



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**título del TFG
Documentación Técnica**



Presentado por nombre alumno
en Universidad de Burgos — 24 de diciembre
de 2024

Tutor: nombre tutor

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	iv
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	19
Apéndice B Especificación de Requisitos	25
B.1. Introducción	25
B.2. Objetivos generales	25
B.3. Catálogo de Requisitos	26
B.4. Especificación de requisitos	29
Apéndice C Especificación de diseño	35
C.1. Introducción	35
C.2. Diseño de datos	35
C.3. Diseño procedimental	38
C.4. Diseño arquitectónico	40
Apéndice D Documentación técnica de programación	47
D.1. Introducción	47
D.2. Estructura de directorios	47
D.3. Manual del programador	49

D.4. Compilación, instalación y ejecución del proyecto	51
D.5. Pruebas del sistema	53
Apéndice E Documentación de usuario	57
E.1. Introducción	57
E.2. Requisitos de usuarios	57
E.3. Instalación	57
E.4. Manual del usuario	58
Apéndice F Anexo de sostenibilización curricular	65
F.1. Introducción	65

Índice de figuras

B.1. Diagrama de casos de uso.	30
C.1. Diagrama de clases, carpetas.	42
C.2. Diagrama de clases, relaciones modelos.	43
C.3. Diagrama de clases, relaciones vistas.	44
C.4. Diagrama de clases, relaciones controladores.	45
E.1. Pantalla inicial de la aplicación	58
E.2. Pantalla para la creación de un nuevo experimento	59
E.3. Pantalla para la creación de un nuevo experimento con datos . .	59
E.4. Pantalla de carga de datos	60
E.5. Pantalla de carga de datos cargando	61
E.6. Pantalla de visualización de datos sin tratar	61
E.7. Pantalla de visualización de mapas con resultados representados	62
E.8. Pantalla de en bruto	62
E.9. Modal de confirmación para borrar experimento	63

Índice de tablas

A.1. Amortización de hardware y materiales.	20
A.2. Gastos fijos estimados.	21
A.3. Resumen de costes del proyecto.	21
A.4. Licencias de las herramientas utilizadas.	22
A.5. Licencias de las bibliotecas utilizadas.	23
B.1. CU-1 Crear un nuevo experimento.	31
B.2. CU-2 Cargar un experimento existente.	32
B.3. CU-3 Descargar los resultados de un experimento.	33
B.4. CU-4 Borrar un experimento existente.	34

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Este proyecto ha estado marcado por pequeñas y grandes variaciones en el código y el objetivo, debido a su naturaleza como un estudio profundo del algoritmo TRACLUS. Estas modificaciones y adaptaciones han sido gestionadas gracias a la implementación de metodologías ágiles durante todo el desarrollo.

En este apartado se relatará cómo se organizó el proyecto a lo largo del tiempo, detallando las estimaciones realizadas, las tareas planificadas y los resultados obtenidos en cada etapa.

A.2. Planificación temporal

El proyecto se estructuró en sprints de entre quince y veinte días cada uno. Esta metodología ágil permitió dividir el trabajo en ciclos cortos y manejables, lo que facilitó la adaptación a los cambios, la retroalimentación constante y la mejora continua en el desarrollo del software.

Cada sprint incluyó la planificación, ejecución y revisión de tareas específicas que se enfocaban en aspectos clave del proyecto, asegurando un progreso constante y alineado con los objetivos generales.

Primer *sprint* (14-12-2023 / 17-01-2024)

El proyecto comenzó oficialmente con la primera reunión, en la que se establecieron las bases iniciales para el desarrollo. En esta etapa se definieron los siguientes puntos clave:

- **Selección del algoritmo:** Se acordó que el foco principal del proyecto sería el algoritmo TRACLUS, con el objetivo de estudiarlo, implementarlo y evaluar posibles variantes.
- **Estudio de la base teórica:** Se identificaron y seleccionaron las fuentes bibliográficas fundamentales, incluyendo el artículo original de TRACLUS y otras investigaciones relacionadas que pudieran servir como referencia teórica.
- **Definición del objetivo final:** Se estableció que el producto final sería una aplicación web que permitiera visualizar y comparar los resultados del algoritmo TRACLUS con diferentes configuraciones e integraciones.
- **Selección de herramientas:** Se eligieron las herramientas clave para el desarrollo:
 - **Lenguaje de programación:** Python, debido a su potencia en el cálculo numérico, amplia biblioteca para el desarrollo de algoritmos y existencia de implementaciones previas del algoritmo TRACLUS (`TRACLUS_library`).
 - **Framework para la aplicación web:** Dash, una biblioteca de Python diseñada para la representación interactiva de datos gráficos.
 - **Sistema de control de versiones:** GitHub, para almacenar el código, registrar cambios y gestionar el flujo de tareas.
- **Plazos y flujo de trabajo:** Se establecieron reuniones periódicas para evaluar el progreso y ajustar la planificación si fuera necesario.

Durante este sprint, se planificaron las siguientes tareas:

1. Estudiar la base teórica para comprender en detalle el funcionamiento del algoritmo TRACLUS y sus fundamentos matemáticos.

2. Preparar una estructura base para documentar adecuadamente el proyecto a medida que avanza.
3. Realizar una primera carga de datos de prueba para evaluar el formato necesario y posibles librerías adicionales.
4. Seleccionar las librerías para la representación gráfica de los datos y realizar una prueba inicial de implementación.
5. Ejecutar y evaluar la funcionalidad básica de la librería `TRACLUS_library`.

Resultados del primer *sprint*

Los resultados del primer sprint fueron satisfactorios en términos generales, aunque surgieron algunos desafíos que permitieron identificar áreas de mejora para los siguientes sprints:

- **Estudio inicial de la base teórica:** Se completó el estudio de la literatura relacionada con el algoritmo TRACLUS y sus aplicaciones, lo que proporcionó una comprensión sólida de sus fundamentos y permitió sentar las bases para su implementación.
- **Definición de objetivos y herramientas:** Se definieron los objetivos específicos del proyecto y se documentaron las herramientas y tecnologías que se utilizarían a lo largo del desarrollo. Esto incluyó la selección de Python como el lenguaje principal de programación y Dash para la visualización interactiva de datos.
- **Primera prueba de carga de datos:** Se realizó una primera prueba de carga de datos, lo que permitió identificar posibles problemas relacionados con el formato y la calidad de los datos. Esto también abrió la discusión sobre la necesidad de limpiar y preparar adecuadamente los datos antes de su procesamiento.
- **Investigación de librerías para la representación de mapas:** Se exploraron diversas librerías capaces de representar trayectorias y mapas de calor. Además, se identificaron los primeros problemas de rendimiento, especialmente con la carga inicial de las trayectorias en un mapa. Esto llevó a la necesidad de optimizar esta parte del flujo de trabajo.
- **Problemas con la ejecución de `TRACLUS_library`:** Aunque se intentó ejecutar la librería `TRACLUS_library`, la ejecución no fue exitosa.

Esto se debió a la falta de familiaridad con el código base de la librería y la complejidad inherente a su funcionamiento.

A pesar de los logros y avances iniciales, se comenzaron a evidenciar las primeras dificultades, principalmente debido al uso de herramientas y tecnologías nuevas, en las cuales no se tenía experiencia previa, y al manejo del gran volumen de datos que se deseaba procesar.

La estimación inicial para este sprint era de aproximadamente 30 horas, pero al final se emplearon unas 40 horas para completar todas las tareas asignadas. Este desfase en el tiempo se debió principalmente a la curva de aprendizaje de las herramientas y a la necesidad de hacer ajustes en los datos y en las librerías utilizadas.

Segundo *sprint* 18-01-2024 / 30-01-2024

La segunda reunión comenzó con la exposición y discusión sobre los avances y problemas enfrentados en las tareas asignadas durante el primer sprint. A partir de ahora, todas las reuniones seguirían este formato, centradas en la revisión del trabajo realizado y la planificación de las nuevas tareas.

Para este sprint, se tomó en cuenta la tarea no lograda del sprint anterior: la ejecución del algoritmo TRACLUS. En esta fase, se debía estudiar más profundamente cada una de las funciones de la librería **TRACLUS_library** y su relación teórica con el algoritmo TRACLUS, con el objetivo de lograr una comprensión más detallada de su funcionamiento y poder implementarlo correctamente.

Además, con el avance en la representación inicial de las trayectorias en los mapas, se buscó añadir nuevas funcionalidades, como la capacidad de hacer zoom o mostrar las trayectorias de diferentes formas, lo que permitiría una mejor visualización de los datos.

Por otro lado, la página web debía comenzar a ser considerada. Se propuso investigar las herramientas más adecuadas para su desarrollo, y se decidieron investigar dos opciones: Plotly Dash y Flask. La tarea consistió en analizar ambas tecnologías y decidir cuál de ellas se adaptaba mejor a las necesidades del proyecto.

Finalmente, se continuó trabajando en la memoria del proyecto, lo cual era una tarea constante a lo largo de todo el trabajo. El registro de investigaciones, avances y decisiones es fundamental para mantener una documentación clara y precisa.

Resultados del segundo *sprint*

Los resultados de este segundo sprint fueron los siguientes:

- Se profundizó en el estudio de la librería TRACLUS_library, lo que permitió mejorar el conocimiento de su funcionamiento y relación con el algoritmo TRACLUS.
- Se implementó un código para probar la ejecución de los datos leídos en el sprint anterior, lo que permitió avanzar en la integración de la librería con los datos.
- Se decidió utilizar Dash como la librería base para la página web, debido a su capacidad para representar datos de forma interactiva, lo que se ajusta a los objetivos del proyecto.
- Se comenzó a desarrollar la página web, incluyendo la creación de un prototipo inicial para la visualización de trayectorias.
- Se priorizó la eficiencia en la carga y ejecución de los mapas de trayectorias, sacrificando algo de calidad visual en favor de tiempos de ejecución más rápidos.
- Se dedicó tiempo a continuar con la memoria del proyecto, registrando los avances y decisiones tomadas durante este sprint.

