ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

APLICACIÓN DE SOFTWARE DE GESTIÓN DE HORARIOS ACADÉMICOS

APLICACIÓN DE AUTOMATIZACIÓN DE HORARIOS ACADÉMICOS (CAPA DE ALGORITMO DE EJECUCIÓN)

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN.

JOSEPH EMMANUEL REYES PÉREZ

joseph.reyes@epn.edu.ec

DIRECTOR: FRANKLIN LEONEL SÁNCHEZ CATOTA

franklin.sanchez@epn.edu.ec

DMQ, agosto 2023

CERTIFICACIONES

Yo, JOSEPH EMMANUEL REYES PÉREZ declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

_____

JOSEPH EMMANUEL REYES PÉREZ

Certifico que el presente trabajo de integración curricular fue desarrollado por JOSEPH EMMANUEL REYES PÉREZ, bajo mi supervisión.

_____

FRANKLIN LEONEL SÁNCHEZ CATOTA

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

JOSEPH EMMANUEL REYES PÉREZ

FRANKLIN LEONEL SÁNCHEZ CATOTA

DEDICATORIA

Quiero dedicar este trabajo a mis padres, Fernando y Cecilia por su amor y apoyo durante toda mi vida y por estar cuando más lo he necesitado.

A mis hermanos, Dennis, Cristian y Tahis por su respaldo inquebrantable el cual ha sido mi fuente constante de fuerza y alegría en las buenas y malas circunstancias. Siempre han sido mi motivación más grande.

A mi abuelita, que desde el cielo me observa con cariño y aunque no pudo presenciar este logro, sé que estaría orgullosa.

A mis amigos y compañeros de universidad que compartimos la misma ilusión de ser profesionales, estoy seguro que lo lograremos.

Joseph Reyes

AGRADECIMIENTO

Deseo expresar mi profundo agradecimiento a mis padres, cuyo sacrificio diario y esfuerzo incansable han permitido que pueda estudiar y perseguir mis metas.

A mis hermanos, quienes han sido mi fuente constante de inspiración y fortaleza, siempre a mi lado para respaldarme y motivarme en cada circunstancia, tanto en los momentos favorables como en los desafiantes.

A mis amigos, Javier y Laura, quienes han compartido este recorrido desde sus inicios, acompañándome durante numerosas horas de clases y experiencias.

A mi amigo y compañero de Trabajo de Integración, Samuel, cuya colaboración y compromiso fueron esenciales en el desarrollo de la aplicación.

Mi agradecimiento también se extiende al Ingeniero Franklin Sánchez, cuyo apoyo ha sido valioso a lo largo de la ejecución de este Trabajo de Integración.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El propósito central de este Trabajo de Integración Curricular es elaborar y poner en funcionamiento un algoritmo de automatización destinado a simplificar el procedimiento de asignación de aulas en la Facultad de Ingeniería Eléctrica y Electrónica de la Escuela Politécnica Nacional. Este algoritmo desempeñará un rol esencial como componente central del software de administración de horarios académicos, la aplicación tiene como objetivo facilitar todos los procesos de gestión y generación horarios mediante una interfaz con la que el usuario pueda interactuar y utilizar el algoritmo. Dicho algoritmo será parte integral de una API que interactuará con el frontend de la aplicación, procesando solicitudes y proporcionando una lista de aulas, según los criterios definidos por el usuario. En adición a su función principal, el algoritmo también ofrecerá la capacidad de verificar la disponibilidad de aulas, en respuesta a las solicitudes de los usuarios.

En el Capítulo 1 se detalla las herramientas seleccionadas para el desarrollo del algoritmo. Además, se lleva a cabo un estudio de los diversos paradigmas de diseño de algoritmos presentes en la programación.

En el Capítulo 2, se aborda el proceso completo de diseño e implementación del algoritmo. Esto incluye la obtención de los requisitos funcionales del algoritmo y la creación de las funcionalidades correspondientes. El énfasis está en la creación de un algoritmo efectivo y eficiente para la asignación de aulas, asegurando que satisfaga las necesidades de los administradores de horarios de la Facultad de Ingeniería Eléctrica y Electrónica.

Finalmente, el Capítulo 3 se enfoca en la verificación del cumplimiento de los requisitos y el logro de los objetivos fijados para el algoritmo. Se verifica la satisfacción de los usuarios finales, asegurando que el algoritmo funcione de manera adecuada y cumpla con su propósito. Esta etapa será crucial para determinar el éxito del algoritmo implementado.

Con esta estructura y enfoque metodológico, se busca desarrollar un algoritmo que simplifique la asignación de aulas en la Facultad de Ingeniería Eléctrica y Electrónica, contribuyendo significativamente al proceso de gestión de horarios académicos de la institución.

PALABRAS CLAVE: Python, FastAPI, Microsoft Visual Studio, Kanban, Algoritmo.

ABSTRACT

The central purpose of this Curricular Integration Work is to elaborate and put into operation an automation algorithm destined to simplify the procedure of assigning classrooms in the Faculty of Electrical and Electronic Engineering of the National Polytechnic School. This algorithm will play an essential role as a central component of the academic timetable management software. The application aims to facilitate all the timetable generation and management processes through an interface with which the user can interact and use the algorithm. Said algorithm will be an integral part of an API that will interact with the frontend of the application, processing requests and providing a list of classrooms, according to the criteria defined by the user. In addition to its main function, the algorithm will also offer the ability to verify the availability of classrooms, in response to user requests.

Chapter 1 details the tools selected for the development of the algorithm. In addition, a study of the various algorithm design paradigms present in programming is carried out.

In Chapter 2, the complete algorithm design and implementation process is discussed. This includes obtaining the functional requirements of the algorithm and creating the corresponding functionalities. The emphasis is on the creation of an effective and efficient algorithm for classroom assignment, ensuring that it meets the needs of the schedule administrators of the Faculty of Electrical and Electronic Engineering.

Finally, Chapter 3 focuses on the verification of compliance with the requirements and the achievement of the objectives set for the algorithm. The satisfaction of end users is verified, ensuring that the algorithm works properly and fulfills its purpose. This stage will be crucial to determine the success of the implemented algorithm.

With this structure and methodological approach, it seeks to develop an algorithm that simplifies the assignment of classrooms in the Faculty of Electrical and Electronic Engineering, contributing significantly to the process of managing academic schedules of the institution.

KEY WORDS: Python, FastAPI, Microsoft Visual Studio, Kanban, Algorithm.

1. INTRODUCCIÓN

La Escuela Politécnica Nacional tiene varios sistemas de gestión entre los cuales se puede resaltar el sistema SAEw y SII Académico. Estos sistemas se utilizan para organizar y estructurar los procesos administrativos de la universidad. No utilizan algoritmos de automatización de procedimientos, sino que se centran en proporcionar un repositorio centralizado de información. Esto permite a los usuarios consultar diferentes aspectos relacionados con la gestión de información de la institución, como la matrícula, las calificaciones y los datos de los empleados.

La administración de horarios de las aulas de la Facultad de Ingeniería Eléctrica y Electrónica es un proceso importante que se realiza de manera manual. Esto significa que no existe un algoritmo de automatización que facilite el procedimiento de asignación de aulas. Como resultado, la persona encargada de la asignación de aulas debe seguir un proceso ambiguo y poco eficiente cada vez que una asignatura abre un nuevo curso. El proceso consiste en comparar el horario de cada una de las aulas de la facultad de Ingeniería Eléctrica y Electrónica en una hoja de cálculo, con el objetivo de encontrar un espacio disponible en un aula que cumpla las condiciones necesarias para impartir la asignatura. Este proceso es lento, propenso a errores y consume mucho tiempo.

La implementación de un algoritmo de automatización para la asignación de aulas sería una mejora significativa para la facultad de Ingeniería Eléctrica y Electrónica. El algoritmo haría que el proceso fuera más eficiente, preciso y menos propenso a errores. Esto liberaría a la persona encargada de la asignación de aulas para que se concentre en otras tareas más importantes.

1.1. OBJETIVO GENERAL

Diseñar e implementar un algoritmo que permita automatizar el proceso de búsqueda de aulas, reduciendo errores y optimizando los recursos destinados al procedimiento de asignación de aulas disponibles dentro de la Facultad de Ingeniería Eléctrica y Electrónica.

1.2. OBJETIVOS ESPECÍFICOS

1. Investigar sobre los diversos paradigmas algorítmicos y seleccionar aquel que resulte más adecuado para el propósito del algoritmo de asignación de aulas.
2. Diseñar el algoritmo que permita la asignación automática de aulas, teniendo en cuenta los criterios y variables identificados.

3. Implementar el algoritmo de asignación de aulas utilizando el paradigma algorítmico apropiado y verificar su correcto funcionamiento a través de la utilización de un entorno de pruebas diseñado para simular situaciones de la vida real.
4. Verificar el cumplimiento de requerimientos y funcionalidades del algoritmo.

1.3. ALCANCE

El funcionamiento básico del algoritmo es devolver una lista de las mejores opciones de aulas según los parámetros ingresados previamente por el usuario. El objetivo del algoritmo es optimizar el procedimiento de búsqueda de aulas, por lo tanto, su tiempo de respuesta debe ser óptimo.

El algoritmo utilizará el principio de programación dinámica, el cual resolverá primero los problemas más pequeños hasta los problemas más complejos [1], de tal manera que al final del proceso se resolverá el problema original que en este caso será el problema de búsqueda de las mejores aulas. Lo que se busca al aplicar el principio de programación dinámica es optimizar el tiempo de respuesta del algoritmo para hallar las posibles aulas.

Siguiendo el principio de programación dinámica, el algoritmo debe considerar restricciones y prioridades para hallar la solución más óptima para la asignación de aulas. Se debe considerar que el problema de asignación de aulas es un problema no determinístico, es decir, no tiene una única solución [2]. Por tal razón el algoritmo brindará una lista de aulas acopladas a los requerimientos del usuario, sin embargo, el encargado de elegir la mejor opción será el usuario final.

1.4. MARCO TEÓRICO

En el **Apartado 1.4.1**, se presenta la importancia de los algoritmos de automatización y su influencia en la automatización de procesos. En el **Apartado 1.4.2** se estudia el estado del arte en paradigmas algorítmicos. Luego, en el **Apartado 1.4.3** se introduce al paradigma de programación dinámica, el mismo que se utilizó para el desarrollo del proyecto. En el **Apartado 1.4.4** se detalla las principales herramientas y bibliotecas utilizadas en este proyecto. Finalmente, en el **Apartado 1.4.5** se describe la metodología Kanban, la misma que se utilizó para la planificación y gestión de este proyecto.

1.4.1. ALGORITMOS DE AUTOMATIZACIÓN

El propósito principal de los algoritmos de automatización de procesos es mejorar la eficiencia del flujo de trabajo en una organización. A través de la automatización, es factible disminuir costos, tiempo y desperdicio, así como incrementar la productividad y

minimizar errores. La automatización puede reemplazar actividades manuales con procesos automatizados, o utilizar software y sistemas para dar soporte a diversas tareas [3].

Para la automatización de procesos mediante algoritmos contenidos en programas, servicios y aplicaciones, es necesario que la organización busque la mejora continua mediante la implementación del enfoque de Gestión de Procesos de Negocio (BPM).

Las etapas BPM utilizadas en el proceso de implementación del algoritmo de automatización para la asignación de aulas son los siguientes:

- Identificación del proceso: En la fase inicial del proceso BPMN, se identifican y definen los procesos de negocio. En este caso, el proceso es la asignación de aulas basada en los parámetros del usuario. Esta etapa implica comprender los objetivos del proceso, las entradas, las salidas y las interacciones con otros procesos.
- Análisis y diseño: En esta fase, se detalla cómo funcionará el proceso. Aquí se define en detalle cómo se recopilan los parámetros del usuario, cómo se realiza la búsqueda de aulas, los criterios de selección, las reglas de asignación, etc.
- Implementación y ejecución: Una vez diseñado, el proceso se implementa en un sistema o plataforma. Aquí es donde la automatización real de la búsqueda de aulas se produce según las reglas y lógica definidas en las fases anteriores.
- Monitorización: Después de la implementación, se supervisa el proceso para asegurarse de que está funcionando como se esperaba.
- Optimización: Con el tiempo, se podría buscar formas de optimizar aún más el proceso. Esto podría incluir la incorporación de técnicas de aprendizaje automático para mejorar la precisión de la asignación de aulas.
- Integración con sistemas existentes: En un entorno más amplio de gestión educativa, el algoritmo de asignación de aulas automático podría integrarse con otros sistemas, como sistemas de gestión de horarios, sistemas de seguimiento de estudiantes u otros sistemas de la Facultad de Ingeniería Eléctrica y Electrónica.

1.4.2. ESTADO DEL ARTE EN PARADIGMAS ALGORÍTMICOS

Existen varios paradigmas enfocados al diseño e implementación de algoritmos de programación. Cada paradigma algorítmico tiene sus ventajas y desventajas, y la

elección del enfoque adecuado depende del tipo de problema que se está abordando, la eficiencia deseada, la complejidad del problema y otros factores específicos del contexto.

