Programa_ Academico, Admision* y *Contacto*.

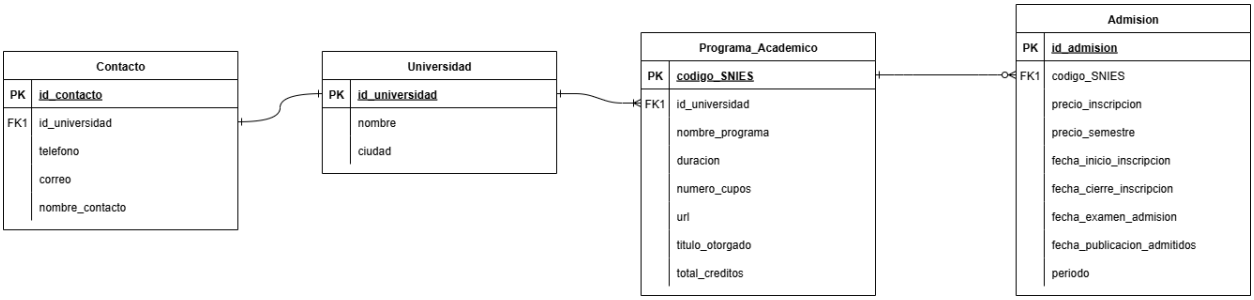


Figura 4.5: Diagrama Modelo Relacional

La entidad Universidad almacena la información básica de cada institución, como su nombre y ciudad. Cada universidad puede tener múltiples programas académicos asociados, los cuales se representan en la entidad *Programa_ Academico*, que contiene atributos clave como el nombre del programa, duración, número de cupos, perfil del aspirante y egresado, nombre del director y periodicidad de admisiones. Esta entidad está relacionada con Universidad mediante una clave foránea (*id_ universidad*), ya que cada programa pertenece a una única universidad.

Por otro lado, la entidad *Admision* se diseñó para almacenar la información asociada a los procesos de ingreso, como fechas de inscripción, precio de matrícula, exámenes, publicación de admitidos y el paso a paso para realizar la inscripción. Está relacionada con *Programa_ Academico* a través de la clave foránea *id_ programa*, lo que permite registrar múltiples períodos de admisión por programa, en caso de ser necesario.

Finalmente, la entidad Contacto registra los números telefónicos disponibles para consultas o atención para los aspirantes, enlazados a cada universidad mediante una clave foránea (*id_ universidad*), permitiendo almacenar múltiples contactos por institución.

Una vez definido el modelo relacional, se implementó la lógica de integración entre los scrapers desarrollados y la base de datos PostgreSQL. Para este propósito, se diseñó una función encargada de actualizar o agregar registros a las distintas tablas del modelo, según los datos extraídos en tiempo real desde los sitios web de las universidades.

Esta decisión se basó en el hecho de que los parámetros e información capturada para cada entidad (como programas académicos, procesos de admisión y contactos) no suelen variar estructuralmente entre ejecuciones, sino que pueden actualizarse (por ejemplo, nuevas fechas de admisión o cambios en el precio de matrícula).

Adicionalmente, se decidió que la actualización de datos se realice de forma planificada cada cierto tiempo, en lugar de hacer scraping en tiempo real cada vez que el usuario realice una búsqueda.

Esta estrategia permite reducir el tiempo de respuesta de la aplicación, mejorar la experiencia de usuario y disminuir la carga sobre los sitios web fuente, sin sacrificar la vigencia de la información. Este enfoque garantiza que la base de datos mantenga la información actualizada sin duplicados, y facilita la sincronización periódica de los datos sin necesidad de vaciar o reconstruir la base en cada ejecución del scraper.

Capítulo 5

Modelado del sistema mediante C4: Niveles 1 a 3

5.1. Nivel 1 - Diagrama de Contexto

El modelo C4 es una técnica para documentar arquitecturas de software mediante una jerarquía de vistas que representan el sistema en diferentes niveles de abstracción: contexto, contenedores, componentes y código. Este enfoque facilita la comprensión de cómo interactúan los usuarios, sistemas externos y los distintos componentes de software, permitiendo una comunicación efectiva entre perfiles técnicos y no técnicos [27].

El diagrama de nivel 1, o diagrama de contexto, proporciona una visión general del sistema desde una perspectiva externa. Muestra cómo los actores (usuarios o sistemas) interactúan con el sistema central, y qué otros sistemas están involucrados en el ecosistema de información.

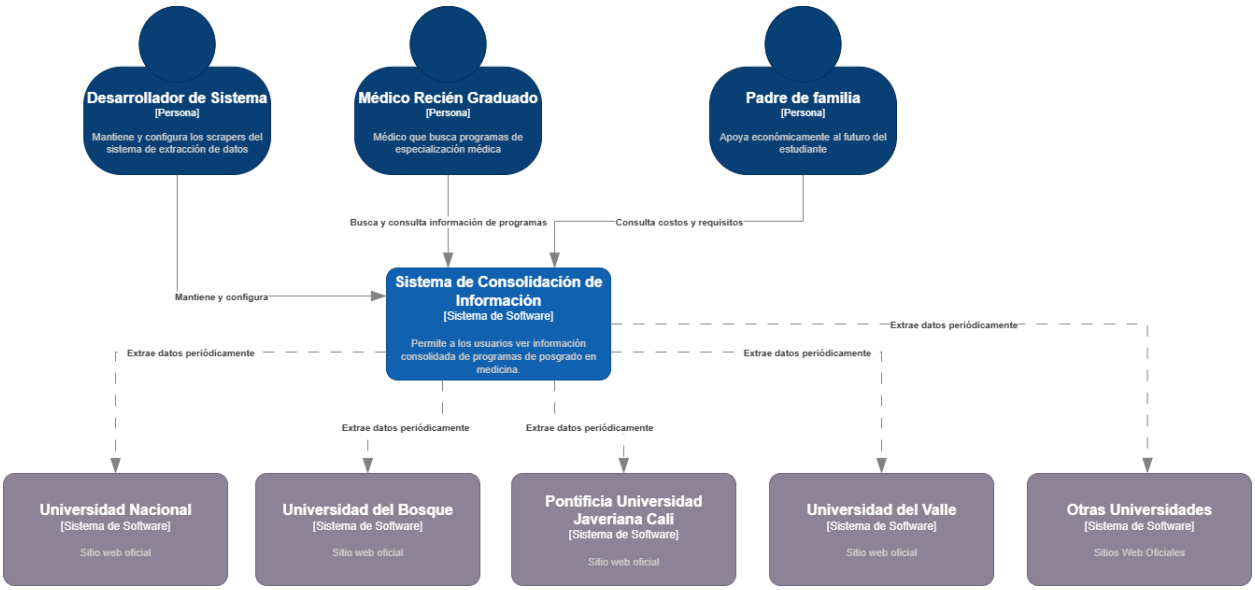


Figura 5.1: Modelo C4 - Nivel 1

En el diagrama de contexto (Figura 5.1), representa al sistema central encargado de consolidar la información de especializaciones médicas. Este sistema interactúa con tres actores principales:

el médico recién graduado, quien busca programas de posgrado; el padre de familia, que consulta costos y requisitos; y el desarrollador, encargado del mantenimiento y configuración de los scrapers. Además, se visualiza la relación del sistema con los portales web de distintas universidades colombianas, desde donde se extraen periódicamente los datos académicos relevantes.

5.2. Nivel 2 - Diagrama de Contenedores

A diferencia del diagrama de contexto, el diagrama de contenedores profundiza en la arquitectura interna del sistema de consolidación de información, mostrando los principales bloques de software que lo componen y cómo se comunican entre sí. Cada contenedor representa una unidad funcional que ejecuta un rol específico dentro del sistema, como la interfaz de usuario, el procesamiento de datos o el almacenamiento [27].

En este caso, se identifican tres contenedores principales: la aplicación web desarrollada con Django, los scrapers implementados con Python y Selenium, y la base de datos relacional PostgreSQL. Estos elementos permiten organizar el flujo completo de la información, desde su extracción hasta su presentación al usuario final.