Se estimaron unas 15 horas para completar este sprint, pero finalmente se emplearon unas 20 horas.

Tercer *sprint* 31-01-2024 / 13-02-2024

En esta reunión se llegó a la conclusión de que, para usar correctamente la librería TRACLUS, se debían realizar pruebas internas más detalladas para entender cómo funciona el algoritmo y qué resultados proporciona. Entre las métricas que se analizaron se incluyeron: cuántas trayectorias se asignaban a cada clúster, cuáles eran los clústeres con más trayectorias, la longitud de los clústeres, cómo se segmentan las trayectorias, y otros datos que podrían ser útiles para la evaluación de los resultados y la representación de las trayectorias.

Además de estudiar la librería, se consideró el hecho de que el algoritmo TRACLUS, aunque no haya sido ampliamente utilizado, existe desde 2007. Por lo tanto, se decidió buscar estudios previos y pseudocódigos de diversas

variantes del algoritmo que han sido creadas a lo largo de los años, con el fin de evaluar si podrían implementarse en el futuro para mejorar el proyecto.

Los datos también fueron un punto clave durante este sprint. Hasta ese momento, solo se había estado trabajando con un conjunto de datos de Trayectoria de Taxis, el cual, aunque útil para las pruebas iniciales, era limitado. Por ello, se comenzó a investigar otros conjuntos de datos, como el conjunto de datos Geolife, para explorar su utilidad en los experimentos. Además, se empezaron a crear filtros específicos para asegurar que los experimentos realizados fueran correctos y los resultados adecuados.

Por último, se planteó realizar un cambio en uno de los mapas que se habían probado inicialmente, el mapa de calor. Este mapa se rediseñó para mejorar su visualización y usabilidad.

Resultados del tercer *sprint*

Los resultados del tercer sprint fueron los siguientes:

- Se identificó que la librería TRACLUS utiliza `sklearn` para la clusterización, con el algoritmo OPTICS como predeterminado. Esto abrió una vía para continuar la investigación, ya que existen varios algoritmos de clustering que tiene propiedades similares a las de OPTICS.
- Se analizó y se obtuvo una comprensión adecuada de los datos generados por TRACLUS, lo cual fue útil para la presentación de resultados en la exposición.
- Se identificaron y evaluaron diversas variantes del algoritmo TRACLUS, como N-TRACLUS y ST-TRACLUS, pero se concluyó que no sería posible implementarlas debido a la limitación de tiempo del proyecto.
- Se investigó el conjunto de datos Geolife y se observó que su formato era muy diferente al de los datos de trayectorias de taxis, por lo que se decidió dejarlo apartado por el momento. No obstante, se planteó la posibilidad de incluirlo en el futuro, si se consideraba necesario y el tiempo lo permitía.
- Se rediseñó el mapa de calor para mejorar su visualización. En lugar del diseño original, se optó por un histograma creado con la librería `numpy`, representado en un mapa, lo que mejoró la claridad de los datos.

- Se completaron correctamente los filtros necesarios para los experimentos, asegurando que los datos estuvieran bien preparados para su análisis.

La estimación inicial de tiempo para este sprint fue de unas 20 horas, pero finalmente se superaron las expectativas, ya que se emplearon más de 30 horas debido al extenso estudio y análisis requerido para comprender la librería, los algoritmos asociados y los datos utilizados.

Cuarto *sprint* 14-02-2024 / 27-02-2024

En esta reunión se discutió cómo deberían ser expuestos correctamente los datos generados por el algoritmo, además de cómo debían ser tratados para evitar problemas como errores de formato. Se decidió que se debían generar estadísticas de manera representativa, utilizando tablas que mostraran todos los datos de un clúster, histogramas y otras visualizaciones. Asimismo, se investigaron algoritmos y librerías que pudieran ayudar a extraer estadísticas complementarias para enriquecer los resultados.

Hasta este punto, los mapas seguían basándose en un conjunto de librerías específico, pero únicamente se había desarrollado el código para representar las trayectorias iniciales en el mapa. Ahora, con el algoritmo TRACLUS ejecutado y conociendo los datos obtenidos, se decidió desarrollar diversos mapas que representaran los cambios y resultados de cada uno de los pasos del algoritmo.

Adicionalmente, se continuó con la investigación teórica sobre el algoritmo TRACLUS y se realizó un registro detallado del conocimiento adquirido. Esto incluyó comentarios y anotaciones en la librería TRACLUS, facilitando revisiones y posibles modificaciones futuras.

Resultados del cuarto *sprint*

Los resultados obtenidos en este sprint fueron los siguientes:

- Se desarrollaron diversas funciones para mostrar los datos generados por el algoritmo TRACLUS, incluyendo mapas, tablas y diagramas. Algunos de estos resultados fueron descartados, pero otros se consideraron útiles para la exposición final.
- Se crearon múltiples mapas para representar las trayectorias y los cambios en las distintas etapas del algoritmo. Esto permitió visualizar los pasos del algoritmo de una forma más clara y comprensible.

- Aunque se investigaron librerías y algoritmos para calcular estadísticas adicionales, no se encontraron resultados que aportaran mejoras significativas al entendimiento de los datos generados.
- Se realizó un *fork* de la librería TRACLUS y se añadieron comentarios extensos para mejorar la comprensión del código, facilitando su revisión y posibles modificaciones futuras.
- Se continuó con el análisis teórico de la matemática aplicada en el algoritmo TRACLUS, lo que ayudó a consolidar el conocimiento sobre su funcionamiento.

En cuanto a las estimaciones de tiempo, estas dejaron de ser completamente útiles. Esto se debió a las constantes pruebas necesarias del algoritmo TRACLUS para generar mapas, extraer datos y entender su funcionamiento. Estas pruebas resultaron extremadamente lentas, lo que afectó el tiempo total del sprint. Aunque se habían estimado aproximadamente 25 horas para las tareas de este sprint, el tiempo real de trabajo neto superó las 35 horas y en términos de horas brutas contando las esperas, el tiempo total invertido fue significativamente mayor.

Quinto sprint 28-02-2024 / 13-03-2024

Hasta este punto, la página web había quedado en segundo plano debido a la prioridad de garantizar el correcto funcionamiento del algoritmo TRACLUS, además de definir qué datos y cómo debían ser expuestos. Con estos aspectos ya avanzados, se decidió comenzar la creación de la página web en su etapa inicial. Esto incluyó la implementación del diseño basado en los prototipos discutidos en reuniones anteriores, evaluando qué era posible y útil, y estableciendo un patrón de diseño estándar a partir de las primeras páginas creadas.

La excesiva demora en las ejecuciones del algoritmo fue otro tema central en esta reunión. Dado que se buscaba analizar grandes volúmenes de datos, era necesario reducir los tiempos de ejecución. En este momento, no se contaba con un conocimiento profundo de las herramientas disponibles en Python para optimizar procesos, por lo que se decidió investigar este aspecto.

Complementariamente, se propuso realizar un análisis de rendimiento de la aplicación, dividiendo las mediciones por funciones, para identificar áreas específicas a mejorar.

Por último, aunque previamente se había descartado implementar variantes del algoritmo TRACLUS, se sugirió investigar algoritmos de clustering que pudieran integrarse al flujo del algoritmo para comparar su rendimiento y resultados.

Resultados del quinto *sprint*

Los resultados obtenidos en este sprint fueron los siguientes:

- Se avanzó significativamente en el desarrollo de la página web:
 - Se creó una página inicial para subir el archivo de datos y seleccionar la cantidad de datos a utilizar.
 - Tras cargar el archivo, se implementó una página para visualizar los mapas de las trayectorias iniciales.
 - A través de una barra de navegación, el usuario podía acceder a:
 - Una página para ejecutar el algoritmo TRACLUS (aún no implementado en esta etapa).
 - Una página para visualizar los resultados del algoritmo, con mapas actualizados y una tabla de resultados.
- Se investigaron herramientas de Python para optimizar procesos, encontrando las siguientes:
 - `numpy` y `pandas` para realizar operaciones más rápidas mediante paralelización.
 - `Threading` para implementar hilos y dividir la ejecución en subprocesos.
- El análisis de rendimiento reveló:
 - La librería TRACLUS tenía una complejidad $O(n^2)$ en el cálculo de distancias entre trayectorias, ya que cada una debía compararse con todas las demás.
 - Otras funciones, como la clusterización, también presentaban tiempos elevados, aunque menos pronunciados, y utilizaban la librería `scikit-learn`, lo que limitaba las posibilidades de optimización interna.
- Se identificaron cinco algoritmos de clustering adecuados para integrar y comparar con el flujo del algoritmo TRACLUS:

- OPTICS
- DBSCAN
- HDBSCAN
- SpectralClustering
- AgglomerativeClustering

En cuanto a tiempos, se habían estimado 30 horas para este sprint, sin contar las horas de pruebas se cumplieron las estimaciones.

Sexto sprint 14-03-2024 / 11-04-2024

Ya teniendo claro como iba a ser el desarrollo de la web se decidió que se debía continuar implementando las diferentes funciones planteadas y probar nuevas como cargar mapas con variaciones de aumentos en el mapa.

El rendimiento era en este momento el principal obstáculo y ya se conocieron diversas formas de como mejorar esto con el estudio que se hizo en el anterior sprint así que se planteó intentar reducir el tiempo de carga todo lo que se pudiera enfocándose en la función que media las tres distancias (angular, paralela y perpendicular).

