

# Desarrollo del Perfil del estudiante dentro de No estás solo

Federico Melo Barrero<sup>1</sup>

22 de enero de 2025

## **Resumen**

El presente proyecto de grado describe el desarrollo del Perfil del estudiante al interior de la plataforma *No estás solo*. El Perfil del estudiante consiste en un conjunto de visualizaciones interactivas que presentan información académica y contextual sobre los estudiantes de la Universidad de los Andes. Originalmente dirigido a profesores en su rol como consejeros, su uso se extendió a estudiantes, coordinadores y administrativos, quienes encontraron valiosa la información para procesos de consejería y toma de decisiones. El documento relata el contexto que suscitó la creación del Perfil y los objetivos asociados a la misma; además, detalla el desarrollo técnico del módulo, incluyendo las capas de datos, lógica y presentación. Se describen las tecnologías empleadas, las prácticas de ingeniería de software implementadas y los principios de claridad y relevancia para la exposición de la información que fueron tenidos en cuenta, para finalizar con los resultados obtenidos.

## **Abstract**

This thesis project presents the development of the Student Profile within the *No estás solo* platform. The Student Profile is a set of interactive visualizations that display academic and contextual information about students at Universidad de los Andes. Initially designed for professors in their role as advisors, its use expanded to students, coordinators, and administrative staff, who found the information valuable for counseling processes and decision-making. The document outlines the context that motivated the creation of the Student Profile and the objectives set for its development. It also details the technical aspects of the module, including the data, logic, and presentation layers. Additionally, thorough descriptions are provided of the technologies used, the software engineering practices implemented, the principles of clarity and relevance applied to information presentation, and the results achieved.

# Índice general

<b>1. Introducción</b>	<b>9</b>
1.1. Contexto . . . . .	9
1.1.1. La plataforma <i>No estás solo</i> . . . . .	9
1.1.2. La necesidad del Perfil del estudiante . . . . .	10
1.2. Objetivos . . . . .	10
1.2.1. Objetivos específicos . . . . .	11
1.2.2. Resultados esperados . . . . .	11
1.3. Usuarios y requerimientos . . . . .	12
1.4. Arquitectura general del sistema . . . . .	12
<b>2. Capa de datos</b>	<b>13</b>
2.1. Fuentes de datos . . . . .	13
2.2. Proceso de extracción, transformación y carga . . . . .	14
2.2.1. Extracción de los datos . . . . .	14
2.2.2. Transformación de los datos . . . . .	16
2.2.3. Carga de los datos . . . . .	16
2.3. Flujo de los datos . . . . .	16
<b>3. Capa de lógica</b>	<b>19</b>
3.1. ¿Por qué un API REST? . . . . .	19
3.2. Diseño del API . . . . .	20
3.2.1. Estructura de clases del API . . . . .	20
3.3. Implementación del API . . . . .	27
3.3.1. Lenguaje y framework . . . . .	27
3.3.2. Estructura de directorios . . . . .	29
3.4. Calidad de código . . . . .	35

3.4.1. Análisis estático . . . . .	35
3.4.2. Documentación . . . . .	36
3.4.3. Pruebas y automatización . . . . .	37
3.5. Infraestructura . . . . .	37
3.5.1. Despliegue del API . . . . .	37
3.5.2. Base de datos . . . . .	38
<b>4. Capa de presentación</b>	<b>39</b>
4.1. Implementación del frontend . . . . .	39
4.1.1. Lenguaje y framework . . . . .	39
4.1.2. Estructura de directorios del proyecto . . . . .	40
4.1.3. Ruta al perfil del estudiante . . . . .	42
4.2. Interfaz del Perfil del estudiante . . . . .	42
4.2.1. Acceso al perfil del estudiante . . . . .	43
4.2.2. Disposición general . . . . .	46
4.2.3. Pestañas del Perfil del estudiante . . . . .	47
<b>5. Diseño Conceptual</b>	<b>67</b>
5.1. Principios de Exposición de Información . . . . .	67
5.2. Visualizaciones y Tablas . . . . .	67
<b>A. Contribuciones al CAPP</b>	<b>69</b>
A.1. Contexto . . . . .	69
A.2. Recursos expuestos . . . . .	69
<b>B. Apuntes para el Ecosistema de analítica institucional</b>	<b>71</b>
<b>Glosario</b>	<b>77</b>
<b>Bibliografía</b>	<b>79</b>

# Agradecimientos

Este proyecto fue desarrollado a lo largo del año 2024, mi último año como estudiante de pregrado en la Universidad de los Andes. El proyecto recoge mi trabajo como desarrollador en la Vicedecanatura de Asuntos Estudiantiles de la Facultad de Ingeniería. Sin embargo, da la sensación de ser el cierre de mi recorrido académico en la Universidad. Por esta razón, no quiero limitar estos agradecimientos únicamente a quienes contribuyeron directamente al desarrollo del proyecto, sino extenderlos a todas las personas que, de una u otra manera, han sido imprescindibles en mi formación académica y personal.

El año 2024 fue un periodo de muchos cambios y de trabajo diligente. Además de trabajar en este proyecto, desempeñé el rol de monitor de investigación en el proyecto Cupi2, enfocado en mejorar la enseñanza de programación en la Universidad; trabajé como desarrollador de software en el equipo de analítica de datos de la multinacional canadiense Caseware; y concluí las materias necesarias para graduarme como Ingeniero de Sistemas y Computación. Este arduo esfuerzo se vio recompensado con los mejores resultados académicos de mi trayectoria y me permitió culminar mis estudios con el mejor promedio general acumulado de la Facultad de Ingeniería en los últimos quince años.

Eso, por supuesto, no hubiese sido posible solo. Antes que a nadie, quiero agradecer profundamente a Angélica, William y Sebastián: mi familia. Su crianza amorosa forjó mi persona y su apoyo incondicional ha sido el cimiento de mi vida. A ellos les debo este y todos mis proyectos.

En segundo lugar, debo un enorme agradecimiento a mis amigos más cercanos. Ellos son un pilar fundamental en mi vida y en mi felicidad, contribuyendo constantemente a una ya extensa colección de momentos inolvidables. A David, Santiago y Andrés, y a Laura, Laura, Mariana y Sarah: gracias por estar en mi vida. Sin esos dos conjuntos de personas, el camino no hubiese sido tan agradable.

Sumado a lo anterior, por supuesto debo expresar mi gratitud a todos quienes han invertido su tiempo, recursos y esfuerzo en mi educación. Eso incluye a todos quienes fueron partícipes de mi formación tanto en el Colegio San Carlos como en la Universidad de los Andes. No ignoro el inmenso privilegio que constituye haber estudiado en las mejores instituciones educativas del país y haber contado con mentores verdaderamente excepcionales. Mis agradecimientos más profundos van para Jaime Rueda Moreno y Héctor Carranza Granados, quienes me enseñaron a estudiar; y para Nicolás Rincón Sánchez y Alejandro Arturo Pérez Ramírez, quienes me forzaron a hacerlo con rigor.

También quiero agradecer a algunas personas que confiaron en mí y me ofrecieron oportunidades que resultaron ser puntos de inflexión en mi formación. En particular, a

Iván David Salazar Cárdenas, quien apostó por mí en el proyecto Cupi2; y a José Joaquín Bocanegra García, quien despertó mi interés por el desarrollo web y auspició mi primera experiencia como desarrollador de software.

Finalmente, quisiera agradecer a quienes estuvieron directamente involucrados en este proyecto. A Mariana y Nicolás, mis compañeros en el desarrollo de No estás solo; a Catalina, Manuel y (nuevamente) Santiago, cuya pericia en ingeniería de datos hizo posible el desarrollo de la capa de datos del proyecto; a Óscar, siempre pendiente de todo; y, por supuesto, a Marcela Hernández, quien confió en mí, me dio la oportunidad de trabajar en este proyecto y me acompañó en todo el proceso.

A todos los acá mencionados y quienes no lo están, pero que han sido parte de mi vida en los últimos años, gracias por acompañarme en esta travesía y por contribuir a que esta etapa de mi vida sea tan significativa. Los resultados son muy buenos y se deben a ustedes.

*Big Brother is watching you.*

—George Orwell, *Nineteen Eighty-Four*



# Capítulo 1

## Introducción

### Resumen del capítulo

NES es una plataforma web de la Universidad de los Andes que centraliza sus recursos y servicios. El Perfil del estudiante complementa esta plataforma al proveer información académica y socioeconómica del estudiantado, antes faltante e imprescindible para mejorar las consejerías a estudiantes y las decisiones administrativas.

### 1.1. Contexto

La Universidad de los Andes es una institución educativa colombiana de gran renombre, reconocida por su excelencia académica y su compromiso con la formación integral de sus estudiantes. En aras de impulsar el éxito de sus estudiantes en todo ámbito, la Universidad realiza un esfuerzo constante por ofrecer servicios y recursos que fomenten el desarrollo académico, social, personal y profesional de sus estudiantes, así como el bienestar de la comunidad universitaria en general.

#### 1.1.1. La plataforma *No estás solo*

En los últimos dos años, la Vicedecanatura de Asuntos Estudiantiles de la Universidad de los Andes identificó dos dolencias significativas concernientes al estudiantado y al profesorado, respectivamente. La primera, consiste en el desconocimiento por parte de los estudiantes de la inmensa cantidad de recursos y servicios de apoyo que la Universidad pone a su disposición en pos de su progreso académico, social, personal y profesional. La segunda, radica en la dificultad y falta de recursos que tenían los profesores a la hora de ejercer su labor como consejeros en apoyo a los estudiantes.

Como herramienta para subsanar estas dolencias, la Vicedecanatura ha impulsado el desarrollo y la extensión de una aplicación web: *No estás solx* (en adelante, NES). Esta plataforma fue concebida por la Vicedecanatura de Asuntos Estudiantiles de la Facultad de Ingeniería como una solución integral para garantizar que todos los miembros de la comunidad, especialmente los estudiantes, puedan acceder de manera centralizada y clara a todos los recursos y servicios disponibles.

Desde la perspectiva del estudiantado, NES centraliza el acceso a todas las herramientas que fomentan su éxito académico, personal y profesional. Esto incluye información sobre eventos académicos y culturales, procesos y solicitudes administrativas,

y servicios de apoyo como consejerías y tutorías. Sumado a eso, facilita el acceso de cada estudiante a su red de apoyo, compuesta por profesores consejeros, coordinadores académicos y otros profesionales de la Universidad. Toda esta información se encontraba dispersa en las distintas plataformas de la Universidad, lo que dificultaba su acceso y su uso conjunto. Esto suponía un reto importante para los estudiantes, quienes debían navegar entre múltiples sistemas y portales para encontrar la información que necesitaban, además de constituir una barrera de entrada significativa para los estudiantes menos experimentados.

Para los profesores y otros usuarios administrativos, NES constituye una herramienta que facilita su labor como consejeros y mejora su capacidad de toma de decisiones. La plataforma incluye una base de preguntas frecuentes y guías para poder orientar a los estudiantes en dificultades académicas, personales o financieras. Finalmente, para los coordinadores académicos, NES contribuye a descongestionar las solicitudes frecuentes poniendo a disposición herramientas de autogestión que permiten que los estudiantes resuelvan problemas simples por sí mismos, lo cual optimiza el uso del tiempo en las coordinaciones.

### **1.1.2. La necesidad del Perfil del estudiante**

NES representa un avance significativo en la centralización y claridad de los recursos y servicios ofrecidos por la Universidad. Sin embargo, previo a la implementación del Perfil del estudiante, los profesores y administrativos enfrentaban desafíos al intentar ofrecer consejerías personalizadas o tomar decisiones informadas. La información sobre los estudiantes estaba dispersa en múltiples sistemas y formatos, lo que dificultaba tener una visión integral de su contexto académico y socioeconómico de forma oportuna.

El Perfil del estudiante surge como una respuesta directa a esta necesidad. Este módulo se integra dentro de NES para ofrecer una herramienta que:

- Centralice y organice la información relevante sobre los estudiantes.
- Presente esta información de manera intuitiva, clara e inmediatamente accesible, sin sacrificar la exhaustividad.
- Facilite a profesores y administrativos brindar consejerías personalizadas y tomar decisiones basadas en datos.

Con estas características, el Perfil del estudiante no solo mejora la experiencia de los usuarios de NES, sino que también fortalece su capacidad de impacto al abordar de manera directa las problemáticas identificadas por la Vicedecanatura de Asuntos Estudiantiles.

## **1.2. Objetivos**

El objetivo general de este proyecto es construir el Perfil del estudiante.

Más específicamente, el objetivo radica en proporcionar a todo el estudiantado y al profesorado de la Universidad de los Andes una herramienta digital que les permita consultar información relacionada con el desempeño académico actual y pasado, así como con el contexto socioeconómico, de los estudiantes de la Universidad.

### **1.2.1. Objetivos específicos**

El objetivo general enunciado recién se desglosa en los siguientes objetivos específicos:

- Conseguir, mediante una herramienta de software, el acceso a la información académica de los estudiantes de la Universidad de los Andes de manera que pueda interactuar con una aplicación web, idealmente siguiendo los lineamientos de un API REST.
- Diseñar e implementar un módulo de la aplicación web NES que extraiga y exhiba la información académica de cualquier estudiante de la Universidad de los Andes. Realizar la distinción entre la forma en la que se presenta la información para estudiantes de pregrado y estudiantes de posgrado, en particular, distinguir entre los resultados de un mismo estudiante en su pregrado y en su posgrado, teniendo completa trazabilidad de su trayectoria académica cuando sea el caso.
- Conectar el módulo con la navegación y funcionalidades ya existentes en la aplicación web NES, de forma que cada estudiante pueda tener acceso a su información académica (mas no a la de otros estudiantes), cada profesor consejero pueda tener acceso a la información académica de sus estudiantes aconsejados y otros usuarios específicos, determinados por la Vicedecantura, puedan tener acceso a la información académica de los estudiantes que les conciernan.
- Garantizar la integridad y seguridad de la información académica de los estudiantes de la Universidad de los Andes, tanto informáticamente como en su presentación, de forma que se cumpla con la normativa de protección de datos personales y de información académica de la Universidad.

### **1.2.2. Resultados esperados**

El presente proyecto se puede dar por culminado una vez se satisfaga el siguiente conjunto de resultados:

- Existencia de un módulo de la aplicación web NES que permita a cada estudiante de la Universidad de los Andes, tanto de pregrado como de posgrado, consultar información relacionada con su desempeño académico.
- Existencia de un módulo de la aplicación web NES que permita a cada profesor de la Universidad de los Andes consultar información relacionada con el desempeño académico de cada uno de sus estudiantes aconsejados, de forma individual.
- Existencia de un módulo de la aplicación web NES que permita a usuarios específicos, como decanos, directores de programas u otros directivos de la Universidad de los Andes, consultar información relacionada con el desempeño académico de los estudiantes de la Universidad que les conciernan.

### **1.3. Usuarios y requerimientos**

Describe a los usuarios principales del sistema, sus roles y las necesidades específicas identificadas. Aquí también puedes incluir cómo se realizó la recolección y análisis de requerimientos, y un resumen de las historias de usuario que guiaron el diseño del sistema.

### **1.4. Arquitectura general del sistema**

Introduce la arquitectura three-tier (datos, lógica y presentación). Proporciona una visión general de cómo estas capas están integradas y cómo contribuyen al funcionamiento del Perfil del estudiante.

# **Capítulo 2**

## **Capa de datos**

El presente capítulo se ocupa de la capa de datos de la arquitectura del Perfil del estudiante. En particular, describe en detalle el proceso de extracción, transformación y carga al que los datos son sometidos. Adicionalmente, provee una perspectiva, a un más alto nivel de abstracción, de cómo los datos fluyen desde su origen hasta su destino final en el Perfil del estudiante, que resulta útil como contexto y preámbulo de los capítulos subsiguientes.

### **2.1. Fuentes de datos**

En aras de que el Perfil del estudiante satisfaga su expectativa como núcleo de información académica y socioeconómica de los estudiantes de la Universidad de los Andes, es necesario que unifique la información que se encuentra dispersa en múltiples sistemas y formatos. En particular, se espera que el Perfil del estudiante sea capaz de reproducir, como mínimo, la información disponible en los sistemas de información académica de la Universidad, como Banner y Advise, así como información en el sistema de gestión financiera y en el portal de oferta de cursos. La tabla 2.1 detalla las fuentes de datos cuya información debe ser reproducida por el Perfil del estudiante, incluyendo el nombre formal de la fuente y una breve descripción de la información que contienen.

Para el acceso a la información contenida en estas fuentes de datos, la Universidad dispone de un Ecosistema de analítica institucional, cuyo propósito es precisamente facilitar el acceso a la información y la generación de conocimiento a partir de los datos. El Ecosistema tiene entre sus objetivos específicos “Entregar los datos con estándares de calidad y seguridad para suplir las necesidades de la Universidad”, “Hacer el uso de datos para garantizar una toma de decisiones informada” e “Impulsar una cultura de uso de datos en la Universidad” [1]. El uso del Ecosistema evita tener que acceder directamente a las fuentes de datos, lo cual simplifica el proceso de extracción de datos y terceriza la responsabilidad de la consistencia de los mismos, al menos en su origen, a la Universidad. En la subsección 2.2.1 se detalla cómo se realiza la extracción de los datos del Ecosistema.

Como complemento a lo anterior, teniendo en mente la enorme responsabilidad que recae sobre el Ecosistema respecto a la calidad de los datos, lo cual es un aspecto crítico para que el Ecosistema pueda cumplir con sus objetivos y en efecto ser útil y confiable para las diversas dependencias de la Universidad, se realizó el esfuerzo de registrar

**Tabla 2.1***Fuentes de datos utilizadas por el Perfil del estudiante*

Nombre	Nombre Formal	Descripción
Banner	Ellucian Banner	Sistema de planificación de recursos empresariales (ERP) diseñado para instituciones de educación superior. Es capaz de gestionar procesos académicos y administrativos, incluyendo matrículas, registros académicos, finanzas y recursos humanos. [5] Almacena el registro académico completo de cada uno de los estudiantes de la Universidad en todos los niveles académicos, así como algunos de los antecedentes académicos del estudiante.
Advise	Ellucian CRM Advise	Sistema de gestión de relaciones con los estudiantes (CRM) que permite a las instituciones identificar y apoyar proactivamente a estudiantes en riesgo, facilitando intervenciones personalizadas para mejorar su éxito académico. [6]
Oferta de Cursos	Portal de Oferta de Cursos	Plataforma web que permite a los estudiantes consultar los cursos disponibles en cada periodo académico, incluyendo información detallada sobre horarios y profesores. [3]
Sistema de Gestión Financiera		Representación genérica de los sistemas contables y financieros de la universidad, en los que se maneja la información relacionada con becas, créditos educativos y pagos realizados por los estudiantes, al igual que información socioeconómica del estudiante relevante para la asignación de auxilios financieros.

en el apéndice B todos los problemas de calidad de los datos que se han encontrado en el Ecosistema y cómo se han abordado.

