Programación Digital Avanzada

SEDE: FRAY BENTOS

INGENIERÍA BIOMÉDICA

Nombre del trabajo: ***Sistema de matriculación v.1***

Autor: *Iturriaga Julia.*
Pereyra Magalí.

Fecha de entrega: **25/04/2025**

Índice

1. Introducción

En el presente informe se describe la implementación de un Sistema de Matriculaciones (SdM) orientado a la carrera de Ingeniería Biomédica (IBIO), inicialmente probado con el plan de estudios 2021. El objetivo principal del sistema es automatizar el control de previaturas necesarias para la inscripción a Unidades Curriculares (UCs) o exámenes, con el fin de reducir el trabajo administrativo manual y minimizar errores en el proceso de matriculación. El desarrollo del sistema se realizó en Python, aplicando principios de Programación Orientada a Objetos (POO).

Una de las características destacadas del SdM es su diseño flexible, que permite trabajar con múltiples planes de estudio. Los planes se cargan a través de archivos en formato JSON, lo que facilita su actualización o extensión a otros años académicos. Si bien durante las pruebas se utilizó el plan 2021, el sistema está preparado para gestionar diferentes cohortes de estudiantes matriculados en planes distintos de forma simultánea. Esta funcionalidad está pensada para facilitar el trabajo tanto de la coordinación como de la secretaría académica.

2. Estructura general

En nuestro diseño optamos por concentrar la mayor parte de la lógica del sistema en clases que constituyen el núcleo estructural del modelo y no representan directamente a personas. Estas clases son: UnidadCurricular, PlanDeEstudio, Curso e InstanciaDeExamen. A lo largo de la implementación, estas clases acumularon la lógica principal del sistema: relaciones entre materias, verificación de previaturas, listado de inscripciones y reglas de matriculación definidas según el plan de estudios.

Por otra parte, las clases Estudiante, Secretaria y Coordinador funcionan como fachadas o interfaces de interacción con el sistema. Estas clases no contienen lógica propia, sino que delegan sus funcionalidades al sistema, utilizando métodos definidos en el núcleo del sistema. De esta manera, cada actor accede únicamente a las funciones necesarias (como matricular, desmatricular o consultar inscripciones), mientras que la validación y el control permanecen centralizados en el núcleo lógico del sistema.

Esta separación surgió de manera natural mientras escribíamos el código, y nos resultó útil para mantener organizadas las responsabilidades. Por ejemplo, Estudiante tiene métodos como inscribirse_a_uc() que en realidad llaman a funciones internas de Curso, que es el que evalúa si cumple con las condiciones. Lo mismo ocurre con Secretaria, que puede

matricular o desmatricular estudiantes, pero sin tener que conocer ni aplicar directamente las reglas de previaturas.

Además, el sistema genera archivos CSV con información relevante para la gestión académica, como los listados de estudiantes matriculados a cada curso y las inscripciones a instancias de examen. Esto permite a la coordinación y a la secretaría disponer de datos organizados en formatos exportables y fácilmente integrables a otros sistemas o documentos institucionales.

En pocas palabras, elegimos estructurar la implementación de modo que las clases que representan personas (Estudiante, Secretaria, Coordinador) sean las que usan el sistema, mientras que la lógica del dominio esté concentrada en clases como Cursos, InstanciaDeExamen, UnidadCurricular y PlanDeEstudio, que funcionan como motor interno.

Fig 1. Analogía del funcionamiento del SdM.

Imaginemos que las clases Curso, UnidadCurricular, InstanciaDeExamen, PlanDeEstudio son un SdM, que es una casa con muchas salas, donde cada sala representa una funcionalidad: cargar un plan, inscribirse, ver listas, generar CSV, etc.

Cada actor (Coordinadora, Secretaria, Estudiante) tiene una llave distinta, que le permite acceder solo a las salas necesarias para cumplir su función.