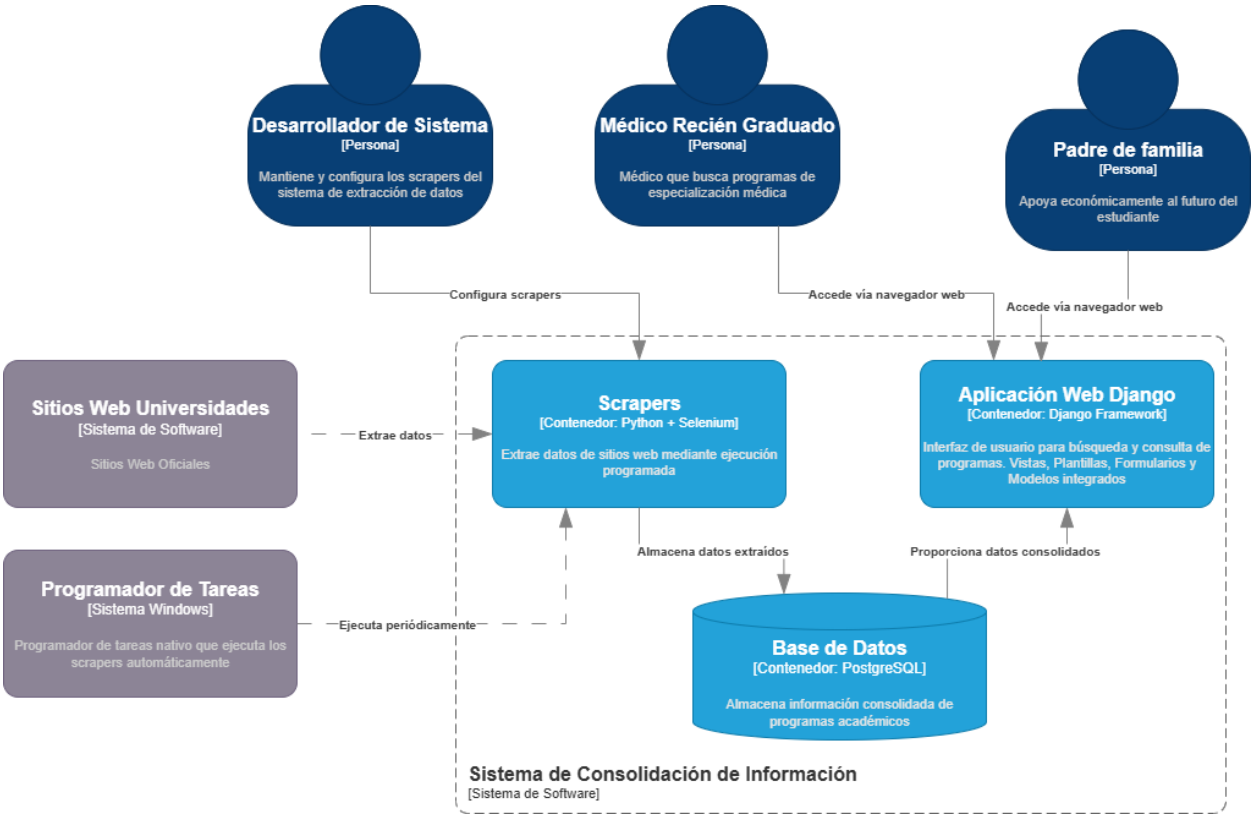


Figura 5.2: Modelo C4 - Nivel 2

El diagrama de la Figura 5.2 representa cómo se organiza internamente el sistema a nivel de contenedores. En primer lugar, los scrapers, desarrollados con Python y Selenium, son responsables de extraer automáticamente los datos desde los sitios web oficiales de las universidades. Esta ejecución se programa y automatiza mediante el uso del Programador de Tareas de Windows, que permite establecer una frecuencia de actualización periódica sin intervención manual.

Los datos obtenidos por los scrapers se almacenan en una base de datos PostgreSQL, la cual actúa como repositorio central de información consolidada. Esta base de datos contiene campos estructurados que permiten almacenar aspectos clave de cada programa académico, como costos, fechas, duración, entre otros.

Por otro lado, la aplicación web Django cumple el rol de interfaz de usuario. Este contenedor permite a los usuarios (médicos recién graduados y padres de familia) consultar la información desde un navegador web. La aplicación está compuesta por vistas, formularios y plantillas que acceden a los datos de la base y los presentan de forma organizada y filtrable según criterios de búsqueda.

Finalmente, el desarrollador de sistema desempeña un papel clave en la configuración y mantenimiento de los scrapers, asegurando que las extracciones continúen funcionando incluso si las estructuras de las páginas web cambian. De esta manera, se garantiza la operatividad del flujo completo de datos dentro del sistema.

5.3. Nivel 3 - Diagrama de Componentes

El diagrama de nivel 3 del modelo C4 permite descomponer un contenedor en sus componentes internos [27]. En este nivel, se detalla la arquitectura de la aplicación web construida con el framework Django, resaltando los principales módulos funcionales que la componen, así como sus interacciones con la base de datos y el frontend.

Este nivel facilita la comprensión de la lógica interna del sistema, mostrando cómo se estructuran las funcionalidades de búsqueda y consulta, y cómo se integran los modelos, formularios, vistas y plantillas del lado del servidor. A través del ORM (Object-Relational Mapping) de Django, estos componentes se conectan con la base de datos PostgreSQL para entregar información consolidada al usuario final.

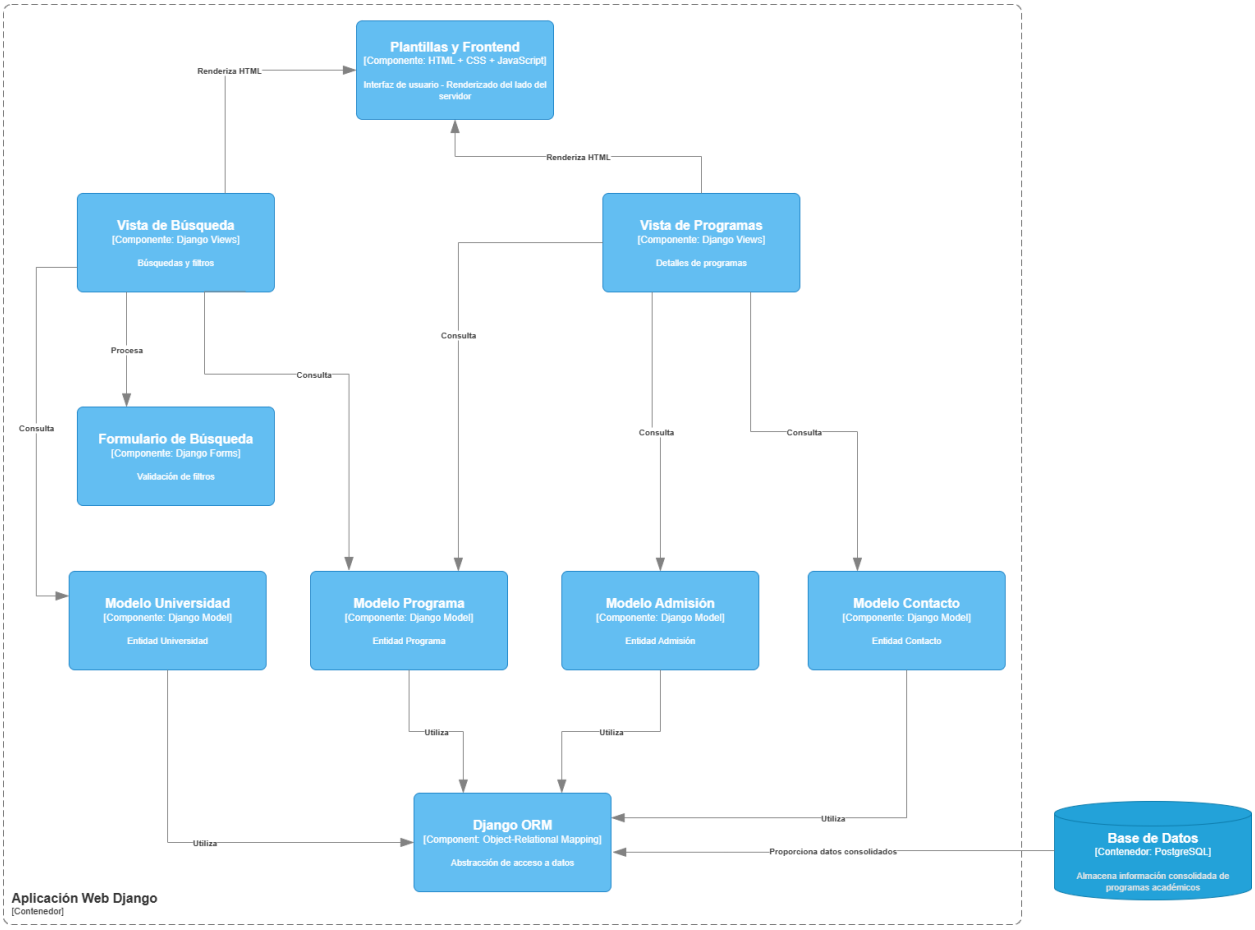


Figura 5.3: Modelo C4 - Nivel 3

En la Figura 5.3 se visualiza la estructura interna de la aplicación web Django. El sistema se organiza en dos flujos principales: la búsqueda de programas y la visualización de detalles.

La Vista de Búsqueda permite aplicar filtros a través del Formulario de Búsqueda, que valida la entrada del usuario. Este formulario interactúa con los Modelos Universidad y Programa, los cuales representan las entidades principales del sistema y permiten obtener resultados relevantes.

Por otro lado, la Vista de Programas se encarga de mostrar información detallada de cada especialización médica. Esta vista consulta directamente los modelos relacionados con Admisión y Contacto, accediendo a los campos necesarios para presentar datos como fechas, requisitos, costos y medios de contacto.

Ambas vistas se conectan con las Plantillas y Frontend, que son responsables de renderizar las páginas HTML usando los datos obtenidos. Este renderizado se realiza del lado del servidor.

Toda la interacción con la base de datos se canaliza mediante el Django ORM, que permite realizar consultas, inserciones y actualizaciones en la base de datos PostgreSQL sin escribir sentencias

SQL directamente. Esta capa de abstracción facilita el mantenimiento y la escalabilidad del sistema.

La representación del sistema mediante el modelo C4 permitió documentar de manera progresiva su arquitectura, desde una vista general hasta un nivel detallado. A través del Nivel 1, se identificaron los actores principales y su relación con el sistema, evidenciando su propósito como puente entre usuarios y los sitios web institucionales. El Nivel 2 expuso la organización interna del sistema en contenedores funcionales, destacando el flujo de datos desde los sitios web de las universidades hacia una base consolidada, mediante el uso de scrapers y una interfaz web desarrollada en Django. Finalmente, el Nivel 3 profundizó en la aplicación web, descomponiendo sus funcionalidades clave y mostrando cómo los distintos componentes (vistas, modelos, formularios y plantillas) contribuyen a ofrecer una experiencia coherente y funcional al usuario final.