## 2.2. Proceso de extracción, transformación y carga

La recuperación y el procesamiento de los datos se realizaron mediante un *pipeline* de analítica implementado en cuadernos de Jupyter en Python, en colaboración con los estudiantes Santiago Martínez Novoa y Manuel Felipe Porras Tascón \*. El pipeline de analítica realiza las tres etapas clásicas de un proceso de ETL: extracción, transformación y carga de los datos. A cada una de estas etapas se le dedica una subsección en las siguientes líneas.

### 2.2.1. Extracción de los datos

El proceso de extracción, a alto nivel, consiste en tomar la información relevante del Ecosistema de analítica institucional.

\*Mi más sincero agradecimiento a ambos: a Manuel, en aquel momento estudiante de la Maestría en Ingeniería de la Información, por su valiosa labor al crear y trabajar inicialmente en el cuaderno; y a Santiago, entonces estudiante de Ingeniería de Sistemas y Computación, por tomar la posta, mejorar y mantener este trabajo. Sin ellos, este proyecto no habría sido posible.

## 15 • Desarrollo del Perfil del estudiante

La información del Ecosistema se encuentra almacenada en Azure Blob Storage, que es un servicio de almacenamiento de objetos en la nube de Microsoft Azure. Está distribuida en diversas cuentas de almacenamiento, cada una de las cuales contiene archivos en formato PARQUET y XLSX. Para el Perfil del estudiante, se hizo uso de diez de esos archivos, distribuidos en dos cuentas de almacenamiento. La figura 2.1 muestra la estructura de directorios y archivos en el Blob Storage que contienen la información relevante para el Perfil del estudiante.

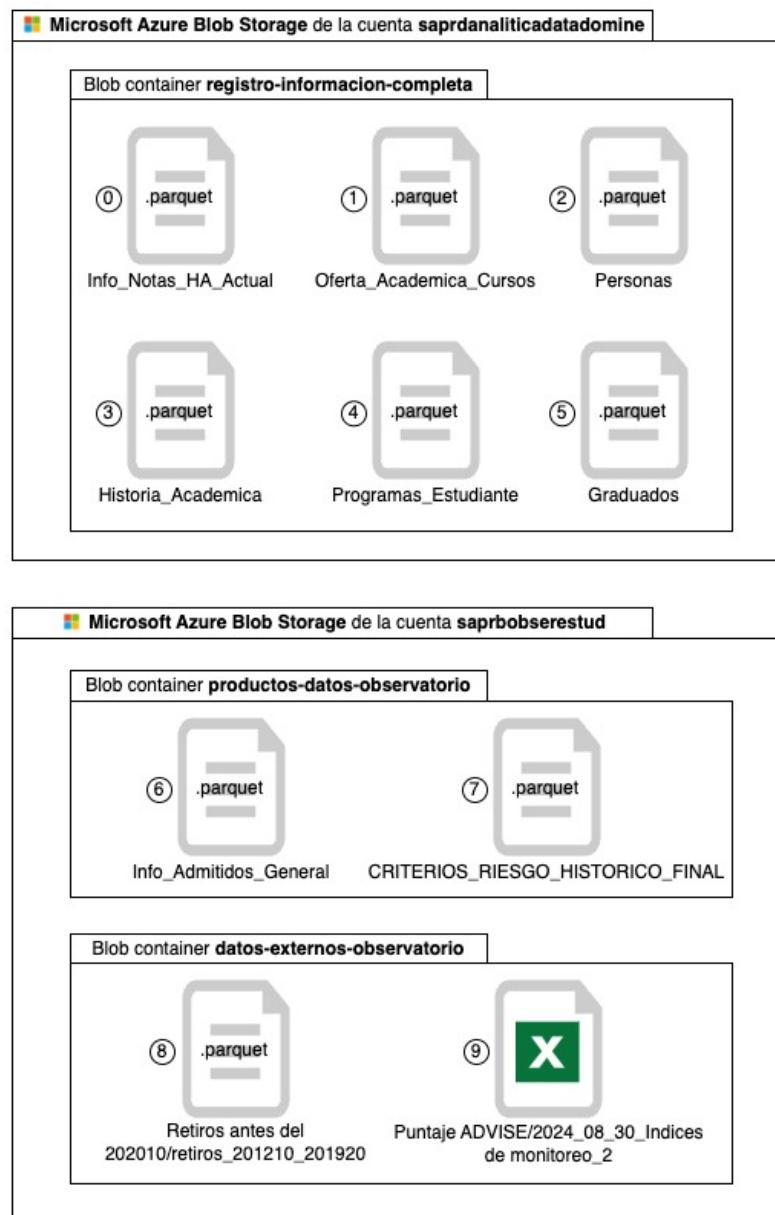


Figura 2.1: Archivos fuente y estructura de directorios en Azure Blob Storage

Cada uno de los archivos presentados en la figura 2.1 se carga en el cuaderno de Jupyter y se extrae la información relevante para el Perfil del estudiante. La tabla 2.2 detalla el orden en el que se realiza cada extracción, describe la información que se extrae y especifica el archivo fuente del cual se obtiene, empleando la numeración de los archivos

definida en la figura 2.1.

**Tabla 2.2**  
*Extracción de datos*

Orden	Información	Descripción	Archivos fuente
1	Histórico académico	Información de las notas obtenidas por los estudiantes en cada una de las asignaturas que han cursado.	0
2	Oferta académica	Información de las asignaturas que se ofertan en cada uno de los semestres académicos.	1
3	Estudiantes	Información básica de los estudiantes.	2, 3, 6
4	Programas académicos	Información de los programas académicos en los que se encuentran inscritos los estudiantes.	4
5	Información adicional de retiros	Información sobre los retiros de los estudiantes para periodos anteriores al 2019-20, que no se encuentra en el Histórico académico.	8
6	Graduados	Información de los estudiantes que se han graduado.	5
7	Criterios de riesgo	Información de los criterios de riesgo que se utilizan para identificar a los estudiantes en riesgo académico.	7
8	Advise	Información de los estudiantes tomada de la plataforma Advise.	9

### 2.2.2. Transformación de los datos

### 2.2.3. Carga de los datos

## 2.3. Flujo de los datos

Esta última sección de la capa de datos se ocupa de ofrecer una visión de alto nivel del flujo que los datos atraviesan desde su origen hasta su destino final en el perfil del estudiante. Es sensato presentar este flujo en este punto del documento, pues ya se han detallado las fuentes de datos y el proceso de ETL, por lo que se da un abrebotas del paso de los datos por el API REST, que es el siguiente componente de la arquitectura del Perfil del estudiante, y del consumo de los datos por parte del frontend. Por ende, esta sección contextualiza al lector y opera como preámbulo a los próximos capítulos.

La figura 2.2 muestra el flujo de los datos mencionado. Los datos se extraen del Ecosistema de analítica institucional, se transforman y se cargan en el API REST, que a su vez los almacena en una base de datos PostgreSQL. El API REST expone una interfaz de programación que permite a los clientes, como el frontend del Perfil del estudiante, consumir los datos de manera estructurada y segura.



Figura 2.2: Flujo de los datos, desde su origen en el Ecosistema de analítica institucional hasta su visualización en el frontend del Perfil del estudiante



# **Capítulo 3**

## **Capa de lógica**

Este capítulo se ocupa de describir la capa de lógica del sistema, que se encarga de procesar los datos extraídos y exponerlos como recursos accesibles mediante una API REST. Primeramente, inicia por justificar la construcción de un API REST como interfaz de la capa de lógica. Posteriormente, se detalla el diseño del API, describiendo la estructura de los datos y cómo se representan como objetos, en particular como clases, agnósticas de la fuente de los datos y de la implementación del API. Tras eso, se dedican algunas secciones a la implementación del API, abordando los aspectos técnicos relacionados con el desarrollo del API, incluyendo el lenguaje de programación, el framework, la estructura de directorios de la implementación, así como las rutas y recursos expuestos por el API. A eso le sigue una sección dedicada al despliegue del API, tratando todos los detalles técnicos correspondientes. Finalmente, se dedica una sección a la calidad del código, detallando las herramientas y prácticas empleadas para garantizar la calidad del código.

### **3.1. ¿Por qué un API REST?**

Antes de entrar en detalles sobre el diseño del API, es importante justificar la elección de un API REST como interfaz de la capa de lógica del sistema. En primer lugar, un API REST es una interfaz de programación de aplicaciones que sigue los principios del estilo arquitectónico REST. Este estilo se basa en la transferencia de representaciones de recursos, que son identificados por URIs, y la manipulación de estos recursos mediante métodos estándar de HTTP. En particular, un API REST es una interfaz que permite la interoperabilidad de aplicaciones heterogéneas, permitiendo a un programa acceder a las funciones y datos de otro. En el caso de este proyecto, un API REST es la interfaz ideal para exponer los datos procesados por la capa de lógica, ya que permite a cualquier aplicación, como el frontend del perfil del estudiante, acceder a estos datos de manera sencilla y eficiente.

En los últimos años, los API REST han ganado popularidad en el desarrollo de aplicaciones web, ya que son fáciles de entender, escalables y flexibles. Además, un API REST es independiente de la implementación subyacente, lo que significa que el frontend del perfil del estudiante no necesita conocer los detalles de cómo se procesan los datos o cómo se almacenan, sino que solo necesita conocer la interfaz del API. Por último, un API

REST es fácil de probar y depurar, ya que se basa en estándares abiertos y bien conocidos, como HTTP y JSON.

Todo esto hace que un API REST sea la elección ideal para la capa de lógica de este sistema, ya que permite exponer los datos procesados de manera sencilla y eficiente, independiente de la implementación subyacente, y fácil de probar y depurar.

## 3.2. Diseño del API

El diseño del API se divide en dos partes. La primera parte está en el marco de la programación orientada a objetos, y se encarga de describir cuál es la estructura de clases del API: qué clases existen, cómo se relacionan entre sí y qué atributos tienen. La segunda parte se enfoca en la interfaz del API, es decir, en cómo se exponen los datos procesados como recursos accesibles mediante una API REST. Eso incluye cuáles son los recursos que se exponen, cómo se representan, en qué formato se devuelven al cliente y cuáles son los endpoints que brinda el API para acceder a estos recursos.

Es importante señalar que el diseño del API es completamente independiente de dos factores: por un lado, de la fuente de los datos, es decir, de cómo se extraen los datos de las fuentes originales; por otro lado, de la implementación del API, es decir, de qué lenguaje y cuál framework se utiliza para escribir el código del API. En este sentido, esta sección es independiente del capítulo anterior, que estudia la capa de datos, y de la sección siguiente, que explica la implementación del API.

### 3.2.1. Estructura de clases del API

Para modelar la información del Perfil del estudiante como objetos se optó por realizar un diagrama de clases conforme al estándar UML. Esto presenta dos ventajas. Por un lado, permite visualizar de manera clara y concisa de la estructura de clases del API. Por otro lado, es un artefacto bien conocido y ampliamente utilizado en el desarrollo de software, que no resultará ajeno a cualquier desarrollador que deba trabajar con el API en el futuro.

#### Diagrama de clases

Se realizaron varias iteraciones en la construcción del diagrama de clases, teniendo en cuenta las necesidades del frontend del perfil del estudiante y las restricciones de la capa de datos. La figura 3.1 muestra el diagrama de clases final del API.

El diagrama de clases está escrito en inglés, siguiendo las convenciones de UML. Cada clase tiene un nombre, que es un sustantivo en singular, y una serie de atributos. Los atributos están escritos en el formato `nombre : tipo`, donde `nombre` es el nombre del atributo y `tipo` es el tipo de dato del atributo. Por conveniencia, los tipos de datos empleados son los mismos que se utilizan en el lenguaje de programación Python, que es el lenguaje en el que se implementa el API, pero bien podrían usarse los tipos de datos de cualquier otro lenguaje de programación orientado a objetos sin afectar semánticamente el diagrama de clases, por lo que mantiene su independencia de la implementación.

## 21 • Desarrollo del Perfil del estudiante

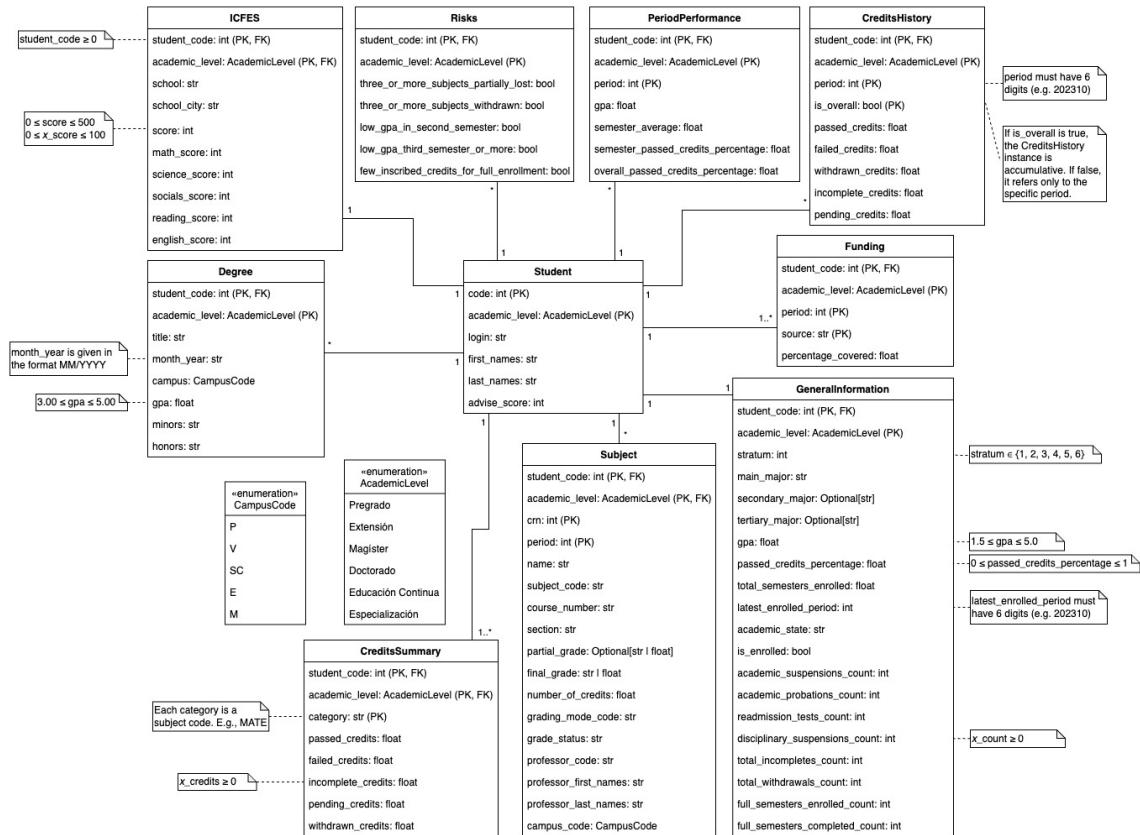


Figura 3.1: Diagrama de clases del API

### Clases del API

La estructura de clases del API se basa en el principio de responsabilidad única, que establece que cada clase debe tener una sola responsabilidad y representar un solo concepto en el dominio del problema. Las clases propuestas en el diagrama se presentan alfabetizadas en la tabla 3.1, con su nombre original en inglés, la traducción al español y una breve descripción de su responsabilidad.

#### Tabla 3.1

*Clases del API y sus responsabilidades*

Nombre original	Nombre traducido	Responsabilidad
CreditsHistory	Historial de créditos	Rastrea el estado global de los créditos del estudiante, incluyendo: aprobados, reprobados, retirados, pendientes e incompletos, tanto para períodos específicos, por ejemplo un semestre particular, como en general y hasta la fecha.

Continúa en la página siguiente

Tabla 3.1: Clases del API y sus responsabilidades (Continuación)

<b>Nombre original</b>	<b>Nombre traducido</b>	<b>Responsabilidad</b>
CreditsSummary	Resumen de créditos	Proporciona un desglose categórico de los créditos aprobados, reprobados, retirados, pendientes e incompletos, para una categoría específica de materias. Algunos ejemplos de categoría pueden ser: las materias de la carrera principal, las materias de matemáticas, las materias de física, entre otras.
Degree	Título	Almacena información sobre los títulos académicos obtenidos por el estudiante, como el nombre del título, el campus, el promedio y menciones especiales, como títulos con honores.
Funding	Financiamiento	Detalla las fuentes de financiamiento del estudiante, que pueden incluir becas y créditos educativos, con el porcentaje cubierto por cada fuente.
GeneralInformation	Información general	Contiene datos generales del estudiante, como su carrera principal, estrato, GPA actual, entre otros indicadores relevantes.
ICFES	ICFES	Almacena los puntajes del estudiante en las áreas evaluadas por el examen ICFES, como matemáticas, lectura y ciencias.
PeriodPerformance	Rendimiento por periodo	Guarda información del desempeño académico del estudiante en un periodo específico, incluyendo el promedio del semestre y el porcentaje de créditos aprobados.
Risks	Riesgos	Identifica indicadores de riesgo académico, como bajo GPA en los primeros semestres o retiro de múltiples materias.

Continúa en la página siguiente

Tabla 3.1: Clases del API y sus responsabilidades (Continuación)

Nombre original	Nombre traducido	Responsabilidad
Student	Estudiante	Representa a un estudiante de la universidad cuando se encontraba en un nivel académico específico. Almacena la información esencial para la identificación del estudiante, como nombres, apellidos, código y usuario (login, que corresponde al nombre de usuario en el correo electrónico institucional).
Subject	Materia	Representa una materia específica cursada por el estudiante, incluyendo el nombre, el código, los créditos, el profesor, la nota obtenida y el estado de la materia.

Como complemento del diagrama de clases y de la tabla 3.1, vale la pena ahondar en algunas de las decisiones de diseño que se tomaron en la creación de las clases con base en el dominio del negocio:

- La clase `Student` es la clase principal del modelo. Salta a la vista que no representa a un estudiante por completo, sino que representa a un estudiante en un nivel académico específico. Esto se debe a que resulta conveniente tratar de forma distinta al mismo estudiante en diferentes niveles académicos, ya que su contexto puede haber cambiado significativamente: el título al que aspira es distinto, las materias que cursa son distintas, su desempeño académico puede haber mejorado o empeorado, e incluso su situación socioeconómica puede haber cambiado. La misma persona en pregrado no puede ser tratada de la misma forma que en posgrado.
- La clase `CreditsSummary` es una clase auxiliar que se encarga de proporcionar un desglose categórico de los créditos aprobados, reprobados, retirados, pendientes e incompletos, para una categoría específica de materias. Naturalmente, no para todos los estudiantes se incluyen las mismas categorías. Es decir, para un estudiante de ingeniería, Categorías relevantes pueden ser las materias Específicas de su rama del ingeniería, las materias de matemática y las materias de física. Sin embargo, para un estudiante de arte, incluso si el estudiante ha optado por cursar materias de matemática y física, esas categorías pueden no ser relevantes para su desempeño académico. Esto es un matiz que es difícil representar en un diagrama de clases, pues corresponde a especificidades del dominio del problema, por lo cual se aclara en este apartado.
- Puede que el lector se haya percatado que la clase `Risks` está nombrada con un sustantivo en plural, lo cual aparentemente transgrede las convenciones de construcción de diagramas de clase. Eso no es un error, sino una decisión consciente de diseño. Se debe a que la clase `Risks` no representa un riesgo académico en

particular, sino que cada instancia de la clase corresponde a un conjunto de riesgos académicos, entre ellos bajo GPA y retiro de múltiples materias. Por lo tanto, el nombre en plural es más adecuado que el nombre en singular para representar la naturaleza de la clase.