Los paradigmas algorítmicos proporcionan una forma estructurada y sistemática de resolver problemas, lo que facilita el diseño y la comprensión de los algoritmos, así como su aplicación en diferentes áreas de la ciencia de la computación y la ingeniería. A continuación, se enumera los principales paradigmas algorítmicos en la **Tabla 1.1**:

Tabla 1.1 Paradigmas Algorítmicos

Paradigma	Definición
Divide y Vencerás [4]	El algoritmo de divide y vencerás es una estrategia que aborda problemas dividiéndolos en subproblemas más pequeños y resolviendo cada subproblema por separado. Luego, los resultados de los subproblemas se combinan para obtener la solución del problema original. El algoritmo de divide y vencerás se destaca por su eficacia en la resolución de problemas que son demasiado grandes o complejos para ser abordados en su totalidad de una sola vez. Esta técnica es versátil y puede aplicarse en una amplia variedad de problemas.
Programación Dinámica [5]	La programación dinámica es un modelo algorítmico de optimización que involucra una serie de decisiones secuenciales para construir una solución óptima a un problema. En este enfoque, las decisiones tomadas en etapas sucesivas condicionan la evolución futura del sistema, lo que resulta en una solución óptima para el problema en cuestión.
Greedy (voraz) [6]	El paradigma greedy se basa en tomar decisiones que parecen ser las mejores en un instante determinado, sin tener en cuenta las consecuencias futuras ni reevaluar las decisiones previas. Un algoritmo greedy avanza rápidamente hacia una solución sin cuestionarla, adoptando una estrategia "miope" al centrarse únicamente en la mejor opción disponible en cada paso sin considerar el panorama completo. Es debido a su naturaleza voraz que no necesariamente se llegue a una solución óptima, pero se obtiene una solución adecuada rápidamente.

Vuelta Atrás (Backtracking) [7]	El Paradigma de Vuelta Atrás es un enfoque algorítmico empleado para resolver problemas de búsqueda exhaustiva, especialmente aquellos que implican generar todas las soluciones posibles para un problema dado. Este método resulta útil cuando se busca encontrar todas las soluciones posibles o una solución específica en un problema que presenta diversas opciones y restricciones. El algoritmo de Backtracking explora metódicamente todas las opciones posibles para una solución, pero, en lugar de examinar todas las combinaciones posibles, realiza un recorrido recursivo en profundidad en el árbol de soluciones. Si se alcanza una solución parcial que no cumple con las restricciones del problema, el algoritmo retrocede a un estado anterior y prueba otras opciones. Es precisamente este proceso de retroceso lo que le da el nombre de "vuelta atrás".
Fuerza Bruta [8]	El algoritmo de búsqueda exhaustiva, también conocido como fuerza bruta, es una técnica sencilla pero comúnmente utilizada. Consiste en listar de manera sistemática todos los posibles candidatos para la solución de un problema y luego verificar si alguno de estos candidatos satisface la solución buscada. Es el algoritmo más simple de todos y a menudo es el algoritmo que más recursos consume pues no le importa el tiempo ni los recursos que utiliza para encontrar las soluciones al problema.
Programación Lineal y Entera [9]	El Paradigma de Programación Lineal y Entera es una estrategia algorítmica empleada para resolver problemas de optimización, con el objetivo de maximizar o minimizar una función objetivo, considerando un conjunto de restricciones lineales o enteras. Este enfoque es especialmente valioso en contextos donde se requiere tomar decisiones óptimas y eficientes, tomando en cuenta ciertas limitaciones o restricciones en el problema. Un modelo de programación lineal y entera se caracteriza por tener variables que son números enteros no negativos. En situaciones prácticas, el analista se encuentra con "decisiones sí o no", que pueden ser representadas mediante variables denominadas binarias. Estas variables binarias toman el valor de 0 o 1, lo que refleja la elección entre dos opciones excluyentes.

1.4.3. PROGRAMACIÓN DINÁMICA.

Richard Bellman fue el creador original de la programación dinámica, según se describe en su trabajo de 1957. Posteriormente, esta técnica fue ampliada y extendida en diversos campos por otros investigadores, como Howard en 1960, Bertsekas en 1987 y Lew & Mauch en 2007. La programación dinámica ha encontrado aplicaciones frecuentes en áreas como economía, ingeniería y robótica, y también ha sido utilizada en contextos de ecología [5].

La programación dinámica es un paradigma algorítmico que aborda problemas complejos descomponiéndolos en subproblemas más pequeños y resolviéndolos de manera recursiva. Lo que la hace especial es que almacena los resultados de los subproblemas previamente resueltos para evitar recalcularlos repetidamente. Esta optimización sencilla reduce drásticamente la complejidad temporal del algoritmo de exponencial a polinomial [10].

La programación dinámica se utiliza para resolver una amplia gama de problemas, incluyendo:

- El problema del camino más corto
- El problema del cambio mínimo
- El problema de la mochila
- El problema de la asignación de recursos

Dos propiedades clave de un problema que sugieren que puede resolverse mediante programación dinámica son: la presencia de subproblemas superpuestos y una subestructura óptima [11].

Al igual que el enfoque "divide y vencerás", la programación dinámica también se basa en combinar soluciones de subproblemas. La programación dinámica se utiliza principalmente cuando se deben resolver los mismos subproblemas repetidamente. En este método, las soluciones calculadas para los subproblemas se guardan en una tabla (tabulación) o en memoria (memoización) para evitar recálculos innecesarios. Por lo tanto, la programación dinámica no es apropiada cuando no existen subproblemas comunes (superpuestos), ya que no tendría sentido almacenar las soluciones si no se requieren nuevamente [11].

La programación dinámica emplea dos técnicas, la memoización o memorización y la tabulación, para optimizar la ejecución de una función que implica cálculos repetidos y

costosos. Aunque ambas técnicas persiguen el mismo objetivo, existen algunas diferencias entre ellas.

La memorización es un enfoque de arriba hacia abajo, en el cual almacenamos en caché los resultados de las llamadas a funciones y, si la función se vuelve a llamar con las mismas entradas, se devuelve el resultado previamente almacenado en caché. Esta técnica se utiliza cuando el problema puede dividirse en subproblemas con subproblemas superpuestos. La memorización se implementa generalmente mediante recursividad y es adecuada para problemas con un conjunto de entradas relativamente pequeño [12].

Mientras que la tabulación es un enfoque de abajo hacia arriba, donde almacenamos los resultados de los subproblemas en una tabla y los utilizamos para resolver subproblemas más grandes, hasta llegar a la solución completa. Esta técnica se emplea cuando el problema se puede definir como una secuencia de subproblemas sin superposición entre ellos. La tabulación se implementa generalmente mediante iteración y es adecuada para problemas con un conjunto de entradas más grande [12].

La programación dinámica permite resolver subproblemas de manera más eficiente, utilizando técnicas como la recursividad y la memorización, lo que disminuye la complejidad del código y mejora su velocidad de ejecución. Los siguientes puntos son fundamentales [10]:

- Descomponer un problema en subproblemas más pequeños.
- Resolver cada subproblema de forma independiente.
- Almacenar las soluciones a los subproblemas para evitar cálculos redundantes.
- Utilizar las soluciones de los subproblemas para construir la solución global.
- Aplicar el principio de optimalidad para asegurar que la solución sea óptima.

El paradigma de Programación Dinámica se revela como la elección idónea para la implementación del algoritmo de asignación de aulas. Dado que este proceso implica una gran cantidad de cálculos basados en los parámetros del usuario, el objetivo es evitar cálculos redundantes de subproblemas y, en su lugar, almacenar las soluciones en memoria para su posible reutilización posterior. Este enfoque se traduce en una reducción en el tiempo de respuesta del proceso.

1.4.4. FASTAPI

Una API, que significa Interfaz de Programación de Aplicaciones, es un conjunto de definiciones y protocolos utilizados para desarrollar e integrar el software de aplicaciones. Estas interfaces permiten que productos y servicios se comuniquen entre sí, sin requerir conocimiento sobre cómo están implementados internamente. En resumen, una API facilita la interacción y la integración de diferentes componentes de software sin necesidad de conocer los detalles de su funcionamiento interno [13].

FastAPI es un moderno y veloz framework web para construir APIs con Python 3.6+ que se fundamenta en las anotaciones de tipos estándar de Python.

En la **Tabla 1.2** se presentará las principales características de FastAPI que lo han convertido en uno de los frameworks web de Python más populares [14].

Tabla 1.2 Características FastAPI

Característica	**Descripción**
Alto Rendimiento	FastAPI supera a otros frameworks de Python en términos de rendimiento. Gracias a su enfoque orientado a la velocidad, aprovecha las funcionalidades de `Starlette`, lo que resulta en una ejecución más rápida y un mejor rendimiento. Esto lo convierte en una opción preferida para crear APIs y generar códigos listos para producción.
Velocidad en la Codificación	Los desarrolladores no necesitan escribir código desde cero, ya que FastAPI ofrece códigos listos para producción. Esto acelera el proceso de desarrollo en un aproximado del 200% al 300%. FastAPI se basa en `Starlette` y proporciona una amplia gama de funciones para la validación y serialización de datos. Al contar con todos los componentes necesarios, el tiempo necesario para codificar se reduce significativamente.
Mínimos Errores	El marco incluye funciones de autocompletado que insertan automáticamente gran parte del código en el proyecto. Esto minimiza la probabilidad de errores y mejora la calidad del resultado final.

Fácil de Comprender	FastAPI sigue el estándar de Python Moderno, siendo una versión modificada de Python 3.6 sin introducir nueva sintaxis. Como resultado, el marco puede ser utilizado por cualquier persona que tenga un buen conocimiento del lenguaje Python.
Mínimo Esfuerzo en Documentación	FastAPI genera automáticamente documentación de OpenAPI con un esfuerzo mínimo por parte del desarrollador. Esta información detallada sobre los puntos finales de la API, códigos de respuesta, parámetros y otros detalles se encuentra en el directorio/doc de la aplicación.
Soporte para Editores	Todos los componentes del marco están diseñados considerando el autocompletado, lo que permite a los desarrolladores generar código listo para producción de manera sencilla. También se puede probar el código directamente desde el navegador utilizando la documentación interactiva.
Adherencia a Estándares Abiertos	La compatibilidad de FastAPI con OpenAPI, anteriormente conocido como `Swagger`, es una gran ventaja. Esta característica permite crear APIs que se adhieren a estándares abiertos, facilitando la comunicación e integración con otras aplicaciones y sistemas.

Según [14] algunas de las ventajas que posee FastAPI en comparación a otros frameworks web son las siguientes:

- Permite validar el tipo de datos del desarrollador, incluso con consultas JSON anidadas.
- Ofrece autocompletado, lo que acelera el desarrollo de aplicaciones y requiere menos esfuerzo.
- Es compatible con el vocabulario OpenAPI y JSON Schema, lo que facilita la validación y anotación de documentos JSON.

- Posibilita la construcción rápida de una API GraphQL, utilizando la biblioteca de Python llamada "grafeno-pitón".
- Es compatible con OAuth 2.0 y proveedores externos.

Según [14] las limitaciones que presenta FastAPI en comparación a otros frameworks web son las siguientes:

- Aun siendo un framework relativamente nuevo, cuenta con una comunidad de guías limitada, lo que resulta en una escasa disponibilidad de información educativa externa, como libros, cursos o tutoriales.
- Debido a la necesidad de unir todos los componentes en FastAPI, el archivo principal puede volverse extenso o desordenado.

1.4.5. HERRAMIENTAS, BIBLIOTECAS Y MÓDULOS PARA LA IMPLEMENTACION DEL ALGORITMO DE AUTOMATIZACIÓN.

Para el desarrollo de este algoritmo, se han empleado diversas herramientas y bibliotecas, lo que ha permitido lograr un despliegue ágil y efectivo del mismo, asegurando así el cumplimiento de los requerimientos del algoritmo.

La elección de las herramientas y bibliotecas utilizadas para la codificación del algoritmo se ha basado en la búsqueda de la mejor opción. En consecuencia, se han utilizado las herramientas y bibliotecas, que se detallan en la **Tabla 1.3**:

Tabla 1.3 Herramientas y Tecnologías a utilizar.

Elementos utilizados	
CORSMiddleware (FastAPI) [15]	El término `CORSMiddleware` se utiliza para describir situaciones en las que una interfaz que se ejecuta en un navegador web contiene código JavaScript que establece comunicación con un servidor backend, y este último tiene un "origen" o dominio diferente al de la interfaz en el navegador. En otras palabras, se trata de una situación en la que el cliente y el servidor se encuentran en dominios distintos, lo que puede plantear desafíos y requerir configuraciones específicas para permitir la comunicación entre ambos debido a las políticas de seguridad del mismo origen (Same-Origin Policy) aplicadas por los navegadores web.