Capítulo 6

Diseño e implementación de la interfaz de usuario para la consulta de programas académicos

6.1. Selección del framework web

El desarrollo del componente web del sistema requería la elección de un framework que facilitara la creación de una aplicación escalable y de fácil mantenimiento. Para ello, se realizó una revisión comparativa de distintos frameworks en Python, priorizando aquellos que tuvieran buen soporte para la construcción de APIs REST, integración con bases de datos y documentación clara. Entre las opciones evaluadas se encontraron Django, Flask y FastAPI, tres herramientas reconocidas por su rendimiento, modularidad y capacidad de adaptación a distintos contextos de desarrollo web.

- **Flask** es un microframework que proporciona una estructura flexible para el desarrollo de aplicaciones. Destaca por su simplicidad y bajo nivel de acoplamiento, lo que permite al desarrollador tener mayor control sobre la arquitectura del sistema. Sin embargo, este enfoque también implica que funcionalidades comunes como autenticación, paneles administrativos o validación de formularios deben implementarse manualmente o mediante bibliotecas externas, lo cual puede extender los tiempos de desarrollo inicial [28].

- **FastAPI** es un framework moderno diseñado para el desarrollo rápido de APIs con alto rendimiento. Utiliza anotaciones de tipo (type hints) y validación automática mediante Pydantic, lo cual lo convierte en una opción atractiva para arquitecturas orientadas a microservicios. A pesar de sus ventajas en cuanto a velocidad y documentación automática, también requiere una mayor configuración al integrar herramientas administrativas o manejar formularios complejos [29].

- **Django** es un framework de alto nivel que promueve el desarrollo rápido y limpio mediante una estructura integral que incorpora múltiples herramientas listas para usar, como un sistema de autenticación, ORM, panel de administración y gestión de formularios. Al contar con estas funcionalidades básicas necesarias para construir una aplicación web, se logra agilizar el proceso de desarrollo desde las primeras etapas [30].

Dado que los tres frameworks evaluados cumplían con los requerimientos técnicos del proyecto, la decisión final se inclinó por Django, debido a la familiaridad previa del equipo con el entorno,

lo cual permitió reducir la curva de aprendizaje y acelerar el proceso de implementación. Además, su enfoque monolítico y estructurado favorece el mantenimiento y escalabilidad de la aplicación, aspectos clave para un sistema académico con posibilidad de crecimiento.

6.2. Mockups

Como parte del proceso de diseño de la interfaz de usuario, se elaboraron mockups que permitieron anticipar la organización visual y funcional del sistema antes de su desarrollo. Estos diseños se enfocan en garantizar una navegación intuitiva para los usuarios principales: médicos en búsqueda de programas de especialización y personas encargadas de su apoyo económico. A través de estos prototipos se definieron las secciones clave, el flujo de consulta y los elementos visuales que facilitan el acceso a la información consolidada sobre especializaciones médicas. A continuación, se presentan los mockups desarrollados para las dos vistas principales del sistema.

Figura 6.1: Mockup - Página para las búsquedas de programas

Esta vista representa la página principal del sistema, en la cual se listan las especializaciones médicas disponibles, cada una en una tarjeta individual. El usuario puede aplicar filtros por nombre de especialización o por universidad, facilitando la navegación. En cada tarjeta se muestra información clave como el nombre del programa, la universidad correspondiente, el código SNIES y los precios (cuando están disponibles).

Figura 6.2: Mockup - Página con la información detallada de un programa

Esta vista se activa cuando el usuario selecciona una especialización específica. Presenta información detallada del programa organizada en secciones como duración, número de cupos, periodicidad, perfiles académicos y fechas importantes. También se incluyen bloques dedicados a costos, información de contacto y un botón de acción para inscribirse. Esta estructura busca ofrecer de forma clara y jerárquica todos los datos relevantes para tomar una decisión informada.

6.3. Implementación del sistema

A partir de los mockups definidos, se procedió a la implementación de la aplicación web utilizando el framework Django. La estructura del sistema se compone de vistas, modelos, formularios y

plantillas que permiten consultar y visualizar de forma organizada la información consolidada sobre programas de especialización médica.

La interfaz desarrollada permite a los usuarios buscar especializaciones filtrando por nombre o universidad, visualizar tarjetas con los datos más relevantes de cada programa y acceder a una vista detallada que presenta información adicional, como duración, número de cupos, periodicidad, fechas de admisión y contacto.

En la implementación final se priorizó la claridad y la simplicidad en la presentación de los datos.

Así mismo, todos los datos que se muestran en la aplicación son cargados dinámicamente desde una base de datos PostgreSQL, la cual es actualizada periódicamente por los scrapers desarrollados previamente. Esto asegura que la información disponible en la plataforma esté sincronizada con los sitios web oficiales de las universidades.

Figura 6.3: Implementación de la Página para las búsquedas de programas

La Figura 6.3 muestra la vista de búsqueda del sistema, donde el usuario puede consultar programas de especialización médica a través de filtros por nombre de especialización y universidad. Esta funcionalidad fue diseñada para facilitar el acceso rápido a la información de interés, permitiendo combinar criterios en una sola búsqueda. La interfaz se alinea con el diseño propuesto en los mockups, y responde de manera dinámica a las consultas realizadas, mostrando solo los programas que coinciden con los parámetros seleccionados. Esta vista es el punto de entrada principal del sistema

y uno de los componentes clave en la experiencia de navegación del usuario.

Figura 6.4: Implementación de la Página con la información detallada de un programa

La Figura 6.4 muestra una de las vistas desarrolladas del sistema, correspondiente al detalle de un programa de especialización. Puede observarse que la implementación sigue la estructura definida en los mockups, conservando tanto la disposición de los bloques informativos como la jerarquía visual. Esta coherencia entre diseño e implementación facilitó una transición fluida desde la etapa de prototipado al desarrollo funcional, asegurando una experiencia de usuario clara, accesible y alineada con los objetivos del sistema.

Capítulo 7

Pruebas Funcionales

7.1. Plan de pruebas

Las pruebas son una etapa fundamental en el desarrollo de sistemas de software, ya que permiten garantizar su correcto funcionamiento, detectar errores tempranamente y validar que las funcionalidades implementadas respondan adecuadamente a las necesidades de los usuarios.

Para este proyecto de grado, el plan de pruebas se estructuró de la siguiente manera:

- Se realizaron pruebas unitarias sobre las funciones del módulo de *scraping*, evaluando la extracción individual por universidad, la normalización y limpieza de datos, y el almacenamiento en la base de datos.
- Se llevaron a cabo pruebas de integración para verificar la correcta interacción entre los *scrapers* y la base de datos, así como entre esta y la interfaz web, asegurando el flujo completo desde la extracción hasta la visualización.
- Se aplicaron pruebas funcionales al sistema web desarrollado con Django, enfocadas en validar las funcionalidades de búsqueda, filtrado, navegación y el manejo adecuado de situaciones como búsquedas sin resultados o filtros vacíos.
- Finalmente, se realizaron pruebas de usabilidad con usuarios reales, con el fin de evaluar la experiencia de uso y verificar que las funcionalidades implementadas respondieran a sus necesidades.

7.2. Herramientras utilizadas para las pruebas

Para cada tipo de prueba se utilizaron herramientas específicas, adaptadas a las tecnologías empleadas en el desarrollo del sistema.

7.2.1. Pytest

Es una herramienta de pruebas unitarias utilizada para verificar el correcto funcionamiento de las funciones del módulo de scraping desarrolladas en Python. Su sintaxis clara y modularidad permitieron escribir pruebas concisas, reutilizables y fácilmente ejecutables durante el desarrollo. Se diseñaron pruebas para cada función crítica, incluyendo aquellas responsables de la limpieza de campos, la validación de datos requeridos y la integración con el modelo de datos.

7.3. Implementación de tipos de pruebas

7.3.1. Pruebas unitarias

Las pruebas unitarias son el proceso mediante el cual se valida el funcionamiento de unidades pequeñas de código, como funciones o métodos individuales. Este tipo de pruebas permite detectar errores de lógica en etapas tempranas del desarrollo y facilita la refactorización del código con mayor seguridad [31].

7.3.1.1. Módulo del scraper

Para este proyecto, se implementaron pruebas unitarias sobre las funciones del módulo de scraping. Para esto se utilizó la herramienta pytest, una librería de pruebas para Python que permite escribir casos de prueba de forma clara y concisa, así como generar reportes detallados sobre su ejecución.

En total, se implementaron pruebas unitarias para todas las funciones que conforman el módulo de scraping, de manera que cada componente pudiera ser validado de forma independiente y se asegurara su correcto funcionamiento antes de la integración completa del sistema. Sin embargo, en este documento se presenta únicamente el proceso aplicado a una de ellas como ejemplo ilustrativo, con el fin de mostrar la metodología seguida.

A continuación, en la Figura 7.1 se puede observar un ejemplo de una prueba unitaria sobre el método `parsear_fecha`

```python
def test_fechas_formato_texto_espanol_completas(self):
        """Prueba fechas en formato texto español con año completo"""
        casos = [
            ("15 de marzo de 2025", date(2025, 3, 15)),
            ("3 de enero de 2024", date(2024, 1, 3)),
            ("25 de diciembre de 2024", date(2024, 12, 25)),
            ("1 de febrero de 2025", date(2025, 2, 1)),
            ("31 de mayo de 2023", date(2023, 5, 31)),
        ]

        for fecha_str, fecha_esperada in casos:
            resultado = parsear_fecha(fecha_str)
            assert (
                resultado == fecha_esperada
            ), f"Falló para '{fecha_str}': esperado {fecha_esperada}, obtuvo {resultado}"
```

Figura 7.1: Ejemplo prueba unitaria método `parsear_fecha`

Esta prueba tuvo como objetivo verificar que el método `parsear_fecha` sea capaz de interpretar correctamente fechas escritas en formato textual en español, donde se incluye el día, el mes (en letras) y el año completo. Para ello, se definió una lista de casos con entradas de texto representando fechas y sus respectivas salidas esperadas en formato *date* de Python.