Además de todo esto se planteó comparar los resultados que daban las funciones de clustering entre ellas.

Sexto sprint 14-03-2024 / 11-04-2024

Con un diseño claro para la página web, se decidió continuar implementando las diversas funciones planteadas, además de probar nuevas características, como cargar mapas con diferentes niveles de zoom.

El rendimiento seguía siendo el principal obstáculo, pero gracias al estudio realizado en el sprint anterior, se identificaron formas de mejorar los tiempos de carga. El enfoque principal fue optimizar la función que calculaba las tres distancias fundamentales del algoritmo TRACLUS (angular, paralela y perpendicular).

Finalmente, se planteó comparar los resultados de los distintos algoritmos de *clustering* identificados en el sprint anterior, con el fin de entender sus diferencias y evaluar su impacto en el análisis.

Resultados del sexto *sprint*

Los resultados obtenidos fueron los siguientes:

- **Avances en la web:**

- Se continuó con el desarrollo de las funciones principales.
- Se descartaron cambios en los mapas debido a problemas de rendimiento.
- Se plantearon ajustes en la forma de ejecutar las funciones:
 - Separar la ejecución del algoritmo TRACLUS de la visualización de los mapas, ya que realizarlas en la misma página provocaba problemas al volver a los datos cargados previamente.
- Se trasladaron los estilos integrados en los archivos de Python a un archivo CSS, mejorando la comprensión y mantenibilidad del código.

- **Comparación de algoritmos de *clustering*:**

- Se evidenciaron diferencias significativas entre los algoritmos evaluados (OPTICS, DBSCAN, HDBSCAN, SpectralClustering y AgglomerativeClustering).
- Cada algoritmo utilizaba diferentes enfoques para segmentar los datos y construir los clústeres, lo que resultaba en variaciones notables en los resultados.
- Se identificó que los algoritmos permitían ajustar parámetros para modificar el procesamiento y los resultados obtenidos.

- **Optimización del rendimiento:**

- La optimización fue una de las tareas más desafiantes debido a las limitaciones de Python en la creación de hilos y la paralelización en comparación con otros lenguajes como Java.
- Se realizaron pruebas con diferentes librerías, pero los cálculos matemáticos mostraban variaciones en los resultados dependiendo de la librería utilizada.
- Se ejecutaron cientos de pruebas del algoritmo TRACLUS durante este sprint, ya que cada tarea requería verificaciones constantes, lo que incrementó los tiempos significativamente.

Inicialmente se estimaron alrededor de 60 horas para este sprint. Sin embargo, las horas reales superaron ampliamente la estimación, siendo aproximadamente tres veces mayor debido a las pruebas continuas y los desafíos encontrados.

Séptimo *sprint* 12-04-2024 / —

En este punto, el proyecto experimentó una pausa significativa debido a un grave accidente sufrido, lo cual me impidió continuar con el desarrollo durante el resto del semestre.

Por lo tanto, este séptimo *sprint* se considera una pausa en el proyecto, sin avances ni resultados relevantes que reportar.

Octavo *sprint* 09-09-2024 / 16-10-2024

El proyecto se retomó con el comienzo del primer semestre del nuevo curso. Debido al tiempo transcurrido y al tiempo limitado al inicio del semestre, la reunión formal se llevó a cabo el 27 de septiembre. Sin embargo, el *sprint* se considera iniciado desde el momento en que se reactivaron las actividades relacionadas con el proyecto.

Para comenzar, se revisaron todos los documentos externos e internos relacionados con el proyecto con el fin de ponerse al día y continuar sin errores derivados de la pérdida de conocimiento durante la pausa.

Durante la reunión, se evaluó el estado de avance del proyecto, las tareas propuestas en la última reunión previa al accidente y cómo proceder a partir de allí. Las decisiones y tareas principales planteadas incluyeron:

- Pulir y comprobar las funcionalidades existentes de la página web, dividiendo correctamente sus partes para mayor claridad y eficiencia.
- Cambiar la página inicial, permitiendo al usuario elegir entre crear un nuevo experimento o cargar uno ya existente. Se planteó implementar un mecanismo para guardar y reutilizar experimentos previamente creados.
- Añadir la funcionalidad de descargar los resultados obtenidos tras la ejecución del algoritmo *TRACLUS*, entregando los datos al usuario en formato *.zip*.

- Implementar correctamente el algoritmo *TRACLUS* en la aplicación. A partir de este momento, el algoritmo se ejecutaría en la pantalla de carga de datos, unificando la interacción con los datos para reducir tiempos de espera.
- Finalizar la comparación de algoritmos de *clustering* para determinar cuáles podrían ser integrados en la aplicación.
- Continuar el desarrollo y la documentación de la memoria.

Resultados del octavo *sprint*

Los avances en la aplicación fueron significativos. Se implementaron todas las nuevas funcionalidades propuestas, destacando:

- Se integró correctamente el algoritmo *TRACLUS* en la aplicación, permitiendo su ejecución y el tratamiento de datos desde la pantalla de carga de datos. Los resultados obtenidos tras la ejecución del algoritmo se guardan automáticamente dentro del programa.
- Se creó una nueva página de inicio que permite cargar datos de experimentos anteriores o dirigirse a la pantalla de carga para comenzar un nuevo experimento.
- La funcionalidad de descarga de datos se implementó con éxito. Los datos, ya sean nuevos o previamente cargados, pueden descargarse desde la visualización de resultados a través de la barra de navegación, siendo entregados en un archivo comprimido `.zip`.
- La barra de navegación fue reformada para garantizar que las páginas solo sean accesibles cuando sea posible, según el estado actual de los datos y las acciones realizadas.
- Se analizaron los parámetros configurables en `sklearn` que provocan cambios en los algoritmos de clustering.
- Algunos parámetros útiles son específicos y están interrelacionados con otros datos de entrada.
- Se continuó trabajando en la memoria del proyecto para documentar los avances.

En este *sprint*, las estimaciones de tiempo fueron más precisas gracias a una mayor división en las tareas. Se estimaron 45 horas de trabajo, que prácticamente coincidieron con las horas efectivamente dedicadas, sin contar el tiempo necesario para recuperar conocimientos previos al accidente.

Noveno *sprint* 17-10-2024 / 03-11-2024

Ya teniendo los resultados de los diferentes algoritmos de *clustering* y los datos que causaban modificaciones en los cálculos, se propuso implementar una nueva página para la web. Esta estaría ubicada en la sección de creación de un nuevo experimento, y su función sería darle al usuario la posibilidad de elegir qué algoritmos y con qué parámetros ejecutar el experimento, permitiendo realizar comparaciones.

Además, la aplicación había avanzado rápidamente, dejando el formato relegado a favor de la velocidad, lo cual, aunque útil inicialmente, podría generar problemas en el futuro en cuanto a mantenimiento y comprensión. Por tanto, se propuso remodelarla para adherirse lo mejor posible al modelo Vista-Controlador (MVC).

Finalmente, se identificó que los datos devueltos no eran del todo satisfactorios, por lo que se propuso reformar esta parte y buscar formas más claras y útiles de mostrarlos.

Resultados del noveno *sprint*

- Se implementó una nueva pantalla entre la página de inicio y la página de carga de datos:
 - La pantalla permite seleccionar entre los cinco algoritmos de *clustering*, pudiendo elegir desde uno hasta todos.
 - Es obligatorio integrar los datos de todos los algoritmos seleccionados.
 - Una vez completada esta acción, se accede a la página de carga de datos donde se introducen los datos y se ejecuta el algoritmo TRACLUS tantas veces como algoritmos hayan sido seleccionados.
 - Esta funcionalidad incrementa el tiempo de ejecución, pero permite al usuario comparar los resultados de los distintos algoritmos.
- Para que la representación de los datos funcione correctamente:

- Se crearon desplegables que permiten al usuario seleccionar qué algoritmo utilizar para cada representación de datos.
 - Estos desplegables se activan o desactivan dependiendo de si el algoritmo ha sido utilizado.
- Se reorganizó la estructura interna del programa:
- Esto facilitó la búsqueda de funciones y los cambios necesarios en el código.

La estimación de tiempo fue de 45 horas, las cuales se cumplieron como en el *sprint* anterior.

Décimo *sprint* 04-11-2024 / 18-11-2024

En este punto, la web estaba bastante avanzada, por lo que se abrió una nueva línea de trabajo necesaria para cualquier página web: su despliegue. Durante la reunión se discutió cuál sería el medio para lograrlo, llegando a la conclusión de que debía usarse un servicio remoto para ello.

Además, aún quedaban muchos aspectos por pulir en la aplicación. Se propusieron nuevas funcionalidades para los experimentos, como la posibilidad de realizar peticiones adicionales o la capacidad de llamar al mismo algoritmo con diferentes datos.

Por último, debido a la proximidad de la fecha de entrega, se destacó la necesidad de avanzar en la redacción de la memoria con mayor celeridad, ya que varios apartados con mayor carga de trabajo aún no habían sido completados.

Resultados del décimo *sprint*

- Despliegue de la web:
- La web fue desplegada correctamente tras analizar diversas opciones, entre las que se seleccionó Render como la solución ideal.
 - Para el despliegue, fue necesario modificar varios apartados de la aplicación, ya que algunas características no eran compatibles con un entorno remoto.
- Reorganización y ajustes:

- Se reorganizó nuevamente la estructura de la web para mejorar su comprensión y facilitar su mantenimiento.
- Las nuevas funcionalidades propuestas para los experimentos, como las peticiones adicionales y la ejecución del algoritmo con diferentes datos, fueron descartadas debido al aumento excesivo de la complejidad y los tiempos de carga.