	Este es un middleware o intermediario que posibilita la configuración de políticas de control de acceso a recursos compartidos entre dominios (CORS) en una aplicación desarrollada con FastAPI. Su utilidad radica en la capacidad de permitir o restringir el acceso a recursos de una API desde diferentes dominios. En otras palabras, el middleware CORS facilita la gestión de las solicitudes que provienen de distintos dominios y asegura que los recursos de la API sean accedidos de manera controlada y segura.
Uvicorn [16]	`Uvicorn` es un servidor de alto rendimiento basado en ASGI (Asynchronous Server Gateway Interface). Su función principal es ejecutar y ofrecer servicios a aplicaciones web que utilizan el protocolo ASGI, como FastAPI. Gracias a su rendimiento óptimo, `Uvicorn` es una elección ideal para implementaciones de producción, especialmente en aplicaciones web asincrónicas, donde la eficiencia y la capacidad de manejar múltiples solicitudes concurrentes son esenciales.
Python [17]	Python es un lenguaje de programación de código abierto, de alto nivel y multiplataforma, creado por Guido Van Rossum en 1989. Esto significa que Python es gratuito y está disponible para su uso en sistemas operativos como Windows, Unix, Linux y otros. Además, se destaca por su sintaxis más simple y elegante en comparación con otros lenguajes de programación.
Requests [18] [19]	`Requests` es una librería de Python utilizada para enviar solicitudes HTTP. Se trata de una herramienta valiosa que facilita la interacción con APIs externas, la realización de peticiones a servicios web y la obtención de información desde la web. `Requests` simplifica el proceso de trabajar con HTTP/1.1 en Python, ofreciendo una integración fluida con servicios web. Elimina la necesidad de lidiar con detalles complicados, como agregar manualmente consultas a las URLs o convertir datos en formularios para realizar peticiones GET. Además, facilita la reutilización de conexiones HTTP y la gestión de keep-alive, lo que contribuye a un funcionamiento más eficiente y sin

	complicaciones. En resumen, gracias a `Requests`, trabajar con servicios web se vuelve más transparente y sencillo en Python.
JSON [20]	La biblioteca `json` en Python proporciona funcionalidades para trabajar con el formato de intercambio de datos JSON (JavaScript Object Notation). JSON es ampliamente utilizado para representar datos estructurados de manera legible tanto para humanos como para máquinas. La biblioteca `json` en Python permite convertir objetos Python en cadenas JSON y viceversa, lo que resulta especialmente útil al interactuar con servicios web que envían o reciben datos en este formato. Con esta librería, los desarrolladores pueden fácilmente serializar y deserializar datos, lo que les permite compartir información con otras aplicaciones, realizar análisis de datos, y en general, facilitar la comunicación y el intercambio de información entre diferentes sistemas y plataformas. La biblioteca `json` es una parte integral de Python y es una herramienta esencial para cualquier proyecto que involucre la manipulación de datos en formato JSON.
Datetime [21]	El módulo `datetime` en Python ofrece clases que permiten el manejo y manipulación de fechas y horas de manera efectiva. Aunque es posible realizar operaciones aritméticas con fechas y horas, su enfoque principal es la extracción eficiente de campos para su posterior manipulación o formateo. Esta biblioteca es especialmente útil para trabajar con datos temporales, ya que proporciona métodos para acceder y manipular fácilmente componentes como años, meses, días, horas, minutos y segundos. Además, el módulo `datetime` facilita la conversión entre diferentes formatos de fecha y hora, lo que lo convierte en una herramienta esencial para tareas relacionadas con el tratamiento de datos temporales en Python.
Union [22]	La librería `Union` pertenece al módulo `typing` de Python y se utiliza para proporcionar anotaciones de tipo que indican que una variable o parámetro puede aceptar más de un tipo de dato. En otras palabras, `Union` se utiliza para especificar que una variable o parámetro puede ser de uno de varios tipos diferentes. Esta funcionalidad es útil en situaciones en las que una función o

	método puede aceptar diferentes tipos de entradas y se quiere asegurar que el código sea más flexible y versátil.

1.4.6. KANBAN

La metodología Kanban es un método ágil que busca optimizar la eficiencia y gestión de proyectos. Surgió en la industria manufacturera japonesa, concretamente en Toyota, y ha sido ampliamente adoptada en diversos campos como el desarrollo de software, la gestión de proyectos y las operaciones empresariales, entre otros [23].

El tablero Kanban ha sido incorporado en el proceso de programación debido a los beneficios que brinda a los desarrolladores, permitiéndoles lograr un trabajo productivo, rápido y eficiente. Esto destaca la importancia de la organización al llevar a cabo cualquier tipo de proyecto. La adopción de esta metodología ha generado resultados muy positivos, siendo destacadas grandes empresas, como Nike, que han mejorado considerablemente desde su implementación y se han convertido en marcas líderes a nivel mundial. Kanban es una parte fundamental de una metodología de manufactura basada en el enfoque JIT (Just in Time), que se enfoca en producir únicamente lo necesario en el momento oportuno.

Kanban se basa en la visualización de las tareas, cada tarea o elemento de trabajo se representa mediante una tarjeta Kanban, que contiene información relevante, como detalles de la tarea, estado actual y propietario. Se utiliza un tablero Kanban, que es una herramienta visual dividida en columnas que representan las diferentes etapas del proceso de trabajo las cuales son:

- “TO DO”: Representa a todas las actividades pendientes por realizar
- “IN PROGRESS”: Representa las actividades que se están realizando actualmente
- “DONE”: Representa todas las actividades finalizadas

2. METODOLOGÍA

En esta sección, se detalla el enfoque utilizado en la creación del algoritmo de automatización para el proceso de búsqueda y asignación de aulas en la Facultad de Ingeniería Eléctrica y Electrónica. Se aborda el proceso desde su fase inicial, que comprende la planificación de actividades, hasta la culminación con la construcción final del algoritmo.

El **Apartado 2.1**, conocido como "Diseño", es dedicado a revisar minuciosamente la estrategia y los fundamentos empleados en la construcción del algoritmo. Aquí se presenta el enfoque empleado, analizando cómo se ha estructurado el algoritmo y los conceptos clave que lo sustentan. Se destaca los aspectos fundamentales que guiaron cada decisión tomada durante su desarrollo.

Por otro lado, en el **Apartado 2.2**, correspondiente al proceso de "Implementación", se explora en detalle los aspectos prácticos y técnicos del diseño previamente planificado. En esta sección, se describe las etapas de implementación paso a paso, destacando las herramientas y tecnologías específicas que se utilizaron para la materialización del algoritmo.

A lo largo de ambos apartados, se brinda una visión integral y completa de cómo se ha llevado a cabo la elaboración del algoritmo, desde sus cimientos conceptuales hasta su ejecución práctica. Se espera que esta amplia perspectiva permita una comprensión detallada y una apreciación sólida del valor y el impacto de este algoritmo de automatización.

2.1. DISEÑO DEL ALGORITMO

El paradigma algorítmico elegido fue el de programación dinámica, el mismo que fue separado por subproblemas y a su vez fue separado por subproblemas superpuestos. En el proceso de diseño se tomó en cuenta los requerimientos funcionales y las historias de usuario correspondientes al desarrollo del algoritmo.

2.1.1. INICIALIZACIÓN DEL TABLERO KANBAN

Basándonos en los conceptos presentados en el **Apartado 1.4.7**, se definen las principales actividades que conformarán el tablero Kanban, enfocándose en la fase de diseño, donde se abordarán aspectos cruciales relacionados con los paradigmas algorítmicos y la abstracción de requerimientos funcionales, los cuales son esenciales para la implementación del algoritmo de la aplicación.

Durante la fase de diseño, se priorizan ciertas actividades fundamentales que se consideran vitales para el éxito del proceso:

- Toma de requerimientos funcionales del algoritmo: En esta etapa, se llevará a cabo una interacción cercana con los interesados y usuarios del algoritmo para comprender a fondo sus necesidades y expectativas. Se identificarán los requisitos funcionales clave que el algoritmo debe cumplir para satisfacer las demandas del proyecto.

- Diseño de casos de uso del algoritmo: En esta actividad, se identificarán y describirán los diversos casos de uso que el algoritmo debe manejar. Se considerarán diferentes situaciones y escenarios para asegurarse de que el algoritmo sea robusto y versátil en su funcionamiento.
- Elaboración de diagrama de flujo del algoritmo: Con base en los requerimientos funcionales definidos, se procederá a desarrollar un diagrama de flujo que represente visualmente el flujo lógico y la estructura del algoritmo. Este diagrama ayudará a comprender claramente la secuencia de pasos y la lógica detrás del algoritmo.

Es importante considerar que estas actividades se llevan a cabo con una perspectiva orientada a obtener respuestas precisas en un tiempo de respuesta reducido, buscando garantizar que el algoritmo diseñado cumpla con los requerimientos funcionales establecidos y sea capaz de resolver los problemas específicos para los que fue creado. El proceso de diseño es esencial en el desarrollo del algoritmo, ya que sienta las bases para su implementación exitosa y eficaz en la aplicación final.

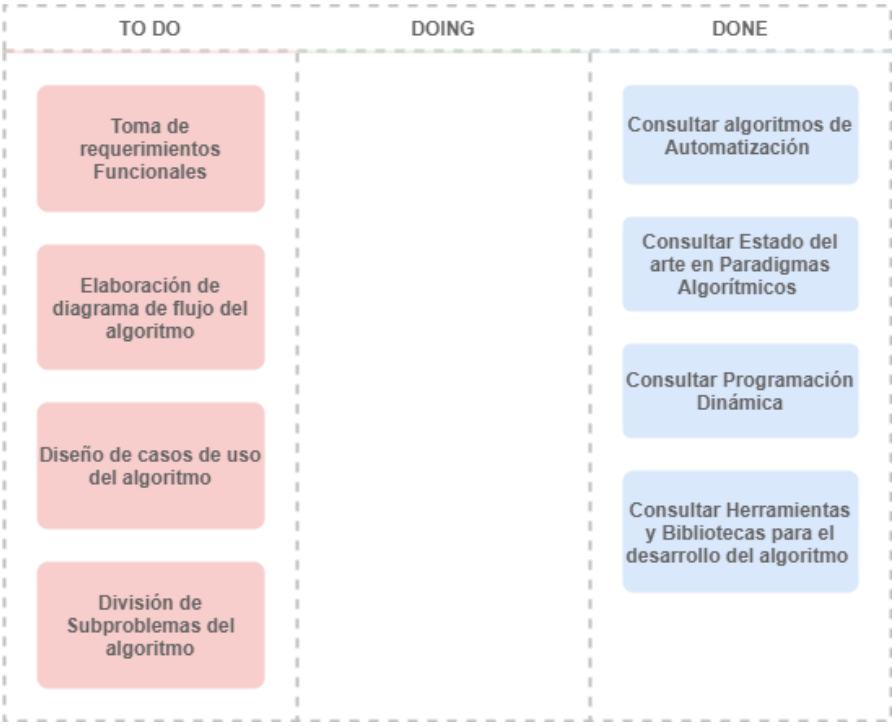

Figura 2.1 Tablero Kanban en Etapa de Diseño del Algoritmo

2.1.2. REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales son un elemento esencial en el ciclo de desarrollo de software, ya que delinean las capacidades y comportamientos que un sistema debe poseer para satisfacer las necesidades específicas del cliente o del negocio. Las reglas

de negocio proporcionan una visión general de los objetivos y directrices del sistema, mientras que los casos de uso ofrecen detalles específicos sobre cómo debe operar en diferentes situaciones [25].

Para llevar a cabo el proceso de abstracción de requerimientos funcionales para el algoritmo, se establecieron reuniones semanales con el encargado de la gestión de horarios de las aulas de la Facultad de Ingeniería Eléctrica y Electrónica. El objetivo de estas reuniones era comprender a fondo el proceso que se pretendía automatizar. A través de un enfoque colaborativo, se recopilaron los detalles y particularidades del método existente, identificando las necesidades y desafíos clave que requerían una solución automatizada.

Los requerimientos funcionales referentes al desarrollo del algoritmo son:

- El sistema permitirá al usuario las opciones de ingresar hora, día, edificio y piso del aula que se requiera encontrar para generar un horario.
- El sistema permitirá al usuario saber la disponibilidad de un aula siguiendo parámetros de entrada proporcionados por el usuario.

2.1.3. CASOS DE USO

En el contexto del desarrollo de software, un caso de uso representa una serie de acciones que describen cómo un actor externo interactúa con el sistema para lograr un objetivo específico. Estos actores pueden ser usuarios reales, otros sistemas o incluso dispositivos. Los casos de uso ofrecen una forma efectiva de capturar los requisitos funcionales del sistema, centrándose en las interacciones entre los usuarios y el software [24].

En base a la información obtenida en las reuniones semanales y abstracción de requerimientos funcionales, se procedió a desarrollar los casos de uso. Estas historias de usuario representaban casos específicos y escenarios relevantes para la aplicación del algoritmo. Cada caso de uso describe un objetivo concreto que se buscaba alcanzar con la automatización, brindando una visión clara y concisa de las funcionalidades que el algoritmo debía abordar.

Este proceso de abstracción y definición de requerimientos funcionales jugó un papel crucial en el diseño del algoritmo de automatización. Al comprender a fondo los requisitos y necesidades del sistema, se pudo desarrollar una solución adaptada y enfocada en satisfacer los objetivos específicos del algoritmo. Además, la colaboración cercana con el encargado de la gestión de horarios permitió obtener información valiosa y realista, lo que contribuyó a la eficiencia y efectividad del algoritmo en la resolución de

problemas reales en el contexto académico. Las historias de usuario y los casos de uso obtenidos pueden ser visualizados en el **ANEXO I.**

2.1.4. DIVISIÓN DE SUBPROBLEMAS

En el **Apartado 1.4.3**, se presenta el paradigma de Programación Dinámica, una estrategia que exige que el problema original se descomponga en subproblemas más pequeños y resolubles de manera independiente. La clave para la eficiencia de este enfoque radica en evitar recalcular los mismos subproblemas repetidamente. En lugar de ello, se adopta la práctica de almacenar los resultados intermedios en memoria y de ser necesario reutilizarlos cuando sea necesario.

Al dividir el problema original en subproblemas más pequeños y manejables, la Programación Dinámica permite una aproximación sistemática y escalable, lo que resulta especialmente valioso cuando se trata de problemas con múltiples posibilidades y soluciones. Además, el enfoque de almacenar y reutilizar los resultados intermedios optimiza el uso de recursos computacionales, lo que es esencial en escenarios donde la eficiencia y la velocidad de cálculo son prioritarias.

2.1.3.1 GENERACIÓN AUTOMÁTICA DE HORARIOS

Los principales subproblemas en los que se dividió el proceso de Generación Automática de Horarios son los siguientes:

- Subproblema 1: En este subproblema se desea obtener todas las aulas que pertenecen a la categoría seleccionada por el usuario, es decir, verifica si un aula es laboratorio o no. Con el objetivo de disminuir la lista de aulas con las que se va a iterar.

- Subproblema 2: El objetivo de este subproblema es obtener una lista de aulas que cumpla con las condiciones correspondientes a día y hora. Este subproblema iterará sobre la lista de aulas obtenidas en el Subproblema 1 y generará una nueva lista con las aulas disponibles en el día y hora proporcionados por el usuario.