Durante la ejecución de la prueba, cada cadena fue procesada por `parsear_fecha`, y el resultado obtenido se comparó contra la fecha esperada mediante una aserción. En caso de alguna falla, se generaba un mensaje indicando cuál fue la entrada que produjo el error, qué se esperaba y qué se obtuvo, lo cual facilitaba la depuración del comportamiento de la función.

Se desarrolló un conjunto de pruebas unitarias para validar el comportamiento de la función `parsear_fecha` frente a distintos formatos de entrada, tanto en texto como en formato numérico. Estas pruebas incluyeron fechas con distintos separadores, fechas escritas en español con y sin año, entradas vacías, entre otros escenarios. Posteriormente, se ejecutaron todos los casos. La Figura 7.2 muestra los resultados obtenidos en una primera ejecución.

```
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_formato_texto_espanol_completas PASSED [  7%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_formato_texto_espanol_sin_ano PASSED [ 14%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_numericas_dd_mm_yyyy PASSED [ 21%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_numericas_yyyy_mm_dd FAILED [ 28%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_guiones FAILED [ 35%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_puntos FAILED [ 42%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_texto_adicional PASSED [ 50%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_limite_validas PASSED [ 57%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_entradas_nulas_o_vacias PASSED [ 64%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_invalidas FAILED [ 71%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_texto_sin_fechas PASSED [ 78%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_tipos_pandas_nan FAILED [ 85%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_ambiguas_dd_mm_vs_mm_dd PASSED [ 92%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_diferentes_mayusculas PASSED [100%]
```

Figura 7.2: Resultados de la primera ejecución de pruebas unitarias sobre la función `parsear_fecha`

Como se puede observar, algunas pruebas no fueron exitosas, lo que indicó posibles errores lógicos en la implementación de la función. Uno de los casos que falló fue `test_fechas_numericas_yyyy_mm_dd`, que corresponde a fechas en formato *yyyy/mm/dd*, como *2025/03/15*, que debían ser interpretadas en formato *yyyy-mm-dd*, es decir, como *2025/03/15*.

Sin embargo, el resultado fue interpretado como *None*, indicando que el formato no había sido reconocido correctamente. En la Figura 7.3 se observa el código de la prueba unitaria y el mensaje de error generado por Pytest.

```python
def test_fechas_numericas_yyyy_mm_dd(self):
    """Prueba fechas en formato yyyy/mm/dd"""
    casos = [
        ("2025/03/15", date(2025, 3, 15)),
        ("2024/12/05", date(2024, 12, 5)),
        ("2025 / 01 / 01", date(2025, 1, 1)),
        ("2023/12/31", date(2023, 12, 31)),
    ]

    for fecha_str, fecha_esperada in casos:
        resultado = parsear_fecha(fecha_str)
        assert (
            resultado == fecha_esperada
        ), f"Falló para '{fecha_str}': esperado {fecha_esperada}, obtuvo {resultado}"
        AssertionError: Falló para '2025/03/15': esperado 2025-03-15, obtuvo None
        assert None == datetime.date(2025, 3, 15)
```

Figura 7.3: Código y mensaje de error de la prueba unitaria `test_fechas_numericas_yyyy_mm_dd`

El análisis del error evidenció que no se estaba evaluando correctamente la fecha ingresada debido al orden de validación de condicionales. En consecuencia, se ajustó la lógica para asegurar que este patrón fuera identificado correctamente. Así mismo, se hicieron las revisiones y ajustes correspondientes a las demás pruebas fallidas.

Una vez aplicadas las correcciones, se volvió a ejecutar el conjunto completo de pruebas. Esta nueva ejecución arrojó resultados exitosos en todos los casos, como se muestra en la Figura 7.4, lo que permitió confirmar que la función `parsear_fecha` podía manejar adecuadamente los formatos considerados.

```
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_formato_texto_espanol_completas PASSED [  7%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_formato_texto_espanol_sin_ano PASSED [ 14%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_numericas_dd_mm_yyyy PASSED [ 21%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_numericas_yyyy_mm_dd PASSED [ 28%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_guiones PASSED [ 35%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_puntos PASSED [ 42%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_texto_adicional PASSED [ 50%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_limite_validas PASSED [ 57%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_entradas_nulas_o_vacias PASSED [ 64%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_invalidas PASSED [ 71%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_texto_sin_fechas PASSED [ 78%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_tipos_pandas_nan PASSED [ 85%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_ambiguas_dd_mm_vs_mm_dd PASSED [ 92%]
tests/unit/test_parsear_fecha.py::TestParsearFecha::test_fechas_con_diferentes_mayusculas PASSED [100%]
```

Figura 7.4: Resultados exitosos en la segunda ejecución de las pruebas unitarias sobre la función `parsear_fecha`

7.3.1.2. Sistema web

En el sistema web se realizaron pruebas unitarias sobre las funcionalidades de búsqueda, filtrado y visualización de resultados. Estas pruebas permitieron verificar que las vistas encargadas de procesar

las solicitudes del usuario devolvieran correctamente los datos esperados.

Para esto, se utilizó el cliente de pruebas que proporciona el framework Django, el cual permitió simular peticiones HTTP, sin necesidad de levantar el servidor. En la Figura 7.5 se puede observar un ejemplo de prueba unitaria sobre la funcionalidad que permite el listado de programas académicos.

```python
def test_lista_programas(self, clean_db, programa_cardiologia, programa_neurologia):
        """Prueba que la lista muestre los programas académicos válidos"""
        client = Client()
        response = client.get(reverse("lista_programas"))

        assert response.status_code == 200
        assert "programas_academicos" in response.context
        assert "universidades" in response.context

        # Debe mostrar 2 programas (excluyendo el que no tiene SNIES)
        programas = response.context["programas_academicos"]
        assert len(programas) == 2

        # Verificar que estén ambos programas
        nombres_programas = [programa.nombre_programa for programa in programas]
        assert programa_cardiologia.nombre_programa in nombres_programas
        assert programa_neurologia.nombre_programa in nombres_programas
```

Figura 7.5: Prueba unitaria sobre funcionalidad de listado de programas académicos

Esta prueba tuvo como objetivo verificar que la vista encargada de listar los programas académicos respondiera correctamente ante una solicitud *GET*. Para ello, se comprobó que el código de estado de la solicitud fuera **200**, que el contexto incluyera las claves esperadas `programas_academicos` y `universidades`, y que se listaran todos los programas, siendo 2 programas en este caso.

Además, se verificó que estuvieran presentes los programas previamente cargados en la base de datos de prueba, siendo estos cardiología y neurología. Para esto, se hizo una comparación entre los nombres esperados y los resultados retornados por la vista.

Posteriormente, se ejecutó el conjunto completo de pruebas del sistema web. La Figura 7.6 muestra los resultados obtenidos tras esta ejecución. Como se observa, la mayoría de las pruebas fueron superadas exitosamente, con excepción de una correspondiente al filtro por universidad.

```
tests/test_views.py::TestListaProgramas::test_lista_programas PASSED                    [ 12%]
tests/test_views.py::TestListaProgramas::test_busqueda_por_nombre_programa PASSED       [ 25%]
tests/test_views.py::TestListaProgramas::test_busqueda_sin_resultados PASSED            [ 37%]
tests/test_views.py::TestListaProgramas::test_busqueda_case_insensitive PASSED          [ 50%]
tests/test_views.py::TestListaProgramas::test_filtro_por_universidad FAILED             [ 62%]
tests/test_views.py::TestListaProgramas::test_filtros_combinados PASSED                 [ 75%]
tests/test_views.py::TestListaProgramas::test_exlusion_snies_invalidos PASSED           [ 87%]
tests/test_views.py::TestListaProgramas::test_parametros_vacios PASSED                  [100%]
```

Figura 7.6: Resultados de la ejecución de pruebas unitarias sobre la vista del sistema web.

La prueba que falló correspondió a la funcionalidad de filtrado por universidad, como se muestra en la Figura 7.7. En este caso, se envió una solicitud *GET* incluyendo el parámetro `id_universidad`, con el objetivo de verificar que solo se retornaran los programas asociados a dicha universidad.

```
    def test_filtro_por_universidad(
        self, clean_db, universidad_test, programa_cardiologia, programa_neurologia
    ):
        """Prueba que el filtro por universidad funcione correctamente"""
        client = Client()
        response = client.get(
            reverse("lista_programas"),
            {"id_universidad": universidad_test.id_universidad},
        )

        assert response.status_code == 200
        programas = response.context["programas_academicos"]
        assert len(programas) == 1
        assert programas[0].nombre_programa == programa_cardiologia.nombre_programa
>       assert programas[0].id_universidad == universidad_test.id_universidad
E       assert <Universidad: Universidad de Pruebas> == 1
E        +  where <Universidad: Universidad de Pruebas> = <ProgramaAcademico: Especialización en Cardiología - Universidad de Pruebas>.id_universidad
E        +  and   1 = <Universidad: Universidad de Pruebas>.id_universidad

tests\test_views.py:79: AssertionError
```

Figura 7.7: Prueba unitaria correspondiente al filtrado de programas por universidad.