■ **Avance de la memoria:**

- Se avanzó considerablemente en la redacción de la memoria, incluyendo la descripción detallada de todos los experimentos realizados hasta la fecha en el proyecto.
- También se actualizaron y rediseñaron diagramas creados previamente para reflejar los importantes cambios que la aplicación había sufrido en los últimos *sprints*.

Se estimaron unas 45 horas para este *sprint*, las cuales se cumplieron como en los anteriores.

Undécimo *sprint* 19-11-2024 / 03-12-2024

Con la web ya muy avanzada, se buscó darle más solidez. Para ello, se propuso solucionar los errores más visibles y comunes, como un error de conexión que ocasionalmente ocurría al cargar los datos o ejecutar el algoritmo TRACLUS.

Complementario a lo anterior, se buscó completar la funcionalidad existente. Se propuso añadir la opción de eliminar experimentos creados anteriormente para mantener un entorno de trabajo limpio. Además, se trabajó en reorganizar la interfaz, reubicando mapas y botones para mejorar la visualización y asegurarse de que los resultados fueran correctos y los datos pudieran ampliarse para un análisis más exhaustivo.

Con el objetivo de mejorar la calidad del desarrollo, se propuso implementar dos tipos de pruebas comunes en el desarrollo de software: análisis estático y pruebas unitarias. Además, como en los sprints anteriores, se continuó avanzando en la memoria del proyecto, centrándose en esta ocasión en los anexos.

Fuera de la reunión, surgieron dos tareas adicionales:

- Paralelizar la ejecución de varios algoritmos de clustering de manera simultánea, ya que el objetivo principal de la aplicación es comparar sus resultados.
- Explorar nuevas fuentes de datos para el análisis, evaluando cuáles de ellas podrían ser útiles.

Resultados del undécimo *sprint*

Las tareas propuestas en este sprint mejoraron significativamente la aplicación web:

- **Resolución de errores:** El problema de conexión se identificó como una incompatibilidad de versiones, que se solucionó actualizando el archivo `requirements.txt`.
- **Eliminación de experimentos:** Esta funcionalidad se implementó en la pantalla de inicio. Ahora, tras seleccionar un experimento anterior, el usuario puede acceder a sus resultados o eliminarlo. Para evitar eliminaciones accidentales, se añadió una ventana modal de confirmación.
- **Mejoras en la interfaz:** Se corrigieron errores en la tabla de datos relacionados con los índices de las trayectorias utilizadas. Además, se agregó a la página de estadísticas la capacidad de visualizar la tabla de segmentos por clúster para cada algoritmo de clustering. También se reubicaron varios botones para mejorar su posición.
- **Testing:**
 - Gracias al análisis estático realizado mediante SonarQube, se identificaron y corrigieron redundancias, variables mal nombradas o sin uso, y otros errores.
 - Las pruebas unitarias intentadas con `pytest` no fueron completamente útiles, ya que no se encontró una forma viable de probar los `callbacks` de Dash durante la ejecución de la aplicación. Por lo tanto, se decidió continuar con pruebas manuales para evaluar el flujo de la aplicación.
- **Optimización del código:** Se optimizaron las funciones de la clase `clustering.py` dividiendo la ejecución en hilos. Se implementó una solución donde cada algoritmo seleccionado se ejecuta en un hilo separado. Aunque se intentó paralelizar también la generación de mapas y

tablas, esto no fue completamente posible debido a la incompatibilidad de `matplotlib` con la biblioteca `threading`. Solo se logró paralelizar la creación de tablas.

- **Nuevas fuentes de datos:** Se evaluaron tres posibles formas de obtener datos. Dos de ellas, basadas en bibliotecas que generaban trayectorias artificiales, fueron descartadas porque sus resultados no aportaban un análisis significativo. La tercera fuente, un conjunto amplio de archivos con coordenadas registradas en distintos contextos (taxis, movimiento urbano, movimiento de animales, etc.), se consideró prometedora y se decidió investigar más a fondo.

Inicialmente, se estimaron 45 horas para completar este sprint. Sin embargo, con la inclusión de las dos tareas adicionales, la estimación se incrementó a 50 horas. Finalmente, el tiempo real invertido fue cercano a las 60 horas.

Duodécimo *sprint* 04-12-2024 / 15-12-2024

Con los resultados del *sprint* anterior, se identificó la necesidad de mejorar el proceso de testeo mediante la implementación de pruebas unitarias enfocadas en las funciones clave de la aplicación, tales como los modelos, la carga de datos, el algoritmo TRACLUS y la generación de mapas. El objetivo principal era garantizar el correcto funcionamiento de estas partes sin depender exclusivamente de pruebas manuales.

Además, se detectó un área de mejora en la organización del apartado de pruebas y experimentos. Este se encontraba en una carpeta externa a la aplicación principal y contenía numerosos archivos utilizados durante el desarrollo del proyecto. Aunque algunos de estos archivos ya se habían subido a GitHub, otros permanecían sin integrar adecuadamente. Se propuso realizar un análisis exhaustivo para seleccionar aquellos archivos que pudieran documentar mejor el proceso de desarrollo, así como optimizar su organización para facilitar la comprensión.

Por otro lado, se planteó analizar nuevos conjuntos de datos disponibles con el fin de evaluar su compatibilidad con el algoritmo TRACLUS. En caso de ser compatibles, se realizarían pruebas exhaustivas del algoritmo con diferentes tamaños y cantidades de datos, buscando validar su rendimiento y eficacia.

Finalmente, como en los *sprints* anteriores, se continuó trabajando en la documentación del proyecto y los anexos.

Resultados del duodécimo *sprint*

Avances en el testing Se crearon dos archivos de pruebas unitarias utilizando `pytest`, que validaron el correcto funcionamiento de los modelos de la aplicación y otras funciones esenciales. Esto permitió automatizar el proceso de testeо y asegurar la calidad del código de manera más robusta.

Mejoras en la aplicación Se realizaron ajustes adicionales en la aplicación, incluyendo la implementación de modales para guiar al usuario y notificar errores. Esto mejoró significativamente la experiencia de usuario y redujo posibles confusiones durante la interacción con la página web.

Organización del apartado de experimentos Tras analizar los archivos de la carpeta de experimentos, se seleccionaron y documentaron los siguientes:

- `Cluster_algorit_test.ipynb`: Contiene las funciones utilizadas para analizar los resultados del TRACLUS aplicando diferentes algoritmos de clustering fuera de la página web.
- `Data_comparative_test.ipynb`: Desarrolla representaciones comparativas de los datos obtenidos.
- `Optimization_TRACLUS.ipynb`: Explora diversos intentos de optimización del algoritmo TRACLUS.
- `Geolife_test.ipynb`: Representa datos provenientes de un conjunto externo (Geolife). Sin embargo, este conjunto fue descartado por no cumplir con el formato requerido por el TRACLUS.

Búsqueda y adaptación de nuevos conjuntos de datos Se exploraron múltiples conjuntos de datos disponibles. Aunque ninguno cumplía directamente con el formato necesario, se desarrolló una metodología para adaptarlos. Generalmente, estos conjuntos presentaban coordenadas individuales por fila; para ajustarlos, se agruparon las coordenadas en listas de listas según criterios lógicos, como la hora, el IP del dispositivo o cualquier identificador común.

Progreso en la documentación Se avanzó significativamente en los anexos y la memoria, consolidando la información y resultados obtenidos durante el desarrollo del proyecto.

Aunque se estimaron 35 horas para completar las tareas, interrupciones imprevistas redujeron el tiempo efectivo a 22 horas. A pesar de esto, todos los objetivos planteados para este *sprint* se cumplieron satisfactoriamente.

Décimo tercer *sprint* 16-12-2024 / —

A.3. Estudio de viabilidad

En este apartado se analizará la viabilidad de este proyecto en un entorno profesional, evaluando los aspectos económicos y legales.

Viabilidad económica

La viabilidad económica es un aspecto absolutamente necesario en cualquier proyecto de software, ya que estos, para bien o para mal, suelen depender de los costes. Para este análisis se tomarán en cuenta todas las herramientas utilizadas, el coste profesional y el desgaste de los dispositivos empleados.

Salarios

Si este proyecto se hubiera llevado a cabo en una entidad privada, un ingeniero informático habría estado a cargo del desarrollo. El salario de este empleado dependería de la ubicación, el cargo y la experiencia. En este caso, tomaremos como referencia el salario medio bruto de un ingeniero informático en España. Según diversas fuentes como UAX, Bankinter y Talent.com, este ronda los 20.000 euros anuales para perfiles con poca experiencia.

Teniendo en cuenta que el proyecto consumió alrededor de **X horas**, el salario correspondiente sería de aproximadamente **X euros**.

Además, el trabajador podría enfrentarse a circunstancias imprevistas, como una baja médica, tal como ocurrió en este caso. Una situación así representaría un coste adicional para la empresa, que tendría que decidir entre sustituir al ingeniero durante la baja o postergar el proyecto. En España, durante una baja médica, el estado abona una parte del salario, mientras que la empresa debe cubrir otra. Si esta pausa hubiera durado 3 meses, el coste adicional podría ascender a aproximadamente **X euros**.

Hardware

Otro coste adicional son los materiales utilizados. Aunque estos suelen durar más tiempo que el propio proyecto, deben ser amortizados anualmente. Para este análisis se aplicó una amortización de 5 años, es decir, el 20 % anual.

Descripción	Coste Total	Amortización Anual
Ordenador	900€	180€
Periféricos	260€	52€
Material de oficina	20€	4€
Imprevistos	40€	8€
Total	1,220€	244€

Tabla A.1: Amortización de hardware y materiales.