- Subproblema 3: El propósito de este subproblema es conseguir una lista de aulas que cumpla con las condiciones de capacidad y ubicación (edificio y piso). En este proceso se iterará en la lista obtenida en el Subproblema 1 y se generará una nueva lista con las aulas que se encuentren en el edificio y piso proporcionado por el usuario. Además, verificará que la capacidad del aula sea igual o mayor a la capacidad insertada por el usuario.

- Subproblema 4: En este subproblema se combinarán las soluciones del Subproblema 2 y Subproblema 3. Se obtendrá una lista de aulas que se encuentren tanto en la lista del Subproblema 2 y Subproblema 3. Con esto se obtienen las aulas que cumplen las condiciones de día, hora, capacidad y ubicación.

- Subproblema 5: En caso de que no se encuentren aulas que satisfagan las condiciones de ubicación y capacidad propuestas en el Subproblema 3. Se procederá a buscar aulas en el piso anterior y posterior al ingresado por el usuario y volverá a resolver el Subproblema 3 con los nuevos pisos en el edificio. En este proceso se reutilizarán las respuestas de los Subproblemas 1 y 2 guardadas en memoria.

2.1.3.2 DISPONIBILIDAD DE AULAS

Los principales subproblemas en los que se dividió el proceso de Disponibilidad de Aulas son los siguientes:

- Subproblema 1: El propósito de este subproblema es obtener las horas ocupadas de cada una de las aulas. La lista de aulas con las que trabajará este subproblema se obtiene a partir de la resolución del Subproblema 3 definido en el **Apartado 2.1.3.1**.

- Subproblema 2: Como respuesta a esta Subproblema se obtendrá una lista con las horas disponibles de cada aula.

- Subproblema 3: Este subproblema combina las soluciones de los Subproblemas 1 y 2. Como resultado se podrá obtener la disponibilidad de las aulas en la “Mañana” considerando un horario desde las 7:00 AM hasta las 14:00 PM o en la “Tarde” considerando un horario desde las 14:00 PM hasta las 20:00 PM. El resultado de este subproblema dependerá de las condiciones proporcionadas por el usuario en la petición a la API.

2.1.5. DIAGRAMA DE FLUJO DEL ALGORITMO

Un diagrama de flujo es una representación gráfica que se utiliza para describir de manera detallada un proceso o algoritmo. Esta herramienta esencial es ampliamente empleada en la programación y el análisis, ya que permite visualizar la secuencia de acciones que deben seguirse paso a paso para lograr un objetivo concreto [26].

Estas representaciones gráficas tienen una amplia aplicación en diversas áreas, como la informática, la ingeniería, la programación y otras disciplinas en las que es necesario

visualizar claramente un conjunto de pasos para llevar a cabo una tarea o resolver un problema. La simplicidad y claridad de los diagramas de flujo los hacen especialmente útiles para comunicar de manera eficiente la lógica detrás de una secuencia de acciones.

Además de su valor en el diseño y creación de algoritmos, los diagramas de flujo también se utilizan para documentar los procedimientos empresariales, planificar proyectos y mejorar la comprensión de los procesos. En contextos académicos y profesionales, estos diagramas son una herramienta valiosa para analizar y mejorar la eficiencia de operaciones y procesos complejos.

En el caso específico del proceso de Generación Automática de Horarios, detallado en el **Apartado 2.1.3.1**, el diagrama de flujo proporciona una representación visual de los pasos clave involucrados en la generación de horarios de forma automatizada. Este diagrama permitirá a los lectores comprender claramente la secuencia de acciones implementadas y la lógica detrás del proceso, facilitando así su comprensión y posible implementación en futuros proyectos relacionados con la gestión de horarios en instituciones académicas.

El diagrama de flujo correspondiente al proceso de Generación Automática de Horarios detallado en el **Apartado 2.1.3.1** es:

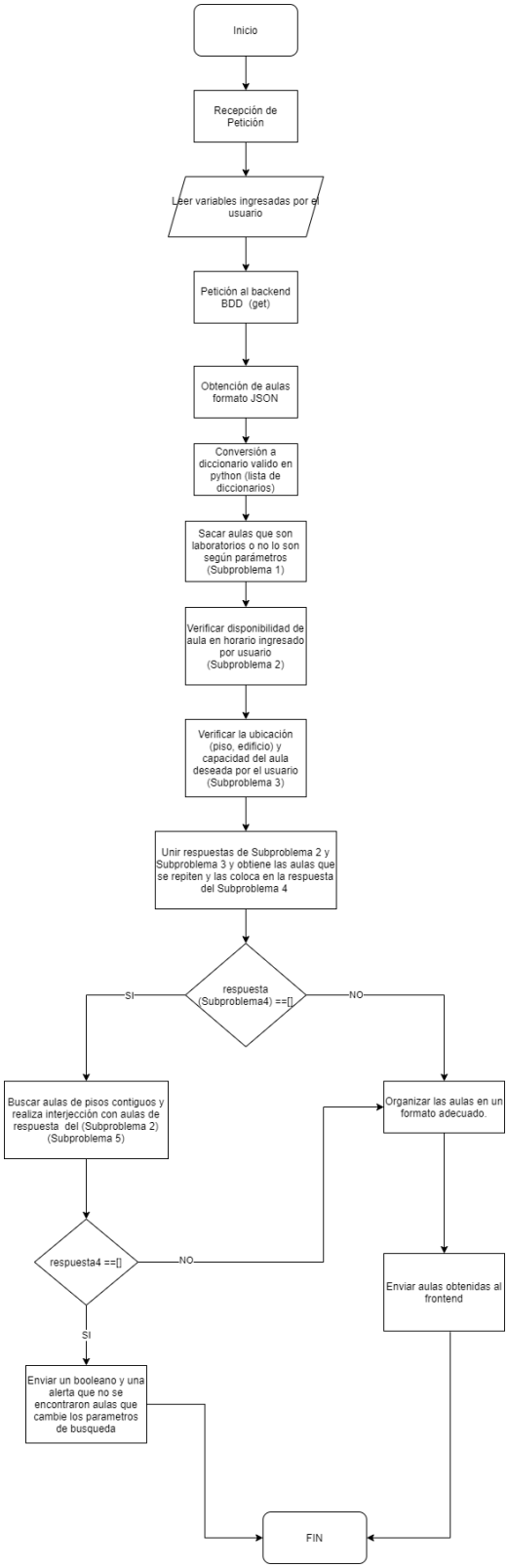

Figura 2.2 Diagrama de Flujo del Proceso de Generación Automática de Horarios

El diagrama de flujo correspondiente al proceso de Disponibilidad de Aulas detallado en el **Apartado 2.1.3.2** es:

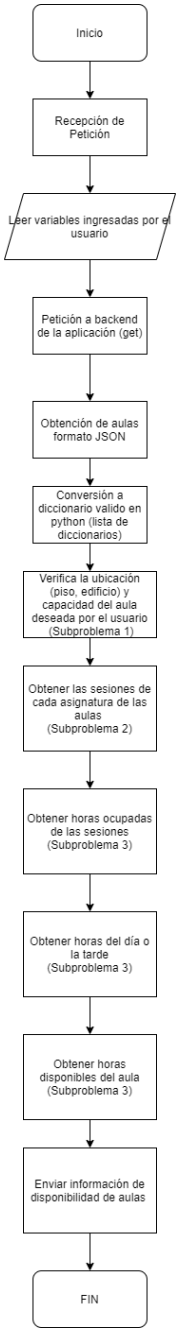

Figura 2.3 Diagrama de Flujo del Proceso de Disponibilidad de Aulas

2.2. IMPLEMENTACION DEL ALGORITMO

En esta etapa, se presentan en detalle todas las actividades llevadas a cabo para la codificación del algoritmo. Durante este proceso, se han utilizado no solo programas, sino también diversas librerías que han permitido desarrollar cada una de las capas del programa de manera efectiva y eficiente. Además, se ha realizado una actualización constante del Tablero Kanban para mantener un seguimiento adecuado del progreso y asegurar una gestión óptima de las tareas.

2.2.1. ACTUALIZACIÓN DE ACTIVIDADES DEL TABLERO KANBAN

Una vez finalizada la fase de diseño, se procede a actualizar el Tablero Kanban con las nuevas actividades y tareas que se desprenden de esta etapa. El Tablero Kanban se convierte en una herramienta esencial para gestionar el flujo de trabajo y seguir el progreso del proyecto en su siguiente fase de implementación.

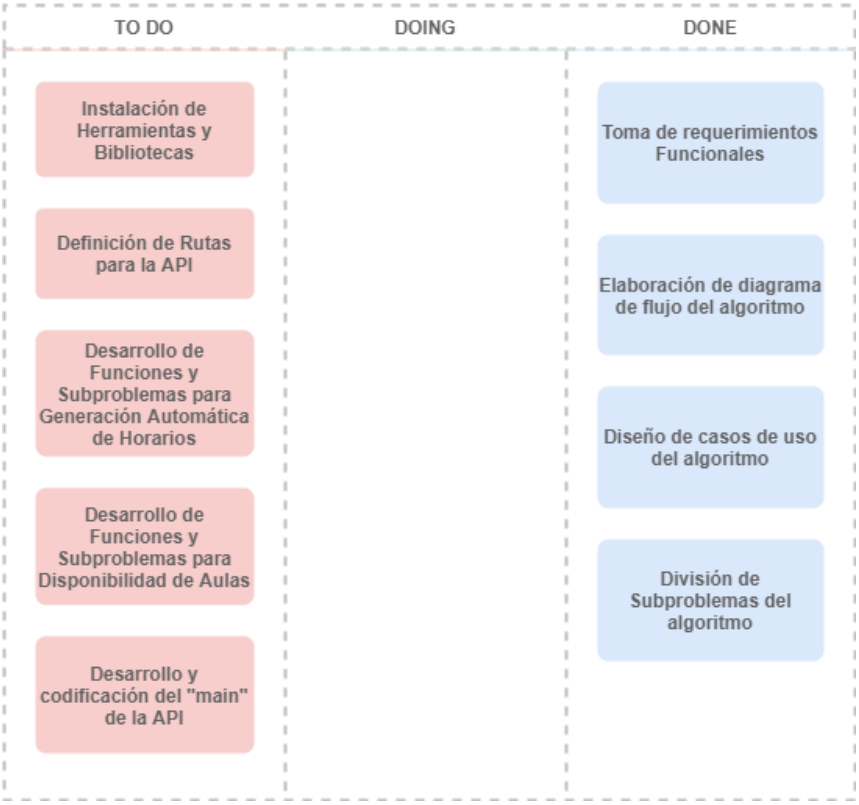

Figura 2.4 Tablero Kanban en Etapa de Implementación del Algoritmo.

2.2.2. INSTALACIÓN DE HERRAMIENTAS DE DESARROLLO

Durante la fase de implementación, se utilizarán aplicaciones y bibliotecas específicas que se instalarán según lo indicado en la **Tabla 2.1** Esta tabla proporciona una lista detallada de las aplicaciones y bibliotecas necesarias para llevar a cabo la implementación del proyecto de manera efectiva.

Tabla 2.1 Instalación de aplicaciones y bibliotecas.

Aplicación	Instalación
Visual Studio Code [25]	1. Comprobar requisitos mínimos necesarios para proceder a la instalación del programa. 2. Ingresar a la página oficial de Microsoft https://visualstudio.microsoft.com/es/downloads/ y descargar el archivo.exe de la versión del editor de código que cumpla con los requisitos.

	3. Ejecutar la instalación del archivo .exe con permisos de administrador. 4. Seguir las instrucciones proporcionados por el instalador de Visual Studio Code. 5. Iniciar la ejecución de Visual Studio Code
Postman [26]	1. Verificar que el sistema cumpla con los requisitos mínimos necesarios para proceder con la instalación del programa. 2. Acceder al sitio web oficial de Postman en https://www.postman.com/downloads/ y seleccionar la versión gratuita de escritorio correspondiente. 3. Ejecutar el instalador de Postman con privilegios de administrador. 4. Seguir las instrucciones proporcionadas por el instalador de Postman para completar el proceso de instalación. 5. Una vez finalizada la instalación, abrir la aplicación de Postman. 6. Si es la primera vez que se utiliza Postman, crear una nueva cuenta en la plataforma para acceder a todas las funcionalidades y servicios adicionales.
Python [27]	1. Acceder al sitio web oficial de Python en https://www.python.org desde el navegador web. 2. En la página de descargas, se mostrará la versión más reciente de Python disponible. 3. Hacer clic en el enlace de descarga correspondiente al sistema operativo que estés utilizando. 4. Una vez descargado el archivo de instalación, ejecutarlo. Asegurarse de marcar la casilla "Agregar Python a PATH" durante la instalación, esto permitirá acceder a Python desde la línea de comandos. 5. Seguir las instrucciones del instalador para completar la instalación de Python. 6. Después de finalizar la instalación, verificar que Python se ha instalado correctamente ejecutando el comando '*python --versio n*' en el terminal de comandos.
FastAPI [14]	1. Verificar que Python se encuentre instalado correctamente en el sistema ejecutando el comando '*python –version*' en el terminal.

	2. Abre una terminal o línea de comandos. 3. Utilizando el gestor de paquetes “pip” para instalar FastAPI se ejecuta el comando ‘*pip install fastapi*’ 4. Verificar a que la instalación de los paquetes y las dependencias se instale correctamente. 5. Comprobar que FastAPI se encuentre instalada ejecutando el comando ‘*pip install fastapi*’
Uvicorn [16]	1. Verificar de tener Python instalado en tu sistema verificando su versión con el comando *'python --version'* en la terminal. 2. Abre una terminal o línea de comandos. 3. Utiliza el gestor de paquetes "pip" para instalar Uvicorn con el comando *'pip install uvicorn'*. 4. Verifica que la instalación de los paquetes y dependencias se realice correctamente. 5. Comprueba que Uvicorn esté instalada ejecutando el comando *'pip show uvicorn'* en la terminal. Si la instalación fue exitosa, se mostrará información detallada sobre el paquete; de lo contrario, se mostrará un mensaje de error.