Sin embargo, la prueba arrojó un error de aserción, ya que el identificador de universidad retornado no coincidía con el esperado. Esto ocurrió porque el campo `id_universidad` del modelo `ProgramaAcademico` se comparó directamente con una instancia de `Universidad`, en lugar de su identificador. Para corregir este error, se modificó la comparación en la aserción.

Tras aplicar esta corrección, se ejecutaron nuevamente todas las pruebas unitarias del sistema web. Como se observa en la Figura 7.8, todas fueron superadas exitosamente.

```
tests/test_views.py::TestListaProgramas::test_lista_programas PASSED                    [ 12%]
tests/test_views.py::TestListaProgramas::test_busqueda_por_nombre_programa PASSED       [ 25%]
tests/test_views.py::TestListaProgramas::test_busqueda_sin_resultados PASSED            [ 37%]
tests/test_views.py::TestListaProgramas::test_busqueda_case_insensitive PASSED          [ 50%]
tests/test_views.py::TestListaProgramas::test_filtro_por_universidad PASSED             [ 62%]
tests/test_views.py::TestListaProgramas::test_filtros_combinados PASSED                 [ 75%]
tests/test_views.py::TestListaProgramas::test_exlusion_snies_invalidos PASSED           [ 87%]
tests/test_views.py::TestListaProgramas::test_parametros_vacios PASSED                  [100%]
```

Figura 7.8: Resultados finales tras corregir el error en el filtrado por universidad.

7.3.2. Pruebas de usabilidad

Con el fin de evaluar la facilidad de uso y el grado de aceptación del prototipo, se realizaron pruebas de usabilidad con usuarios finales que representan a los públicos objetivos de la herramienta. El ejercicio consistió en permitir que cada participante explorara el sistema y, una vez familiarizado con sus funcionalidades, respondiera un cuestionario mixto (afirmaciones tipo Likert, preguntas dicotómicas y campos abiertos) con el objetivo de conocer su percepción y recomendaciones de mejora.

La primera pregunta del cuestionario buscaba identificar el perfil de los participantes. En la siguiente imagen se muestra cómo fue planteada esta pregunta dentro del formulario y la distribución de respuestas obtenida.

Figura 7.9: Caracterización de los participantes

Como se observa en la figura 7.9, participaron cinco personas vinculadas al ámbito médico: dos estudiantes de Medicina próximos a graduarse (40 %), dos médicos en ejercicio (40 %) y un acudiente interesado en costear la especialización de un familiar (20 %). Esta distribución refleja de manera equilibrada los perfiles definidos en el diseño del prototipo, combinando expectativas académicas, profesionales y financieras.

La segunda sección del cuestionario se enfocó en evaluar la experiencia del módulo de búsqueda de especializaciones. A continuación, se muestran las preguntas y las respuestas obtenidas:

3. En escala: 1 (Muy en desacuerdo) a 5 (Muy de acuerdo).

5 Respuestas

ID ↑	Nombre	Respuestas			
		La función de búsqueda por nombre de programa le pareció clara y fácil de usar	La búsqueda por nombre de universidad funcionó correctamente y fue intuitiva	Los resultados de búsqueda fueron relevantes según los criterios que selecciono	La velocidad de respuesta del sistema fue adecuada
1	anonymous	Opción 5	Opción 5	Opción 5	Opción 5
2	anonymous	Opción 5	Opción 5	Opción 5	Opción 5
3	anonymous	Opción 3	Opción 4	Opción 4	Opción 4
4	anonymous	Opción 5	Opción 5	Opción 5	Opción 5
5	anonymous	Opción 5	Opción 5	Opción 5	Opción 5

Figura 7.10: Percepción sobre el módulo de busqueda

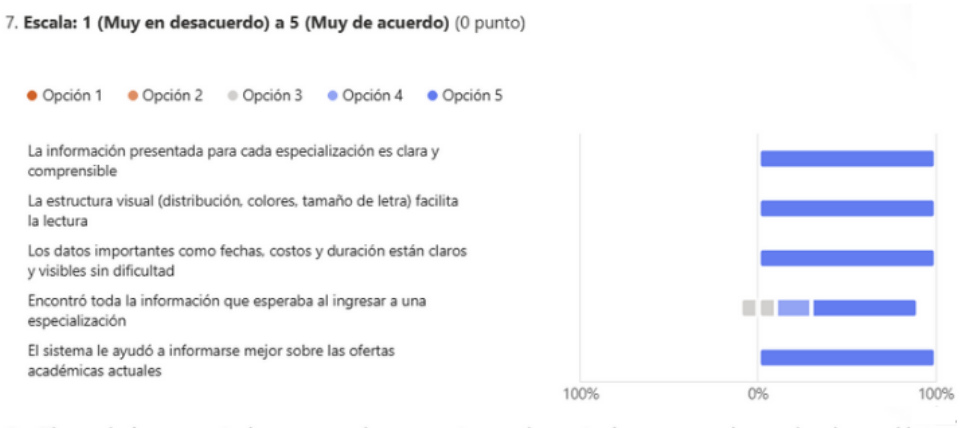
En esta parte, se presentaron cuatro afirmaciones que los participantes valoraron en una escala de 1 (muy en desacuerdo) a 5 (muy de acuerdo). La valoración fue notablemente positiva: el 100 % de los participantes calificó con 4 o 5 la claridad y facilidad de la búsqueda por nombre de universidad (80 % le otorgó la nota máxima). En la búsqueda por nombre de programa, el 60 % eligió 5 y el 40 % restante se situó entre 3 y 4, sin respuestas negativas. La relevancia de los resultados alcanzó un 80 % de puntuaciones de 5 y un 20 % de 4, lo que indica que los filtros satisfacen la expectativa general. Por último, la velocidad de respuesta fue considerada adecuada por la totalidad de los usuarios (100 % entre 4 y 5).

Para complementar estos datos cuantitativos, se plantearon dos preguntas abiertas:

- "¿Tiene algún comentario, sugerencia o aspecto que mejoraría sobre la sección de búsqueda de especializaciones?": Las respuestas coincidieron en la necesidad de "organizar mejor los cuadros", "habilitar una comparación directa entre especialidades o universidades" y "añadir un filtro por precio".
- "¿Le hubiera gustado contar con filtros adicionales?" (respuesta Sí/No): El 60 % respondió Sí, mientras que el 40 % indicó que los filtros actuales resultan suficientes. De los participantes que contestaron si deseaban más filtros, propusieron: rangos de matrícula, temáticas de los exámenes de admisión, comparaciones entre programas y requisitos de inscripción.

En conjunto, los comentarios cualitativos refuerzan la buena acogida del módulo de búsqueda, pero señalan oportunidades claras de mejora: optimizar la organización visual de los resultados, ofrecer comparaciones rápidas y ampliar los criterios de filtrado (en especial por precio y requisitos académicos).

La tercera sección del cuestionario se centró en la percepción de los usuarios frente a la información detallada que se ofrece para cada especialización. En la siguiente figura se puede ver algunas de las preguntas realizadas en esta sección, así como un resumen gráfico de las respuestas.



8. ¿Tiene algún comentario, sugerencia o aspecto que le gustaría que se mejore sobre la sección donde se muestra la información completa de cada especialización?

1 Respuestas

ID ↑	Nombre	Respuestas
1	anonymous	Me gustaría que hubiera más universidades en la lista y que muestre que acreditaciones tiene cada universidad.

Figura 7.11: Percepción sobre la información detallada

En esta sección se plantearon cinco afirmaciones evaluadas en la misma escala Likert. Los resultados revelaron una aceptación prácticamente unánime:

- Claridad de la información: todos los participantes (100 %) puntuaron con 5 la comprensibilidad del contenido presentado.
- Estructura visual: la totalidad de participantes otorgó nuevamente la nota máxima, validando que la distribución, los colores y el tamaño de letra facilitaron la lectura.
- Visibilidad de datos clave (fechas, costos, duración): se obtuvo un consenso absoluto en 5, lo que confirmó la correcta jerarquización de los elementos más críticos.
- Completitud de la información encontrada: tres usuarios (60 %) señalaron que obtuvieron exactamente lo que esperaban (5), mientras que uno (20 %) matizó con un 4 y otro (20 %) consideró que la ficha cubría la mayor parte, pero aún podía ampliarse (3).
- Utilidad para informarse mejor sobre la oferta académica: nuevamente se alcanzó el 100 % de valoraciones de 5, lo que sugirió un impacto positivo en la toma de decisiones del usuario.

Adicionalmente, uno de los participantes dejó un comentario en el que sugiere incluir más universidades en la lista y mostrar qué acreditaciones tiene cada una. Esta observación refuerza la importancia de ampliar la cobertura institucional.

En conjunto, los resultados confirmaron que la ficha informativa cumple con los principios de claridad, jerarquía visual y valor añadido para la toma de decisiones, aunque hay la oportunidad de complementar la información con los indicadores de acreditación y ampliar la cantidad de universidades.