### Relaciones entre las clases

Las relaciones entre las clases se representan mediante líneas que unen las clases. Las relaciones en este diagrama son muy simples, lo cual fue una elección deliberada en pos de la simplicidad y la claridad. Cada línea representa una asociación entre dos clases y en sus extremos cuenta con números que indican la multiplicidad de la asociación, es decir, cuántos objetos de una clase están asociados con cuántos objetos de la otra clase. Un asterisco indica que hay muchos objetos asociados, mientras que un número indica la cantidad exacta de objetos asociados.

Como ejemplo, considérese la asociación entre la clase `Student` que representa a un estudiante y la clase `Degree` que representa un grado. Un estudiante puede estar asociado con algún grado, en caso de ya haberse graduado, con varios, en caso de haber obtenido varios grados, o con ninguno, en caso de no haberse graduado. Sin embargo, cada grado corresponde únicamente a un estudiante. Por ende, la asociación entre la clase `Student` y la clase `Degree` es de uno a muchos, representada por la línea que une ambas clases con un 1 en el extremo de la clase `Student` y un asterisco en el extremo de la clase `Degree`.

La tabla 3.2 detalla las relaciones entre las clases del diagrama de clases, con una breve descripción de la relación y la multiplicidad de la asociación. Se organizan en orden de aparición, de arriba a abajo y de izquierda a derecha, en el diagrama de clases.

**Tabla 3.2**

*Relaciones entre las clases del diagrama de clases*

Clase 1	Clase 2	Descripción y Multiplicidad
<code>Student</code>	<code>GeneralInformation</code>	Un estudiante está asociado con una única instancia de información general. Multiplicidad: 1 a 1.
<code>Student</code>	<code>ICFES</code>	Un estudiante puede tener una única instancia de resultados del ICFES. Multiplicidad: 1 a 1.
<code>Student</code>	<code>Risks</code>	Un estudiante tiene una única instancia de indicadores de riesgos académicos. Multiplicidad: 1 a 1.
<code>Student</code>	<code>PeriodPerformance</code>	Un estudiante puede estar asociado con muchos períodos de desempeño académico. Multiplicidad: 1 a muchos.

Continúa en la página siguiente

Tabla 3.2: Relaciones entre las clases del diagrama de clases (Continuación)

<b>Clase 1</b>	<b>Clase 2</b>	<b>Descripción y Multiplicidad</b>
Student	CreditsHistory	Un estudiante puede tener muchos historiales de créditos, cada uno relacionado con un periodo específico o de manera acumulativa. Multiplicidad: 1 a muchos.
Student	CreditsSummary	Un estudiante tiene un resumen de créditos categorizado. Multiplicidad: 1 a 1.
Student	Subject	Un estudiante está asociado con muchas materias, pero cada materia pertenece a un único estudiante. Multiplicidad: 1 a muchos.
Student	Degrees	Un estudiante puede haber obtenido múltiples títulos. Multiplicidad: 1 a muchos.
Student	Funding	Un estudiante puede tener múltiples fuentes de financiamiento. Multiplicidad: 1 a muchos.
Subject	CreditsSummary	Cada materia está categorizada en un resumen de créditos. Multiplicidad: 1 a 1.

### Reglas de negocio en el diagrama de clases

En el diagrama de clases se pueden evidenciar anotaciones en algunos atributos. Varias de estas anotaciones corresponden a restricciones de integridad que se deben cumplir en el modelo de datos, que en últimas representan reglas del dominio del problema. A esto, usualmente se le denomina *reglas de negocio* y son restricciones que se deben cumplir en el modelo de datos para que reflejen la realidad de manera fiel.

La tabla 3.3 detalla las anotaciones en el diagrama de clases que tienen esa función, junto con su significado en términos del dominio del problema, es decir, en el contexto de la Universidad. Se organizan en orden de aparición en el diagrama de clases, de arriba a abajo y de izquierda a derecha.

**Tabla 3.3***Anotaciones del diagrama de clases y su significado*

<b>Anotación</b>	<b>Significado</b>	<b>Justificación</b>
student_code $\geq 0$	El código del estudiante debe ser mayor o igual a 0	El código del estudiante debe ser un entero positivo que identifica únicamente a cada estudiante. No se permiten códigos negativos.
$0 \leq score \leq 500$	El puntaje del ICFES (formalmente, la prueba Saber 11) debe estar entre 0 y 500	El puntaje máximo en la prueba Saber 11 es 500, y el mínimo es 0.
$0 \leq x_score \leq 100$	Los puntajes de las áreas evaluadas por el ICFES deben estar entre 0 y 100	Los puntajes de las áreas evaluadas por el ICFES se normalizan a una escala de 0 a 100.
month_year is given in the format MM/YYYY	La fecha de graduación, que se representa como un mes y un año, debe tener el formato MM/YYYY	La fecha de graduación se codifica en el formato de mes y año para garantizar consistencia en los registros.
$3.00 \leq gpa \leq 5.0$	El GPA debe estar entre 3.0 y 5.0	En el sistema de la Universidad, el GPA mínimo para graduarse es 3.0 y el máximo posible es 5.0.
Each category is a subject code. E.g., MATE	Cada categoría de un Resumen de créditos es un código de materia. Por ejemplo, MATE representa las materias de matemáticas	Las categorías de un Resumen de créditos se representan con códigos de materia. Esta decisión de diseño se explica en detalle arriba, en la descripción de la clase CreditsSummary.
$x_credits \geq 0$	Los créditos aprobados, reprobados, retirados, pendientes e incompletos deben ser mayores o iguales a 0	No tiene sentido registrar un número negativo de créditos en ningún contexto académico.

Continúa en la página siguiente

Tabla 3.3: Anotaciones del diagrama de clases y su significado (Continuación)

Anotación	Significado	Justificación
period must have 6 digits (e.g. 202310)	El periodo académico debe tener 6 dígitos	Los periodos académicos se codifican en el formato de año seguido por el indicador del tipo de periodo. Usualmente se usa un guión entre el año y el tipo de periodo, e.g., 2023-10, pero en este caso resulta más conveniente usar un número entero de 6 dígitos.
stratum ∈ $\{1, 2, 3,$ $4, 5, 6\}$	El estrato socioeconómico debe estar entre 1 y 6	Los estratos socioeconómicos en Colombia están normados en este rango por ley. Si el estudiante no reside en Colombia, no se registra su estrato.
$1.5 \leq gpa \leq 5.0$	El GPA debe estar entre 1.5 y 5.0	En el sistema de la Universidad, el GPA mínimo posible es 1.5, y el máximo posible es 5.0.
$0 \leq$ passed_- credits_- percentage $\leq 1$	El porcentaje de créditos aprobados debe estar entre 0 y 1	Todo porcentaje es un número entre 0 y 1, inclusive.
latest_- enrolled_- period must have 6 digits (e.g. 202310)	El periodo académico más reciente en el que el estudiante estuvo inscrito debe tener 6 dígitos	Esta explicación es análoga a la de la anotación period must have 6 digits (e.g. 202310).

### 3.3. Implementación del API

Esta sección se ocupa de la implementación del API REST. En particular, se centra en los aspectos técnicos relacionados con el desarrollo del API, incluyendo el lenguaje de programación, el framework y las herramientas asociadas a la implementación.

#### 3.3.1. Lenguaje y framework

El API REST está implementado en Python, utilizando el framework FastAPI. Se dedica un apartado a justificar la elección de cada uno de ellos.

## Python

Un breve pero excelente resumen de las características del lenguaje se encuentra en la documentación oficial:

Python es un lenguaje interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipado dinámico, tipos de datos dinámicos de muy alto nivel y clases. Además de la programación orientada a objetos, admite múltiples paradigmas de programación, como la programación procedural y funcional. Python combina una potencia notable con una sintaxis muy clara. Ofrece interfaces a numerosas llamadas y bibliotecas del sistema, así como a distintos sistemas de ventanas, y es extensible en C o C++. También puede utilizarse como lenguaje de extensión para aplicaciones que requieran una interfaz programable. Finalmente, Python es portátil: se ejecuta en diversas variantes de Unix, incluyendo Linux y macOS, así como en Windows. [7]

Existe una infinidad de argumentos técnicos para usar Python en cualquier proyecto de software, que están bien documentadas en muchísimos recursos; por ejemplo, [8] tiene en su primer capítulo una sección dedicada exclusivamente a las razones por las cuales millones de desarrolladores alrededor del mundo usan el lenguaje. Por ende, no ahondaré en razones técnicas para elegir Python.

No obstante, cabe mencionar una que existe una justificación contextual para elegir Python para la implementación del API REST, la cual es independiente de sus virtudes técnicas y responde a preocupaciones por la mantenibilidad del proyecto. En concreto, Python es el principal lenguaje de programación que se enseña en la Universidad. Bajo la estructura curricular actual, en todos los programas de las facultades de Ingeniería, Ciencias y Economía, el curso ISIS-2021: Introducción a la Programación es obligatorio y constituye el primer curso de programación.

Este curso, que “introduce conceptos básicos de programación, utilizados para resolver problemas con un programa de computadora” [2], se enseña enteramente en Python. El siguiente curso de programación, ISIS-1225: Estructuras de Datos y Algoritmos, que es obligatorio para los programas de Matemática e Ingeniería de Sistemas y Computación, también se enseña en Python. Por lo tanto, la mayoría de los estudiantes de la Universidad que cursan programas del área de STEM tienen experiencia previa en Python y varios no tienen experiencia en ningún otro lenguaje de programación. Esta restricción contextual implicó que la elección de Python para la implementación del API REST fue prácticamente obligatoria en aras de garantizar la sostenibilidad del proyecto a largo plazo.

## FastAPI

“FastAPI es un framework moderno, rápido (de alto rendimiento) y fácil de usar para construir APIs con Python basado en las anotaciones de tipo estándar de Python” [9].

La principal justificación para la elección de FastAPI va de la mano con la alta prioridad que se le ha dado a la mantenibilidad del proyecto. Se buscaba un framework que tuviera una curva de aprendizaje corta para desarrolladores familiarizados con

Python, que fuera fácil de mantener y extender, y que facilitara construir buena documentación. Sorprendentemente, FastAPI cumple con todos estos requisitos, sin comprometer su rendimiento. De hecho, FastAPI es uno de los frameworks más rápidos para construir APIs con Python, superando a otros frameworks populares como Flask y Django [11].

Respecto a la facilidad de aprendizaje, la sintaxis FastAPI está basada en las anotaciones de tipo estándar de Python. Esto permite que para cualquier desarrollador que conozca Python, sea sencillo aprender FastAPI, pues no requiere aprender sintaxis adicional. [10]. Respecto a la mantenibilidad del framework, se destaca su integración con herramientas de validación de datos, como Pydantic, que permite definir modelos de datos y validar automáticamente los datos entrantes. Esta funcionalidad facilita el mantenimiento del código en tanto que cualquier cambio en los modelos de datos se refleja inmediatamente en la validación de los datos entrantes [10].

Por último, se destaca la integración con herramientas de documentación automática. FastAPI genera automáticamente documentación interactiva de la API REST, que se puede acceder a través de dos rutas: /docs y /redoc. La primera ruta genera una interfaz interactiva basada en Swagger, mientras que la segunda ruta genera una interfaz interactiva basada en Redoc [10]. Ambas interfaces permiten explorar los recursos del API REST, probarlos y ver la documentación generada automáticamente, lo cual simplifica enormemente la tarea de mantener el API REST, ya que cualquier cambio en el código se ve reflejado en la documentación.

Las razones anteriores hacen de FastAPI una excelente elección para la implementación de cualquier API, y más para este proyecto en particular, considerando las restricciones contextuales enunciadas en la sección anterior.

### 3.3.2. Estructura de directorios

FastAPI es un framework no opinionado, es decir, que permite al desarrollador elegir cómo estructurar su proyecto. Esto significa que hay una infinidad de formas de estructurar un proyecto de FastAPI. La siguiente fue la estructura de directorios elegida para este proyecto:

- README.md
- app.log
- db/
- .env
- dev.log
- requirements.txt
- sonar-project.properties
- src/
  - \_\_init\_\_.py

- main.py
- authentication/
- azure\_interface/
- config/
- constants/
- files/
- models/
- persist\_from\_blobs/
- routers/
- schemas/
- services/
- utils/
- tests/

Nótese que no se reflejan todos los archivos y directorios del proyecto, sino únicamente aquellos que son relevantes para la estructura de código del API REST. La tabla 3.4 detalla la función de cada directorio y archivo en el directorio raíz del proyecto.

**Tabla 3.4**

*Estructura de archivos y directorios del proyecto*

Directorio/Archivo	Descripción
README .md	Archivo con documentación fundamental del proyecto, en concreto una breve descripción e instrucciones de ejecución del proyecto en un ambiente local. Está escrito en Markdown, que es un lenguaje de marcado ligero, es decir, que permite dar formato al texto de forma sencilla.
app.log	Archivo de registro ( <i>log</i> ) generado automáticamente en el que se almacena información como errores, advertencias y notificaciones generadas durante la ejecución de la aplicación. El archivo app.log corresponde al entorno de producción; en el entorno de desarrollo se genera el archivo dev.log, cuyo propósito es análogo. Se ahonda en el uso de este archivo en la sección 3.5.1
db/	Directorio que almacena scripts relacionados con respaldos y restauración de la base de datos, además de otros archivos necesarios para la gestión de datos persistentes.

Continúa en la página siguiente

Tabla 3.4: Estructura de archivos y directorios del proyecto (Continuación)

Directorio/Archivo	Descripción
.env	Archivo de configuración que almacena variables de entorno. Estas variables permiten definir valores sensibles o específicos del entorno, como: credenciales, configuraciones de la base de datos, claves o modos de ejecución (desarrollo, pruebas y producción), sin incluirlos directamente en el código fuente. El archivo no se incluye en el control de versiones por contener información sensible, pero en el README se incluyen instrucciones para construirlo, mientras se cuente con las credenciales necesarias.
sonar-project.properties	Archivo de configuración para SonarQube. El uso del archivo y de SonarQube se explica en la sección 3.4.
src/	Directorio principal que contiene el código fuente del proyecto. Su nombre es una abreviación convencional para <i>source</i> , que en inglés significa <i>fuente</i> .
tests/	Directorio que contiene las pruebas unitarias del proyecto. Se ahonda en las pruebas unitarias en la sección 3.4.

Como complemento a la tabla 3.4, la tabla 3.5 especifica la función de cada directorio y archivo dentro del directorio `src/`.

### Tabla 3.5

Estructura de archivos y directorios dentro de `src/`

Directorio/Archivo	Descripción
<code>__init__.py</code>	Archivo vacío cuyo nombre indica que el directorio es un paquete en Python.
<code>main.py</code>	Punto de entrada de la aplicación. Contiene la configuración del objeto <code>FastAPI</code> , que es la instancia principal de la aplicación, la importación e inicialización de las rutas y la configuración de los <i>middlewares</i> .
<code>authentication/</code>	Módulo que gestiona la autenticación y autorización de usuarios en el proyecto.
<code>azure-interface/</code>	Módulo que contiene la lógica para interactuar con servicios de Azure.
<code>config/</code>	Módulo que almacena configuraciones del proyecto y de su interacción con la base de datos.
<code>constants/</code>	Módulo que define constantes compartidas utilizadas en el proyecto.

Continúa en la página siguiente

Tabla 3.5: Estructura de archivos y directorios dentro de `src/` (Continuación)

Directorio/Archivo	Descripción
<code>files/</code>	Directorio que almacena archivos estáticos o temporales necesarios para el proyecto.
<code>models/</code>	Módulo en el que se definen las clases que representan las tablas de la base de datos, apalancándose de SQLAlchemy, que actúa como ORM.
<code>persist_from_blobs/</code>	Módulo encargado de la persistencia de datos provenientes de BLOBs o almacenamiento de objetos.
<code>routers/</code>	Módulo que define las rutas del API REST, organizando los puntos de entrada según su funcionalidad.
<code>schemas/</code>	Módulo que define los esquemas de datos utilizados en el API REST, como las estructuras para solicitudes y respuestas.
<code>services/</code>	Módulo que implementa las operaciones que es posible realizar sobre los recursos del API con base en la lógica de negocio del proyecto.
<code>utils/</code>	Módulo que agrupa funciones auxiliares utilizadas en diferentes partes del proyecto.

Vale la pena ahondar en cuatro de los directorios mencionados arriba, que constituyen el esqueleto de cualquier API: `models`, que representa el modelo de datos del API; `schemas`, que define los esquemas de datos utilizados en las solicitudes y respuestas del API; `services`, que contiene la lógica de negocio del proyecto; y `routers`, que define las rutas del API REST. Se dedica un apartado a cada uno de ellos.

## Modelos de datos

El módulo `models` contiene las clases que representan las tablas de la base de datos. En particular, cada archivo de Python en el directorio `models` define una clase que abstrae la estructura de una de las tablas de la base de datos relacional. La mayoría de los atributos de la clase corresponden a las columnas de la tabla y algunos atributos especiales definen las relaciones entre las tablas y las restricciones de integridad. Por ejemplo, la clase `Student` en el archivo `models/student.py` plasma la tabla `students` en la base de datos.

Esta abstracción, en la que el modelo de datos relacional se expresa como un modelo de objetos, es una tarea común en el desarrollo de software en lenguajes orientados a objetos y convencionalmente se realiza siguiendo el patrón ORM (*Object-Relational Mapper*). Dicho patrón permite representar objetos como tablas en una base de datos relacional y viceversa, a pesar de la diferencia en las estructuras orientada a objetos y relacional (problema conocido como *impedancia de mapeo*, en inglés *object-relational impedance mismatch*). En este proyecto, la librería SQLAlchemy encapsula la lógica de mapeo objeto-relacional, permitiendo que las clases de Python se mapeen directamente a tablas de la base de datos relacional. El lector puede ser familiar con otros ORMs, como JPA (*Java Persistence API*) en Java o Entity Framework en C#, que cumplen una función

análoga.