- La Coordinadora tiene la llave maestra: puede entrar a casi todas las salas. Puede cargar planes, crear instancias de exámenes o cursos, y modificar estados de estudiantes.

- La Secretaria tiene acceso a salas administrativas: puede matricular y desmatricular estudiantes, consultar inscripciones, y generar listas en CSV.

- El Estudiante tiene una llave limitada: puede acceder a su panel personal, donde puede inscribirse a materias y exámenes si cumple con las condiciones, y ver sus propias inscripciones.

Pero ninguno de ellos construye la casa ni cambia cómo funcionan las salas. Lo único que hacen es usar sus llaves para pedirle al sistema (la casa) que haga algo. Todo lo que ocurre adentro (las reglas, las validaciones, las decisiones) lo maneja la estructura de la casa: las puertas automáticas que se abren si tenés la llave correcta, las reglas de circulación internas, etc.

2.2 Diagrama UML

Luego de establecer cómo queríamos que funcionara el sistema desde una perspectiva funcional, avanzamos hacia una etapa de diseño más formal. En esta fase, elaboramos los diagramas UML que representan la arquitectura del sistema, definiendo las clases que lo componen, sus métodos y las relaciones entre ellas. Este paso nos permitió consolidar la estructura lógica del sistema antes de comenzar con la implementación.

Relaciones internas en el núcleo del sistema:

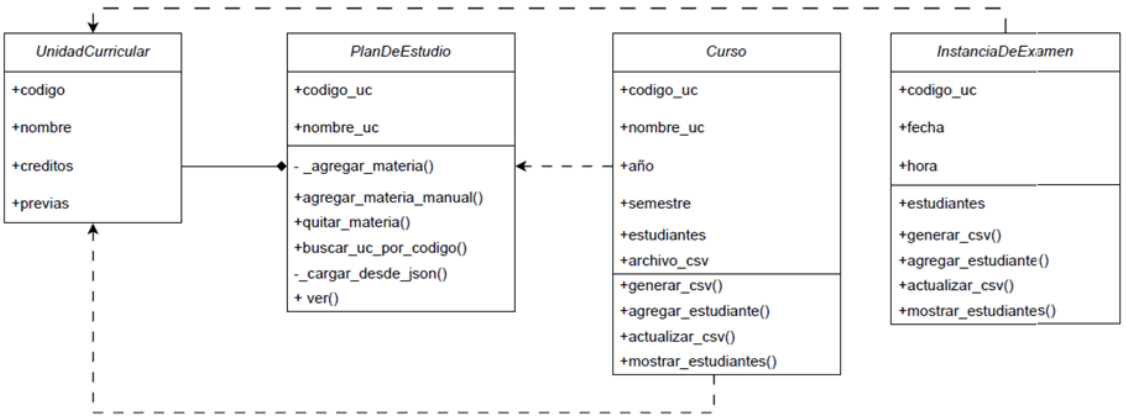

Fig. 2 UML del núcleo del sistema.

Relación entre los actores del sistema:

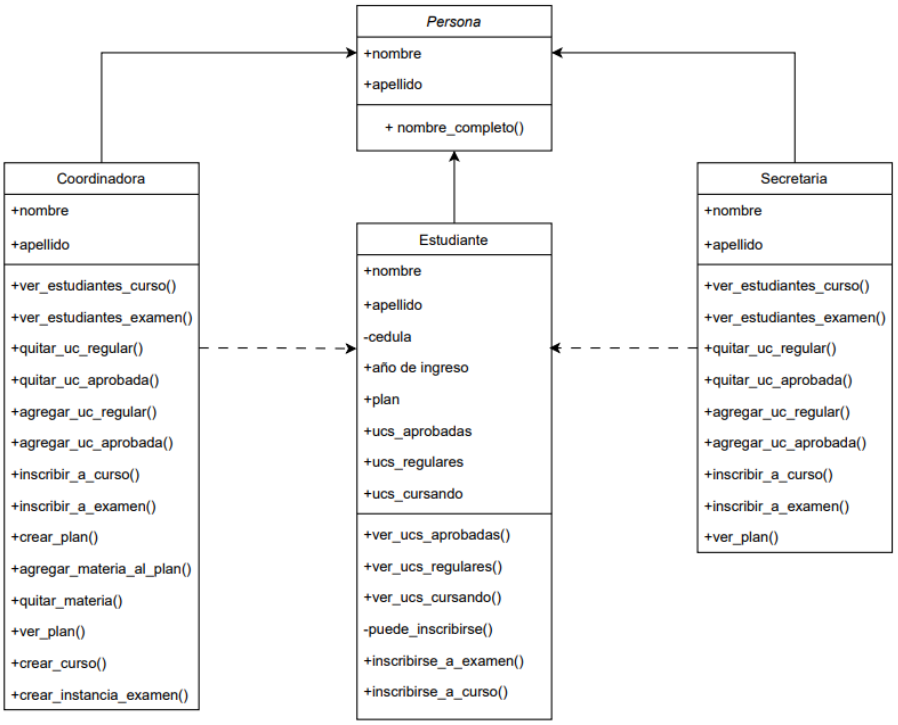

Fig 3. UML de los actores del sistema.

Interacción entre los actores y el núcleo del sistema:

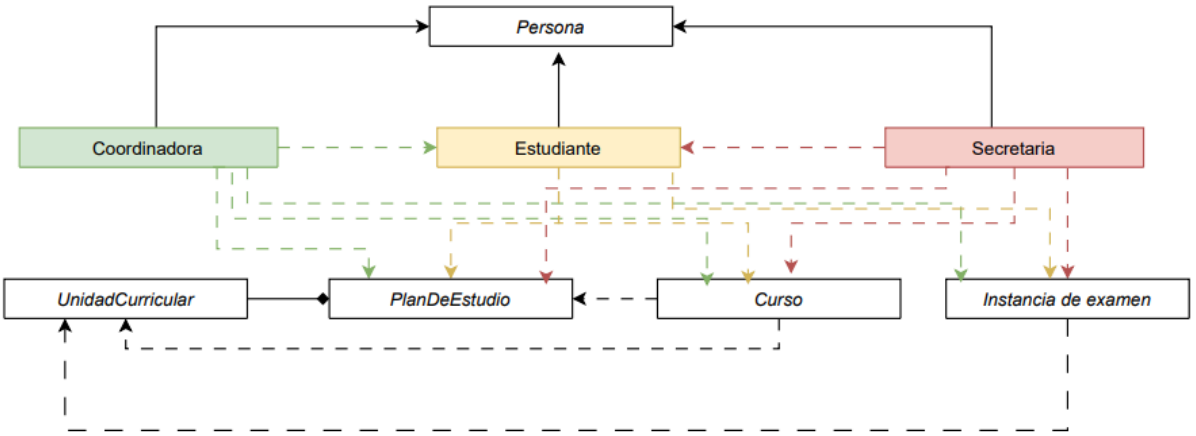

Fig 4. UML de la relación entre actores y el núcleo.

2.3 Núcleo del sistema

2.3.1 Unidad Curricular

La clase UnidadCurricular almacena en sus atributos información sobre una materia, incluyendo su código, nombre, créditos y las materias previas necesarias para cursarla.

2.3.2 Plan de Estudio

La clase PlanDeEstudio representa una estructura académica que agrupa múltiples materias (unidades curriculares) que componen un plan de estudios universitario. Su objetivo principal es organizar, gestionar y visualizar dichas materias, permitiendo además cargarlas automáticamente desde un archivo en formato JSON.

Atributos:

- nombre_plan: El nombre del plan de estudio.
- materias (list[UnidadCurricular]): Lista que contiene los objetos UnidadCurricular que forman parte del plan.del plan. Cada materia es un objeto de la clase UnidadCurricular.