La última parte del cuestionario se enfocó en conocer la percepción general de los usuarios sobre la herramienta, su nivel de confianza en los datos presentados, la comodidad durante la navegación, y su disposición a volver a usar el sistema en el futuro. La siguiente figura muestra cómo fue presentada esta sección en el formulario:

9. Pregunta

5 Respuestas

ID ↑	Nombre	Respuestas				
		En general, la navegación por el sistema fue fácil	Considera que esta herramienta podría ser útil para otras personas como usted	Le gustaría volver a usar esta herramienta en el futuro	Confía en los datos que muestra el sistema	Se sintió cómodo(a) usando la interfaz sin necesidad de ayuda adicional
1	anonymous	Opción 5	Opción 5	Opción 5	Opción 5	Opción 5
2	anonymous	Opción 5	Opción 5	Opción 5	Opción 5	Opción 5
3	anonymous	Opción 5	Opción 5	Opción 5	Opción 5	Opción 5
4	anonymous	Opción 5	Opción 5	Opción 5	Opción 5	Opción 5
5	anonymous	Opción 5	Opción 5	Opción 5	Opción 5	Opción 5

Figura 7.12: Percepción de general de la herramienta

En esta sección se plantearon seis afirmaciones, todas evaluadas con una escala Likert. Los resultados fueron consistentes y altamente favorables: los cinco participantes seleccionaron la opción 5 en todos los ítems, lo que indica un alto grado de satisfacción con la herramienta. Destacan especialmente la facilidad de navegación, la utilidad del sistema para otros usuarios, la intención de volver a usarlo, la confianza en la información y la comodidad de uso sin necesidad de asistencia. Esta unanimidad refuerza la percepción positiva obtenida a lo largo de todo el cuestionario.

Posteriormente, se solicitó a los usuarios que compartieran comentarios o sugerencias para mejorar la apariencia o la forma de interactuar con el sistema. Las respuestas abiertas, recopiladas en la figura 7.13, ofrecieron observaciones para futuras mejoras del prototipo.

11. ¿Hay algo en la apariencia o en la forma de usar el sistema que te gustaría que fuera diferente o más fácil?

5 Respuestas

ID ↑	Nombre	Respuestas
1	anonymous	Un poco más de imágenes
2	anonymous	organizar la información por universidad
3	anonymous	No
4	anonymous	por cantidad de cupos, convocatorias mas proximas o por fercha
5	anonymous	No

Figura 7.13: Comentarios sobre apariencia y facilidad de uso de la interfaz

Algunas recomendaciones incluyeron incorporar más imágenes, organizar la información por universidad y permitir el ordenamiento por criterios como cantidad de cupos, fechas próximas o convocatorias.

También, se preguntó por errores o enlaces defectuosos, y los cinco participantes respondieron "No" o "ninguno", indicando que no se detectaron fallos de carga ni enlaces defectuosos durante las pruebas.

Para finalizar el cuestionario, se incluyó una pregunta abierta sobre la utilidad de la herramienta para encontrar información sobre especializaciones médicas de interés. La figura 7.14 muestra cómo se presentó esta pregunta.

13. ¿Esta herramienta le ayudó a encontrar más fácilmente información sobre las especializaciones de su interés?

5 Respuestas

ID ↑	Nombre	Respuestas
1	anonymous	100%
2	anonymous	si
3	anonymous	Si
4	anonymous	claro que si . seria genial que vinculara mas especializaciones.
5	anonymous	Es de gran ayuda y muy organizado

Figura 7.14: Percepción de utilidad de la herramienta

Los comentarios fueron positivos y directos: frases como “100 %”, “sí”, “claro que sí”, “es de gran ayuda y muy organizado” y “sería genial que vinculara más especializaciones” reflejan una percepción general favorable frente al objetivo principal del sistema. La herramienta cumplió con su propósito de centralizar y facilitar el acceso a la información, permitiendo a los usuarios ahorrar tiempo y consultar los datos de forma ordenada.

En conjunto, las respuestas mostraron una aceptación global muy alta: los usuarios confiaron en la información, percibieron práctica la herramienta y no encontraron errores. Las oportunidades de mejora se concentraron en enriquecer los elementos visuales, ampliar los filtros de organización y sumar más programas académicos a la base de datos.

Capítulo 8

Conclusiones y trabajos futuros

8.1. Conclusiones

1. Cumplimiento de objetivos. El prototipo alcanzó los cuatro objetivos específicos: identificación de estructuras web, extracción y estandarización de datos, diseño de una interfaz de consulta y validación funcional mediante pruebas controladas y con usuarios reales.

2. Viabilidad técnica demostrada. La combinación de Selenium y un modelo de configuración por universidad permitió extraer información de páginas con DOM dinámico, menús desplegables y modales sin necesidad de reescribir el núcleo del scraper, probando la escalabilidad de la arquitectura.

3. Integridad y fiabilidad de los datos. El esquema relacional en PostgreSQL eliminó duplicidades y preservó la consistencia entre programas, procesos de admisión y contactos, facilitando consultas complejas y futuras ampliaciones curriculares.

4. Experiencia de usuario destacada. Las pruebas de usabilidad revelaron unanimidad (valoración 5) en facilidad de navegación, confianza en la información e interés en volver a utilizar la herramienta; ningún participante reportó enlaces defectuosos, ni errores de carga.

5. Aporte académico y social. El sistema desarrollado contribuye a reducir las barreras de acceso a la información para quienes aspiran a especializaciones médicas. Además, permitió evidenciar que la dispersión de datos en los sitios web de muchas universidades representa un obstáculo real para los usuarios. Al centralizar esta información, se demostró que herramientas similares pueden ser de gran utilidad en otros contextos educativos, facilitando la búsqueda y ampliando el acceso a opciones que, de otro modo, podrían pasar desapercibidas.

6. Limitaciones y potencial de mejora. El alcance se restringió a cuatro universidades y a un subconjunto de filtros; ampliar la cobertura nacional, incorporar métricas de acreditación y habilitar comparaciones directas entre programas, son pasos recomendados a corto plazo.

8.2. Trabajos futuros

Como sucede con un sistema prototipo, este proyecto no queda exento de áreas con posibilidad de mejora. Este capítulo tiene como objetivo documentar algunas ideas que surgieron durante el proceso de desarrollo y que podrían servir de guía para futuras iteraciones, tanto en lo funcional como en lo

técnico. Las propuestas incluidas se basan en la experiencia adquirida durante la implementación, la validación del sistema y el análisis de las necesidades reales del usuario final.

8.2.1. Funcionalidades planificadas para el usuario

Desde el planteamiento inicial del proyecto se definieron historias de usuario que demostraban el alcance del prototipo presentado: la posibilidad de que los interesados se registren, marquen programas como favoritos y reciban notificaciones automáticas sobre aperturas de inscripciones, cambios en costos o requisitos y fechas límite próximas.

8.2.2. Expansión del scraping y cobertura nacional

Como parte del enfoque inicial, el prototipo se limitó a cuatro universidades para demostrar la viabilidad técnica del scraping; la siguiente fase contempla incorporar la totalidad de las instituciones con Acreditación de Alta Calidad que ofrezcan especializaciones médicas, garantizando así una cobertura nacional robusta y homogénea. Una vez consolidado ese repertorio, cabría extender la base de datos a otros niveles de la educación superior (pregrado, maestría y doctorado) y a programas de disciplinas distintas, siempre que se disponga de fuentes confiables.

8.2.3. Mejora de la experiencia de búsqueda

En versiones futuras, la ampliación de contenido debe venir acompañada de mejoras en la experiencia de búsqueda. Filtros avanzados por rango de matrícula, ciudad, modalidad, cupos disponibles, acreditaciones o proximidad temporal de la convocatoria, y la opción de ordenar o comparar varias especializaciones en una misma vista resultan indispensables para mantener la usabilidad demostrada. Sumado a ello, un diseño con refuerzo visual (imágenes representativas).

8.2.4. Mejoras en la arquitectura del sistema

En cuanto a la arquitectura técnica del sistema, se identificaron diversas posibilidades de mejora. Entre ellas, se considera la automatización del proceso de actualización mediante trabajos programados que vigilen cambios en los sitios web y detecten enlaces rotos, así como la migración hacia una estructura de microservicios con una API pública que permita a terceros consumir los datos e integrarlos en sus propios sistemas.

8.2.5. Ampliación y profundización en las pruebas

En futuras versiones del sistema, se tiene previsto ampliar significativamente la cobertura de pruebas, abordando tanto aspectos funcionales como no funcionales. Se planea llevar a cabo pruebas de integración completas que consideren escenarios con múltiples universidades operando simultáneamente, además de abarcar flujos completos que vayan desde la extracción de datos hasta la visualización de los programas académicos. Asimismo, se contempla incorporar pruebas de rendimiento para analizar la capacidad de escalabilidad del sistema frente al aumento en el volumen de

datos, junto con pruebas de regresión, diseñadas para garantizar que las nuevas modificaciones no afecten las funcionalidades previamente validadas.

8.2.6. Sugerencias de usuarios

Finalmente, las sugerencias recogidas en la encuesta (mayor variedad de filtros, organización de la información por universidad y visualización de rankings o acreditaciones) aportan una visión adicional. De este modo, el proyecto podría convertirse en un buscador académico integral y sostenible, alineado con las necesidades reales de los futuros usuarios.

Bibliografía

[1] M. de Educación Nacional de Colombia, "Estudiantes graduados 2023 [base de datos en línea]," 2023, recuperado de https://snies.mineducacion.gov.co/portal/ESTADISTICAS/Bases-consolidadas/.

[2] C. Slamet, R. Andrian, D. S. Maylawati, Suhendar, W. Darmalaksana, and M. A. Ramdhani, "Web scraping and naïve bayes classification for job search engine," *IOP Conference Series: Materials Science and Engineering*, vol. 288, no. 1, p. 012038, Jan. 2018.

[3] S. F. Rashid and R. P. Qasha, "Extracting and archiving data from social media to support cultural heritage preservation in nineveh," in *Proceedings of the 2nd 2022 International Conference on Computer Science and Software Engineering (CSASE 2022)*, 2022, pp. 295–300.

[4] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill, 2020.