### Esquemas de datos

El módulo `schemas` define las clases que corresponden a los esquemas de datos utilizados en las solicitudes y respuestas del API. Estas clases están implementadas utilizando Pydantic, una librería que provee validación de datos y la generación automática de documentación para los modelos. Todas las clases relacionadas a una entidad específica se definen en un solo archivo, de manera que en el archivo `schemas/student.py` se definen las clases que representan los esquemas de datos de la entidad `Student`.

En general, los esquemas de datos son una abstracción que permite definir la estructura de los datos que se envían y reciben en el API, y que facilita la validación de los datos y la generación de documentación. Para lectores familiarizados con otros lenguajes o frameworks de desarrollo web, esta capa del API es análoga a los *serializers* de Django en Python, los DTOs (Data Transfer Objects) de Spring en Java, las clases de contratos en ASP.NET en C# y las interfaces o tipos definidos en NestJS en TypeScript, por mencionar algunos ejemplos.

Cada esquema se compone de múltiples clases, cada una de las cuales refleja el propósito de la operación para la que se utiliza. Se escriben cinco clases, cada una con un sufijo que indica su propósito: la clase base (con sufijo `Base`, de forma que para una entidad `Entity` la clase base se llamaría `EntityBase`), cuyo propósito es declarar todos los atributos comunes a todas las clases en pos de minimizar la duplicación de código; la clase para creación de datos (`EntityCreate`), que consta de los atributos necesarios para crear un objeto; la clase para actualización de datos (`EntityUpdate`), que contiene los atributos necesarios para actualizar un objeto; y la clase para lectura de datos (`EntityResponse`), que consta de los atributos mínimos que se devuelven en la respuesta del API. Algunos objetos tienen además otra clase de respuesta extendida, con prefijo `ExtendedResponse`, que contiene atributos adicionales que no son necesarios en todas las respuestas, pero que pueden ser útiles en ciertos contextos. Cada clase contiene configuraciones adicionales que proporcionan ejemplos representativos de los datos, que se ven reflejados automáticamente en la documentación del API.

Esta estructura modular y estándar para todos los esquemas de datos facilita la reutilización del código y mantiene la consistencia dentro del proyecto. Incluso si los atributos necesarios para crear una clase y los necesarios para actualizarla son los mismos, se definen en clases separadas para mantener la coherencia y facilitar la extensión en el futuro. Al mantener una separación tan definida entre las estructuras de datos utilizadas en las peticiones y respuestas del API, y las utilizadas internamente (como los modelos de la base de datos), se facilita el mantenimiento del proyecto, se evitan dependencias innecesarias y se reduce el riesgo de sobreexposición, es decir, de entregar más información de la necesaria en las respuestas del API. Eso último es especialmente importante en un proyecto que maneja datos sensibles, como el presente.

### Lógica de negocio

El módulo `services` contiene las funciones que implementan la lógica de negocio del proyecto. Cada archivo en este módulo encapsula todas las operaciones relacionadas con una clase específica. Por ejemplo, el archivo `services/student.py` contiene todas las operaciones que se pueden realizar sobre la entidad `Student`.

Al interior de cada servicio existen, como mínimo, las funciones necesarias para realizar todas las operaciones CRUD (crear, leer, actualizar y eliminar, del inglés *Create, Read, Update, Delete*) sobre la entidad correspondiente, y en algunos casos, funciones adicionales que implementan operaciones más complejas. Cada operación se organiza claramente y se nombra con un prefijo que indica su responsabilidad: las funciones para creación (prefijadas con `create`, de manera que para una entidad `Entity` la función se llamaría `create_entity`) reciben datos validados y los almacenan en la base de datos; las funciones para lectura (`get_entity_by_id`, `get_all_entities`) realizan consultas específicas o generales sobre los modelos; las funciones para actualización (`update_entity`) modifican registros existentes; y las funciones para eliminación (`delete_entity`) eliminan los registros de la base de datos de manera controlada. Esta estructura estándar asegura que las operaciones básicas son consistentes en todas las entidades.

En cada función, el servicio interactúa tanto con los modelos como con los esquemas de datos. Los esquemas de datos se utilizan para validar los argumentos que cada función recibe. Se mencionó en la subsección 3.3.2 que en cada esquema de datos se definen clases para creación (`Create`) y actualización (`Update`); estas clases se utilizan para validar los datos que ingresan a las funciones de creación (`create_entity`) y actualización (`update_entity`) de los servicios, respectivamente. Por su parte, los modelos cumplen su función como abstracción de la base de datos ofreciendo una interfaz orientada a objetos para interactuar con las tablas, permitiendo que cada función realice las operaciones correspondientes sobre los registros de la base de datos. Por ejemplo, la función `create_student`, cuya responsabilidad es crear un estudiante, recibe un objeto de tipo `CreateStudent`, cuya estructura es validada automáticamente por Pydantic, y lo utiliza para crear una instancia del modelo `Student`, que representa una nueva entrada en la tabla de estudiantes de la base de datos.

Durante el proceso de ejecutar la operación que le corresponde, cada función del servicio se encarga de verificar las reglas de negocio asociadas que son demasiado intrincadas como para ser validadas por Pydantic. Verbigracia, Pydantic puede validar fácilmente que el periodo académico ingresado sea un número, pero no puede verificar que el periodo académico sea válido en el contexto de la Universidad; esta verificación corresponde a la lógica de negocio y se realiza en la función del servicio correspondiente. Varios ejemplos de estas reglas de negocio se mencionan en la tabla 3.3.

En términos de analogía con otras tecnologías, el módulo `services` es equivalente a las capas de servicio (*service layers*) en frameworks como Spring en Java, los servicios en ASP.NET en C#, y las clases de servicios en NestJS en TypeScript. Todos estos enfoques tienen como objetivo separar la lógica de negocio tanto de los objetos que representan los datos como de las rutas del API y las representaciones de los recursos.

## Rutas del API

El módulo `routers` define las rutas del API REST, sirviendo como punto de entrada para las solicitudes HTTP realizadas al sistema. Cada archivo en este módulo agrupa las rutas relacionadas con una entidad específica. Por ejemplo, el archivo `routers/student.py` define las rutas bajo el prefijo `/students`, donde cada ruta corresponde a una operación específica relacionada con la entidad `Student`.

Cada ruta está asociada a una operación del API cuya lógica está implementada en los servicios. Por ende, dependiendo del contenido del servicio, una ruta puede corresponder a una operación básica CRUD o una operación personalizada que satisface necesidades específicas del negocio. Cada ruta está implementada como una función dentro del archivo y está decorada con un decorador de FastAPI que indica el método HTTP correspondiente. Los decoradores disponibles son `@router.get`, `@router.post`, `@router.put` y `@router.delete`, correspondientes a las operaciones estándar de HTTP: GET, POST, PUT y DELETE.

Un aspecto clave del diseño es la interacción entre las rutas, los servicios (`services`) y los esquemas (`schemas`). Las funciones de los *routers* actúan como intermediarias: reciben las solicitudes HTTP, validan los datos de entrada utilizando esquemas (`schemas`), delegan la lógica de negocio a las funciones correspondientes en los servicios y entregan respuestas estructuradas basadas en los esquemas de datos. Por ejemplo, una función que crea un estudiante utilizará un esquema `StudentCreate` para validar los datos de entrada, invocará a la función `create_student` del servicio que encapsula la creación del estudiante y entregará una respuesta estructurada utilizando el esquema `StudentResponse` (o `StudentExtendedResponse` si la ruta en cuestión requiere información adicional).

A causa del flujo anterior, los esquemas de tipo `Response`, como `StudentResponse`, desempeñan un papel fundamental en este proceso. Estos esquemas aseguran que las respuestas del API estén estructuradas de manera consistente, incluyendo únicamente los atributos necesarios para el cliente y excluyendo información interna o sensible.

Además de las operaciones CRUD, los *routers* incluyen validaciones adicionales para garantizar que las solicitudes sean válidas antes de delegar el procesamiento a los servicios. Estas validaciones pueden incluir la verificación de que los identificadores no sean nulos, que los datos ingresados sean válidos y que los recursos existan antes de realizar operaciones sobre ellos. Por ejemplo, una solicitud para eliminar un estudiante puede comprobar primero si el estudiante existe, averiguándolo a través del servicio, antes de llamar a la función de eliminación del servicio.

En términos de arquitectura, el módulo `routers` es análogo a los controladores (`controllers`) en frameworks como Spring en Java, ASP.NET en C#, o NestJS en TypeScript. Al igual que en estas tecnologías, el propósito de los *routers* es mantener la independencia entre la lógica de negocio y la lógica de la interfaz de usuario, de forma que los cambios en la lógica de negocio no afecten la interfaz de usuario y viceversa.

## 3.4. Calidad de código

Teniendo en cuenta que el software producido es de alto valor para la universidad y, por ende, probablemente deberá ser mantenido, extendido y reutilizado en el futuro, su desarrollo fue sometido a rigurosos estándares de calidad de código. Esta sección detalla las herramientas y prácticas empleadas para garantizar la calidad del código.

### 3.4.1. Análisis estático

El código fuente del API está escrito para que sea fácil de entender y mantener.

Se configuró SonarQube para realizar análisis estático sobre el código, utilizando el perfil de calidad estándar para Python 3.11. Esta herramienta identifica posibles problemas en el código y genera métricas detalladas para el monitoreo continuo de la calidad. En particular, la herramienta se cerciora de que el código cumple con estándares mínimos de mantenibilidad, confiabilidad y seguridad.

En este caso, los resultados más recientes del análisis de SonarQube se muestran en la figura ??.

Más aún, en favor de garantizar un estilo de código uniforme y limpio, se emplearon las siguientes herramientas:

- isort y black como formateadores de código. El primero se encarga de ordenar las importaciones de los módulos de Python de manera consistente, mientras que el segundo reformatea el código para asegurar un formato consistente. Black garantiza que todos los proyectos de Python que lo usan tengan el mismo estilo de código, lo que facilita la lectura y el mantenimiento del código a desarrolladores nuevos.
- Flake8 y SonarLint como linters, para verificar la adherencia a las mejores prácticas y estándares de calidad de código. Flake8 es una herramienta de verificación de código que busca errores en el estilo del código Python, mientras que SonarLint es un linter que se integra con SonarQube y proporciona análisis en tiempo real del código.

### 3.4.2. Documentación

Existe extensa documentación del proyecto, tanto específica del API como general, incluyendo este documento. La documentación del API se genera automáticamente a partir de los docstrings de las funciones y métodos, gracias a las funcionalidades nativas de FastAPI mencionadas arriba, y se puede acceder a través de las rutas /docs y /redoc.

Para que la documentación generada automáticamente resulte genuinamente útil, es indispensable que los parámetros usados por las funciones estén semánticamente bien nombrados, que los tipos de datos sean explícitos y que todas las funciones estén debidamente documentadas. Para garantizar que ese sea el caso, se emplearon las prácticas descritas a continuación.

En primer lugar, para asegurar que los parámetros de las funciones estén bien nombrados y que los tipos de datos sean explícitos, resulta útil el uso de Pydantic en la definición de los esquemas de datos. Pydantic permite definir los tipos de datos de los atributos de las clases y automáticamente valida los datos que se le pasan a la clase. Por ejemplo, si se define una clase `StudentCreate` con un atributo `name` de tipo `str`, Pydantic se encargará de validar que el valor de `name` sea una cadena de texto. Esto garantiza que los datos que se pasan a las funciones del API sean válidos y que los desarrolladores que utilicen el API sepan qué tipo de datos esperar.

En segundo lugar, respecto a la documentación de las funciones y métodos al interior del código, todas las funciones y métodos del API están documentados mediante *docstrings*, siguiendo las convenciones definidas en el estándar PEP 257. Debido a que

estas disposiciones son notablemente flexibles, se optó por la convención de docstrings de NumPy. Más específicamente, se satisfacen todos los lineamientos de la guía de docstrings de Pandas.

Para evitar omisiones, se emplea la librería `flake8-docstrings` para verificar que los docstrings sigan la convención de NumPy. Esto se configura mediante la opción `docstring-convention=numpy` en el archivo de configuración de Flake8.

### 3.4.3. Pruebas y automatización

Sumado a lo anterior, se integraron diversas herramientas de automatización y pruebas para garantizar la calidad del código en cada modificación. En particular, se destacan las siguientes prácticas:

- Se hizo uso de Pytest para la creación y ejecución de pruebas unitarias. Pytest es una librería de pruebas que facilita la escritura y ejecución de pruebas en Python. Se eligió Pytest por su simplicidad y flexibilidad, que permiten escribir pruebas de manera clara y concisa.
- Se implementó un hook de pre-commit de Git, configurado para garantizar que el código está formateado y probado antes de cada *commit*. El hook de pre-commit se encarga de ejecutar `isort`, `black`, `Flake8` y `Pytest` antes de permitir que se realice un *commit*. Si alguna de estas herramientas reporta un error, el *commit* es rechazado y se notifica al desarrollador.
- Se dispone de un pipeline en GitHub Actions que ejecuta las pruebas, genera un reporte de cobertura y lo envía a SonarQube. El pipeline se ejecuta automáticamente cada vez que se realiza un *push* a la rama principal del repositorio. Si alguna de las pruebas falla, el pipeline se detiene y se notifica al desarrollador.

Esto garantiza que el código que se envía a producción esté formateado correctamente, sea legible, esté probado y cumpla con los estándares de calidad definidos.

## 3.5. Infraestructura

Esta sección describe la infraestructura utilizada para el despliegue del API y la base de datos. La descripción se realiza a alto nivel y sin entrar en las características de la infraestructura, ya que estas son específicas de la Universidad y no son relevantes para el desarrollo del API. Se hace hincapié en la elección de Docker como tecnología de contenedores y en la separación de la base de datos en una máquina virtual aparte, por cuestiones de seguridad.

### 3.5.1. Despliegue del API

El despliegue del API se realiza mediante el uso de contenedores Docker, que permiten garantizar un entorno de ejecución uniforme y reproducible. Estos contenedores se alojan en una máquina virtual proporcionada por la universidad, la cual actúa como servidor central y asegura que el API esté disponible de manera continua. La configuración

del servidor garantiza que el contenedor de Docker se mantenga siempre en ejecución, asegurando así la disponibilidad y la estabilidad del servicio.

El uso de Docker ofrece múltiples beneficios. En primer lugar, permite simplificar el proceso de despliegue al encapsular todas las dependencias y configuraciones necesarias en el contenedor. Esto asegura que el API se comporte de manera consistente en cualquier entorno, eliminando problemas derivados de diferencias en sistemas operativos o configuraciones locales. Además, facilita significativamente las actualizaciones y el mantenimiento. Cuando se realizan cambios en el API, basta con hacer un *pull* del repositorio en la máquina virtual y reiniciar el contenedor para que los cambios sean efectivos. Este enfoque reduce tiempos de inactividad y minimiza errores humanos durante el proceso de despliegue. De hecho, se realizaron pruebas de modificación y despliegue y se determinó que el tiempo de indisponibilidad del servicio tras una actualización es inferior a 5 minutos.

### 3.5.2. Base de datos

La base de datos utilizada es PostgreSQL, una solución robusta, escalable y ampliamente reconocida. PostgreSQL ofrece características avanzadas como soporte para transacciones ACID, manejo eficiente de consultas complejas y extensibilidad mediante módulos adicionales. Sumado a eso, es predominantemente la tecnología de bases de datos que se enseña en la Universidad, por lo cual se consideró una elección natural, pensando en la continuidad del proyecto.

Por razones de seguridad, la base de datos PostgreSQL reside en una máquina virtual separada, accesible únicamente a través de la red interna Séneca de la universidad. Este diseño implementa una práctica conocida como *air-gapping*, la cual consiste en aislar físicamente o lógicamente un sistema del acceso directo desde redes externas. En este caso, el *air-gapping* asegura que la base de datos no pueda ser accedida desde dispositivos o redes externas a la Universidad, incluso si se dispone de las credenciales.

El uso del *air-gapping* tiene particular sentido en este contexto, considerando la sensibilidad de la información gestionada. Los datos académicos y socioeconómicos del estudiantado están además protegidos por normativas de privacidad y protección de datos, lo cual impulsó la implementación de esta arquitectura.

# **Capítulo 4**

## **Capa de presentación**

El presente capítulo detalla el diseño y la implementación de la capa de presentación, cuya responsabilidad es proporcionar una interfaz de usuario intuitiva, interactiva, accesible y estéticamente consistente para el Perfil del estudiante.

Este es probablemente el capítulo más extenso y llamativo. Además de diversos aspectos técnicos correspondientes a la implementación del frontend, exhibe y describe minuciosamente la interfaz del Perfil del estudiante, apoyándose en varias capturas de pantalla. Sin embargo, el lector debe tener presente que el desglose que aquí se realiza con la interfaz no suple la navegación real en el Perfil del estudiante, directamente en la aplicación web NES. Por tanto, se recomienda al lector que, en caso de tener acceso a la aplicación, explore el Perfil del estudiante directamente en ella.

El capítulo se ocupa primeramente de las cuestiones técnicas de la implementación de la capa de presentación, incluyendo el lenguaje y framework utilizados y la estructura de directorios del frontend. Tras eso, se describe, tan bien como es posible en un documento estático, la interfaz interactiva del Perfil del estudiante.

### **4.1. Implementación del frontend**

#### **4.1.1. Lenguaje y framework**

El frontend está implementado en el framework React, utilizando TypeScript como lenguaje principal. El uso de React responde al hecho de que el resto de NES está desarrollado bajo ese framework, por lo que se optó por mantener la consistencia con los demás módulos de la aplicación. Se descartó la implementación de microfrontends o alternativas similares puesto que se consideró que introducía complejidad adicional sin aportar beneficios significativos.

Por otro lado, se eligió TypeScript por encima de JavaScript, debido a que el tipado estático facilita la detección temprana de errores y mejora la mantenibilidad del código. Además, TypeScript es un superset de JavaScript, por lo que resulta compatible con el resto de módulos de la aplicación que se encuentran escritos en JavaScript.

### 4.1.2. Estructura de directorios del proyecto

El frontend sigue una arquitectura modular, con el código al interior de la carpeta `src` y cada página encapsulada en un directorio dentro de `src/pages`. Esto facilita la organización del código y su mantenibilidad. Debido a que no compete a este trabajo detallar la estructura de directorios de NES en su totalidad, se describe solo superficialmente la estructura general del frontend, en la tabla 4.1, a manera de contexto.