2.2.3. DEFINICIÓN DE RUTAS PARA LA API

La API contempla la creación de cuatro rutas, cada una con su función específica. Las primeras tres rutas están dedicadas a la implementación de la funcionalidad de generación automática de horarios, permitiendo obtener las aulas disponibles para diferentes configuraciones de sesiones de asignaturas en la semana. La última ruta permite obtener el rango de disponibilidad en la mañana o en la tarde de cada una de las aulas que cumplan los requerimientos proporcionados por el usuario. La **Tabla 2.2** proporciona un análisis de las rutas de la API junto con su funcionalidad correspondiente. En ella se detalla cuidadosamente la información sobre cada ruta y las acciones específicas que se pueden llevar a cabo a través de ellas.

Tabla 2.2 Rutas de la API.

Rutas	**Funcionalidad**
"/aulas1"	Esta ruta se ha diseñado para obtener las aulas disponibles considerando únicamente una sesión de la asignatura en la semana. Al hacer uso de esta ruta, el sistema proporcionará la lista

	de aulas disponibles que cumplen con los requisitos establecidos para una sola sesión académica. Esto es especialmente útil para situaciones donde solo se necesita una sesión de clase para una materia específica.
"/aulas2"	Mediante esta ruta, es posible obtener información acerca de las aulas disponibles cuando se definen dos sesiones de una asignatura en la semana. Al realizar la solicitud a esta ruta, el sistema ofrecerá una lista de aulas que se ajustan a los criterios establecidos para dos sesiones académicas. Esto permite una mayor flexibilidad en la programación de asignaturas que requieren dos sesiones por semana.
"/aulas3"	Esta ruta ofrece la opción de obtener las aulas disponibles al definir tres sesiones de una asignatura en la semana. Al acceder a esta ruta, el sistema mostrará las aulas disponibles que cumplen con los requisitos establecidos para tres sesiones académicas. Esto resulta beneficioso para asignaturas que necesitan una mayor cantidad de tiempo de clase a lo largo de la semana.
"/disponibility"	Esta ruta permite a los usuarios acceder a información detallada sobre los horarios y la disponibilidad de las aulas durante un período específico, ya sea en la mañana o en la tarde. Esta funcionalidad resulta de gran valor al proporcionar una visión completa de cuándo y por cuánto tiempo las aulas están disponibles para ser utilizadas en actividades académicas u otros propósitos. Los usuarios pueden obtener datos precisos que les ayudarán a planificar y programar de manera eficiente el uso de las instalaciones, asegurando una mejor organización de las actividades y una distribución adecuada del tiempo en las aulas.

Cada una de las rutas implementadas en la API utiliza el método de solicitud GET. Mediante el método GET, la información destinada al servidor se incorpora directamente en la URL de la página. Esto resulta visible en la barra de dirección del navegador. Los elementos de datos se convierten en parte de la dirección URL, y en consecuencia, son perceptibles en el enlace del navegador. Es posible guardar los parámetros URL junto con la dirección como marcadores, permitiendo almacenar una búsqueda en particular para su futura referencia. [28]. Los parámetros de las solicitudes GET utilizadas en el proceso de Generación Automática de Horarios, se detallan en la **Tabla 2.3**.

Tabla 2.3 Parámetros de las rutas Generación Automática de Horarios.

Parámetros	Tipo de dato	Descripción
"idSchedule"	*String*	Representa el identificador del horario académico con el que se trabajará.
"isLab"	*Boolean*	Indica si el aula es un laboratorio o no.
"startTimeTest"	*String*	Denota la hora de inicio de la sesión para la cual se busca un aula disponible.
"endTimeTest"	*String*	Representa la hora de finalización de la sesión en la que se requiere encontrar un aula disponible.
"dayTest"	*Int*	Representa el identificador del día de la semana de la sesión de la materia para la que se necesita un aula disponible.
"piso"	*String*	Parámetro opcional que indica el piso del edificio donde se desea el aula.
"capacity"	*String*	Parámetro opcional que indica la capacidad mínima necesaria que se requiere para el aula.
"buildCode"	*String*	Parámetro opcional que indica el código del edificio donde se necesita el aula.
"nameLab"	*String*	Parámetro adicional que, en caso de ser aplicable, muestra el nombre del laboratorio para el cual se busca verificar la disponibilidad en la sesión propuesta.

Cabe recalcar que existen parámetros adicionales como "startTimeTest2", "endTimeTest2", "startTimeTest3" y "endTimeTest3" que detallan la hora de inicio y hora final de las sesiones de la signatura. Estas se utilizan cuando la asignatura tiene dos y tres sesiones respectivamente.

Por otra parte, los parámetros de la solicitud GET utilizada en el proceso de Disponibilidad de Aulas, se detallan en la **Tabla 2.4**.

Tabla 2.4 Parámetros de la ruta Disponibilidad de Aulas.

Parámetros	Tipo de dato	Descripción
"idSchedule"	*String*	Representa el identificador del horario académico con el que se trabajará.

"piso"	*String*	Parámetro opcional que indica el piso del edificio donde se desea el aula.
"dayTest"	*Int*	Representa el identificador del día de la semana de la sesión de la materia para la que se necesita un aula disponible.
"morning"	*Boolean*	Indicador que representa si el horario de disponibilidad es la mañana o en la tarde. Si el valor es True se busca en el horario de la mañana (7-14 Hrs.) caso contrario en la tarde (14-20 Hrs.).
"buildCode"	*String*	Parámetro opcional que indica el código del edificio donde se necesita el aula.

2.2.4. DESARROLLO DE FUNCIONES Y SUBPROBLEMAS PARA LA GENERACIÓN AUTOMÁTICA DE HORARIOS

En el próximo apartado, se proporciona una explicación detallada sobre cómo se codificarán las funciones y se resolverán los subproblemas planteados. Esto conducirá a la obtención de la respuesta necesaria para el proceso de búsqueda de aulas, que es un paso fundamental en la generación automática de horarios.

Para obtener la lista completa de aulas con sus horarios correspondientes, se ha implementado la función "obtenerHorarios", la cual requiere el parámetro de entrada "idSchedule" para indicar el horario del semestre con el que se trabajará. Una vez obtenidos los datos del aula en formato JSON, se aplica la función "json()" para convertirlos en una estructura de lista en Python. Posteriormente, se utiliza la función "json.dumps()" para transformar la estructura de lista en una cadena JSON válida, con el propósito de estandarizar el formato para todas las aulas. Finalmente, la función "json.loads()" se emplea para convertir la cadena JSON en una lista de objetos Python en formato clave-valor, asegurando una representación coherente y legible de la información de las aulas y sus horarios.

```
def obtenerHorarios(idSchedule):
    peticion  =
f'https://localhost:7130/api/classroom/byCalendar/{idSchedule}
'
    response = requests.get(peticion, verify=False)
    listaJson =  response.json()
    cadenaJson = json.dumps(listaJson)
    listaPython = json.loads(cadenaJson)
    return listaPython
```

Código 2.1 Función "obtenerHorarios"

La función denominada "subproblema1" recibe como parámetros de entrada la variable "lista". Esta variable contiene una lista de aulas con atributos representados en un diccionario mediante pares clave-valor obtenidas mediante la función “obtenerHorarios”. También recibe un valor booleano en la variable "isLab", el cual, cuando es True, indica que se trata de un laboratorio. Además, se cuenta con la variable "nameLab" que, si no es nula, almacena el nombre del laboratorio solicitado por el usuario. Esta función tiene la tarea de verificar si un aula funciona como laboratorio, con el propósito de reducir la cantidad de aulas que necesitan ser iteradas. Si el usuario necesita un laboratorio en específico, el proceso de iteración se limitará únicamente a las aulas designadas como laboratorios, excluyendo así aquellas que no tienen esa función.

```
def subproblema1(lista, isLab, nameLab):
    aulaLab=[]
    if(nameLab == "null"):
        for aula in lista:
            if aula['isLab']== isLab:
                aulaLab.append(aula)
        return aulaLab
    else:
        for aula in lista:
            if aula['isLab']== isLab and aula['name']==
nameLab:
                aulaLab.append(aula)
        return aulaLab.
```

Código 2.2 Función “Subproblema1”

La función "subproblema2" se basa en la lista de aulas previamente categorizadas como laboratorios o no laboratorios, proporcionada por el "subproblema1". Además, requiere las variables "dayTest", "startTimeCastTest" y "endTimeCastTest", que contienen la información del día, hora de inicio y hora de finalización de la sesión de clase que se desea programar. El propósito de esta función es identificar las aulas que están disponibles para albergar la sesión de la asignatura, según los parámetros establecidos por el usuario. Como resultado, la función devuelve una lista de aulas que cumplen con los requisitos de disponibilidad para la asignatura en cuestión.

```
def subproblema2(listaPython,dayTest,startTimeCastTest,
endTimeCastTest):
    aulasAptas=[]
    for aula in listaPython:
        sessions_list=[]
        espacios=aula["groups"]
        for espacVal in espacios:
            sessions = espacVal['sessions']
            sessions_list.extend(sessions)
        if hora(sessions_list,dayTest,startTimeCastTest,
endTimeCastTest):
```

```
            aulasAptas.append(aula)
    return (aulasAptas)
```

Código 2.3 Función “Subproblema2”

De manera similar, la función "Subproblema3" hace uso de la lista generada por el "Subproblema1" como parámetro de entrada. La función se centra en la tarea de descubrir aulas que se ubiquen en un edificio y piso específicos, además de tener la capacidad requerida por el usuario. En respuesta a estos criterios, la función devuelve una lista de aulas que cumplen con los requisitos de ubicación y capacidad proporcionados por el usuario.

```
def subproblema3(inputList, pisoTest, capacity, buildCode):
    aulasObtenidas=[]
    if(pisoTest=='null' and buildCode== 'null' and
capacity=='null' ):
        return (inputList)
    if(pisoTest=='null' and buildCode== 'null' ):
        for aula in inputList:
            if  aula['capacity']>= int(capacity):
                aulasObtenidas.append(aula)
        return (aulasObtenidas)
    if(capacity=='null'):
        for aula in inputList:
            if aula['floor']== pisoTest and
aula["building"]["code"]== buildCode:
                aulasObtenidas.append(aula)
        return (aulasObtenidas)
    else:
        for aula in inputList:
            if  aula['capacity']>= int(capacity) and
aula['floor']== pisoTest and aula["building"]["code"]==
buildCode:
                aulasObtenidas.append(aula)
        return (aulasObtenidas)
```

Código 2.4 Función “Subproblema3”

La función "subproblema4" recibe como parámetros de entrada las respuestas obtenidas previamente del "subproblema2" y "subproblema3". Su objetivo radica en combinar las respuestas del "subproblema2", las cuales corresponden al día y la hora de la materia para la cual se necesita programar una sesión en el aula, con las respuestas del "subproblema3", que se relacionan con los requisitos de ubicación del aula. En esencia, esta función busca fusionar la disponibilidad temporal obtenida en el "subproblema2" con las características de ubicación obtenidas en el "subproblema3" para proporcionar una solución conjunta que cumpla con los requerimientos de horario y ubicación para la asignatura en cuestión.

```
def subproblema4(answer2, answer3):
    aulasObtenidas=[]
```

```
    for aula1 in answer2:
        for aula2 in answer3:
            if aula1['id'] == aula2['id']:
                aulasObtenidas.append(aula1)
    return (aulasObtenidas)
```

Código 2.5 Función “Subproblema4”

La función "subproblema5" toma como parámetros la lista de aulas generada por el "subproblema4". Si la lista resultante del "subproblema4" no está vacía, la función automáticamente retorna esta lista de aulas obtenida. En caso contrario, se ejecuta un proceso de recálculo de la respuesta del "subproblema3". Este recálculo se realiza considerando el piso superior e inferior del mismo edificio que fue proporcionado por el usuario inicialmente. Una vez que se obtienen las respuestas para ambos pisos, se recalcula el "subproblema4" utilizando las nuevas listas de ubicación de aulas y la lista de aulas que cumplen los requisitos de día y hora de la sesión.

Al finalizar, las respuestas se combinan para obtener una lista de aulas que cumplen con los requerimientos tanto de horario de sesión como de ubicación, considerando los pisos superior e inferior al proporcionado originalmente por el usuario. Es importante destacar que, si la respuesta original del "subproblema4" no está vacía, el algoritmo retornará dicha respuesta sin recálculos adicionales de los subproblemas anteriores siguiendo el principio de programación Dinámica.

```
def subproblema5(lista,piso, answer1, answer2,capacity,
buildCode):
    if piso== "null":
        return(lista)
    if lista==[]:
        siguiente = str(int(piso[1:]) + 1)
        anterior = str(int(piso[1:]) - 1)
        siguientePiso = 'P' + siguiente
        anteriorPiso = 'P' + anterior
        resp=subproblema3(answer1, siguientePiso,capacity,
buildCode)
        resp2=subproblema3(answer1, anteriorPiso, capacity,
buildCode)
        final=subproblema4(answer2, resp)
        final1=subproblema4(answer2, resp2)
        ultimate=final+final1
        return (ultimate)
    else:
        return lista
```

Código 2.6 Función “Subproblema5”

2.2.5. DESARROLLO DE FUNCIONES Y SUBPROBLEMAS PARA LA DISPONIBILIDAD DE AULAS

En el proceso de implementar la obtención de la disponibilidad de aulas, se aprovechan funciones previamente establecidas en el **Apartado 2.2.4**. A continuación, se

proporcionará una descripción de cómo se lleva a cabo la codificación de estas funciones y subproblemas utilizados en el proceso de verificación de la disponibilidad de aulas dentro de la API.