[5] K. Schwaber and J. Sutherland, "The scrum guide: The definitive guide to scrum: The rules of the game," 2020, [En línea]. Disponible en: https://scrumguides.org/scrum-guide.html.

[6] Ausum Cloud. (2025) Scrum: La metodología ágil más popular en las empresas. Accedido el 1 de julio de 2025. [Online]. Available: https://ausum.cloud/scrum-metodologia-agil-mas-popular-en-empresas/

[7] R. R. Fayzrakhmanov, E. Sallinger, B. Spencer, T. Furche, and G. Gottlob, "Browserless web data extraction: Challenges and opportunities," in *Proceedings of the 2018 World Wide Web Conference (WWW 2018)*, Lyon, France, Apr. 2018, pp. 1095–1104.

[8] S. Chaudhari, R. Aparna, V. G. Tekkur, G. L. Pavan, and S. R. Karki, "Ingredient/recipe algorithm using web mining and web scraping for smart chef," in *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Bangalore, India, 2020, pp. 1–4.

[9] A. Puscasiu, A. Fanca, D.-I. Gota, and H. Valean, "Data mining for identifying trends in markets," in *2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, Cluj-Napoca, Romania, 2020, pp. 1–6.

[10] U. E. Bosque, "Especializaciones facultad de medicina [sitio web]," 2025, recuperado de https://www.unbosque.edu.co/programas-academicos/especializaciones/facultad-de-medicina/modalidad.

[11] U. N. de Colombia, "Especialidades médicas facultad de medicina - sede bogotá [sitio web]," 2025, recuperado de https://medicina.bogota.unal.edu.co/formacion/especialidades-medicas.

[12] P. U. J. Cali, "Especializaciones médicas [sitio web]," 2025, recuperado de https://www.javerianacali.edu.co/programas/especializaciones.

[13] U. del Valle, "Programas de posgrado facultad de medicina [sitio web]," 2025, recuperado de https://medicina.univalle.edu.co/postgrado.

[14] A. R. O. Tanori and G. A. G. Mireles, "Estimación en proyectos de software basado en "planning poker"," Cartel presentado en la XXXIII Semana Nacional de Investigación y Docencia en Matemáticas, 2022, https://semana.mat.uson.mx/semanaxxxiii/cartel/pdf/CARTELAlanRaulOrtizTanori.pdf.

[15] C. Zheng, G. He, and Z. Peng, "A study of web information extraction technology based on beautiful soup," *Journal of Computers*, vol. 10, no. 6, pp. 381–387, 2015, iSSN: 1796-203X.

[16] Y. Dikilitaş, C. Çakal, A. C. Okumuş, H. N. Yalçın, E. Yıldırım, Ömer Faruk Ulusoy, B. Macit, A. E. Kırkaya, Özkan Yalçın, E. Erdoğmuş, and A. Sayar, "Performance analysis for web scraping tools: Case studies on beautifulsoup, scrapy, htmlunit and jsoup," in *Lecture Notes in Computer Science*. Springer, 2024. [Online]. Available: https://www.researchgate.net/publication/380189817

[17] ScrapingBee. (2024) Web scraping with selenium and python: The definitive guide. Consultado en junio de 2025. [Online]. Available: https://www.scrapingbee.com/blog/selenium-python/

[18] QA Touch. (2024) Selenium web scraping: A beginner's guide with practical examples. Consultado en junio de 2025. [Online]. Available: https://www.qatouch.com/blog/selenium-web-scraping/

[19] Proxyway. (2024) Cheerio vs puppeteer: Which one is better for web scraping? Consultado en junio de 2025. [Online]. Available: https://proxyway.com/guides/cheerio-vs-puppeteer-for-web-scraping

[20] A. Multiple. (2024) Cheerio vs puppeteer: Comparison of two web scraping tools. Consultado en junio de 2025. [Online]. Available: https://research.aimultiple.com/cheerio-vs-puppeteer/

[21] Oxylabs. (2025) Playwright vs puppeteer: Key differences for web scraping. Consultado en junio de 2025. [Online]. Available: https://oxylabs.io/blog/playwright-vs-puppeteer

[22] ScraperAPI. (2021) Cheerio vs puppeteer for web scraping. Consultado en junio de 2025. [Online]. Available: https://www.scraperapi.com/blog/cheerio-vs-puppeteer/

[23] A. Camargo and G. Arciniegas, "Bases de datos sql vs nosql: Una revisión comparativa para el desarrollo de aplicaciones web," *Revista Ingenio Magno*, vol. 13, no. 2, pp. 45–55, 2022, consultado en junio de 2025. [Online]. Available: https://revistas.americana.edu.co/index.php/ingeniomagno/article/view/198

[24] M. Gibert Ginestà and Pérez Mora, *Bases de datos en PostgreSQL*, 2022, p06/M2109/02152.

[25] Y. Castro Castaño and L. A. Montoya, *Bases de Datos MySQL*. Colombia: Universidad Nacional Abierta y a Distancia (UNAD), 2019, iSBN: 978-958-48-7722-2.

[26] J. C. Perales Díaz, *Diseño de bases de datos con SQLite para aplicaciones móviles*. Universidad Politécnica de Madrid, 2021, trabajo de Fin de Grado, Escuela Técnica Superior de Ingenieros de Telecomunicación.

[27] Miro. (2025) ¿qué es el modelo c4? Consultado el 2 de julio de 2025. [Online]. Available: https://miro.com/es/diagrama/que-es-modelo-c4/

[28] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2018.

[29] S. Ramírez, "Fastapi documentation," https://fastapi.tiangolo.com, 2022, accedido en junio de 2025.

[30] Django Software Foundation, "Documentación oficial de django," https://docs.djangoproject.com, 2024, accedido en junio de 2025.

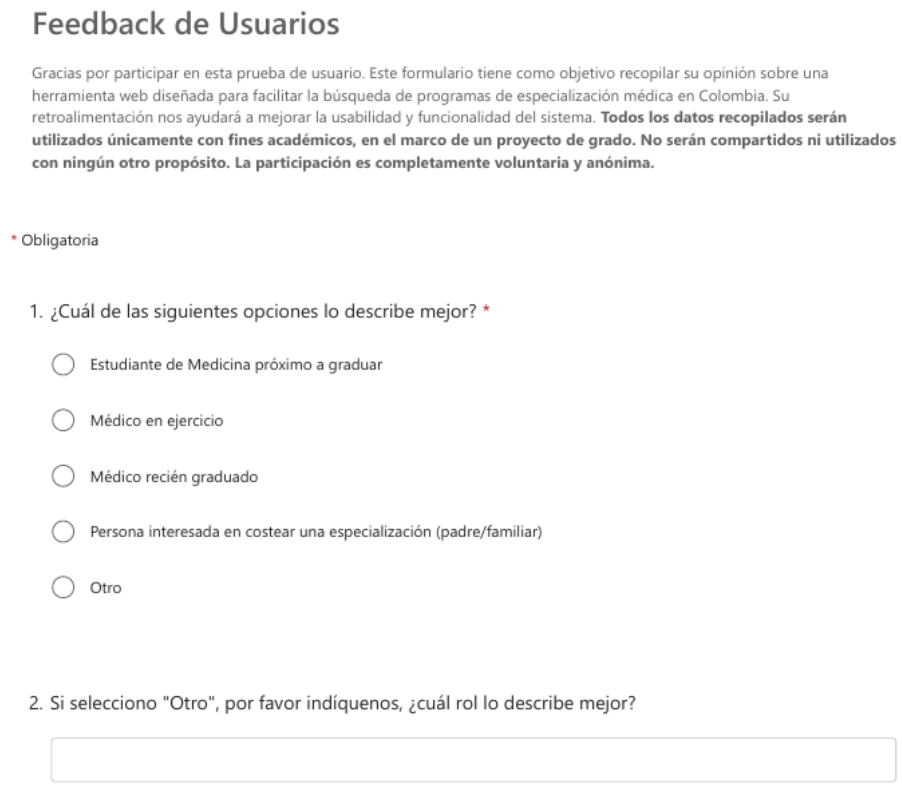
[31] Amazon Web Services. (s.f.) ¿qué son las pruebas unitarias? Consultado en junio de 2025. [Online]. Available: https://aws.amazon.com/what-is/unit-testing/

Capítulo 9

Anexos

9.1. Cuestionario utilizado en las Pruebas de Usabilidad

A continuación se presenta el cuestionario aplicado a los participantes que interactuaron con el prototipo en las pruebas de usabilidad. Este cuestionario se realizó con el propósito de recopilar sus impresiones sobre la utilidad, facilidad de uso y oportunidades de mejora del sistema. El cuestionario se elaboró a través de la plataforma Microsoft Forms, lo que permitió obtener las respuestas de manera anónima y estandarizada.