**Tabla 4.1**

*Estructura de directorios del proyecto*

Directorio/Archivo	Descripción
<code>App.tsx</code>	Componente principal de la aplicación, que sirve como punto de entrada y organiza las rutas.
<code>Router.tsx</code>	Configura las rutas de la aplicación, incluyendo la navegación al perfil del estudiante y a las otras páginas.
<code>MainLayout.tsx</code>	Componente que define el diseño general de la aplicación, incluyendo el encabezado, pie de página y disposición general.
<code>pages/</code>	Directorio que contiene el código asociado a cada página de la aplicación, como el perfil del estudiante.
<code>hooks/</code>	Directorio que almacena hooks reutilizables en toda la aplicación, como manejo del estado global.
<code>components/</code>	Contiene componentes reutilizables que se utilizan en varias páginas.
<code>constants/</code>	Define constantes globales del proyecto, como configuraciones y estilos estándar.
<code>services/</code>	Implementa funciones para interactuar con otros servicios externos (distintos al API REST del Perfil del estudiante) que alimentan el frontend.
<code>assets/</code>	Contiene recursos estáticos como imágenes utilizados en la aplicación.
<code>config/</code>	Directorio con configuraciones específicas librerías como axios y Firebase.
<code>interfaces/</code>	Define interfaces de TypeScript que se utilizan para tipar datos y garantizar consistencia en el código.
<code>recoilAtoms/</code>	Contiene los <i>atoms</i> de Recoil para el manejo del estado global de la aplicación, que se utilizan para compartir información entre componentes.
<code>utils/</code>	Agrupa funciones auxiliares reutilizables en diferentes partes del proyecto.
<code>index.css</code>	Archivo principal de estilos CSS utilizado para definir configuraciones de diseño globales.

El código correspondiente al Perfil del estudiante se encuentra en un solo directorio al interior de `src/pages`, que es `StudentProfilePage`. Naturalmente, en la implementación del Perfil del estudiante se importan y se reutilizan varias utilidades, interfaces, componentes y hooks de las carpetas `src/utils`, `src/interfaces`, `src/components` y `src/hooks`, respectivamente, lo que asegura la consistencia y reduce la duplicación de código.

### Estructura del módulo del Perfil del estudiante

El módulo del Perfil del estudiante está encapsulado en el directorio `src/pages/StudentProfilePage`. Allí, el código a su vez está organizado en un subdirectorios por cada pestaña visual del Perfil del estudiante. Esta organización modulada facilita dos cosas:

- Realizar cambios de manera localizada, ya que cada pestaña es prácticamente un submódulo independiente.
- Asegurar que los permisos de usuario sean administrados correctamente, pues cada pestaña tiene diferentes niveles de acceso. Esto es especialmente importante para la pestaña Financiación, que contiene el detalle de la información financiera del estudiante y, por las políticas de gobierno de datos de la Universidad, solo debe ser accesible para directivos.

La estructura de directorios al interior del módulo del Perfil del estudiante se detalla en la tabla 4.2.

**Tabla 4.2**

*Estructura de directorios del módulo del Perfil del estudiante*

Directorio/Archivo	Descripción
<code>performanceTab/</code>	Contiene la pestaña de rendimiento académico.
<code>fundingTab/</code>	Contiene la pestaña de financiación.
<code>subjectsTab/</code>	Contiene la pestaña con todas las materias inscritas por el estudiante.
<code>scheduleTab/</code>	Proporciona una vista del horario del estudiante.
<code>personalDataCard</code>	Componente que muestra información personal del estudiante y se reutiliza en todas las pestañas.
<code>chips/</code>	Contiene componentes específicos reutilizados únicamente dentro del perfil del estudiante, como botones y etiquetas.
<code>useStudentProfile</code>	Hook que centraliza la obtención de los datos del perfil desde el API REST.

Con respecto a la tabla anterior, cabe señalar que los componentes implementados en `chips/` no se implementaron en la carpeta `components/` del proyecto debido a que son específicos del Perfil del estudiante y no se reutilizan en otras partes de la aplicación. Dicho eso, en caso de que se requiera reutilizar estos componentes en otras partes de la

aplicación, basta con moverlos a la carpeta `components/` y ajustar las importaciones correspondientes.

Bajo un raciocinio similar, el hook `useStudentProfile.tsx` se implementó dentro del directorio del módulo del Perfil del estudiante, en lugar de en `hooks/`, porque se utiliza solamente en el Perfil del estudiante y no en otras partes de la aplicación. Es, además, el único código del frontend que se comunica con el API REST del Perfil del estudiante, por lo que su ubicación en el módulo del Perfil del estudiante facilita la comprensión y el mantenimiento del código.

El hook `useStudentProfile.tsx` solicita toda la información relevante del API REST del Perfil del estudiante en una única petición. Esto se hace así porque resulta más eficiente. Inicialmente, el diseño del API contaba con un endpoint separado para cada recurso, lo que implicaba realizar múltiples peticiones independientes para cada pestaña. Sin embargo, se determinó que esta estructura incrementaba la latencia. Por esta razón, el API fue rediseñado para agrupar todos los recursos en una única representación unificada, accesible mediante un solo endpoint.

#### 4.1.3. Ruta al perfil del estudiante

Como ruta para acceder al perfil del estudiante desde NES se seleccionó `/perfil-estudiante/<correo>`, donde `<correo>` corresponde al correo Uniandes, sin el dominio, del estudiante de quien se quiere ver el perfil. Por ejemplo, la ruta para acceder al perfil del autor de este documento es `/perfil-estudiante/f.melo`.

Cualquier usuario que esté autenticado en NES y cuente con los permisos adecuados, podrá acceder y visualizar directamente el perfil del estudiante al ingresar a la ruta indicada.

Esto permite que entre profesores y directivos de la Universidad se compartan fácilmente perfiles de estudiantes. En caso de que el lector tenga ese rol dentro de la Universidad, puede probar la funcionalidad iniciando sesión en NES y luego pulsando en el siguiente enlace: <https://noestassolo.virtual.uniandes.edu.co/perfil-estudiante/f.melo>.

Se hace hincapié en que el usuario debe estar autenticado en NES y debe contar con los permisos necesarios para acceder al perfil de un estudiante. De lo contrario, no podrá visualizarlo. En particular, ningún alumno podrá acceder al perfil de algún otro de esta manera (ni de ninguna otra, salvo que el otro pupilo o algún otro usuario autorizado deliberadamente se lo permita).

### 4.2. Interfaz del Perfil del estudiante

Esta sección describe la interfaz del Perfil del estudiante, mostrando capturas de pantalla de las diferentes pestañas y secciones que lo componen. Se detallan las funcionalidades y la información presentada en cada una de ellas, así como las decisiones de diseño tomadas para garantizar la claridad, relevancia y simplicidad de la información.

Como se mencionó anteriormente, esta descripción no reemplaza la experiencia de navegación real en el Perfil del estudiante. Por tanto, se recomienda al lector que, en caso de tener acceso a la aplicación, explore el Perfil del estudiante directamente en ella.

Todas las capturas de pantalla en esta sección son reales. La gran mayoría corresponden al Perfil del estudiante del autor de este documento; aquellas que no, lo indican explícitamente, mencionando que corresponden a un estudiante anónimo en la descripción de la figura.

#### 4.2.1. Acceso al perfil del estudiante

Los estudiantes, profesores y directivos de la Universidad cuentan con acceso al Perfil del estudiante, cada uno con diferentes niveles de permisos y restricciones de visualización. Los profesores y directivos pueden acceder a los perfiles de todos los estudiantes, mientras que los estudiantes únicamente pueden acceder a su propio perfil. Más aún, los directivos tienen acceso a la pestaña de Financiación, mientras que los profesores y estudiantes no tienen acceso a esta pestaña.

En la sección 4.1.3 se explica una forma práctica de acceso al Perfil del estudiante mediante una URL específica. Sin embargo, la forma más común e intuitiva de acceso al Perfil del estudiante es desde la interfaz de NES. La manera de realizar este acceso es diferente para los estudiantes y para los profesores y directivos, como se describe a continuación.

##### Acceso al Perfil del estudiante como estudiante

Cada alumno tiene la posibilidad de acceder a su propio Perfil del estudiante y tiene dos maneras de hacerlo. Para ambas, en primer lugar, debe iniciar sesión en NES con su correo Uniandes y su contraseña, pulsando en el botón *Iniciar sesión* en la esquina superior derecha de la página de inicio de NES, que se muestra en la figura 4.1.



Figura 4.1: Página inicial de NES.

Una vez iniciada la sesión, la página inicial de NES se transforma a la vista de estudiante, como se muestra en la figura 4.2. Esta vista difiere de la anterior en tanto que ahora existe un botón adicional, titulado Mi perfil académico. Esta es la primera manera de acceder al Perfil del estudiante como estudiante: pulsando en el botón Mi perfil académico.



Figura 4.2: Página inicial de NES para un estudiante.

La segunda manera de acceder al Perfil del estudiante como alumno es pulsando en su fotografía, que se ubica en la parte superior derecha de la página inicial de NES para estudiantes. Al dar clic en la fotografía, se despliega la ventana modal que se puede ver en la figura 4.3, en el cual se encuentra la opción Ver mi perfil académico. Al pulsar en esta opción, se accede al Perfil del estudiante.

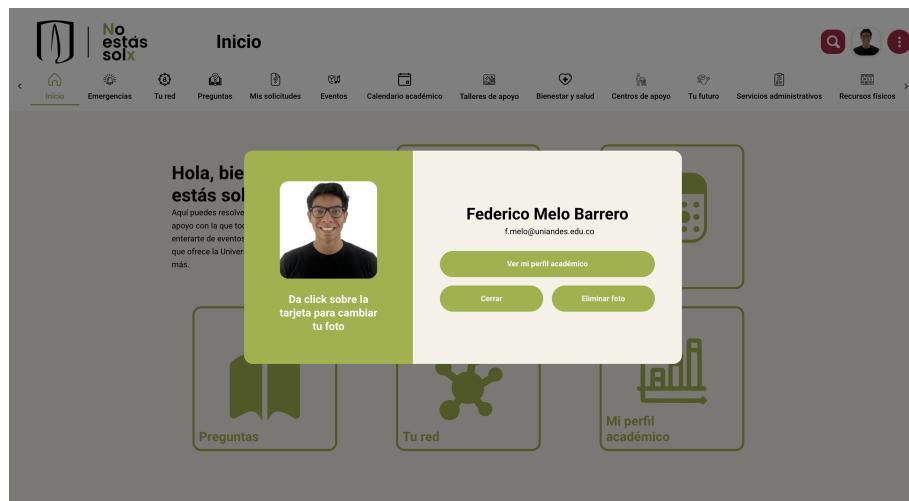


Figura 4.3: Ventana modal de opciones para el estudiante.

### Acceso al Perfil del estudiante como profesor o directivo

El acceso al perfil del estudiante como profesor o directivo difiere del acceso como alumno principalmente en el hecho de que el profesor o directivo debe seleccionar el estudiante cuyo perfil desea visualizar. Para ello, tras haber iniciado sesión en NES con su correo Uniandes y su contraseña, debe dirigirse a la pestaña Tus aconsejados en la barra de navegación superior. Una vez seleccionada esta pestaña, se despliega una lista de estudiantes como la que se muestra en la figura 4.4.

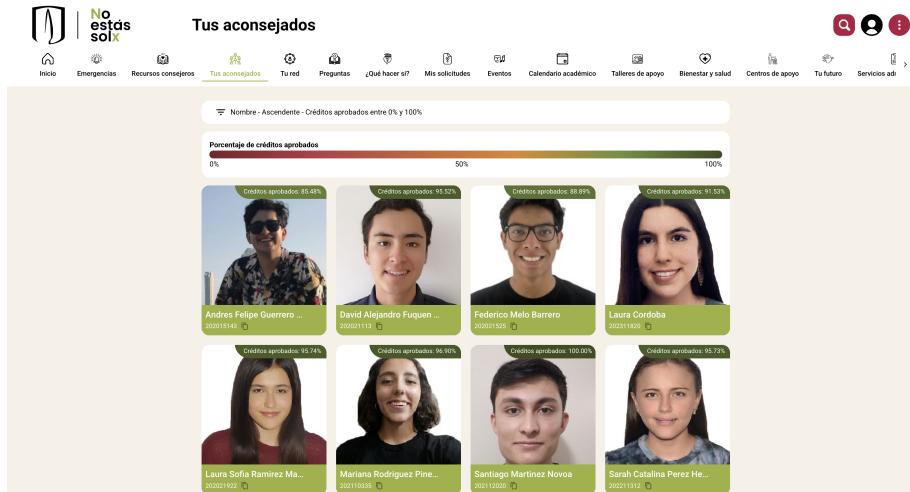


Figura 4.4: Pestaña Tus aconsejados en NES.

En esta lista, el profesor o directivo puede seleccionar al alumno cuyo perfil desea visualizar. Al hacer clic sobre la tarjeta que representa a esa persona, se accede al Perfil del estudiante de ese alumno en particular. Los estudiantes que aparecen en ese listado dependen del rol del profesor o directivo en la Universidad, así como de las políticas de gobierno de datos de la Universidad. Para los profesores consejeros, la lista se limita a sus aconsejados. Dependiendo del rol del usuario en cuestión, es posible que pueda hacer uso de la barra de búsqueda en la parte superior derecha de la interfaz para buscar a cualquier estudiante de la Universidad, ya sea por su nombre, por su correo Uniandes o por su código estudiantil.

Más aún, en la pestaña Tus aconsejados, el profesor o directivo puede pulsar en el ícono de tres rayas horizontales que se encuentra en la parte superior izquierda de la pestaña, para desplegar un menú lateral izquierdo con opciones para filtrar los estudiantes, como se muestra en la figura 4.5. Allí, es posible ordenar los alumnos mostrados por nombre o porcentaje de créditos aprobados, así como utilizar la barra deslizante para fijar un rango de porcentaje de créditos aprobados. Para algunos roles, es posible marcar la opción que permite visualizar a todos los estudiantes de la Universidad, en lugar de solo los aconsejados, con base en las políticas de gobierno de datos de la Universidad.

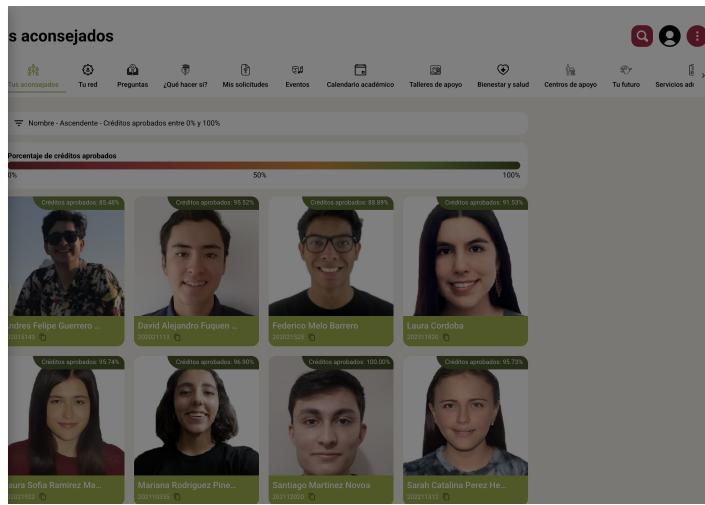


Figura 4.5: Menú de filtros en la pestaña Tus aconsejados.

#### 4.2.2. Disposición general

El Perfil del estudiante tiene la misma disposición general en cualquiera de sus presentaciones, independiente de si el usuario es un estudiante, un profesor o un directivo. La interfaz se divide en dos secciones principales: a la izquierda, se ubica una tarjeta con información personal del estudiante; a la derecha, se encuentra la pestaña que esté activa. La figura 4.6 muestra la disposición general del Perfil del estudiante con la pestaña Desempeño activa, que es la pestaña por defecto.



Figura 4.6: Disposición general del Perfil del estudiante.

La sección de información personal del estudiante contiene toda la información relevante para identificarlo. En la parte superior, se presenta prominentemente la fotografía del alumno, seguida de su nombre completo y el programa académico al que pertenece.

Tras eso, se plasma su nivel académico. En caso de que el estudiante haya estado afiliado a la Universidad en distintos niveles académicos, ese componente se convierte

en un selector que permite seleccionar el nivel académico que se desea visualizar, como se muestra en la figura 4.7. Al cambiar el nivel académico, cambia por completo el Perfil del estudiante, pues toda la información presentada en las pestañas se ajusta al nivel académico seleccionado.

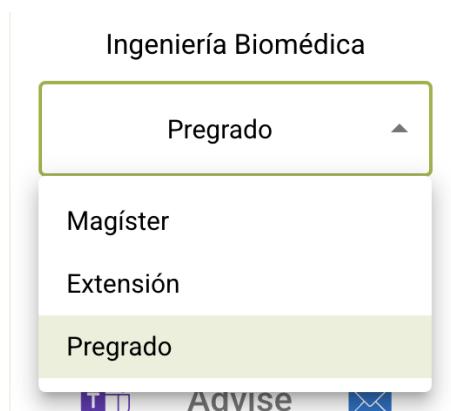


Figura 4.7: Selector de nivel académico en el perfil de un estudiante anónimo.

Más abajo, se detallan datos como el código Uniandes del estudiante, su usuario de correo institucional y su número de contacto, este último solo disponible para directivos. Cada uno de esos tres datos tienen a su derecha un botón que facilita copiarlo al portapapeles. Finalmente, se ofrecen botones de interacción directa, como un enlace para asesoramiento mediante Microsoft Teams y otro para enviar correos electrónicos, lo que facilita la comunicación inmediata con el alumno.

La sección de información personal del estudiante está diseñada para que el Perfil gire en torno a la persona. Independientemente de la pestaña activa en la interfaz, los datos personales del alumno se encuentran siempre visibles en la parte izquierda de la pantalla. La interfaz dispone de un sistema de scroll independiente para las dos secciones, permitiendo navegar en la pestaña seleccionada sin que se pierda de vista la información principal del estudiante. La fotografía, ubicada de forma prominente, jamás queda oculta y refuerza la conexión humana, recordando en todo momento que detrás de los datos hay una persona con identidad y contexto.

#### 4.2.3. Pestañas del Perfil del estudiante

El Perfil del estudiante se compone de cuatro pestañas principales: Desempeño, Materias, Financiación y Horario. Los nombres de las pestañas son cortos y dicitentes: la primera, Desempeño, contiene toda la información asociada al rendimiento académico; la segunda, Materias, consta de un listado exhaustivo de todas las materias inscritas; la tercera, Financiación, presenta información detallada sobre cómo el estudiante ha financiado sus estudios; y la cuarta, Horario, muestra el horario de clases del alumno al momento de la consulta.

Cada una de estas pestañas presenta información relevante para el estudiante, el profesor o el director, dependiendo del rol del usuario que accede al Perfil. A continuación, se dedica una sección a cada una de las pestañas, describiendo la información presentada y las funcionalidades disponibles.

## Pestaña de Desempeño

La pestaña de Desempeño es la pestaña por defecto que se muestra al acceder al Perfil del estudiante. Esta pestaña ofrece un análisis detallado del rendimiento académico a través del tiempo. A continuación, se detallan uno a uno los elementos que componen la pestaña de Desempeño. Para empezar, la figura 4.8 presenta un primer pedazo de la pestaña de Desempeño, que contiene los indicadores claves y la gráfica central.