Métodos:

__init__(self, nombre_plan, ruta_json)

Constructor de la clase. Inicializa el nombre del plan y carga automáticamente las materias desde un archivo JSON.

_agregar_materia(self, unidad_curricular)

Método privado que incorpora un objeto UnidadCurricular a la lista de materias del plan. Se usa internamente al cargar desde JSON.

buscar_uc_por_codigo(self, codigo)

Busca una materia en el plan a partir de su código. Devuelve el objeto UnidadCurricular correspondiente si lo encuentra, o None en caso contrario.

_cargar_desde_json(self, ruta_archivo)

Método privado que lee un archivo JSON y convierte cada materia listada en un objeto UnidadCurricular, el cual es luego agregado al plan mediante _agregar_materia().

ver(self)

Muestra por consola el nombre del plan de estudio y las materias que contiene, incluyendo detalles como el nombre, los créditos y las materias previas.

2.3.3 Curso

La clase Curso representa una instancia específica en la que se dicta una unidad curricular dentro de un determinado año y semestre. Está diseñada para gestionar tanto los datos administrativos del curso como la inscripción de estudiantes, y mantener esta información sincronizada con un archivo CSV.

Atributos:

- codigo_uc (str): Código único de la unidad curricular (string).
- nombre_uc (str): Nombre de la unidad curricular (string).
- año (int): Año en que se imparte el curso (entero).
- semestre(int): Semestre en que se imparte el curso (entero, entre 1 y 10).
- estudiantes (list[Estudiante]): Lista de objetos de la clase Estudiante, donde se almacenan los estudiantes inscritos en el curso.
- archivo_csv (str): Nombre del archivo CSV que contiene la información del curso y estudiantes inscritos (string).
- uc (UnidadCurricular): Objeto recuperado del plan, que representa la unidad curricular asociada.

Métodos:

__init__(self, codigo_uc, año, semestre, plan)

Constructor que inicializa un curso. Realiza validaciones de tipo y contenido, verifica que el código de la UC exista en el PlanDeEstudio y que coincida con el semestre indicado. Si todo es correcto, crea el archivo CSV del curso.

curso_valido(self)

Devuelve True si el curso fue correctamente creado (es decir, si la unidad curricular fue hallada en el plan y pasó todas las validaciones). Usado como verificación previa en otros métodos.

generar_csv(self)

Genera un archivo CSV al momento de la creación del curso con la información base (sin estudiantes). Asegura trazabilidad y persistencia de los datos.

agregar_estudiante(self, estudiante)

Permite agregar un estudiante al curso, siempre que sea una instancia válida de Estudiante y no se encuentre ya registrado. Luego actualiza el archivo CSV correspondiente.

actualizar_csv(self)
Escribe nuevamente el archivo CSV con la información actualizada del curso y los nombres de los estudiantes inscritos. Es llamado automáticamente al agregar un estudiante.

mostrar_estudiantes(self)
Imprime en consola la lista de estudiantes actualmente registrados en el curso. Si no hay estudiantes o el curso no es válido, se indica apropiadamente.

2.3.4 InstanciaDeExamen

La clase InstanciaDeExamen modela una instancia específica de examen para una unidad curricular determinada, permitiendo la gestión de estudiantes inscritos y el almacenamiento persistente de la información en un archivo CSV.

Atributos:

- codigo_uc (str): Código único de la unidad curricular asociada al examen (por ejemplo, "UC2S1").
- fecha(str): Fecha en la que se realiza el examen (por ejemplo, "2025-11-03").
- hora (str): Hora programada del examen (por ejemplo, "14:00").
- estudiantes (list[Estudiante]): Lista de estudiantes inscritos en el examen. Solo pueden agregarse si están regulares en la unidad curricular correspondiente.