Durante el proyecto, pueden surgir imprevistos relacionados con el material. En este caso, uno de los componentes se averió y tuvo que ser sustituido.

Software

La mayoría de las herramientas de software utilizadas en este proyecto son de código abierto, por lo que no generan un coste adicional.

Sin embargo, el único servicio que sí representa un gasto es el proveedor de hosting para la página web, Render. Este cuenta con un plan gratuito, aunque no permite un despliegue sin interrupciones. Para evitar este problema, sería necesario optar por un plan de pago.

El plan más económico cuesta aproximadamente 7€ mensuales, pero tiene capacidades limitadas. En contraste, el plan *Pro Ultra*, que ofrece una mayor capacidad de cómputo adecuada para experimentos a gran escala, tiene un coste de 450€ mensuales. Para esta simulación, se considerará este último plan como el elegido.

Otras opciones que podrían haber sido evaluadas incluyen AWS o Azure. Sin embargo, para una evaluación adecuada sería necesario considerar factores como el uso esperado o el número de usuarios simultáneos.

Gastos fijos

Aunque el teletrabajo es común en la actualidad, muchas empresas continúan operando desde oficinas, lo que conlleva ciertos gastos fijos anuales.

Descripción	Coste Anual
Internet y comunicaciones	600€
Electricidad y mantenimiento	400€
Limpieza	1,000€
Total	2,000€

Tabla A.2: Gastos fijos estimados.

Se han considerado los recursos básicos necesarios para garantizar un entorno funcional en una empresa informática.

Costes y beneficios

Finalmente, al calcular todos los costes, incluyendo los salarios, hardware, software y gastos fijos, se puede realizar una estimación global del coste del proyecto. A continuación, se presenta un desglose detallado:

Descripción	Coste
Personal (Neto)	X€
Hardware (Amortización)	X€
Software (Servicios)	X€
Gastos fijos	X€
Total Anual	X€

Tabla A.3: Resumen de costes del proyecto.

El coste total estimado del proyecto se sitúa en torno a **X euros**, considerando todos los aspectos mencionados.

En cuanto a los beneficios, estos dependerán del valor añadido que el proyecto pueda ofrecer a los usuarios finales y de la capacidad de la empresa para monetizar las funcionalidades desarrolladas. No obstante, este proyecto no parece estar diseñado para una monetización directa. Su principal utilidad radica en su carácter experimental, proporcionando una base sólida para

continuar con la implementación del algoritmo TRACLUS en diferentes ámbitos. Por tanto, su valor reside más en su potencial académico y como base tecnológica que en su rentabilidad económica inmediata.

Viabilidad legal

La viabilidad legal de un proyecto es un aspecto crítico, especialmente cuando se planea publicarlo para obtener un beneficio económico. Este análisis depende de las licencias asociadas a cada una de las herramientas y bibliotecas utilizadas durante su desarrollo.

A continuación, se presentan las herramientas y bibliotecas utilizadas, junto con sus licencias respectivas, así como una descripción de sus implicaciones legales.

Herramientas

Herramienta	Licencia
Python 3.11.2	PSF License (Python Software Foundation License)
Git	GNU General Public License v2.0
Texmaker	GNU General Public License v2

Tabla A.4: Licencias de las herramientas utilizadas.

■ Python 3.11.2

- Licencia: PSF License (Python Software Foundation License).
- Características:
 - Permite usar, modificar y distribuir, tanto para proyectos abiertos como comerciales.
 - Compatible con licencias como la GPL.
 - Distribuido "tal cual", sin garantías.
- Implicaciones: Asegura el acceso y uso de Python en proyectos de todo tipo.

■ Git

- Licencia: GNU General Public License v2.0.
- Características:

- Licencia copyleft fuerte que asegura que las modificaciones al código fuente sean abiertas y compartidas bajo la misma licencia.
- Implicaciones: Es ideal para fomentar la transparencia en sistemas de control de versiones.

■ **Texmaker**

- Licencia: GNU General Public License v2.
- Características:
 - Requiere que cualquier modificación o redistribución mantenga el código fuente abierto bajo la misma licencia.
- Implicaciones: Garantiza que herramientas de edición de LaTeX sigan siendo accesibles para la comunidad.

Bibliotecas

Biblioteca	Licencia
Dash	MIT
Pandas	BSD
Matplotlib	Matplotlib License
Plotly	MIT
NumPy	BSD
sklearn.cluster	BSD
Shapely	BSD
PyProj	MIT
Contextily	BSD personalizada

Tabla A.5: Licencias de las bibliotecas utilizadas.

■ **Licencia MIT**

- Características:
 - Permisiva: permite uso, modificación y redistribución incluso con fines comerciales.
 - Requiere incluir el aviso de copyright y la licencia original.
- Implicaciones: Ideal para proyectos abiertos o comerciales.
- Usada por: Dash, Plotly, PyProj.

■ Licencia BSD

- Características:
 - Permite uso y redistribución con atribución al autor original.
 - Existen dos versiones principales: BSD-2-Clause (simplificada) y BSD-3-Clause (que incluye restricciones adicionales para evitar promoción indebida).
- Implicaciones: Muy permisiva y compatible con proyectos comerciales.
- Usada por: Pandas, NumPy, sklearn, Shapely, Contextily.

■ Licencia Matplotlib

- Características:
 - Basada en la licencia BSD, permite uso y redistribución con atribución.
- Implicaciones: Compatible con proyectos comerciales.
- Usada por: Matplotlib.

■ Licencia Contextily (BSD personalizada)

- Características:
 - Similar a BSD, con ajustes específicos para el uso de datos abiertos y mapas.
- Implicaciones: Garantiza libertad de uso y redistribución.

Conclusión

En general, todas las herramientas y bibliotecas utilizadas en este proyecto están respaldadas por licencias permisivas que permiten su uso, modificación y distribución, incluso para fines comerciales. Sin embargo, para garantizar la viabilidad legal, es fundamental cumplir con los requisitos específicos de cada licencia, como la atribución de autoría en las licencias MIT y BSD, o la redistribución bajo la misma licencia en herramientas GPL como Git y Texmaker.

Esto asegura que el proyecto puede publicarse y distribuirse sin conflictos legales, siempre que se respeten las condiciones de las licencias asociadas.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apartado se detallan los requisitos del proyecto, destacando la importancia de establecerlos antes de iniciar la implementación. Este enfoque estructurado facilita la organización, minimiza errores y proporciona una base sólida para el desarrollo.

B.2. Objetivos generales

El proyecto tenía como objetivo abordar desafíos en el análisis de trayectorias geoespaciales mediante el desarrollo de un sistema innovador y eficiente. A continuación, se detallan los objetivos principales:

- **Desarrollar un algoritmo de Big Data denominado TRA-CLUS para el análisis y agrupación de trayectorias geoespaciales.** El algoritmo TRA-CLUS debía adaptarse a grandes volúmenes de datos geoespaciales, permitiendo identificar patrones y relaciones en trayectorias complejas. Este objetivo implica implementar un modelo eficiente que sea capaz de procesar datos masivos y ofrecer resultados precisos y fiables.
- **Facilitar la interpretación de los datos recogidos mediante representaciones gráficas claras y precisas.** Con el fin de transformar datos complejos en información comprensible, se buscaba desarrollar herramientas de visualización intuitivas. Estas debían permitir a los

usuarios finales analizar patrones, relaciones y tendencias de forma rápida y efectiva.

- **Realizar comparativas de rendimiento y precisión entre diferentes algoritmos de clustering.** Este objetivo se centraba en evaluar el rendimiento del algoritmo TRA-CLUS en comparación con otros algoritmos de clustering establecidos, utilizando métricas como la eficiencia computacional, la precisión en la agrupación y la robustez frente a datos ruidosos o incompletos.
- **Diseñar y desarrollar una aplicación web interactiva para mostrar y analizar los resultados obtenidos por el algoritmo.** La aplicación debía ofrecer una interfaz amigable que permitiera a los usuarios interactuar con los resultados del algoritmo TRA-CLUS. Además, debía incluir funcionalidades como filtros personalizados, visualización en mapas y generación de informes para facilitar el análisis y la toma de decisiones basada en datos.

Estos objetivos generales constituyen la base del proyecto, guiando el diseño y desarrollo hacia la creación de una solución integral y eficiente para el análisis de trayectorias geoespaciales.

B.3. Catálogo de Requisitos

En este apartado se desglosan los requisitos del proyecto en dos categorías principales: funcionales y no funcionales. Los requisitos funcionales describen las características que el sistema debe implementar para cumplir sus objetivos, mientras que los requisitos no funcionales detallan las restricciones y cualidades del sistema.

Requisitos funcionales

A continuación, se enumeran los requisitos funcionales del sistema:

- **RF-1: Gestión de experimentos**
 1. **RF-1.1: Navegar entre experimentos.** El sistema debe permitir al usuario acceder a experimentos existentes o crear nuevos experimentos desde una página inicial.

2. **RF-1.2: Crear y guardar experimentos.** Los usuarios deben poder crear experimentos asignándoles un nombre único y guardar sus datos y resultados asociados.
3. **RF-1.3: Almacenar datos cargados y resultados.** Cada experimento debe guardar los datos cargados y los resultados generados por el algoritmo TRA-CLUS.