Figura 4.8: Parte de arriba de la pestaña de Desempeño.

**Indicadores clave** En la parte superior de la pestaña, se encuentran cuatro indicadores clave que resumen el desempeño académico: el Promedio General Acumulado (PGA), el Puntaje Advise, el número de matrículas que ha realizado y el estado académico actual del estudiante.

- El PGA es el principal indicador de rendimiento académico en la Universidad. De acuerdo con el Reglamento general de estudiantes de pregrado de la Universidad de los Andes, el PGA resulta de multiplicar el número de créditos de cada materia por la calificación obtenida, sumando estos productos y dividiendo el resultado por el número total de créditos calificados numéricamente. Su menor valor es 1.5 y su máximo valor es 5.0. El PGA se presenta con dos decimales.
- El Puntaje Advise es un indicador de la probabilidad de éxito académico del estudiante en la Universidad. Es calculado y provisto por la herramienta de análisis predictivo Advise y oscila entre 0 y 100. Se presenta el puntaje Advise tal y como se extrae de la herramienta.
- El número de matrículas representa la cantidad de semestres completos que han sido pagados y matriculados. Puede ser un número decimal, en caso de que el alumno haya cursado matrículas parciales. En esta medida, un semestre completo se valora como una matrícula; media matrícula se valora como 0.5 matrículas; y un cuarto de matrícula, un periodo intersemestral o una práctica académica se valora como 0.25

matrículas. Esta medida es un buen reflejo de la inversión realizada en la educación del estudiante.

- El estado académico actual del alumno es un término que resume su situación académica. [Falta descripción del estado académico.]

Estos indicadores son la primera pieza de información que el usuario ve al acceder al Perfil del estudiante y le permiten tener una visión general de su rendimiento de un vistazo. Al pasar el cursor sobre cada uno de los indicadores, se despliega una descripción detallada del indicador, que facilita su interpretación. Un ejemplo de esto se presenta en la figura 4.9, en la que se pasa el cursor por encima del indicador de Puntaje Advise.



Figura 4.9: Descripción del indicador de Puntaje Advise al pasar el cursor sobre él.

**Profesor consejero** Inmediatamente debajo de los indicadores clave, se muestra el nombre del consejero académico asignado al alumno. El consejero académico es un profesor de la Universidad, comisionado para asesorar a la persona en temas académicos y administrativos, y que puede ser contactado por ella en caso de necesitar orientación. A los estudiantes que cursan múltiples programas académicos se les asigna un consejero académico por cada programa. El raciocinio detrás de mostrar el consejero académico en una posición tan prominente de la pestaña radica en responsabilizar al consejero académico de la orientación del aconsejado y de su rendimiento académico, en particular cuando este ingrese a visualizar el Perfil del estudiante.

**Gráfica central** A continuación, se encuentra una gráfica interactiva. Esta gráfica es la pieza central de la pestaña de Desempeño y realmente podría desglosarse en doce gráficas distintas. La interactividad de la gráfica es lo que permite que pueda tener muchas presentaciones distintas sin resultar abrumadora. Para explicar la gráfica, en primer lugar se listan las opciones de interacción que esta ofrece y tras eso se listan las 12 visualizaciones distintas que puede presentar al elegir determinadas opciones de interacción.

Respecto a las opciones de interacción, para variar entre las presentaciones de la gráfica, se presentan tres grupos de botones:

- En la parte inferior derecha de la gráfica, se encuentra un botón que permite seleccionar la información que la gráfica despliega. Las posibles opciones son: Promedios, Porcentaje de Créditos Aprobados y Créditos Aprobados.
- En la parte superior derecha, se encuentra un botón que permite alternar entre la Vista por periodo y la Vista acumulada.
- En la parte inferior izquierda, se encuentra un botón que permite alternar entre dos escalas de la gráfica: la escala estándar, que depende del tipo de información

seleccionada, y la escala relativa, que facilita la visualización del comportamiento de la información a través del tiempo.

Cada combinación de botones seleccionada resulta en una visualización distinta de la gráfica. A continuación, se listan las 12 visualizaciones distintas que puede presentar la gráfica:

- Promedios, Vista acumulada, Escala estándar.** Esta visualización se titula “PGA a través del tiempo” y muestra la evolución del Promedio General Acumulado (PGA) del estudiante a lo largo del tiempo, en una escala estándar cuyo valor mínimo es 1.5 y cuyo valor máximo es 5.0. Cada punto de la gráfica representa el PGA del estudiante en un periodo académico específico. En la escala estándar se dibujan dos líneas de referencia: la inferior corresponde al PGA mínimo y se traza en rojo a la altura del 3.0; la superior corresponde al PGA máximo y se traza en verde a la altura del 5.0. Se presenta en la figura 4.10.



Figura 4.10: Visualización de PGA a través del tiempo en escala estándar.

- Promedios, Vista acumulada, Escala relativa.** Esta visualización transmite la misma información que la anterior y se titula igual. La diferencia radica en el enfoque que da a la información. Se modifica la escala de la gráfica para que el menor valor sea el PGA mínimo que el estudiante ha obtenido y el mayor valor sea el PGA máximo que ha alcanzado. Esto pone el énfasis en la variabilidad del PGA del alumno a lo largo del tiempo, permitiendo ver los períodos en los que mejoró o empeoró su rendimiento académico. Se presenta en la figura 4.11.



Figura 4.11: Visualización de PGA a través del tiempo en escala relativa.

3. **Promedios, Vista por periodo, Escala estándar.** Esta visualización se titula “Promedios semestrales a través del tiempo” y muestra los promedios semestrales que el estudiante ha obtenido a lo largo del tiempo. Cada punto de la gráfica representa el promedio semestral del alumno en un período académico específico. Se utiliza la misma escala estándar que en la visualización 1. Se presenta en la figura 4.12.



Figura 4.12: Visualización de promedios semestrales a través del tiempo en escala estándar.

En la figura 4.12 resalta que hay un hueco en la gráfica. Esto puede suceder para alumnos que en un período académico determinado no tuvieron un promedio semestral numérico. Lo anterior puede deberse a varias causas; algunas de las más comunes son que el estudiante haya cursado un intercambio académico o una práctica académica, o que únicamente haya visto materias con calificación alfabética (por ejemplo: deportes o cursos de inglés). En estos casos, no se calcula un promedio semestral numérico, lo cual se ve reflejado en la gráfica mediante un salto. Ese espacio no se evidencia en la gráfica de la visualización 1 porque el PGA siempre es numérico, y si en un semestre no hay promedio semestral, el PGA se calcula con los promedios semestrales de los semestres anteriores.

4. **Promedios, Vista por periodo, Escala relativa.** Similar a la visualización 2, esta visualización corresponde a la misma información de la visualización 3 pero pone el foco en el comportamiento del promedio semestral del estudiante a lo largo del tiempo. Se presenta en la figura 4.13.



Figura 4.13: Visualización de promedios semestrales a través del tiempo en escala relativa.

**5. Porcentaje de Créditos Aprobados, Vista acumulada, Escala estándar.** Esta visualización se titula “Porcentaje acumulado de créditos aprobados” y muestra como ha cambiado el porcentaje de créditos aprobados por el alumno a lo largo del tiempo. En cada periodo, el porcentaje de créditos aprobados es acumulado, por lo cual contempla los períodos anteriores. Se utiliza una escala estándar con un mínimo de 0 % y un máximo de 100 %. Se trazan dos líneas de referencia, que son proporcionales a las trazadas en la visualización 1: la inferior se traza en rojo y corresponde al 60 % de los créditos aprobados; la superior se traza en verde y corresponde al 100 % de los créditos aprobados. Se presenta en la figura 4.14.

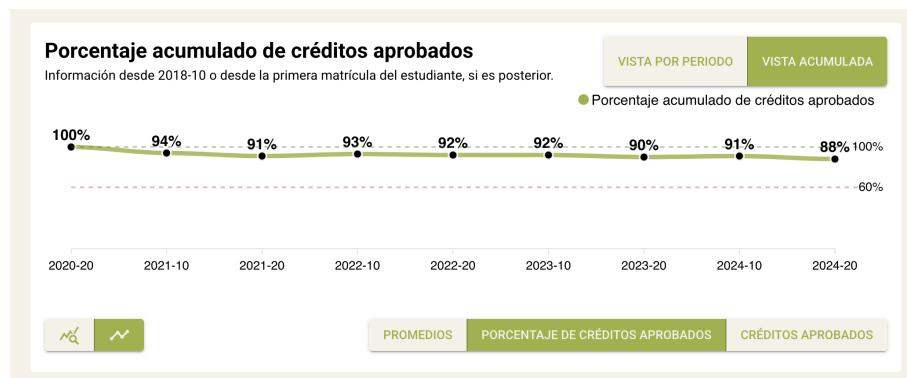


Figura 4.14: Visualización de porcentaje de créditos aprobados a través del tiempo en escala estándar.

**6. Porcentaje de Créditos Aprobados, Vista acumulada, Escala relativa.** Similar a la visualización 5, esta visualización pone el énfasis en la variabilidad del porcentaje de créditos aprobados del estudiante a lo largo del tiempo, en lugar de en los valores. Se presenta en la figura 4.15.

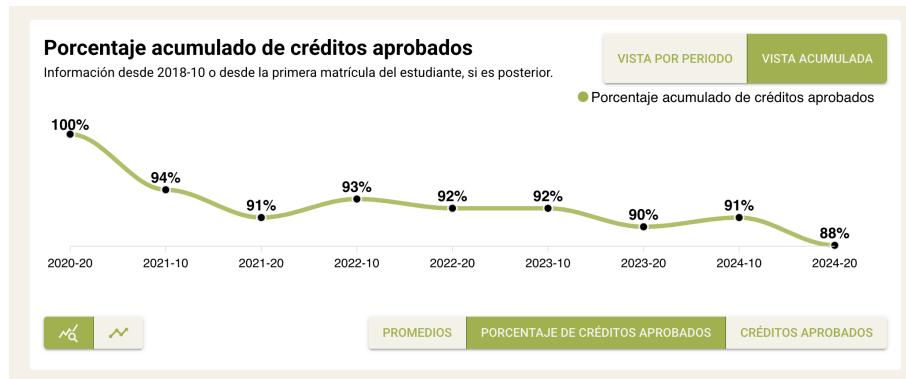


Figura 4.15: Visualización de porcentaje de créditos aprobados a través del tiempo en escala relativa.

7. **Porcentaje de Créditos Aprobados, Vista por periodo, Escala estándar.** Esta visualización se titula “Porcentaje de créditos aprobados por periodo” y muestra el porcentaje de créditos aprobados por el alumno en cada periodo académico, sin contemplar los períodos anteriores. Esto permite estudiar el desempeño del estudiante en cada periodo de forma independiente. Se utiliza la misma escala estándar que en la visualización 5. Se presenta en la figura 4.16.



Figura 4.16: Visualización de porcentaje de créditos aprobados por periodo en escala estándar.

8. **Porcentaje de Créditos Aprobados, Vista por periodo, Escala relativa.** Similar a la visualización 7, esta visualización pone el énfasis en el comportamiento del porcentaje de créditos aprobados del estudiante en cada periodo académico. Se presenta en la figura 4.17.

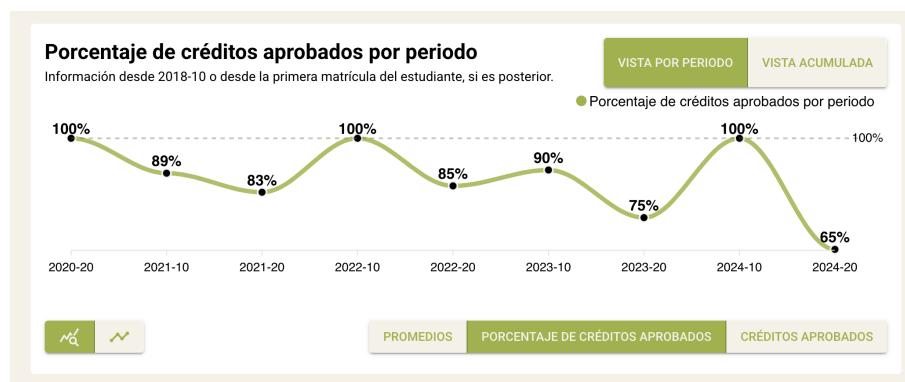


Figura 4.17: Visualización de porcentaje de créditos aprobados por periodo en escala relativa.

9. **Créditos Aprobados, Vista acumulada, Escala estándar.** Esta visualización se titula “Créditos aprobados acumulados” y muestra la cantidad de créditos aprobados por el estudiante a lo largo del tiempo. En cada período, los créditos aprobados son acumulados, por lo cual contempla los períodos anteriores. La escala estándar usa un mínimo de 0 créditos y utiliza como máximo el número de créditos aprobados por el alumno hasta la fecha. No se trazan líneas de referencia, pues no tiene sentido para esta gráfica, en la que un valor bajo en el total de créditos aprobados no indica un mal rendimiento sino simplemente una etapa temprana en la carrera del estudiante. Se presenta en la figura 4.18.

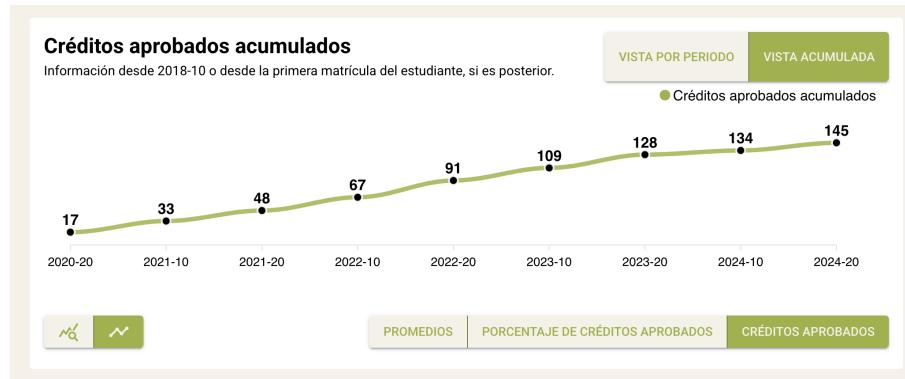


Figura 4.18: Visualización de créditos aprobados a través del tiempo en escala estándar.

10. **Créditos Aprobados, Vista acumulada, Escala relativa.** Esta visualización resulta muy similar a la anterior, con la única diferencia en que el valor mínimo es el número de créditos aprobados por el alumno en el período más temprano, en lugar de ser 0 créditos. Se presenta en la figura 4.19.

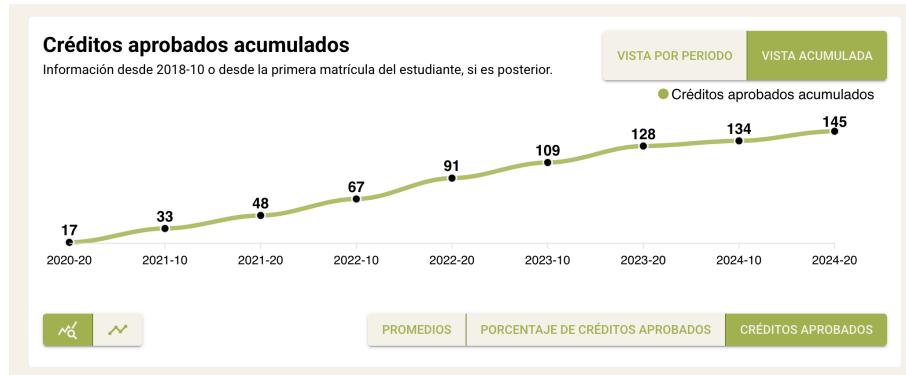


Figura 4.19: Visualización de créditos aprobados a través del tiempo en escala relativa.

11. **Créditos Aprobados, Vista por periodo, Escala estándar.** Esta visualización se titula “Créditos aprobados por periodo” y muestra la cantidad de créditos aprobados por el estudiante en cada periodo académico, sin contemplar los períodos anteriores. La escala estándar usa un mínimo de 0 créditos y un máximo que depende del número de créditos aprobados en el periodo con más créditos aprobados. Se traza una línea de referencia, a la altura de los 11 créditos, que no necesariamente tiene connotaciones negativas, teniendo en cuenta que el alumno puede haber cursado matrículas parciales. Se presenta en la figura 4.20.

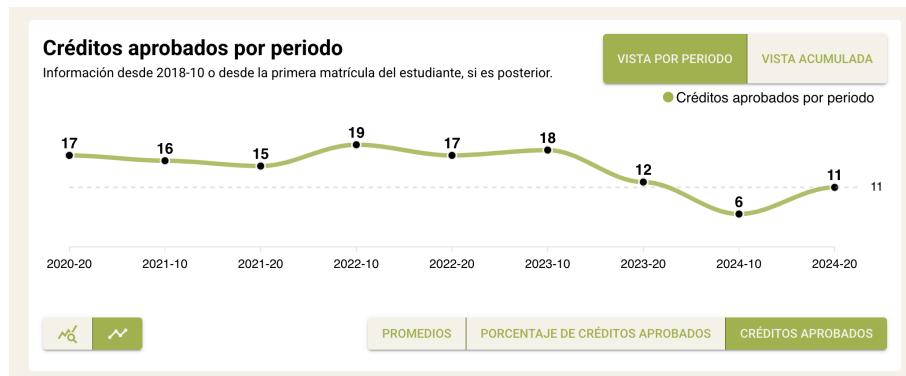


Figura 4.20: Visualización de créditos aprobados por periodo en escala estándar.

12. **Créditos Aprobados, Vista por periodo, Escala relativa.** Esta última visualización presenta la misma información que la visualización 11, pero pone el énfasis en el comportamiento del número de créditos aprobados por el estudiante en cada periodo académico. Se presenta en la figura 4.21.

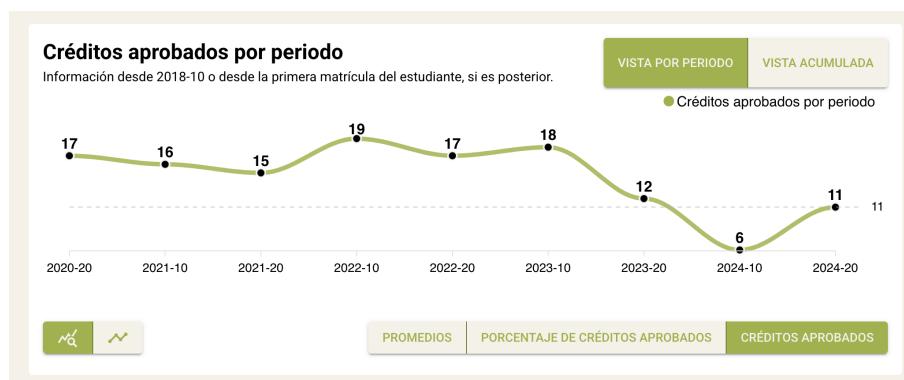


Figura 4.21: Visualización de créditos aprobados por periodo en escala relativa.