El proceso se inicia mediante la obtención del horario de las aulas, aprovechando la función "obtenerHorarios" que se describe en el **Apartado 2.2.4**. Una vez que se ha obtenido el horario de las aulas. Posteriormente, se lleva a cabo una operación de filtrado de las aulas, basada en su ubicación de edificio y piso, empleando la función "subproblema3" detallada previamente. De esta manera, se logra una selección más precisa y específica de las aulas que cumplen con los requisitos de ubicación y criterios establecidos por el usuario.

La secuencia prosigue al extraer exclusivamente los detalles del horario de las sesiones que se llevan a cabo en el aula. Estos detalles son obtenidos a través de la función "sesiones", esta función utiliza una lista de objetos Python representativos de las aulas. Esta función, configurada para recibir esta lista como su parámetro de entrada, retorna una nueva lista que contiene las sesiones asociadas a las aulas específicas que se encuentran en la ubicación proporcionada por el usuario.

```
def sesiones(listaPython):
    sesionesAulas=[]
    for aula in listaPython:
        sessions_list=[]
        espacios=aula["groups"]
        for espacVal in espacios:
            sessions = espacVal['sessions']
            sessions_list.extend(sessions)
        sesionesAulas.append(sessions_list)
    return (sesionesAulas)
```

Código 2.7 Función “Sesiones”

Una vez que se han recuperado las sesiones asociadas a cada aula, se procede a implementar la función "get_available_slots_morning". Esta función tiene como objetivo determinar los intervalos de tiempo en los cuales el aula se encuentra disponible para su uso. La esencia de esta función radica en filtrar las sesiones de las aulas en función del día proporcionado por el usuario, y posteriormente calcular y definir los intervalos en los cuales el aula no está en uso.

El proceso se lleva a cabo mediante iteraciones a lo largo de las sesiones correspondientes al día especificado. De cada una de estas sesiones, se extraen los atributos "startTime" y "endTime", los cuales son convertidos al formato de tipo dateTime. En el contexto de la función, se verifica la disponibilidad de cada intervalo en el horario, comparando las horas de inicio y final de las sesiones. Esto se realiza a lo

largo de intervalos de tiempo predefinidos, que abarcan desde las 07:00 hasta las 13:00 horas.

El resultado es la identificación de uno o más intervalos disponibles en un día determinado para un aula en particular. De esta manera, la función "get_available_slots_morning" logra cumplir su propósito al proporcionar información precisa sobre los momentos en los cuales el aula se encuentra disponible para su utilización en el rango de horas matutinas.

```
def get_available_slots_morning(schedule, day):
    day_schedule = [slot for slot in schedule if slot['day']
== day]
    day_schedule.sort(key=lambda x:
parse_time(x['startTime']))

    available_slots = []
    previous_end = parse_time('07:00')

    for slot in day_schedule:
        start_time = parse_time(slot['startTime'])
        end_time = parse_time(slot['endTime'])

        if start_time > previous_end:
            available_slots.append((previous_end, start_time))

        previous_end = end_time if end_time > previous_end
else previous_end

    if previous_end < parse_time('13:00'):
        available_slots.append((previous_end,
parse_time('13:00')))

    return available_slots
```

Código 2.8 Función “get_available_slots_morning”

Se introduce la función "slotsMorningAulas", cuya finalidad es llevar a cabo el procedimiento de adquirir los intervalos de disponibilidad para múltiples aulas. Esta función se basa en los resultados obtenidos mediante la ejecución de la función "get_available_slots_morning". Su objetivo principal es recolectar y almacenar estos intervalos de disponibilidad para, posteriormente, generar una lista que contenga los rangos de disponibilidad de todas las aulas ubicadas en un piso y edificio específicos.

En esencia, la función "slotsMorningAulas" actúa como un engranaje que conecta la capacidad de obtener intervalos de disponibilidad individuales para cada aula, como lo hace la función "get_available_slots_morning", con la tarea más amplia de recolectar y consolidar estos intervalos en una lista integral. De esta manera, se logra obtener una

vista compuesta de los momentos de disponibilidad de todas las aulas en un piso y edificio determinados durante la franja horaria matutina.

```
def slotsMorningAulas (sessionList, day_input):
    respuesta=[]
    for session in sessionList:
        respuesta1=[]
        available_slots = get_available_slots_morning(session,
day_input)
        for start, end in available_slots:

respuesta1.append(f"{start.time().strftime('%H:%M')} -
{end.time().strftime('%H:%M')}")
        respuesta.append(respuesta1)
    return(respuesta)
```

Código 2.9 Función “slotsMorningAulas”

Se establece la función "get_available_slots_afternoon", la cual presenta una funcionalidad similar a la función "get_available_slots_morning". La distinción principal radica en que esta nueva función se dedica a obtener los intervalos de disponibilidad en un horario diferente, específicamente desde las 14:00 hasta las 20:00 horas.

En términos de operación, la función "get_available_slots_afternoon" sigue el mismo proceso que su contraparte matutina. Realiza una evaluación de las sesiones y sus intervalos para determinar los momentos en los que un aula se encuentra disponible para su uso. Sin embargo, se enfoca en el rango de tiempo de la tarde, ajustando su enfoque a las horas que van desde las 14:00 hasta las 20:00 horas. Esto permite obtener información precisa sobre los intervalos en los cuales las aulas están disponibles durante la tarde, cumpliendo con los requisitos de disponibilidad en este horario específico.

```
def get_available_slots_afternoon(schedule, day):
    day_schedule = [slot for slot in schedule if slot['day']
== day]
    day_schedule.sort(key=lambda x:
parse_time(x['startTime']))

    available_slots = []
    previous_end = parse_time('14:00')

    for slot in day_schedule:
        start_time = parse_time(slot['startTime'])
        end_time = parse_time(slot['endTime'])

        if start_time > previous_end:
            available_slots.append((previous_end, start_time))

        previous_end = end_time if end_time > previous_end
else previous_end
```

```
    if previous_end < parse_time('20:00'):
        available_slots.append((previous_end,
parse_time('20:00')))

    return available_slots
```

Código 2.10 Función “get_available_slots_afternoon”

Igualmente, se establece la función "slotsAfternoonAulas", la cual sigue el mismo enfoque. En este caso, utiliza la función "get_available_slots_afternoon" para recopilar y almacenar la disponibilidad de horarios en la tarde para múltiples aulas.

La función "slotsAfternoonAulas" actúa como un mecanismo que orquesta la aplicación de la función "get_available_slots_afternoon" a lo largo de varias aulas. Esta función permite la compilación de los intervalos de disponibilidad en la tarde para cada aula específica, culminando en una lista que contiene los rangos de disponibilidad de todas las aulas en consideración. De esta manera, se logra obtener una visión global de la disponibilidad de horarios en la tarde para múltiples aulas en un piso y edificio determinados.

```
def slotsAfternoonAulas (sessionList, day_input):
    respuesta=[]
    for session in sessionList:
        respuesta1=[]
        available_slots =
get_available_slots_afternoon(session, day_input)
        for start, end in available_slots:

respuesta1.append(f"{start.time().strftime('%H:%M')} -
{end.time().strftime('%H:%M')}")
        respuesta.append(respuesta1)
    return(respuesta)
```

Código 2.11 Función “slotsAfternoonAulas”

3. RESULTADOS Y DISCUSIÓN

En este capítulo, se detallan los resultados de las pruebas ejecutadas y los resultados obtenidos al concluir la fase de implementación del algoritmo. A lo largo del **Apartado 3.1**, titulado Resultados, se procede a actualizar el tablero Kanban, se establece el entorno de pruebas y se lleva a cabo la validación de los requerimientos definidos para el algoritmo. En el **Apartado 3.2**, se presentan las conclusiones y recomendaciones derivadas del desarrollo de cada una de las etapas del algoritmo.

3.1. RESULTADOS

En esta etapa, se lleva a cabo un análisis de los resultados de las pruebas realizadas en el entorno de pruebas. El propósito es evaluar si el algoritmo cumple con los requisitos establecidos y si se gestionan de manera adecuada todas las situaciones contempladas en el diseño.

3.1.1. ACTUALIZACIÓN DE ACTIVIDADES DEL TABLERO KANBAN

Una vez que las fases previas del algoritmo han sido completadas, se inicia el proceso de actualización de las actividades en el Tablero Kanban. En esta etapa, se incorporan las nuevas actividades y tareas específicas que forman parte de la fase de resultados. Estas actividades están directamente relacionadas con la evaluación y análisis de los datos obtenidos, así como la implementación de las soluciones adecuadas basadas en los resultados y conclusiones extraídos de las pruebas y experimentos realizados.

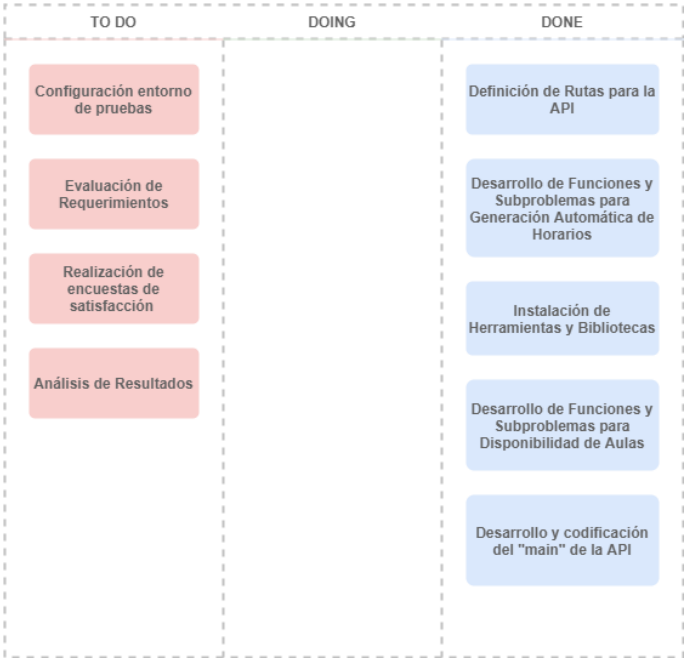

Figura 3.1 Tablero Kanban en Etapa de Resultados del Algoritmo

3.1.2. IMPLEMENTACIÓN DE ENTORNO DE PRUEBAS

Para llevar a cabo la implementación del entorno de pruebas y poner en funcionamiento el algoritmo, se empleó la base de datos proporcionada por el subdecanato de la Facultad de Ingeniería Eléctrica y Electrónica (FIEE). Se utilizó el horario académico correspondiente al periodo 2023-A, que abarca un total de 192 materias, 105 aulas y 626 sesiones. Esta elección se realizó con el propósito de recrear un entorno de pruebas que refleje de manera cercana la realidad, proporcionando una base sólida para evaluar el desempeño y la eficacia del algoritmo en condiciones similares reales.

Durante la fase de pruebas, el algoritmo fue puesto en uso por dos personas encargadas de la administración y gestión de horarios en la facultad. A estos usuarios se les presentaron una serie de tareas que se debían ejecutar utilizando el algoritmo. Entre los problemas propuestos se incluyeron:

- Asignar un grupo con una sesión de clases a la semana.
- Asignar un grupo con dos sesiones de clases a la semana.
- Asignar un grupo con tres sesiones de clases a la semana.
- Verificar la disponibilidad de aulas en un edificio y piso especifico.

3.1.3. EJECUCIÓN DE TAREAS

En la presente fase se utilizó Postman y el frontend de la aplicación desarrollada en el segundo componente de la aplicación de gestión de horarios.

Para la creación de un grupo que tenga una sesión se utilizó la ruta `/aulas1` con los parámetros especificados en la **Tabla 3.1**.

Tabla 3.1 Parámetros de petición GET para problema de una sesión.

Parámetros	Tipo de dato	Valor	Descripción
"idSchedule"	*String*	1	Hace referencia al calendario académico que se va a utilizar en este caso el horario del periodo 2023-A.
"isLab"	*Boolean*	False	Hace referencia a que el aula requerida no es un laboratorio.
"startTimeTest"	*String*	"16:00"	Hace referencia a que la hora de inicio de la sesión es a las 16:00 hrs.
"endTimeTest"	*String*	"18:00"	Hace referencia a que la hora de finalización de la sesión es a las 18:00 hrs.
"dayTest"	*Int*	0	Hace referencia a que la sesión requerida es el día Lunes.
"capacity"	*Int*	10	Hace referencia a que se busca un aula con capacidad mínima de 10 estudiantes.
"buildCode"	*String*	"E17"	Hace referencia a que el aula debe estar en el edificio 17 "Química - Eléctrica".
"piso"	*String*	"P3"	Hace referencia al piso del edificio en el que se requiere el aula.

Como consecuencia, En Postman la API emite una respuesta en formato JSON que se describe detalladamente en el **Código 3.1**:


```
{
    "status": true,
    "aulas": [
        {
            "id": 51,
            "code": "E005",
            "isLab": false,
            "name": null,
            "capacity": 29,
            "floor": "P2",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 52,
            "code": "E009",
            "isLab": false,
            "name": null,
            "capacity": 23,
            "floor": "P2",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 53,
            "code": "E021",
            "isLab": false,
            "name": null,
            "capacity": 56,
            "floor": "P2",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 54,
            "code": "E022",
            "isLab": false,
            "name": null,
            "capacity": 54,
            "floor": "P2",
            "building": {
```

```
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 55,
            "code": "E023",
            "isLab": false,
            "name": null,
            "capacity": 50,
            "floor": "P2",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        }
    ]
}
```


Código 3.1 Respuesta JSON para una sesión

El resultado obtenido al utilizar el algoritmo en el frontend de la aplicación se muestra en la **Figura 3.2**:

Figura 3.2 Resultados de utilización del Algoritmo para una sesión.