- **RF-2: Interacción con datos**

1. **RF-2.1: Selección de datos de entrada.** El sistema debe proporcionar una página para seleccionar los clústeres que se van a usar y configurar cómo se procesarán los datos.
2. **RF-2.2: Carga de datos.** Los usuarios deben poder cargar conjuntos de datos en la aplicación para procesarlos mediante el algoritmo TRA-CLUS.
3. **RF-2.3: Visualización de datos sin tratar.** Los datos originales deben visualizarse en una página que muestre un mapa interactivo de las trayectorias cargadas.
4. **RF-2.4: Visualización de resultados del algoritmo.** Los resultados generados por el algoritmo deben representarse gráficamente en un mapa interactivo que muestre los clústeres resultantes.
5. **RF-2.5: Análisis tabular.** Los resultados deben incluirse en tablas detalladas que permitan a los usuarios analizar los datos agrupados y realizar comparaciones.

- **RF-3: Avance en la implementación del algoritmo TRA-CLUS**

1. **RF-3.1: Optimización del algoritmo TRA-CLUS.** El sistema debe avanzar en la implementación real del algoritmo TRA-CLUS, mejorando su rendimiento y asegurando su capacidad para manejar grandes volúmenes de datos geoespaciales. Este avance debe incluir optimizaciones en la precisión de los clústeres generados y en la eficiencia de procesamiento.

- **RF-4: Descarga de datos**

1. **RF-4.1: Exportar resultados.** El sistema debe incluir un botón en la barra de navegación que permita descargar los resultados generados en formatos .zip.

Requisitos no funcionales

A continuación, se detallan los requisitos no funcionales del sistema:

- **RNF-1: Rendimiento**

1. **RNF-1.1: Procesamiento eficiente.** La carga y el procesamiento de datos debe reducirse todo lo que sea posible para mejorar la viabilidad del algoritmo.
2. **RNF-1.2: Renderización de mapas.** Los mapas interactivos deben renderizarse sin interrupciones.

- **RNF-2: Usabilidad**

1. **RNF-2.1: Navegación intuitiva.** La aplicación debe ofrecer una interfaz fácil de usar con una navegación clara entre páginas y funcionalidades.
2. **RNF-2.2: Diseño responsivo.** El sistema debe adaptarse a diferentes tamaños de pantalla, asegurando una experiencia óptima en dispositivos móviles, tablets y escritorios.
3. **RNF-2.3: Indicadores visuales.** La interfaz debe incluir indicadores que muestren el estado de carga, procesamiento y finalización de tareas.

- **RNF-3: Compatibilidad**

1. **RNF-3.1: Soporte para múltiples navegadores.** La aplicación debe funcionar correctamente en los navegadores web más utilizados, como Chrome, Firefox, Edge y Safari.
2. **RNF-3.2: Integración con formatos estándar.** Los datos descargados deben ser compatibles con programas comunes como Excel y herramientas de análisis como Python o R.

- **RNF-4: Seguridad**

1. **RNF-4.1: Protección de datos.** El sistema debe garantizar que los datos cargados por el usuario se mantengan seguros y no sean accesibles por terceros.
2. **RNF-4.2: Control de errores.** El sistema debe manejar errores comunes como formatos de datos incorrectos, alertando al usuario y ofreciendo posibles soluciones.

- **RF-5: Borrado de un experimento existente**

1. **RF-5.1: Eliminar un experimento previamente guardado de manera irreversible** El sistema permitirá al usuario eliminar un experimento que haya sido previamente guardado en el sistema. Esta acción será irreversible, y no se podrá recuperar el experimento una vez eliminado.
2. **RF-5.2: Asegurarse de que la eliminación es deseada por el usuario** Antes de proceder con la eliminación, el sistema solicitará al usuario una confirmación adicional para evitar la eliminación accidental. Esto puede implicar un cuadro de diálogo de confirmación con opciones como "Sí, eliminar." o "Cancelar".

B.4. Especificación de requisitos

En la ejecución del programa, el actor tiene tres acciones posibles: crear, cargar y descargar un experimento.

La creación de un nuevo experimento requiere que el usuario seleccione los algoritmos y los datos que se van a utilizar, lo cual incluye el paso de cargar los datos. Una vez realizados estos pasos, el nuevo experimento se guarda automáticamente y el usuario es llevado al apartado de visualización de datos.

Para cargar un experimento anterior, debe existir un experimento guardado previamente; en caso contrario, la selección del experimento (necesaria para este proceso) no será posible. Tras cargar el experimento, al igual que en la creación de uno nuevo, el usuario es dirigido al apartado de visualización de datos.

Finalmente, la descarga de un experimento solo será posible si el usuario se encuentra en el apartado de visualización, desde donde el programa permitirá realizar esta acción.

En conjunto, el sistema asegura un flujo coherente de acciones, permitiendo al usuario gestionar experimentos mediante la creación, carga, visualización y descarga de datos.

A continuación se mostrarán las tablas de uso:

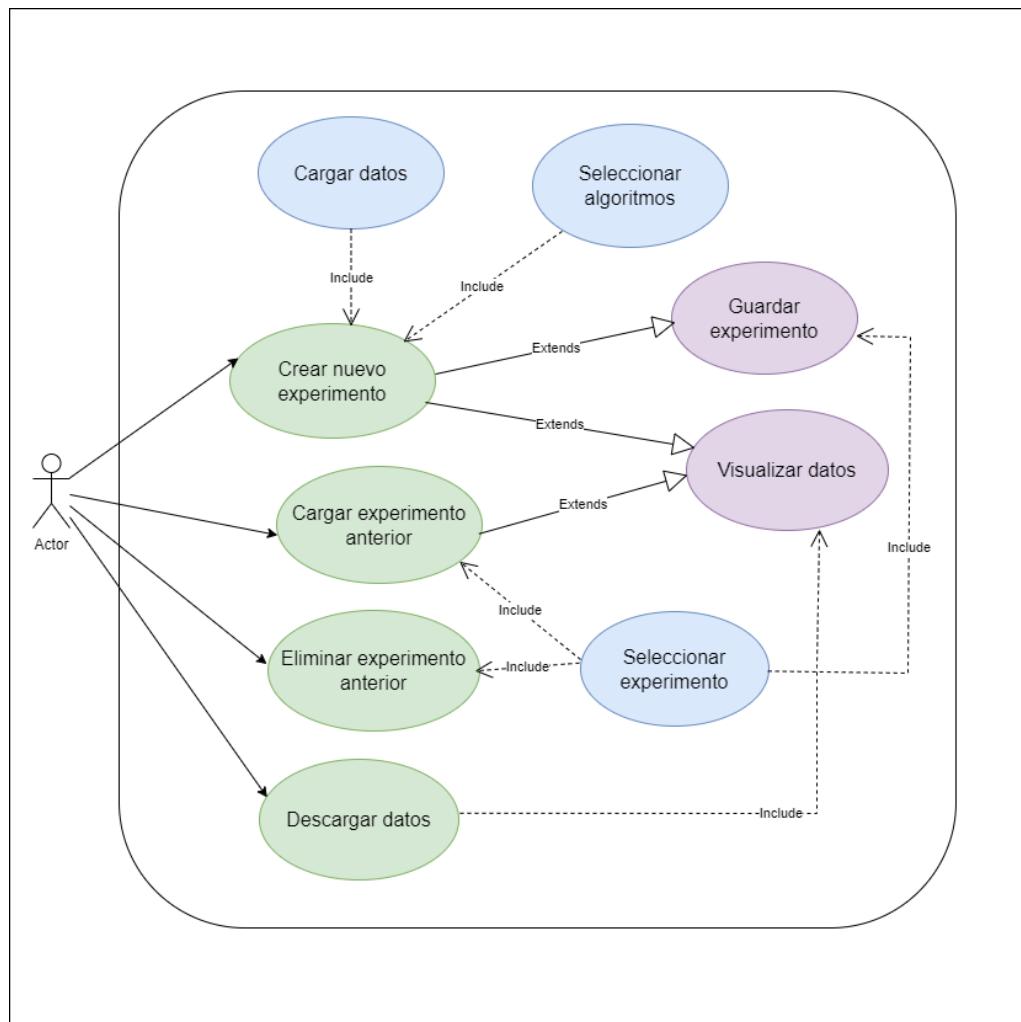


Figura B.1: Diagrama de casos de uso.

CU-1	Crear un nuevo experimento
Versión	1.0
Autor	Álvaro González Delgado
Requisitos asociados	RF-1, RF-2, RF-3
Descripción	El usuario crea un nuevo experimento seleccionando los algoritmos y los datos que se van a usar. Los datos se cargan automáticamente, y el experimento se guarda en el sistema. El usuario es dirigido a la página de visualización de datos.
Precondición	El sistema debe permitir crear un nuevo experimento, no debe haber restricciones previas. Se debe haber seleccionado como mínimo un experimento y rellenando sus datos. El usuario deba haber introducido un conjunto de datos válido.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la página de creación de un experimento. 2. Elige los algoritmos y variables a utilizar. 3. Se accede a la pagina de carga de datos. 4. Se da un nombre Y se introducen los datos. 5. Se carga los datos. 6. Se ejecuta el algoritmo TRA-CLUS con todas las variables introducidas. 7. Se finaliza la carga del algoritmo. 8. El sistema guarda el experimento automáticamente. 9. El sistema redirige al usuario a la página de visualización de datos.
Postcondición	El experimento es guardado en el sistema y se presenta la página de visualización de datos.
Excepciones	Si los datos seleccionados no dan un resultado válido o hay un error de conexión el sistema muestra un error.
Importancia	Alta

Tabla B.1: CU-1 Crear un nuevo experimento.