Métodos:

__init__(self, codigo_uc, fecha, hora)
Constructor de la clase. Inicializa el examen con la UC, fecha y hora especificadas. Crea automáticamente un archivo CSV asociado al examen llamando a generar_csv().

generar_csv(self)
Crea (o sobrescribe) un archivo CSV con la información básica del examen: código de la UC, fecha y hora, y lista de estudiantes (vacía en un inicio). El archivo se nombra como examen_{codigo_uc}_{fecha}.csv.

agregar_estudiante(self, estudiante) Permite agregar un estudiante al examen si cumple las siguientes condiciones:

-Es una instancia de la clase Estudiante.

-Se encuentra en estado "regular" en la unidad curricular del examen.

-No está previamente inscrito en esta instancia.

Al agregar un nuevo estudiante, se actualiza el archivo CSV correspondiente mediante actualizar_csv().

actualizar_csv(self)
Reescribe el archivo CSV del examen, incluyendo la lista actualizada de estudiantes inscritos. Este método se invoca automáticamente después de cada inscripción.

mostrar_estudiantes(self)
Imprime en consola la lista actual de estudiantes inscritos. Si no hay inscriptos, notifica que la lista está vacía.

2.3.5 Persona

La clase Persona representa una entidad básica con nombre y apellido. Es una clase fundamental que puede ser utilizada como clase base para otras entidades más específicas dentro del sistema, como Estudiante, Coordinadora o Secretaria.

Atributos:

nombre(str): Nombre de la persona. Debe ser una cadena no vacía y sin espacios en blanco solamente.

apellido(str): Apellido de la persona. También debe ser una cadena no vacía.

Ambos atributos son normalizados automáticamente para eliminar espacios innecesarios y capitalizar la primera letra de cada palabra (por ejemplo, "jUan" se transforma en "Juan").

Métodos:
__init__(self, nombre, apellido

Inicializa una nueva instancia de la clase Persona, validando que el nombre y el apellido sean cadenas no vacías. En caso contrario, lanza una excepción ValueError.
nombre_completo(self)
Retorna el nombre completo de la persona en el formato "Nombre Apellido".

2.3.6 Estudiante

La clase Estudiante representa a un estudiante dentro del sistema académico. Hereda de la clase Persona, por lo que posee nombre y apellido. Incorpora además atributos propios de un estudiante universitario, incluyendo su cédula, el año de ingreso, y un plan de estudios asociado.

Atributos:

- nombre, apellido: heredados de la clase Persona.
- cedula (int): Número de cédula del estudiante, validado como entero.
- año_ingreso (int): Año en que el estudiante ingresó a la carrera.
- plan (PlanDeEstudio): Plan de estudios al que pertenece el estudiante.
- ucs_aprobadas (list[UnidadCurricular]): Unidades curriculares que el estudiante ya aprobó.
- ucs_regulares (list[UnidadCurricular]): Unidades curriculares que el estudiante tiene regularizadas.
- ucs_cursando(list[UnidadCurricular]): Unidades curriculares que el estudiante está cursando actualmente.
- ucs_a_examen (list[UnidadCurricular]): Unidades a las que el estudiante se ha inscripto para rendir examen.

Métodos:

__init__(self, nombre, apellido, cedula, año_ingreso, plan)
Inicializa los atributos del estudiante, incluyendo la validación de la cédula como número entero. Asocia el estudiante a un plan de estudios y crea las listas que reflejan su estado académico.

@property cedula(self)
Propiedad que devuelve la cédula del estudiante (atributo protegido).

__str__(self)

devuelve una representación como string con nombre, apellido y cédula.

ver_ucs_aprobadas(self)
Imprime la lista de UCs que el estudiante ha aprobado. Si no tiene ninguna, lo indica.

ver_ucs_regulares(self)
Muestra las UCs que el estudiante tiene regularizadas.

ver_ucs_cursando(self)
Muestra las UCs que está cursando en el momento.

ver_plan(self)
Llama al método ver() del objeto PlanDeEstudio asociado, para mostrar el contenido del plan del estudiante.