Para implementar un grupo con dos sesiones a la semana se utiliza la ruta `/aulas2` con los parámetros definidos en la **Tabla 3.2**:

Tabla 3.2 Parámetros de petición GET para problema de dos sesiones.

Parámetros	Tipo de dato	Valor	Descripción

"idSchedule"	*String*	1	Hace referencia al calendario académico que se va a utilizar en este caso el horario del periodo 2023-A.
"isLab"	*Boolean*	False	Hace referencia a que el aula requerida no es un laboratorio.
"startTimeTest"	*String*	"07:00"	Hace referencia a que la hora de inicio de la sesión es a las 07:00 hrs.
"endTimeTest"	*String*	"09:00"	Hace referencia a que la hora de finalización de la sesión es a las 09:00 hrs.
"dayTest"	*Int*	0	Hace referencia a que la sesión requerida es el día Lunes.
"startTimeTest2"	*String*	"14:00"	Hace referencia a que la hora de inicio de la segunda sesión es a las 14:00 hrs.
"endTimeTest2"	*String*	"16:00"	Hace referencia a que la hora de finalización de la segunda sesión es a las 16:00 hrs.
"dayTest2"	*Int*	3	Hace referencia a que la segunda sesión requerida es el día Jueves.

Como consecuencia, en Postman la API emite una respuesta en formato JSON que se describe detalladamente en el **Código 3.2**:

Código 3.2 Respuesta JSON para dos sesiones.


```
{
    "status": true,
    "aulas": [
        {
            "id": 10,
            "code": "E021",
            "isLab": false,
            "name": null,
            "capacity": 50,
            "floor": "P1",
            "building": {
                "id": 1,
                "code": "E16",
                "name": "Edificio de Eléctrica"
            }
        },
        {
            "id": 27,
            "code": "E001",
            "isLab": false,
            "name": null,
            "capacity": 20,
```

```
            "floor": "S1",
            "building": {
                "id": 1,
                "code": "E16",
                "name": "Edificio de Eléctrica"
            }
        },
        {
            "id": 28,
            "code": "E004",
            "isLab": false,
            "name": null,
            "capacity": 50,
            "floor": "S1",
            "building": {
                "id": 1,
                "code": "E16",
                "name": "Edificio de Eléctrica"
            }
        },
        {
            "id": 34,
            "code": "E002",
            "isLab": false,
            "name": null,
            "capacity": 22,
            "floor": "MZ",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 36,
            "code": "E001",
            "isLab": false,
            "name": null,
            "capacity": 22,
            "floor": "P1",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 37,
            "code": "E002",
            "isLab": false,
            "name": null,
            "capacity": 12,
            "floor": "P1",
            "building": {
                "id": 2,
                "code": "E17",
```

```
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 38,
            "code": "E003",
            "isLab": false,
            "name": null,
            "capacity": 12,
            "floor": "P1",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 39,
            "code": "E004",
            "isLab": false,
            "name": null,
            "capacity": 12,
            "floor": "P1",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        }
    ]
}
```


El resultado obtenido al utilizar el algoritmo en el frontend de la aplicación se muestra en la **Figura 3.3**:

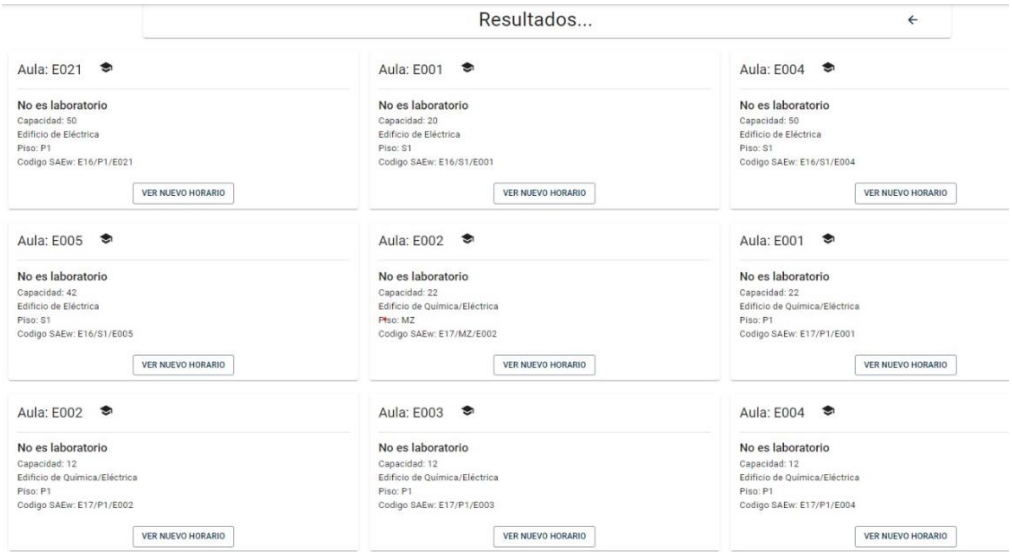

Figura 3.3 Resultados de utilización del Algoritmo en la aplicación para dos sesiones.

Para implementar un grupo con tres sesiones a la semana se utiliza la ruta `/aulas3` con los parámetros definidos en la **Tabla 3.3**:

Tabla 3.3 Parámetros de petición GET para problema de tres sesiones.

Parámetros	Tipo de dato	Valor	Descripción
"idSchedule"	*String*	1	Hace referencia al calendario académico que se va a utilizar en este caso el horario del periodo 2023-A.
"isLab"	*Boolean*	False	Hace referencia a que el aula requerida no es un laboratorio.
"startTimeTest"	*String*	"07:00"	Hace referencia a que la hora de inicio de la sesión es a las 07:00 hrs.
"endTimeTest"	*String*	"09:00"	Hace referencia a que la hora de finalización de la sesión es a las 09:00 hrs.
"dayTest"	*Int*	0	Hace referencia a que la sesión requerida es el día Lunes.
"startTimeTest2"	*String*	"14:00"	Hace referencia a que la hora de inicio de la segunda sesión es a las 14:00 hrs.
"endTimeTest2"	*String*	"16:00"	Hace referencia a que la hora de finalización de la segunda sesión es a las 16:00 hrs.
"dayTest2"	*Int*	3	Hace referencia a que la segunda sesión requerida es el día Jueves.
"startTimeTest3"	*String*	"09:00"	Hace referencia a que la hora de inicio de la tercera sesión es a las 09:00 hrs.
"endTimeTest3"	*String*	"11:00"	Hace referencia a que la hora de finalización de la tercera sesión es a las 11:00 hrs.
"dayTest3"	*Int*	2	Hace referencia a que la tercera sesión requerida es el día Miércoles.

Como consecuencia, en Postman la API emite una respuesta en formato JSON que se describe detalladamente en el **Código 3.3**:

```
{
    "status": true,
    "aulas": [
        {
```

```
            "id": 10,
            "code": "E021",
            "isLab": false,
            "name": null,
            "capacity": 50,
            "floor": "P1",
            "building": {
                "id": 1,
                "code": "E16",
                "name": "Edificio de Eléctrica"
            }
        },
        {
            "id": 27,
            "code": "E001",
            "isLab": false,
            "name": null,
            "capacity": 20,
            "floor": "S1",
            "building": {
                "id": 1,
                "code": "E16",
                "name": "Edificio de Eléctrica"
            }
        },
        {
            "id": 28,
            "code": "E004",
            "isLab": false,
            "name": null,
            "capacity": 50,
            "floor": "S1",
            "building": {
                "id": 1,
                "code": "E16",
                "name": "Edificio de Eléctrica"
            }
        },
        {
            "id": 29,
            "code": "E005",
            "isLab": false,
            "name": null,
            "capacity": 42,
            "floor": "S1",
            "building": {
                "id": 1,
                "code": "E16",
                "name": "Edificio de Eléctrica"
            }
        },
        {
            "id": 34,
            "code": "E002",
            "isLab": false,
            "name": null,
```

```
            "capacity": 22,
            "floor": "MZ",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 36,
            "code": "E001",
            "isLab": false,
            "name": null,
            "capacity": 22,
            "floor": "P1",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 37,
            "code": "E002",
            "isLab": false,
            "name": null,
            "capacity": 12,
            "floor": "P1",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        },
        {
            "id": 38,
            "code": "E003",
            "isLab": false,
            "name": null,
            "capacity": 12,
            "floor": "P1",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            }
        ]
}
```


Código 3.3 Respuesta JSON para tres sesiones.

El resultado obtenido al utilizar el algoritmo en el frontend de la aplicación se muestra en la

Figura 3.4:

Figura 3.4 Resultados de utilización del Algoritmo en la aplicación para tres sesiones.

Finalmente, para obtener la disponibilidad de las aulas en un determinado piso de un edificio se utiliza la ruta `/disponibility` con los parámetros definidos en la **Tabla 3.4**:

Tabla 3.4 Parámetros de petición GET para obtener disponibilidad de aulas.

Parámetros	Tipo de dato	Valor	Descripción

"idSchedule"	*String*	1	Hace referencia al calendario académico que se va a utilizar en este caso el horario del periodo 2023-A.
"isLab"	*Boolean*	False	Hace referencia a que el aula requerida no es un laboratorio.
"dayTest"	*Int*	0	Hace referencia a que se quiere ver la disponibilidad en el día Lunes.
"buildCode"	*String*	"E17"	Hace referencia a que el aula debe estar en el edificio 17 "Química - Eléctrica".
"piso"	*String*	"P3"	Hace referencia al piso del edificio en el que se requiere el aula.
"morning"	*Boolean*	False	Hace referencia a que se requiere ver la disponibilidad en el rango de 14:00 hrs a 20:00 hrs.

Como consecuencia, en Postman la API emite una respuesta en formato JSON que se describe detalladamente en el **Código 3.4**:


```
{
    "status": true,
    "aulas": [
        {
            "id": 57,
            "code": "E001",
            "isLab": false,
            "name": null,
            "capacity": 41,
            "floor": "P3",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            },
            "disponibility": [
                "14:00 - 20:00"
            ]
        },
        {
            "id": 58,
            "code": "E002",
            "isLab": false,
            "name": null,
            "capacity": 50,
            "floor": "P3",
            "building": {
                "id": 2,
```

```
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            },
            "disponibility": [
                "14:00 - 20:00"
            ]
        },
        {
            "id": 59,
            "code": "E014",
            "isLab": true,
            "name": "LAB. COMPUTACION QUIMICA",
            "capacity": 29,
            "floor": "P3",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            },
            "disponibility": [
                "14:00 - 20:00"
            ]
        },
        {
            "id": 60,
            "code": "E016",
            "isLab": true,
            "name": "LAB. CENTRO COMPUTO AGROIND",
            "capacity": 29,
            "floor": "P3",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            },
            "disponibility": [
                "14:00 - 20:00"
            ]
        },
        {
            "id": 61,
            "code": "E021",
            "isLab": false,
            "name": null,
            "capacity": 56,
            "floor": "P3",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            },
            "disponibility": [
                "16:00 - 20:00"
            ]
        },
        {
```

```
            "id": 62,
            "code": "E022",
            "isLab": false,
            "name": null,
            "capacity": 54,
            "floor": "P3",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            },
            "disponibility": [
                "18:00 - 20:00"
            ]
        },
        {
            "id": 63,
            "code": "E023",
            "isLab": false,
            "name": null,
            "capacity": 50,
            "floor": "P3",
            "building": {
                "id": 2,
                "code": "E17",
                "name": "Edificio de Química/Eléctrica"
            },
            "disponibility": [
                "16:00 - 20:00"
            ]
        }
    ]
}
```


Código 3.4 Respuesta JSON para obtener Disponibilidad de aulas.

El resultado obtenido al utilizar el algoritmo en el frontend de la aplicación se muestra en la **Figura 3.5**:

Figura 3.5 Resultados de utilización del Algoritmo en la aplicación para obtener disponibilidad de aulas.

3.1.4. VALIDACIÓN DE REQUERIMIENTOS

A través de una encuesta formada por cinco preguntas, llevada a cabo entre los usuarios de prueba del algoritmo, los cuales fueron la secretaria de subdecanato y el encargado de la gestión de horarios de la facultad. Los resultados de la encuesta se detallan en el **Anexo III**, con este proceso se logró establecer la utilidad del algoritmo, así como su precisión y relevancia en la optimización del proceso de gestión y búsqueda de aulas en la Facultad de Ingeniería Eléctrica y Electrónica. Estos hallazgos validaron con éxito los requerimientos del algoritmo y ratificaron su impacto positivo en la mejora de la operatividad del proceso de gestión de horarios. Como resultado de esta encuesta se obtuvieron un nivel de satisfacción completo en todas las preguntas como se observa en la **Figura 3.6** que representa el resultado de la Pregunta 1 de la encuesta, el mismo que se obtuvo en todas las preguntas.

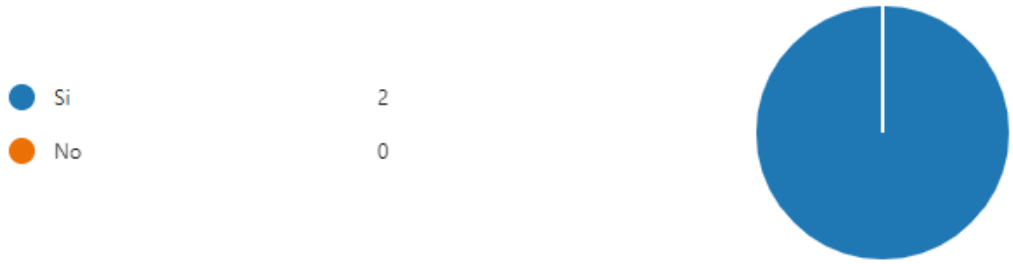

Figura 3.6 Resultado de la Pregunta 1 de la Encuesta

Los resultados de validación de requerimientos por parte de los usuarios de prueba se detallan en la **Tabla 3.5**.