CU-2	Cargar un experimento existente
Versión	1.0
Autor	Álvaro González Delgado
Requisitos asociados	RF-1, RF-2
Descripción	El usuario carga un experimento previamente guardado. El sistema permite seleccionar un experimento de la lista disponible y lo carga para continuar con el análisis.
Precondición	Debe existir al menos un experimento previamente guardado.
Acciones	<ol style="list-style-type: none"> 1. El usuario elige el experimento de la lista de experimentos guardados. 2. El sistema carga el experimento y presenta la página de visualización de datos.
Postcondición	El experimento cargado se muestra en la página de visualización de datos.
Excepciones	Si no existen experimentos guardados, el sistema muestra el desplegable vacío y no puede seleccionarse nada.
Importancia	Alta

Tabla B.2: CU-2 Cargar un experimento existente.

CU-3	Descargar los resultados de un experimento
Versión	1.0
Autor	Álvaro González Delgado
Requisitos asociados	RF-4
Descripción	El usuario puede descargar los resultados generados por el algoritmo TRA-CLUS en formato .zip.
Precondición	El usuario debe estar en una de las páginas de visualización de datos.
Acciones	<ol style="list-style-type: none"> 1. El usuario accede a la página de visualización de datos cargando un experimento nuevo o guardado. 2. El usuario presiona el botón de descarga. 3. El sistema descarga los resultados en el formato .zip.
Postcondición	Los resultados se descargan correctamente al dispositivo del usuario.
Excepciones	Si no hay resultados disponibles para descargar, el sistema no hace nada.
Importancia	Media

Tabla B.3: CU-3 Descargar los resultados de un experimento.

CU-4	Borrar un experimento existente
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-5
Descripción	El usuario puede borrar un experimento previamente guardado. La eliminación es irreversible.
Precondición	El experimento debe existir previamente.
Acciones	<ol style="list-style-type: none"> 1. El usuario elige el experimento que desea borrar de la lista. 2. El sistema despliega la petición de confirmación. 3. El sistema elimina el experimento y confirma la eliminación.
Postcondición	El experimento es eliminado del sistema de forma irreversible.
Excepciones	Si el experimento no existe, el botón no racionara.
Importancia	Media

Tabla B.4: CU-4 Borrar un experimento existente.

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado se explicarán las especificaciones de diseño utilizadas durante el desarrollo de este proyecto, dividido en varios apartados. Cada uno describe los aspectos clave del sistema y las decisiones tomadas para garantizar su funcionalidad, escalabilidad y eficiencia:

- El **diseño de datos** detalla cómo se organizan, estructuran y gestionan los datos utilizados en el sistema.
- El **diseño procedimental** describe los algoritmos, procesos y flujos operativos implementados en el proyecto.
- El **diseño arquitectónico** define la estructura global del sistema, los componentes principales y las relaciones entre ellos.

Los apartados siguientes desglosan cada uno de estos aspectos en detalle.

C.2. Diseño de datos

El diseño de datos de este proyecto se centra en la gestión, almacenamiento y manipulación eficiente de la información utilizada en la aplicación. A continuación, se describen los principales aspectos relacionados con la estructura de los datos, sus fuentes y su tratamiento.

Estructura de los datos

Los datos utilizados en este proyecto provienen de archivos en formato CSV o Excel, los cuales contienen trayectorias geoespaciales representadas mediante coordenadas GPS. Cada fila del archivo representa una trayectoria, que incluye varios atributos necesarios para el análisis.

- **Estructura básica del archivo Trayectorias de Taxis:**

- **TRIP_ID:** Identificador único de cada trayectoria.
- **CALL_TYPE:** Tipo de llamada para el servicio de taxi (A, B o C).
- **ORIGIN_CALL:** Identificador del cliente (si aplica).
- **ORIGIN_STAND:** Identificador del punto de recogida (si aplica).
- **TAXI_ID:** Identificador único del taxi.
- **TIMESTAMP:** Marca temporal del inicio de la trayectoria.
- **DAY_TYPE:** Tipo de día (laboral, fin de semana, festivo).
- **MISSING_DATA:** Indica si faltan datos en la trayectoria.
- **POLYLINE:** Lista de coordenadas GPS que componen la trayectoria.

- **Requisitos de formato:**

- El archivo debe estar correctamente delimitado por comas o tabulaciones.
- La columna POLYLINE debe contener las coordenadas en un formato JSON válido.
- No se permiten valores nulos en las columnas clave, particularmente en POLYLINE, que será la única columna tratada por el algoritmo.

Se han utilizado otros archivos con parámetros diferentes, pero dado que el algoritmo TRACLUS solo necesita trayectorias para funcionar, únicamente se toma en cuenta esta columna para el análisis.

Modelo de datos

Para facilitar el procesamiento y análisis de los datos, estos se estructuran internamente en un modelo basado en un **GeoDataFrame**, que permite integrar datos tabulares y geoespaciales de manera eficiente.

- Cada fila del `GeoDataFrame` representa una trayectoria única.
- Las coordenadas en `POLYLINE` se convierten en objetos de geometría tipo `LineString`, lo que permite su uso en análisis geoespaciales.

Gestión de datos

Los datos son procesados en diferentes etapas para asegurar su calidad y consistencia:

1. **Validación:** Se verifica que el archivo cargado cumpla con los requisitos de formato y no contenga valores nulos en columnas clave.
2. **Transformación:** Las coordenadas en `POLYLINE` son transformadas en objetos geoespaciales.
3. **Almacenamiento temporal:** Los datos procesados se almacenan temporalmente en memoria para su uso en el análisis y visualización.
4. **Exportación:** Los resultados generados (mapas, estadísticas, archivos transformados) se guardan en formato ZIP para su descarga.

Almacenamiento de resultados

Los resultados obtenidos del análisis y ejecución del algoritmo TRACLUS se almacenan en un directorio específico dentro de la aplicación. Este directorio incluye:

- Los datos procesados, guardados en un archivo GeoJSON.
- Archivos CSV con las trayectorias procesadas y los clústeres generados.
- Mapas en formato HTML para su visualización.

Conclusión

Este diseño garantiza que los datos se carguen y procesen correctamente, aplicando el algoritmo TRACLUS de manera eficiente. Además, permite que los experimentos se revisen de forma sencilla y fluida, facilitando su uso tanto para análisis como para futuros desarrollos.

C.3. Diseño procedimental

El diseño procedimental de este proyecto define las operaciones clave que realiza la aplicación para cumplir con sus objetivos. Estas operaciones incluyen la carga de datos, el procesamiento de trayectorias, la ejecución del algoritmo TRACLUS, la visualización de resultados y la exportación de datos. A continuación, se detalla el flujo y los procedimientos más importantes.

Carga y validación de datos

El proceso comienza con la carga del archivo de datos proporcionado por el usuario. Este procedimiento asegura que los datos estén en el formato correcto antes de continuar.

1. El usuario selecciona un archivo CSV o Excel a través de la interfaz.
2. Se valida el formato del archivo:
 - Verificación de la existencia de la columna POLYLINE.
 - Comprobación de que las coordenadas en POLYLINE estén en un formato JSON válido.
 - Verificación de que no haya valores nulos en las columnas clave.
3. Si el archivo es válido, los datos se cargan en memoria como un **DataFrame** de pandas.
4. En caso de error, se muestra un mensaje detallado al usuario en la interfaz.

Procesamiento de trayectorias

Una vez cargados los datos, se realiza el procesamiento para convertir las trayectorias en un formato adecuado para el análisis.

- Las coordenadas GPS en la columna POLYLINE se transforman en objetos **LineString** utilizando la biblioteca **shapely**.
- Se crea un **GeoDataFrame** que combina datos tabulares con las trayectorias geoespaciales.
- Este **GeoDataFrame** se almacena temporalmente en memoria para los siguientes pasos.

Ejecución del algoritmo TRACLUS

El algoritmo TRACLUS se utiliza para agrupar trayectorias similares en clústeres.

1. Los parámetros del algoritmo son definidos por el usuario, dependiendo del método de clustering seleccionado:
 - Número mínimo de puntos por clúster (**SpectralClustering**).
 - Distancia máxima entre segmentos (**OPTICS**).
2. Se realiza un preprocesamiento para dividir las trayectorias en segmentos.
3. El algoritmo TRACLUS agrupa los segmentos en clústeres basados en su proximidad y similitud.
4. Los resultados incluyen:
 - Trayectorias agrupadas por clúster.
 - Trayectorias representativas de cada clúster.

Visualización de resultados

Los resultados del algoritmo se representan visualmente en mapas interactivos. Todos estos datos se almacenan dentro de la aplicación para poder acceder a ellos en el futuro, y los datos actualmente utilizados se guardan en variables globales para reducir los tiempos de carga entre las pantallas de visualización.

- La página inicial muestra las trayectorias originales en un mapa.
- Los resultados del clustering se visualizan en una página específica, donde cada clúster se representa con un color diferente.
- Se incluyen gráficos estadísticos que resumen la distribución de los clústeres.

Exportación de resultados

El sistema permite al usuario descargar los resultados del análisis en un archivo ZIP.

- El archivo ZIP incluye:
 - Mapas en formato PNG.
 - Archivos CSV con los datos procesados y los clústeres generados.

Conclusión

El diseño procedimental asegura un flujo lógico y eficiente desde la carga de datos hasta la exportación de resultados. Cada paso está diseñado para ser modular y reutilizable, lo que facilita la adaptación del sistema a futuros cambios o mejoras.

C.4. Diseño arquitectónico

El diseño arquitectónico de este proyecto sigue el modelo **Vista-Controlador (MVC)**. Este patrón organiza la aplicación en tres componentes principales: el modelo, la vista y el controlador. Adicionalmente, se ha incluido un componente complementario denominado **Utils** para funciones auxiliares, lo que asegura una separación clara de responsabilidades, facilita el mantenimiento y mejora la escalabilidad del sistema.