_puede_inscribirse(self, codigo_uc)
Método privado que verifica si el estudiante puede inscribirse a una UC determinada. Valida que:

-No haya sido aprobada ni esté actualmente en curso.

-Se cumplan todas las previas requeridas. En caso contrario, informa las causas (por ejemplo, previas faltantes).

inscribirse_a_examen(self, instancia_examen)
Inscribe al estudiante en una instancia de examen si:

-La instancia recibida es de tipo InstanciaDeExamen.

-El estudiante está regular en la UC correspondiente.

No haya aprobado ya esa UC. Usa agregar_estudiante() de la clase InstanciaDeExamen para registrar la inscripción.

inscribirse_a_curso(self,curso)
Intenta inscribir al estudiante en un curso. El curso debe ser válido (curso.curso_valido()), y el estudiante debe cumplir con las condiciones de inscripción. Si es posible:

-Se agrega la UC a su lista de materias cursando.

-Se llama a agregar_estudiante(self) del curso para registrarlo y actualizar el CSV.

2.3.7 Coordinadora

La clase Coordinadora representa a una persona encargada de gestionar procesos académicos en una institución. Hereda de la clase Persona, por lo tanto también contiene nombre y apellido.

Su función principal es coordinar actividades como la creación de planes de estudio, gestión de cursos, inscripción de estudiantes y administración de exámenes.

Atributos:

- nombre, apellido: Heredados de la clase Persona.

Métodos:

Gestión de plan de estudios

crear_plan(nombre_plan, ruta_json)

Crea una instancia del PlanDeEstudio a partir del nombre del plan y una ruta al archivo JSON con la información de las materias. Imprime confirmación y retorna el objeto creado.

ver_plan(plan)

Muestra el contenido del plan de estudio, delegando en el método ver() del objeto PlanDeEstudio.

Cursos

crear_curso(codigo_uc, año, semestre, plan)

Crea una instancia de Curso utilizando los datos proporcionados. Si la UC es válida, retorna el curso; si no, informa el error y retorna None.

inscribir_a_curso(estudiante, curso)

Inscribe a un estudiante en un curso si cumple con las condiciones. Verifica:

-Que el estudiante sea instancia de Estudiante.

-Que el curso sea válido.

-Que el estudiante no haya aprobado ni esté cursando la UC, y que cumpla con las previas.

-Si cumple, se lo inscribe en el curso y se actualiza el registro.

ver_estudiantes_curso(curso)
Muestra la lista de estudiantes inscritos en el curso, llamando a curso.mostrar_estudiantes().

Exámenes

crear_instancia_examen(codigo_uc, fecha, hora, plan)

Busca la UC en el plan y, si existe, crea una nueva instancia de InstanciaDeExamen con los datos proporcionados. Si la UC no se encuentra, informa el error y retorna None.

inscribir_a_examen(estudiante, examen)
Verifica si el estudiante está regular en la UC del examen y, si es así, lo inscribe al mismo usando estudiante.inscribirse_a_examen(examen).

ver_estudiantes_examen(examen)
Muestra la lista de estudiantes inscritos en el examen, llamando a examen.mostrar_estudiantes().

Gestión del Estado Académico del Estudiante

agregar_uc_aprobada(estudiante, codigo_uc)
Agrega una UC a la lista de aprobadas del estudiante si aún no la tiene registrada como tal.

quitar_uc_aprobada(estudiante, codigo_uc)
Elimina una UC de la lista de aprobadas del estudiante si se encuentra allí.

agregar_uc_regular(estudiante, codigo_uc)
Agrega una UC a la lista de regulares del estudiante si aún no está presente.

quitar_uc_regular(estudiante, codigo_uc)
Elimina una UC de la lista de regulares si se encuentra en ella.