**Tabla de créditos inscritos** La tabla de créditos inscritos es el siguiente componente de la pestaña de Desempeño. Se muestra en la figura 4.22. La primera fila de la tabla proporciona un resumen del total de créditos inscritos por el alumno, clasificándolos en *aprobados, reprobados, retirados, incompletos y pendientes*. La segunda fila de la tabla permite al usuario visualizar ese desglose de los créditos inscritos pero con relación a un área de conocimiento específica, que el usuario selecciona de un menú desplegable. Por ejemplo, para los estudiantes de ingeniería, se puede seleccionar las áreas de matemáticas, física o su ingeniería específica. La tabla se actualiza automáticamente al seleccionar un área de conocimiento distinta. Como interacción adicional, la tabla incluye la posibilidad de alternar los valores desplegados como cantidad absoluta de créditos o como porcentaje del total de créditos inscritos, facilitando el análisis de los datos por parte del usuario.

Créditos inscritos						#	%
	Inscritos	Aprobados	Reprobados	Retirados	Incompletos	Pendientes	
Total	165	145	0	17	0	3	
Total ▾	165	145	0	17	0	3	

Figura 4.22: Tabla de créditos inscritos.

**Gráfica de créditos por periodo** Tras lo anterior, aparece la gráfica de créditos por periodo, exhibida en la figura 4.23. Esta gráfica desglosa el número de créditos aprobados, retirados y reprobados en cada periodo académico. Traza una curva para cada uno, pintando los créditos aprobados en verde, los retirados en azul y los reprobados en rojo. Esta gráfica, al igual que la primera, también cuenta con múltiples representaciones. Sin embargo, en ese caso es más sensato explicar las posibles interacciones de la gráfica, sin necesidad de listar las doce visualizaciones posibles.

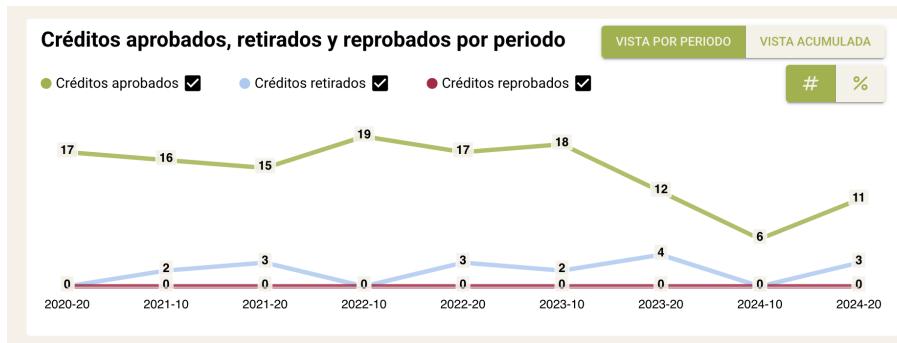


Figura 4.23: Gráfica de créditos por periodo.

- **Esconder alguna, dos o todas las curvas.** La gráfica permite al usuario seleccionar cuáles curvas desea visualizar. Por defecto, se muestran todas las curvas, pero el usuario puede deseleccionar alguna o todas las curvas para enfocarse en una sola o en contrastar dos curvas. Esto resulta útil para reducir el ruido en análisis específicos, como comparar el número de créditos aprobados con el número de créditos reprobados, determinar el comportamiento de específicamente los créditos retirados, entre otros. En la figura 4.24 se muestra un ejemplo en el que se han deseleccionado las curvas de créditos aprobados y reprobados, dejando solo la curva de créditos retirados.

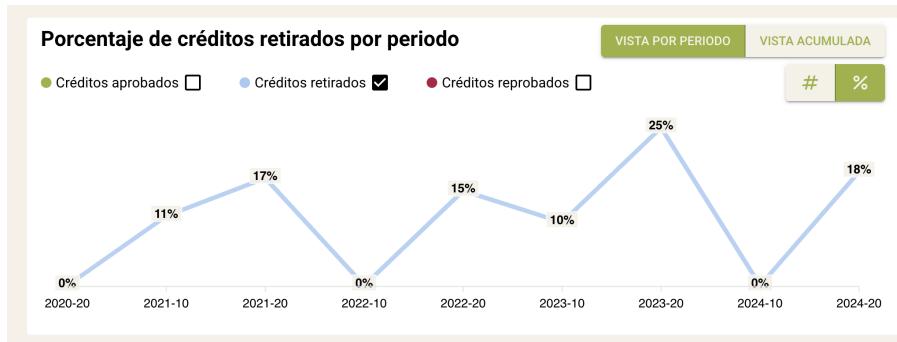


Figura 4.24: Gráfica de créditos por periodo con solo la curva de créditos retirados.

- **Vista por periodo o acumulada.** Al igual que la gráfica central, esta de créditos por periodo permite al usuario alternar entre una vista por periodo y una vista acumulada. La vista por periodo muestra el comportamiento de los créditos aprobados, retirados y reprobados en cada periodo académico, mientras que la vista acumulada muestra la evolución de esos créditos a lo largo del tiempo. La vista acumulada es particularmente útil para ver cómo el número de créditos aprobados, retirados y reprobados ha cambiado a lo largo de la carrera del estudiante. La figura 4.25 muestra un ejemplo de la gráfica de créditos por periodo en vista acumulada.

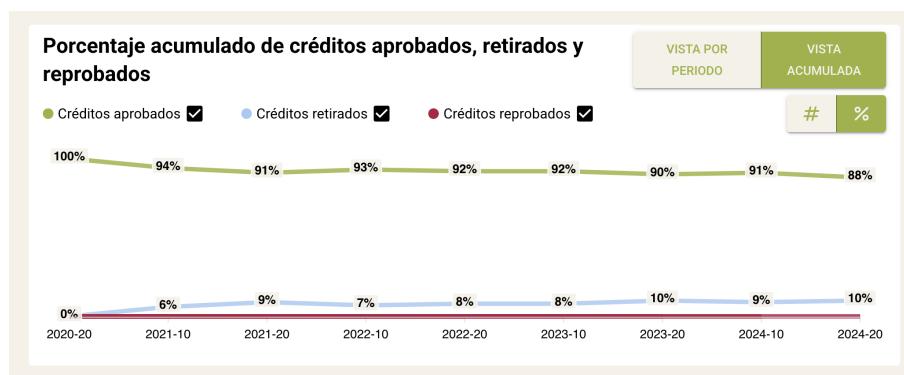


Figura 4.25: Gráfica de créditos por periodo en vista acumulada.

- **Vista porcentual o absoluta.** Al igual que la tabla de créditos inscritos, la gráfica de créditos por periodo permite al usuario alternar entre una vista porcentual y una vista absoluta. La vista porcentual es particularmente útil para comparar el comportamiento de los créditos aprobados, retirados y reprobados en cada periodo académico, independientemente del número total de créditos inscritos. Las figuras 4.24 y 4.25 constituyen ejemplos de la gráfica de créditos por en vista porcentual, mientras que la figura 4.23 muestra la gráfica en vista absoluta.

**Condiciones académicas** En la parte inferior izquierda de la pestaña de Desempeño, se presenta la cantidad de suspensiones académicas, pruebas académicas, pruebas de reingreso e incompletos totales que el estudiante ha tenido a lo largo de su carrera. Se muestra en la figura 4.26. Es un componente simple, al que la interactividad no le agregaría valor, pero constituye una pieza de información crucial para el usuario, sobretodo si el conteo de alguno de esos indicadores es mayor a cero.

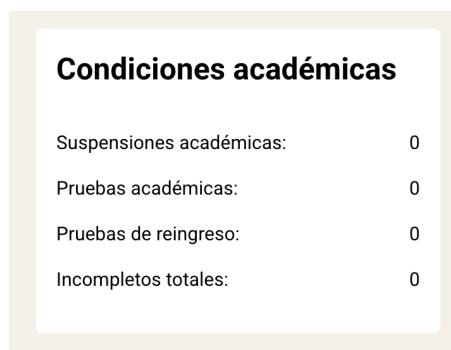


Figura 4.26: Condiciones académicas.

**Resultados del ICFES** En la parte inferior derecha de la pestaña, se resume el desempeño del estudiante en el examen de estado Saber Pro, como se puede evidenciar en la figura 4.27. Se presenta el puntaje global del examen, así como los puntajes específicos obtenidos en cada una de las áreas del examen. Se presenta un gráfico de radar que ilustra los puntajes de cada área y permite reconocer las fortalezas y debilidades del alumno un solo vistazo. Este gráfico es interactivo y permite al usuario visualizar los puntajes de

cada área al pasar el cursor sobre el gráfico. En esa tarjeta se incluye también el nombre de la institución que otorgó el diploma de bachiller al estudiante en cuestión.

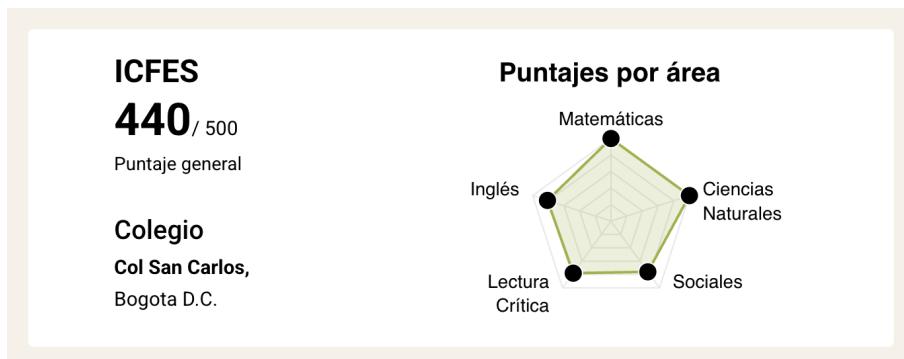


Figura 4.27: Resultados del ICFES.

### Pestaña de Materias

La pestaña de Materias presenta un listado de todas las asignaturas que el estudiante ha inscrito a lo largo de su estancia en la Universidad. Consta de dos componentes principales. En la parte superior, mostrada en la figura 4.28, se encuentra un componente que permite buscar o filtrar materias con facilidad, incluyendo una barra de búsqueda textual y tres filtros: por periodo, por código de área de la materia y por estado de la asignatura (aprobada, retirada, reprobada, pendiente o incompleta). Cada filtro cuenta con un menú desplegable que permite al usuario seleccionar una o varias opciones entre las posibilidades de filtrado. Como ejemplo, la figura 4.29 muestra el filtro por periodo.

**Buscar y filtrar materias**  
Las materias mostradas satisfacen tanto los filtros como la búsqueda.

Buscar por nombre de materia:

Filtrar por periodo:

Filtrar por código de área:

Filtrar por estado:

Figura 4.28: Componente de búsqueda y filtros de materias.

Figura 4.29: Menú desplegable de filtro por periodo en el componente de búsqueda y filtros de materias.

Debajo de ese componente, se encuentra la tabla con el listado de materias, que está en la figura 4.30. Para cada materia, se muestra el periodo en el que se cursó, el código, el nombre, el número de créditos, la calificación parcial obtenida por el estudiante en el 30 % de la materia, la nota final alcanzada y el estado de la materia. Adicionalmente, las materias que se cursaron de forma virtual están marcadas con un ícono que lo indica. La tabla se puede ordenar por cualquiera de las columnas, permitiendo al usuario visualizar la información de la forma que prefiera, en adición a los filtros y la barra de búsqueda.

Convenciones:		Curso virtual	Aprobado	Retirado	Reprobado	
Periodo	Código	Nombre del curso	Créditos	Nota 30%	Nota final	Estado
2023-10	DEPO-1228	Selección Natación	1		Aprobado	
2023-10	ISIS-2007	Diseño Prod. e Innovación Ti	3	4.51	5.00	
2023-10	ISIS-2503	Arquitectura y Diseño de Sw	3	4.61	5.00	
2023-10	MATE-3712	Teoría de Juegos	3	5.00	4.90	
2023-10	ISIS-2203	Infraestructura Computacional	3	4.44	5.00	
2023-10	ISIS-2403	Arquitectura Empresarial	3	4.63	5.00	
2023-10	ISIS-3302	Modelado, Simulación y Optimiz	3	4.79	4.94	
2023-10	CBCO-1345	Los Animales en Colombia	2		Retirado	

Figura 4.30: Tabla de materias inscritas, filtrada para el periodo 2023-10.

Esta pestaña permite varios casos de uso distintos sin necesidad de contar con muchos componentes. A continuación, se describen algunos de los más comunes:

- Consultar si un estudiante ha cursado o no una materia específica (haciendo uso de la barra de búsqueda) y, en caso de haberla cursado, ver el resultado obtenido.

- Visualizar las calificaciones de un estudiante en un área de conocimiento específica, por ejemplo, matemáticas (haciendo uso del filtro por código de área de la materia, en este caso, la opción MATE).
- Ver cuántas y cuáles materias han sido reprobadas o retiradas por un estudiante (haciendo uso del filtro por estado de la materia).
- Ver las materias que el estudiante cursó en un periodo académico específico y sus resultados, o ver las que se encuentra cursando en el periodo actual (haciendo uso del filtro por periodo).
- Consultar todas las materias inscritas por el estudiante en un periodo académico específico.
- Conocer las materias en las que el estudiante ha tenido mejor o peor desempeño, ordenando la tabla por la columna de nota final.

### Pestaña de Financiación

La pestaña de Financiación presenta información detallada sobre cómo el estudiante ha financiado sus estudios en la Universidad. Esta pestaña solo es accesible para los directivos de la Universidad, quienes pueden hallarla útil para la toma de decisiones relacionadas con el otorgamiento de apoyos financieros.

La pestaña contiene dos piezas fundamentales de información:

- El estrato socioeconómico del estudiante. En Colombia, “la estratificación socioeconómica es una clasificación en estratos de los inmuebles residenciales que deben recibir servicios públicos. Se realiza principalmente para cobrar de manera diferencial por estratos los servicios públicos domiciliarios permitiendo asignar subsidios y cobrar contribuciones en esta área.” [4]. A causa de eso, es uno de muchos indicadores importantes para determinar la capacidad de pago de un estudiante y, por ende, su necesidad de apoyo financiero.
- Un gráfico de barras apiladas con la distribución de la financiación del estudiante en cada periodo académico en el que se ha matriculado. Las barras se dividen en segmentos que representan las distintas fuentes de financiación, como becas, créditos, ahorros, recursos propios, entre otros. Cada segmento se colorea de forma distinta y se etiqueta con el porcentaje de financiación que representa para el periodo en cuestión. La altura de la barra representa el total de la financiación en cada periodo académico, que no necesariamente corresponde al 100 % de la matrícula, pues el estudiante puede haber cursado matrículas parciales.

Un ejemplo de la pestaña de Financiación se muestra en la figura 4.31.

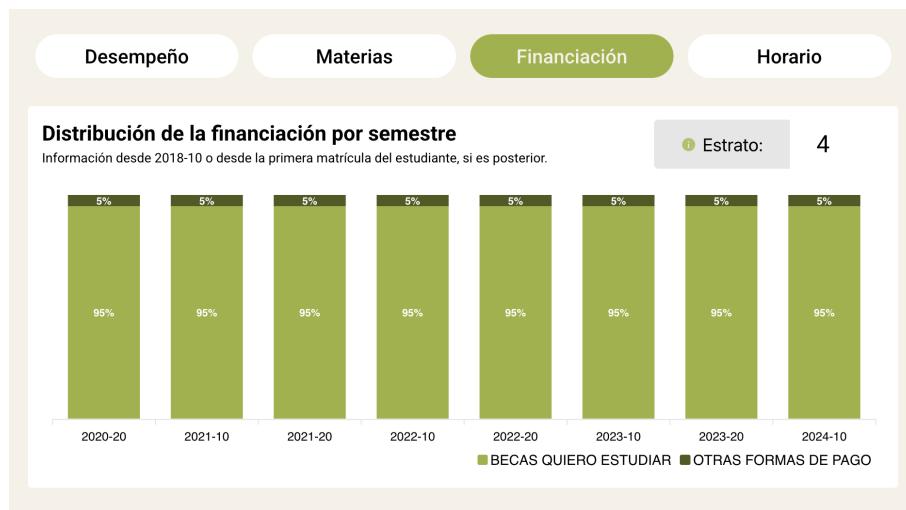


Figura 4.31: Pestaña de Financiación.

### Pestaña de Horario

La pestaña de Horario presenta el horario de clases del estudiante en el momento de la consulta. El horario se muestra en un formato clásico de horario de clases, como se puede evidenciar en la figura 4.32, donde cada columna representa un día de la semana y cada fila representa una franja de media hora. Cada celda de la tabla contiene la información de la materia que el estudiante tiene programada en ese día y a esa hora.

	dom 19/1	lun 20/1	mar 21/1	mié 22/1	jue 23/1	vie 24/1	sáb 25/1
5:30							
6:30							
7:30							
8:30							
9:30			9:30 - 10:50 PROGRAMACION CON TECNOLOGIAS	9:30 - 10:50 INFRAESTRUCTURA DE COMUNICACIONES	9:30 - 10:50 PROGRAMACION CON TECNOLOGIAS		
10:30							
11:30			12:00 - 13:50 HISTORIA(S) DEL ARTE EN OCCIDENTE		12:30 - 13:50 LABORATORIO DE INFRAESTRUCTURA DE		
12:30							
13:30			14:00 - 15:20 ANALISIS DE ALGORITMOS		14:00 - 15:20 ANALISIS DE ALGORITMOS		
14:30							
15:30							
16:30							
17:30				17:30 - 18:50 DISEÑO Y CONSTRUCCION DE APLICACIONES			
18:30							
19:30							
20:30							

Figura 4.32: Horario de clases de un estudiante anónimo.

Al pulsar en una celda de la tabla, se despliega una ventana emergente con información detallada de la materia, incluyendo el código de la materia, el nombre de la materia, el nombre del profesor, el salón de clases y el tipo de clase (magistral, complementaria, laboratorio, taller, entre otros). En caso de que el estudiante tenga una materia virtual programada, aparece un ícono que lo indica. Se muestra la información de una materia en la figura 4.33.



Figura 4.33: Detalle de una materia en el horario de un estudiante anónimo.

Esta pestaña es naturalmente útil para los estudiantes, particularmente al inicio de cada semestre, cuando deben organizar su tiempo y coordinar sus actividades. Sumado a

eso, la pestaña es de gran utilidad para los profesores, quienes pueden visualizar el horario de sus estudiantes y coordinar reuniones, tutorías y otros eventos con ellos. Para servir a ese fin, el horario cambia dinámicamente cada semana como respuesta a modificaciones en los horarios de clase, por ejemplo, a causa de días festivos o de eventos organizados por la Universidad.