Tabla 3.5 Validación de Requerimientos.

Requerimiento	Descripción
El sistema permitirá al usuario obtener recomendaciones de aulas. Utilizando parámetros como hora, día, capacidad, edificio y piso del aula deseada para la creación de horarios para una, dos y tres sesiones a la semana.	Los usuarios de prueba pudieron obtener las recomendaciones de aulas según los parámetros ingresados a través del algoritmo. Esto se comprobó con los resultados de la encuesta de las preguntas: • Pregunta 1 • Pregunta 3 • Pregunta 5
El sistema permitirá al usuario saber la disponibilidad de aulas siguiendo los parámetros de entrada proporcionados por el usuario.	Los usuarios de prueba pudieron observar la disponibilidad de las aulas según los parámetros ingresados a través del algoritmo. Esto se comprobó con los resultados de la encuesta de las preguntas: • Pregunta 2 • Pregunta 4

3.1.5. CIERRE DE TABLERO KANBAN

Una vez concluidas todas las actividades de pruebas del algoritmo y resueltos los problemas identificados durante su ejecución, se procede al cierre del tablero Kanban mostrado en la

Figura 3.7:

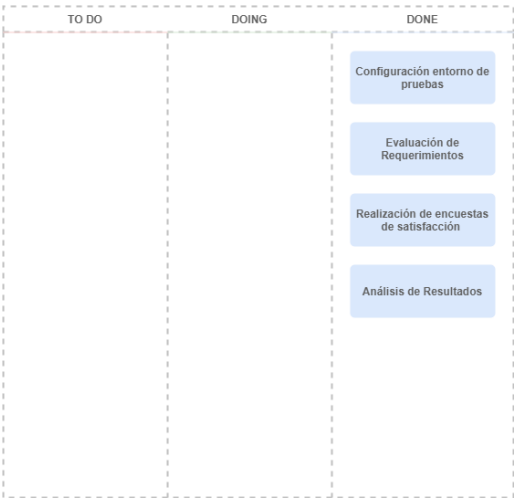

Figura 3.7 Cierre Tablero Kanban

3.2. CONCLUSIONES Y RECOMENDACIONES.

En esta sección, se presentan las conclusiones y recomendaciones derivadas del proceso de desarrollo del presente Trabajo de Integración Curricular.

3.2.1. CONCLUSIONES

A partir de la implementación del algoritmo, se derivaron las siguientes conclusiones:

- La implementación del algoritmo dentro de una API ha posibilitado la creación de modularidad y autonomía en los procesos implicados. La integración de diferentes procesos que conforman la aplicación y la implementación de la API permitió que cada proceso pueda mejorarse individualmente sin alterar la funcionalidad general.
- El proceso de toma de requerimientos fue fundamental en la elaboración exitosa del algoritmo. La identificación precisa y detallada de las necesidades y expectativas de los usuarios proporcionó una base sólida sobre la cual se desarrolló el algoritmo.
- La incorporación de la programación dinámica como elemento central del algoritmo fue fundamental para mejorar la eficiencia y eficacia de la solución. La capacidad de descomponer problemas complejos en subproblemas más manejables, junto con la memorización de soluciones óptimas, ha dado como resultado un algoritmo altamente optimizado y preciso al evitar redundantes cálculos de respuestas para los subproblemas.
- La elección de Python como lenguaje de programación para la implementación del algoritmo ha posibilitado la utilización de diversas bibliotecas, las cuales han tenido gran relevancia en la resolución de problemas específicos durante el proceso de desarrollo. Estas bibliotecas han brindado un soporte esencial para abordar problemas puntuales, garantizando así el desarrollo efectivo del algoritmo.
- La incorporación de la metodología Kanban en las distintas fases de desarrollo del algoritmo fue fundamental para el éxito de este Trabajo de Integración Curricular. La continua supervisión de las tareas a lo largo de cada etapa ha enriquecido la ejecución de las actividades. Este enfoque ha permitido una mejor planificación, seguimiento y ajuste de las tareas, lo que ha contribuido a la eficacia y eficiencia en el proceso de desarrollo del algoritmo.

- A través de los resultados obtenidos en la encuesta dirigida a los usuarios finales, quienes desempeñan un rol fundamental en la generación de horarios académicos en la Facultad de Ingeniería Eléctrica y Electrónica, se llega a la conclusión de que el algoritmo efectivamente optimiza y automatiza el proceso de búsqueda de aulas para la elaboración de horarios.

3.2.2. RECOMENDACIONES

Dadas las circunstancias identificadas durante la creación del algoritmo, se proponen las siguientes recomendaciones con el objetivo de potenciar el procedimiento y asegurar una mejora en la eficacia de los algoritmos a desarrollar en el futuro:

- Se recomienda emplear frameworks actualizados que aprovechen lenguajes de programación con una variedad de bibliotecas disponibles. Esto permitirá hacer uso de aquellas bibliotecas que ofrecen soluciones específicas a los desafíos particulares del algoritmo y que contribuyan a mejorar su rendimiento.

- Se recomienda la implementación de un entorno de pruebas bien estructurado que simule las condiciones reales en las cuales el algoritmo operará. Esto debe incluir un número adecuado de aulas, materias y grupos que reflejen con precisión la complejidad y el alcance del entorno en el que se desplegará el algoritmo. Este enfoque permitirá evaluar su eficiencia y eficacia de manera realista, brindando una visión clara de cómo el algoritmo funcionará en situaciones reales y cómo responderá a desafíos y requisitos del mundo real.

- Se recomienda estructurar el código de manera modular, organizando cada función y separándola en función de su rol dentro del algoritmo. Esta práctica se vuelve esencial, especialmente en el contexto de una API, donde la estructura general puede no ser rígida. La segmentación de funciones permite reutilizar el código de manera eficiente y lograr una estructura de código ordenada.

- Se recomienda adoptar una metodología ágil para administrar las actividades relacionadas con el desarrollo del algoritmo es de suma importancia. En esta situación, se destaca la recomendación de mantener una actualización constante de las actividades en el tablero Kanban. Esta práctica permite una gestión dinámica y transparente del progreso del proyecto, facilitando la asignación de tareas, el seguimiento y la colaboración entre los miembros del equipo.

4. REFERENCIAS BIBLIOGRAFICAS

[1] L. J. R. Fuentes, «Técnicas de diseño de algoritmos,» 2019.

[2] F. A. Ferreira, Programación estocástica: modelo determinista, 2012, p. 38.

[3] SYDLE, «Automatización de procesos: ¿cómo funciona? ¿Cuáles son los beneficios?,» Blog SYDLE, 09 04 2021. [En línea]. Available: https://www.sydle.com/es/blog/automatizacion-de-procesos-. [Último acceso: 22 07 2023].

[4] T. H. Cormen, Introduction to Algorithms, USA: MIT Press, 1990.

[5] SciELO - Scientific Electronic Library Online, «La programación dinámica en el estudio de procesos de migración,» [En línea]. Available: http://www.scielo.org.ar [Último acceso: 25 07 2023].

[6] J. R. M. Torres, «Un procedimiento greedy para el problema de posicionamiento de vehículos en sistemas complejos de transporte automatizado para manufactura,» SciELO, Bogotá, 2008.

[7] C. L. Vida, «Una Revisión sobre la Ejecución Simbólica de Programas,» Santiago, 2014.

[8] L. R. Pichucho, «ALGORITMO DE FUERZA BUSQUEDA,» Scribd, [En línea]. Available: https://es.scribd.com/document/101881852/Fuerza-Bruta. [Último acceso: 2023 07 26].

[9] Y. B. Colina, «Aplicaciones de programación lineal, entera y mixta,» *Ingeniería Industrial. Actualidad y Nuevas Tendencias,* vol. II, nº 7, p. 104, 2011.

[10] GeeksforGeeks, «Complete Tutorial on Dynamic Programming (DP) Algorithm,» [En línea]. Available: https://www.geeksforgeeks.org/introduction-to-dynamic-programming-data-structures-and-algorithm-tutorials/?ref=lbp. [Último acceso: 20 07 2023].

[11] GeeksforGeeks, «Overlapping Subproblems Property in Dynamic Programming,» [En línea]. Available: https://www.geeksforgeeks.org/overlapping-subproblems-property-in-dynamic-programming-dp-1/. [Último acceso: 25 07 2023].

[12] GeeksforGeeks, «Tabulation vs Memoization,» [En línea]. Available: https://www.geeksforgeeks.org/tabulation-vs-memoization/?ref=lbp. [Último acceso: 22 07 2023].

[13] Scribd, «Qué Es Una API | PDF | Transferencia de estado representacional | Jabón,» Scribd, [En línea]. Available: https://es.scribd.com/document/426970149/Que-es-una-API#:~ [Último acceso: 20 07 2023].

[14] FastAPI, «FastAPI,» FastAPI, [En línea]. Available: https://fastapi.tiangolo.com/es/. [Último acceso: 21 07 2023].

[15] FastAPI, «CORS (Cross-Origin Resource Sharing) - FastAPI,» FastAPI, [En línea]. Available: https://fastapi.tiangolo.com/tutorial/cors/?h=corsmiddleware#use-corsmiddleware. [Último acceso: 22 07 2023].

[16] Uvicorn, «Uvicorn,» Uvicorn, [En línea]. Available: https://www.uvicorn.org/. [Último acceso: 25 07 2023].

[17] Python.org, «Welcome to Python.org,» [En línea]. Available: https://www.python.org/. [Último acceso: 22 07 2023].

[18] documentación de Requests - 1.1.0, «Requests: HTTP para Humanos,» documentación de Requests - 1.1.0, [En línea]. Available: https://requests.readthedocs.io/projects/es/es/latest/. [Último acceso: 25 07 2023].

[19] PyPI, «Requests,» PyPI, [En línea]. Available: https://pypi.org/project/requests/. [Último acceso: 2023 07 22].

[20] Python documentation, « Codificador y decodificador JSON,» Python documentation, 19 07 2023. [En línea]. Available: https://docs.python.org/es/3/library/json.html.

[21] Python documentation, «Tipos básicos de fecha y hora,» Python documentation, [En línea]. Available: https://docs.python.org/es/3/library/datetime.html. [Último acceso: 20 07 2023].

[22] Python documentation, «Support for type hints,» Python documentation, [En línea]. Available: https://docs.python.org/3/library/typing.html. [Último acceso: 19 07 2023].

[23] Thinking for Innovation, «Qué es la metodología Kanban y cómo utilizarla,» Thinking for Innovation, 27 07 2023. [En línea]. Available: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/.

[24] IBM - Deutschland | IBM, «IBM Documentation,» [En línea]. Available: https://www.ibm.com/docs/es/engineering-lifecycle-management-suite/lifecycle-management/6.0.3?topic=requirements-defining-use-cases. [Último acceso: 25 07 2023].

[25] Microsoft Learn: Build skills that open doors in your career, «Instalar Visual Studio,» [En línea]. Available: Microsoft Learn: Build skills that open doors in your career. [Último acceso: 31 07 2023].

[26] A. López, «Instalación y primeros usos de Postman,» OpenWebinars.net, 07 06 2019. [En línea]. Available: https://openwebinars.net/blog/instalacion-y-primeros-usos-de-postman/. [Último acceso: 31 07 2023].

[27] Amazon Web Services, «Instalación de Python, pip y la CLI de EB en Windows,» [En línea]. Available: https://docs.aws.amazon.com/es_es/elasticbeanstalk/latest/dg/eb-cli3-install-windows.html. [Último acceso: 31 07 2023].

[28] Equipo editorial de IONOS, «GET vs. POST: los dos métodos de petición HTTP más conocidos cara a cara,» IONOS Digital Guide, 11 08 2020. [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/get-vs-post/. [Último acceso: 05 08 2023].

[29] GeeksforGeeks, «Dynamic Programming - GeeksforGeeks,» [En línea]. Available: https://www.geeksforgeeks.org/dynamic-programming/?ref=lbp. [Último acceso: 24 07 2023].

[30] Amazon Web Services, Inc., «¿Qué es Python? - Explicación del lenguaje Python - AWS,» Amazon Web Services, Inc., [En línea]. Available: https://aws.amazon.com/es/what-is/python/. [Último acceso: 27 07 2023].

[31] Northware, «Requerimientos en el desarrollo de software y aplicaciones,» Northware, [En línea]. Available: https://www.northware.mx/blog/requerimientos-en-el-desarrollo-de-software-y-aplicaciones/. [Último acceso: 29 07 2023].

[32] Lucidchart, «Que es un diagrama de flujo,» Lucidchart, [En línea]. Available: https://www.lucidchart.com/pages/es/que-es-un-diagrama-de-flujo. [Último acceso: 30 07 2023].

5. ANEXOS

La incorporación de anexos aporta información adicional y detalles técnicos cruciales que son indispensables para lograr una comprensión y evaluación completa del presente Trabajo de Integración Curricular.

ANEXO I. Historias de Usuario y Casos de Uso.

ANEXO II. Código del algoritmo.

ANEXO III. Resultados de encuesta.

ANEXO IV. Manual de Usuario.

ANEXO I. HISTORIAS DE USUARIO Y CASOS DE USO.

Historias de Usuario y Casos de Uso

ANEXO II. CÓDIGO DEL ALGORITMO.

Código Algoritmo

ANEXO III. RESULTADOS DE ENCUESTA.

Resultados de la Encuesta

ANEXO IV. MANUAL DE USUARIO.

Manual de Usuario