2.3.8 Secretaria

Hereda de Persona y comparte la mayoría de las operaciones académicas de la Coordinadora, a excepción de las funciones de creación de planes, cursos y exámenes. Su rol se centra en las tareas operativas de inscripción y gestión de estados de las UCs, pero no en la definición estructural del plan ni en la generación de instancias.

Métodos que comparte con Coordinadora:

inscribir_a_curso(estudiante, curso)
inscribir_a_examen(estudiante, examen)
ver_estudiantes_curso(curso)
ver_estudiantes_examen(examen)
agregar_uc_aprobada(estudiante, codigo_uc)
quitar_uc_aprobada(estudiante, codigo_uc)
agregar_uc_regular(estudiante, codigo_uc)
quitar_uc_regular(estudiante, codigo_uc)

De este modo, la Secretaria dispone de todas las herramientas para ejecutar inscripciones y modificaciones en el estado académico de los estudiantes, pero no puede definir ni estructurar nuevos planes, cursos o exámenes.

3. Principios de POO en la implementación

3.1 Herencia

Persona es la clase base de la que derivan Estudiante, Coordinadora y Secretaria, compartiendo atributos comunes (nombre, apellido) y comportamientos generales.

3.2 Encapsulamiento

Los métodos “privados” _cargar_desde_json() y _agregar_materia() en PlanDeEstudio y el atributo protegido _cedula en Estudiante con su propiedad cédula demuestran ocultamiento de detalles internos y control de acceso.

3.3 Polimorfismo

Todas las clases (Estudiante, Coordinadora, Secretaria) exponen un método ver_plan(), que internamente llama a PlanDeEstudio.ver().

-En Estudiante, ver_plan() muestra solo su plan asociado.

-En Coordinadora y Secretaria, ver_plan(plan) puede invocar la vista de cualquier PlanDeEstudio pasado como argumento.

3.4 Composición

–PlanDeEstudio contiene múltiples UnidadCurricular.
– Curso y InstanciaDeExamen guardan listas de Estudiante.
– Cada Estudiante referencia un PlanDeEstudio.

3.5 Abstracción

Métodos como PlanDeEstudio.desde_json(), Curso.generar_csv() o InstanciaDeExamen.actualizar_csv() ocultan la compleja lógica de carga de datos y persistencia, exponiendo interfaces sencillas.

4. Consideraciones a mejorar

Durante el desarrollo e implementación del sistema, identificamos algunos aspectos que podrían ser mejorados o añadidos en futuras versiones. Estas observaciones surgieron tanto del análisis del funcionamiento actual como de pensar en situaciones reales que podrían darse en un entorno universitario.

Ventana temporal para inscripciones: Actualmente, el sistema permite que un/a estudiante se inscriba a una unidad curricular en cualquier momento, sin ninguna restricción temporal. Sería deseable implementar una funcionalidad que permita definir ventanas de tiempo específicas para la inscripción a cursos o exámenes, como sucede en la realidad universitaria.

Fechas incoherentes en cursos y exámenes: En el estado actual, la coordinadora puede crear cursos o instancias de examen con cualquier fecha, incluso si esa fecha ya pasó. Este

comportamiento no genera errores, pero en la práctica podría generar confusión o inconsistencias. Sería conveniente implementar validaciones que impidan registrar eventos académicos con fechas que no tengan sentido (por ejemplo, en el pasado).

Privacidad de los datos: La información de los y las estudiantes (como nombre, materias cursadas, etc.) está accesible sin restricciones. Una posible mejora sería implementar niveles de acceso o permisos, de forma que no todas las clases o usuarios puedan ver o modificar toda la información.

Desinscripción de cursos: Actualmente, una vez que un/a estudiante se matricula a un curso, no hay forma automática de desinscribirse. Estaría bueno implementar una opción que permita a un/a estudiante cancelar su inscripción antes de que comience el curso o durante una ventana de tiempo razonable.