### Información específica para estudiantes en riesgo académico

En el diseño del Perfil del estudiante, se incorporó una funcionalidad específica para alertar sobre estudiantes que cumplen algún criterio de riesgo académico. Hasta el momento, se reportan cinco riesgos académicos, que son los siguientes:

- El estudiante está en segundo semestre y tiene un PGA entre 3.00 y 3.30.
- El estudiante está en tercer semestre o superior y tiene un PGA entre 3.25 y 3.50.
- El estudiante pagó matrícula completa, pero tiene inscritos menos de 13 créditos.
- El estudiante ha retirado  $x$  materias en el semestre actual.
- El estudiante va perdiendo  $x$  materias al corte del 30 %.

En los últimos dos criterios, la  $x$  corresponde al número de materias en cuestión, de acuerdo con la realidad del estudiante. Esos dos criterios se muestran únicamente en caso de que la cantidad de materias sea de 3 o más. Ese valor es parametrizable y puede ser ajustado a conveniencia por los directivos de la Universidad.

En caso de que el alumno satisfaga alguno de esos criterios, se presenta un mensaje de alerta en la parte superior de la pestaña de Desempeño, indicando con gran visibilidad el riesgo académico que cumple. La figura 4.34 presenta un ejemplo de un estudiante con riesgo académico. Estos riesgos se muestran a cualquier profesor o administrativo que consulte el perfil, así como también al estudiante cuando consulta su propio perfil.



Figura 4.34: Mensaje de alerta de riesgo académico en el perfil de un estudiante anónimo.

### Información específica para estudiantes graduados

En el Perfil del estudiante se agregó información específica para estudiantes graduados. En la pestaña de Desempeño, se presenta un mensaje que exalta el logro de haberse graduado y exhibe el título obtenido, con el mes y año en el que se obtuvo. En caso de que el alumno haya completado alguna opción académica, se registra la compleción de la opción, como se puede ver en la figura 4.35.

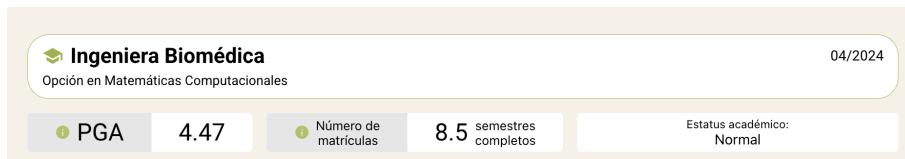


Figura 4.35: Mensaje de graduación en el perfil de un estudiante anónimo.

Sumado a eso, en caso de que haya sido galardonado durante la carrera o se haya graduado con honores, se resalta ese logro. Un ejemplo de un estudiante graduado con honores está en la figura 4.36.

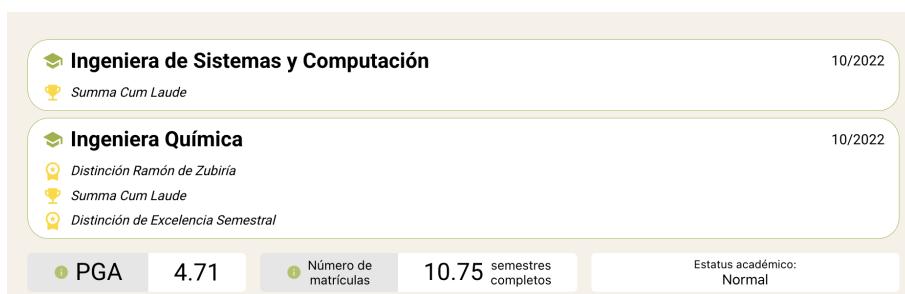


Figura 4.36: Mensaje de graduación con honores en el perfil de un estudiante anónimo.



# **Capítulo 5**

## **Diseño Conceptual**

### **5.1. Principios de Exposición de Información**

Explica los principios fundamentales que guiaron el diseño del Perfil del estudiante. Incluye claridad, relevancia, simplicidad, y cómo se aseguraron de que la información sea comprensible y útil para los diferentes usuarios.

### **5.2. Visualizaciones y Tablas**

Describe las decisiones tomadas para las visualizaciones y tablas interactivas, incluyendo qué información se priorizó y cómo se seleccionaron las herramientas o librerías para representarlas.



# Apéndice A

## Contribuciones al CAPP

Este apéndice describe las contribuciones realizadas al CAPP durante el desarrollo de este proyecto. En particular, se detallan los recursos expuestos por la API REST para consumo exclusivo del CAPP, a petición de Daniel Arango Cruz y Marilyn Stephany Joven Fonseca, quienes lideran el desarrollo de aquel proyecto.

### A.1. Contexto

El CAPP es un sistema que permite a los estudiantes de pregrado de la Universidad de los Andes visualizar su progreso académico. Su propósito principal es determinar si un estudiante ha cumplido con los requisitos mínimos para graduarse de un programa académico. Para ello, el CAPP se conecta con diversas fuentes de datos, entre ellas, con el API REST diseñado y desarrollado en este proyecto.

### A.2. Recursos expuestos

Los recursos provistos fueron diseñados específicamente para el CAPP ante solicitud de los desarrolladores. Se solicitó que la nomenclatura para los recursos y sus atributos fuera en español, en lugar de inglés, como se había manejado en el resto de la API REST.

A causa de eso y como forma de separación semántica, se decidió agrupar los *endpoints* bajo el prefijo `/capp/`. La tabla A.1 detalla los *endpoints* expuestos por la API REST para el CAPP.

**Tabla A.1***Recursos expuestos por la API REST para el CAPP*

Método HTTP	URL	Descripción	Cuerpo de la petición	Respuesta
GET	/capp /materias /estudiante /{codigo estudiante}	Obtiene todas las materias aprobadas por un estudiante en sus estudios de pregrado.	–	Lista de objetos Materia con las materias aprobadas por el estudiante.
GET	/capp /materia /{codigo materia}	Obtiene la información básica de una materia a partir de su código.	–	Objeto Materia correspondiente al código.
PUT	/capp /materias /	Obtiene la información básica de múltiples materias a partir de una lista de sus códigos.	Lista de códigos de materias	Lista de objetos Materia correspondientes a los códigos provistos.
GET	/capp /estudiante /programa /	Obtiene la información de los programas académicos cursados por el estudiante.	–	Lista de nombres de los programas académicos, ordenados desde el principal.

El recurso Materia mencionado en la tabla es una representación simplificada de una materia, que contiene la información mínima necesaria para el CAPP. La figura A.1 muestra un ejemplo de esta representación. No se hace uso de la representación SubjectResponse mencionada en la sección ??, ya que esta contiene información adicional y se incuraría en sobreexposición de datos.

```
{
  "codigo_curso": "MATE",
  "numero_curso": "1104",
  "numero_creditos": 3,
  "nombre_curso": "TEORIA DE GRAFOS",
  "atributo_sec": "",
  "periodo": 202320
}
```

Figura A.1: Ejemplo de representación del recurso Materia para el CAPP

## **Apéndice B**

### **Apuntes para el Ecosistema de analítica institucional**

Este apéndice contiene apuntes y recomendaciones para el Ecosistema de analítica institucional de la Universidad de los Andes. Estas recomendaciones se basan en la experiencia adquirida durante el desarrollo de este proyecto y en la revisión de literatura sobre calidad de datos y buenas prácticas en la arquitectura de malla, que es la utilizada por el Ecosistema.



# Glosario

**API (*Application Programming Interface*)** Mecanismo que permite la interoperabilidad de aplicaciones heterogéneas, permitiendo a un programa acceder a las funciones y datos de otro. 19–23, 27–29, 32, 33, 35–38, 42, 73–76

**API REST** API que sigue los principios arquitectónicos REST, permitiendo interacciones con los recursos usando métodos estándar HTTP. 11, 16, 19, 20, 27–30, 32, 34, 40–42, 69, 70

**axios** Biblioteca de JavaScript que facilita realizar peticiones HTTP desde el navegador o desde Node.js. Más información en <https://axios-http.com>. 40

**Azure Blob Storage** Servicio de almacenamiento de objetos en la nube de Microsoft Azure. Más información en su página oficial. 15

**black** Librería o utilidad de Python que formatea automáticamente el código fuente usando un estilo consistente con cualquier otro código fuente formateado con black, de acuerdo con su página de web. 36, 37

**BLOB (*Binary Large Object*)** Gran objeto binario, un tipo de dato que se utiliza para almacenar cualquier tipo de información binaria, usualmente de gran tamaño, como imágenes, videos o archivos. 32

**CAPP (*Curriculum, Asesoría y Planeación de Programas*)** Sistema de información de la Universidad de los Andes que determina si un estudiante satisface los requisitos para graduarse de un programa académico. 69, 70

**CRUD (*Create, Read, Update, Delete*)** Conjunto de operaciones básicas que se pueden realizar sobre un recurso en un sistema de información: crear, leer, actualizar y eliminar. 34, 35

**Docker** Plataforma de código abierto que permite a los desarrolladores empaquetar, distribuir y ejecutar aplicaciones en contenedores. Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias, de modo que la aplicación se ejecute de la misma manera en cualquier entorno. Más información en <https://www.docker.com>. 37, 38

**docstring** Cadena de texto especiales que se escriben al inicio de un módulo, clase, método o función en Python, que describe su funcionalidad y cómo usarlo. 36, 37

**ETL (*Extract, Transform, Load*)** Proceso utilizado en la integración de datos, que consiste en extraer datos de múltiples fuentes, realizar sobre ellos las transformaciones necesarias y cargarlos en un destino. 14, 16

**FastAPI** Framework moderno, rápido (de alto rendimiento) y fácil de usar para construir APIs con Python basado en las anotaciones de tipo estándar de Python, de acuerdo con su documentación. 27–29, 31, 35, 36

**Firebase** Plataforma de desarrollo de aplicaciones móviles y web desarrollada por Google. Proporciona una base de datos en tiempo real, autenticación de usuarios, almacenamiento en la nube y funciones de backend. Más información en <https://firebase.google.com>. 40

**Flake8** Herramienta de Python que combina varios linters para verificar el cumplimiento de las guías de estilo de Python. Más información en su documentación. 36, 37

**formateador** Herramienta que modifica automáticamente el código fuente para que siga convenciones de estilo predefinidas. 36

**frontend** Parte de una aplicación que interactúa con el usuario, generalmente a través de una interfaz gráfica. 20, 39

**Git** Sistema de control de versiones distribuido. 37

**GitHub Actions** Servicio de integración continua de GitHub que permite automatizar flujos de trabajo como pruebas y despliegues en repositorios de código fuente. Más información en su documentación. 37

**hook** En Git, es un script que se ejecuta automáticamente en momentos específicos del flujo de trabajo, como antes de hacer un commit o un push. 37

**HTTP (*Hypertext Transfer Protocol*)** Protocolo de comunicación que permite la transferencia de información en la web, caracterizado por operar sin estado y basado en el modelo cliente-servidor. 19, 20, 34, 35, 73, 76

**isort** Librería o utilidad de Python que alfabetiza las importaciones en el código fuente y las separa por tipo, de acuerdo con su página de web. 36, 37

**JavaScript** Lenguaje de programación de alto nivel, interpretado y orientado a objetos, ampliamente utilizado en el desarrollo web. Más información en <https://developer.mozilla.org/es/docs/Web/JavaScript>. 39, 73, 75, 76

**JSON (*JavaScript Object Notation*)** Notación de objetos de JavaScript, un formato ligero de intercambio de datos que es fácil de leer y escribir para humanos y fácil de analizar y generar para máquinas. 20

**Jupyter** Software de código abierto que permite crear y compartir documentos interactivos que contienen código, visualizaciones y texto explicativo. Más información en su página oficial. 14, 15

**linter** Herramienta que analiza el código fuente para encontrar errores, malas prácticas o incumplimientos de convenciones de estilo. 36

**Markdown** Lenguaje de marcado ligero que se convierte en HTML y se utiliza comúnmente para escribir documentación en repositorios de código fuente.. 30

**middleware** Software que actúa como intermediario entre dos aplicaciones o sistemas, permitiendo la comunicación entre ellos. 31

**NES (No Estás Solo)** Plataforma web de la Universidad de los Andes para el éxito estudiantil. <https://noestassolo.virtual.uniandes.edu.co/>. 9–11, 39, 42–44

**Node.js** Entorno de ejecución de JavaScript de código abierto que permite ejecutar código JavaScript en el servidor. Más información en <https://nodejs.org>. 73

**NumPy** “El paquete fundamental para la computación científica en Python”, de acuerdo con su página oficial. Librería de Python que añade soporte para arreglos y matrices de gran tamaño, junto con una colección de funciones matemáticas para operarlos. 37

**OpenAPI** Especificación para describir y documentar APIs de manera estandarizada, permitiendo la generación automática de documentación y la validación de APIs. Más información en su especificación oficial. 76

**ORM (*Object-Relational Mapping*)** Patrón de programación que permite representar objetos como tablas en una base de datos relacional y viceversa, trazando una relación entre las disímiles estructuras de la capa de lógica (orientada a objetos) y la capa de datos (relacional), problema que se conoce como *impedancia del mapeo* (del inglés, *object-relational impedance mismatch*). 32, 76

**Pandas** Librería de análisis y manipulación de datos en Python. Más información en su página oficial. 37

**PEP (*Python Enhancement Proposal*)** Documento que propone y describe una nueva característica o modificación en Python. En la PEP 1 se define qué es un PEP. 36

**perfil de calidad** Conjunto de reglas y configuraciones que definen los estándares de calidad del código. 36

**pipeline** Flujo de automatización en el que se definen una serie de pasos o tareas que deben ejecutarse de manera secuencial o en paralelo. 14, 37

**PostgreSQL** Sistema de gestión de bases de datos relacional de código abierto. Más información en <https://www.postgresql.org>. 38

**pruebas unitarias** Pruebas automáticas que se enfocan en verificar el correcto funcionamiento de las unidades más pequeñas de un programa, como funciones o métodos individuales. 31, 37

**Pydantic** Librería de Python que facilita la validación y serialización de datos, especialmente útil para trabajar con datos JSON. Más información en su documentación. 29, 33, 34, 36

**Pytest** Framework de pruebas para Python. Más información en su página oficial. 37

**Python** Lenguaje de programación de propósito general y alto nivel, interpretado y multiparadigma, aunque principalmente orientado a objetos. 14, 20, 27–29, 31–33, 36, 37, 74, 76

**React** Biblioteca de JavaScript de código abierto para construir interfaces de usuario, desarrollada por Facebook. Más información en <https://reactjs.org>. 39, 76

**Recoil** Biblioteca de gestión de estado para aplicaciones React, desarrollada por Facebook. Más información en <https://recoiljs.org>. 40

**Redoc** Herramienta para renderizar documentación de APIs en formato OpenAPI con un diseño limpio y personalizable, orientada a la presentación profesional de la documentación de APIs. Más información en su documentación. 29

**REST (*Representational State Transfer*)** Estilo arquitectónico para sistemas hipermedia distribuidos, como la *World Wide Web*. Se basa en la transferencia de representaciones de recursos, que son identificados por URIs, y la manipulación de estos recursos mediante métodos estándar de HTTP. Para una descripción detallada, leer el quinto capítulo de la tesis doctoral de Roy Fielding. 19, 73

**sobreexposición** Fenómeno que ocurre cuando una API expone más información de la necesaria, lo que puede llevar a problemas de seguridad y a una mayor complejidad en el mantenimiento. 33

**SonarLint** Extensión de análisis estático de código que se integra en el editor o IDE para detectar problemas de calidad en el código y proponer correcciones. 36

**SonarQube** Plataforma que realiza análisis estático de código para detectar vulnerabilidades, errores y problemas de mantenimiento en proyectos de software. 31, 36, 37

**SQLAlchemy** Librería de Python que facilita la interacción con bases de datos relacionales mediante un ORM. Más información en su página oficial. 32

**Swagger UI** Herramienta que permite visualizar y probar APIs de manera interactiva utilizando documentación generada automáticamente en formato OpenAPI. Es especialmente útil para explorar y probar endpoints durante el desarrollo. Más información en su documentación. 29

**TypeScript** Lenguaje de programación de código abierto desarrollado por Microsoft que añade tipado estático a JavaScript. Más información en <https://www.typescriptlang.org>. 39, 40

**UML (*Unified Modeling Language*)** Lenguaje unificado de modelado, un lenguaje estándar para visualizar, especificar, construir y documentar los artefactos de un sistema de software. 20

**URI (*Uniform Resource Identifier*)** Cadena de caracteres con estructura fija que identifica un recurso en la web de forma única. 19, 76



# Bibliografía

- [1] Universidad de los Andes. *Ecosistema de analítica institucional*. Consultado el 20 de noviembre de 2024. 2024. URL: <https://uniandes.sharepoint.com/sites/gobierno-de-datos/SitePages/Estrategia-de-datos.aspx>.
- [2] Universidad de los Andes. *ISIS-1221 - Introducción a la Programación*. Accedido: 2025-01-05. 2024. URL: <https://uniandes.smartcatalogiq.com/2024/catalogo/cursos/isis/1000/isis-1221/>.
- [3] Universidad de los Andes. *Oferta de Cursos*. Consultado el 20 de noviembre de 2024. URL: <https://ofertadecursos.uniandes.edu.co/>.
- [4] Departamento Administrativo Nacional de Estadística (DANE). *Estratificación Socioeconómica*. Accessed: 2025-01-21. 2025. URL: <https://www.dane.gov.co/index.php/servicios-al-ciudadano/servicios-informacion/estratificacion-socioeconomica>.
- [5] Ellucian. *Ellucian Banner*. Consultado el 20 de noviembre de 2024. URL: <https://www.ellucian.com/es/soluciones/ellucian-banner>.
- [6] Ellucian. *Ellucian CRM Advise*. Consultado el 20 de noviembre de 2024. URL: <https://www.ellucian.com/solutions/ellucian-crm-advise>.
- [7] Python Software Foundation. *What is Python?* 2025. URL: <https://docs.python.org/3/faq/general.html#what-is-python>.
- [8] Mark Lutz. *Learning Python*. 5th Edition. Sebastopol, CA: O'Reilly Media, 2013. ISBN: 978-1449355739. URL: <https://www.oreilly.com/library/view/learning-python-5th/9781449355722/>.
- [9] Sebastián Ramírez. *FastAPI*. Accessed: 2025-01-05. 2025. URL: <https://fastapi.tiangolo.com/>.
- [10] Sebastián Ramírez. *FastAPI: Features*. Accessed: 2025-01-05. 2025. URL: <https://fastapi.tiangolo.com/features/>.
- [11] Sebastián Ramírez. *FastAPI: Performance Benchmarks*. Accessed: 2025-01-05. 2025. URL: <https://fastapi.tiangolo.com/benchmarks/>.