Feedback de Usuarios

Gracias por participar en esta prueba de usuario. Este formulario tiene como objetivo recopilar su opinión sobre una herramienta web diseñada para facilitar la búsqueda de programas de especialización médica en Colombia. Su retroalimentación nos ayudará a mejorar la usabilidad y funcionalidad del sistema. **Todos los datos recopilados serán utilizados únicamente con fines académicos, en el marco de un proyecto de grado. No serán compartidos ni utilizados con ningún otro propósito. La participación es completamente voluntaria y anónima.**

* Obligatoria

1. ¿Cuál de las siguientes opciones lo describe mejor? *

- ◯ Estudiante de Medicina próximo a graduar
- ◯ Médico en ejercicio
- ◯ Médico recién graduado
- ◯ Persona interesada en costear una especialización (padre/familiar)
- ◯ Otro

2. Si selecciono "Otro", por favor indíquenos, ¿cuál rol lo describe mejor?

Figura 9.1: Preguntas 1 y 2

3. **En escala: 1 (Muy en desacuerdo) a 5 (Muy de acuerdo).** *

	Opción 1	Opción 2	Opción 3	Opción 4	Opción 5
La función de búsqueda por *nombre de programa* le pareció clara y fácil de usar	○	○	○	○	○
La búsqueda por *nombre de universidad* funcionó correctamente y fue intuitiva	○	○	○	○	○
Los resultados de búsqueda fueron relevantes según los criterios que selecciono	○	○	○	○	○
La velocidad de respuesta del sistema fue adecuada	○	○	○	○	○

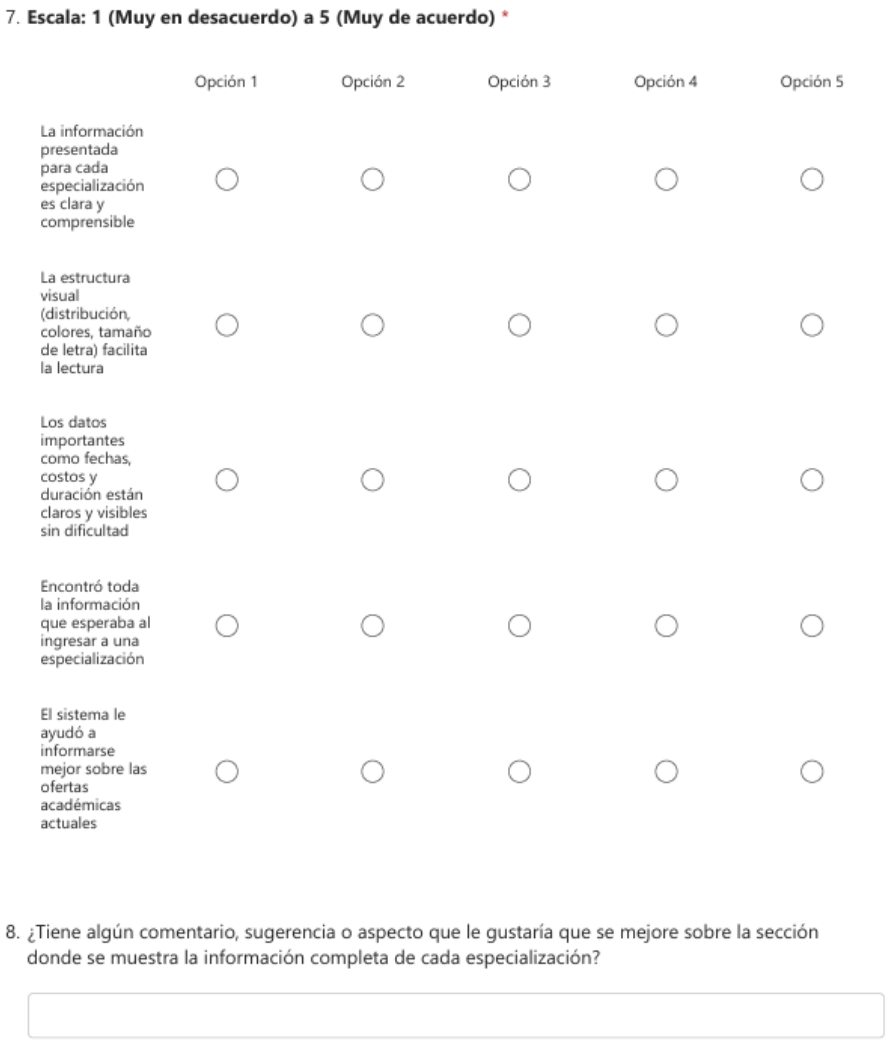
4. ¿Tiene algún comentario, sugerencia o aspecto que mejoraría sobre la sección de búsqueda de especializaciones?

5. ¿Le hubiera gustado contar con filtros adicionales? *

- ○ Si
- ○ No

6. Si contesto en la pregunta anterior "Sí", por favor indique ¿cuáles filtros adicionales le hubiese gustado encontrar?

Figura 9.2: Preguntas 3, 4, 5 y 6



7. **Escala: 1 (Muy en desacuerdo) a 5 (Muy de acuerdo)** *

	Opción 1	Opción 2	Opción 3	Opción 4	Opción 5
La información presentada para cada especialización es clara y comprensible	○	○	○	○	○
La estructura visual (distribución, colores, tamaño de letra) facilita la lectura	○	○	○	○	○
Los datos importantes como fechas, costos y duración están claros y visibles sin dificultad	○	○	○	○	○
Encontró toda la información que esperaba al ingresar a una especialización	○	○	○	○	○
El sistema le ayudó a informarse mejor sobre las ofertas académicas actuales	○	○	○	○	○

8. ¿Tiene algún comentario, sugerencia o aspecto que le gustaría que se mejore sobre la sección donde se muestra la información completa de cada especialización?

Figura 9.3: Preguntas 7 y 8

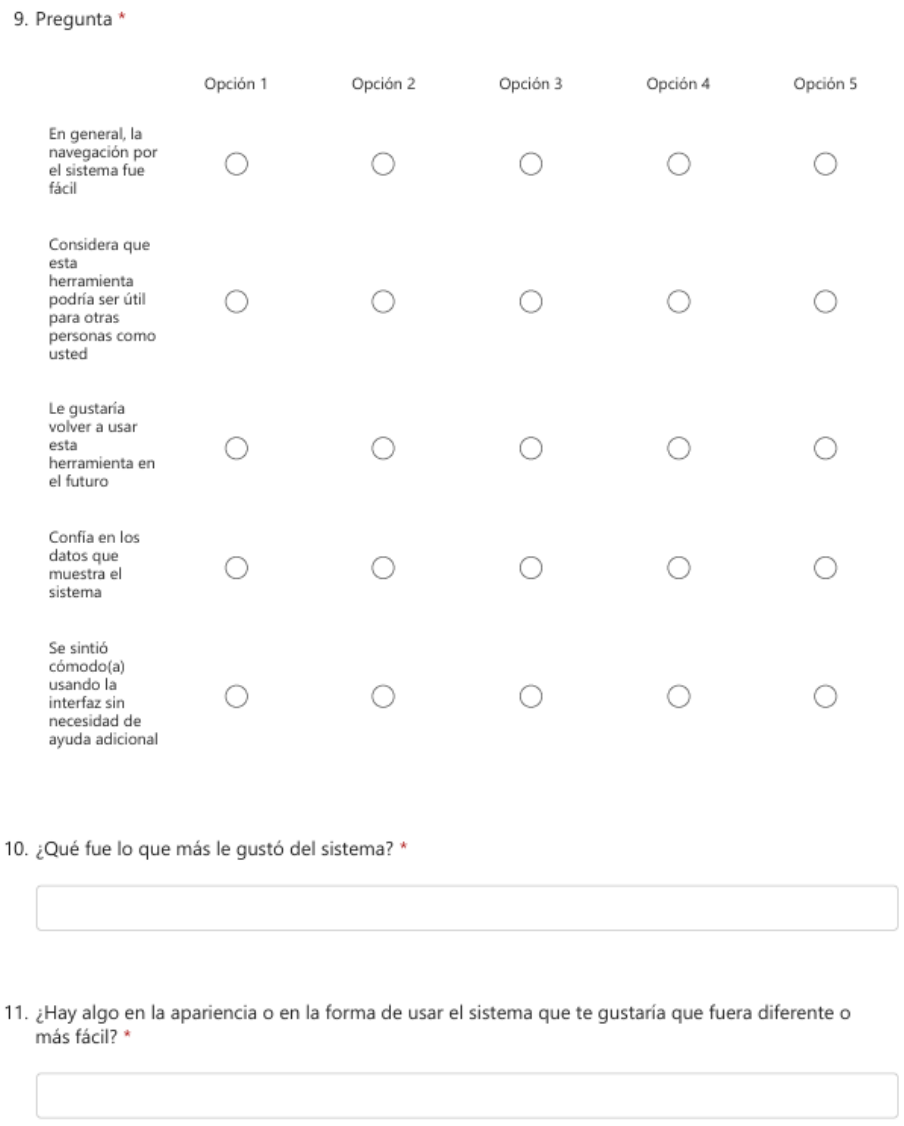


Figura 9.4: Preguntas 9, 10 y 11

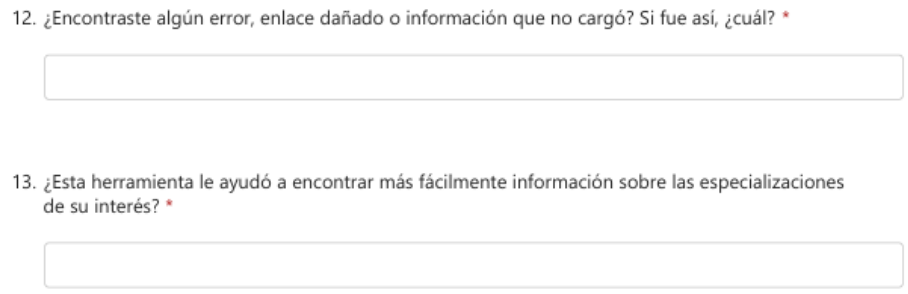


Figura 9.5: Preguntas 12 y 13

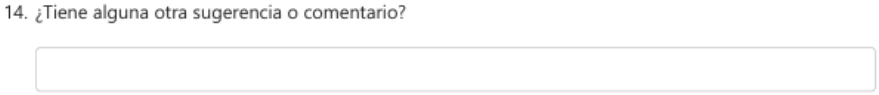


Figura 9.6: Pregunta 14