Estructura general del modelo MVC

- **Modelo (Model):**
 - Responsable de la gestión de los datos y la lógica de negocio.
 - Incluye funciones para la validación, procesamiento y almacenamiento temporal de datos.
 - Implementado mediante estructuras como **DataFrame** y **GeoDataFrame**.
- **Vista (View):**
 - Encargada de la interacción con el usuario mediante una interfaz gráfica.

- Desarrollada utilizando **Dash**, que aprovecha componentes de React para crear layouts interactivos.
- Proporciona pantallas para la carga de datos, la selección de parámetros y la visualización de resultados.

- **Controlador (Controller):**

- Actúa como intermediario entre el modelo y la vista.
- Gestiona las peticiones del usuario, realiza las operaciones necesarias en el modelo y actualiza la vista con los resultados.
- Incluye la lógica de los **callbacks** de Dash y otras funciones de control específicas.

- **Funciones complementarias (Utils):**

- Proveen soporte adicional para transformar y gestionar datos utilizados por la vista y el controlador.
- Contienen variables globales para almacenar información compartida.
- Ayudan en tareas como guardar y cargar datos, reduciendo la complejidad del controlador.

Diagrama de arquitectura MVC

El siguiente diagrama ilustra cómo se organizan e interactúan los componentes del modelo MVC en este proyecto:

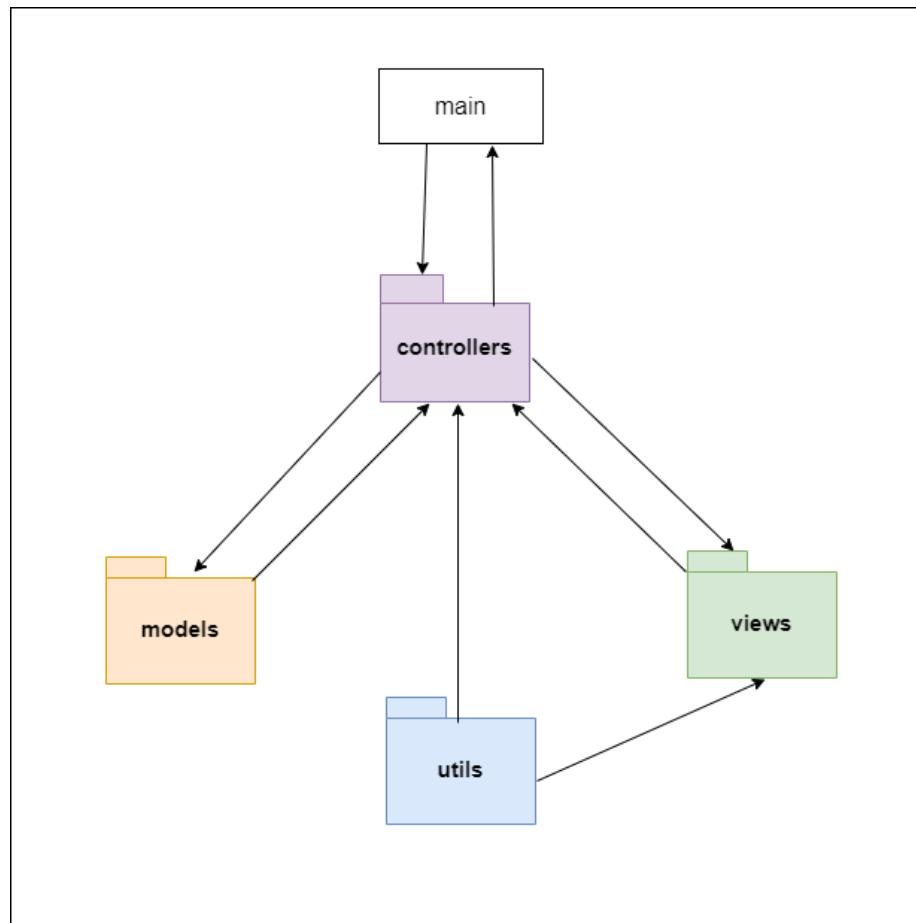


Figura C.1: Diagrama de clases, carpetas.

Flujo de operaciones

El flujo de operaciones en la arquitectura MVC sigue estos pasos principales:

1. El usuario interactúa con la **vista** (por ejemplo, carga un archivo, selecciona parámetros o solicita un análisis).
2. La **vista** envía los eventos generados al **controlador**.
3. El **controlador** procesa las solicitudes y realiza las operaciones necesarias en el **modelo**.
4. El **modelo** devuelve los resultados al **controlador**.

5. El **controlador** actualiza la **vista** con los resultados para que el usuario pueda visualizarlos.

Organización del código

El proyecto está organizado para reflejar la arquitectura MVC, con las siguientes carpetas principales:

- **Main:** Contiene el archivo principal que ejecuta el programa (`main.py`).
- **Modelo (Model):** Almacenado en la carpeta `models`, incluye funciones para el procesamiento de trayectorias, validación de datos y almacenamiento temporal.

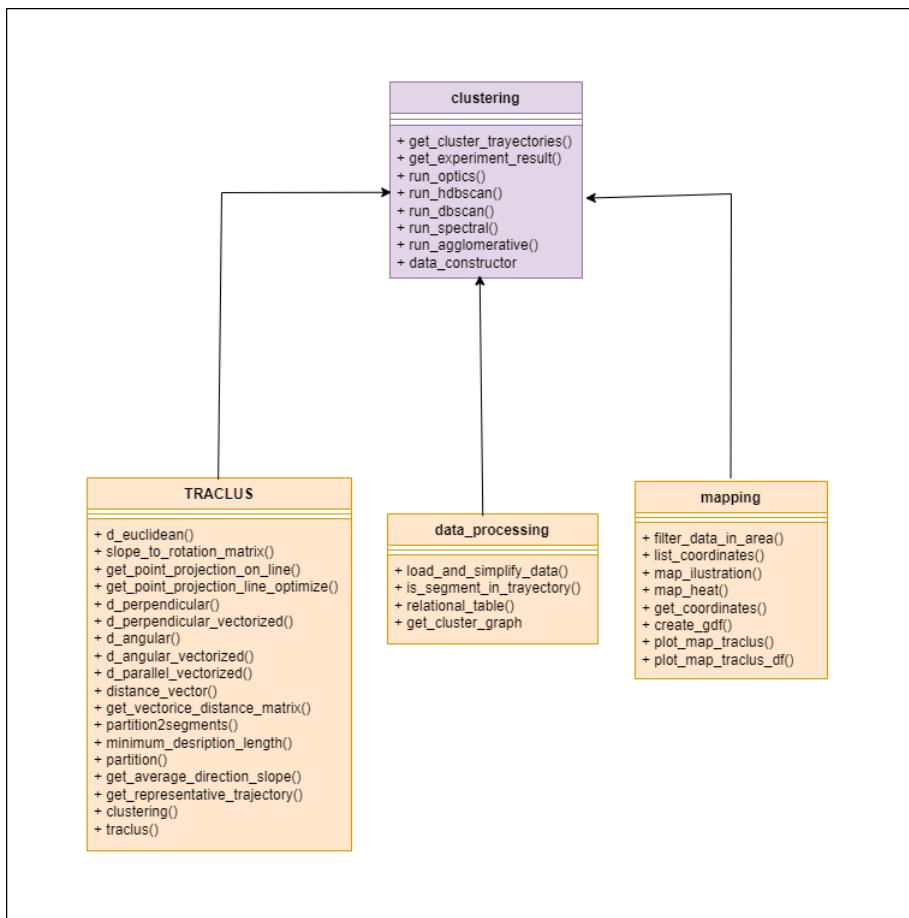


Figura C.2: Diagrama de clases, relaciones modelos.

- **Vista (View):** Situado en la carpeta `views`, contiene los layouts y componentes visuales de Dash.

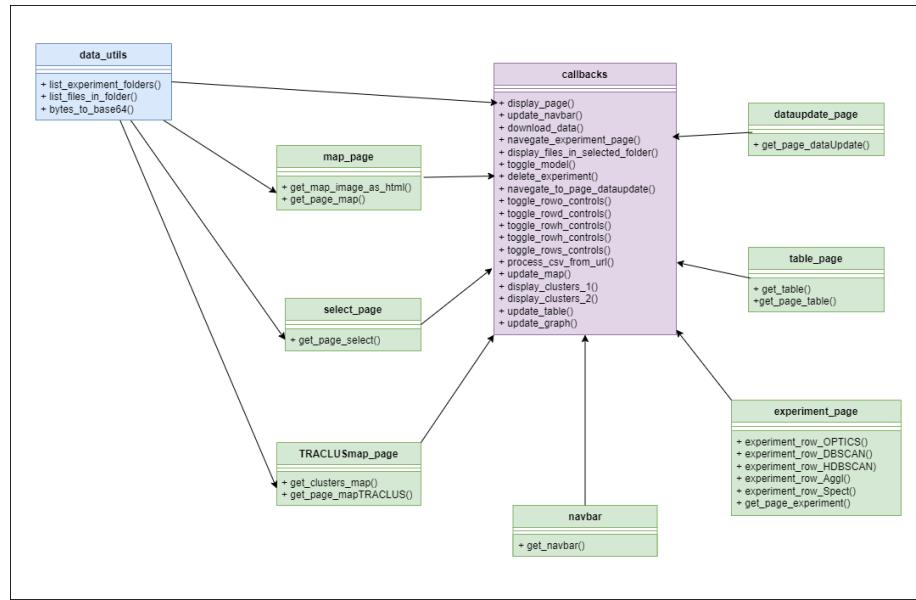


Figura C.3: Diagrama de clases, relaciones vistas.

- **Controlador (Controller):** Ubicado en la carpeta `controllers`, implementa la lógica de los `callbacks` de Dash y coordina las operaciones entre el modelo y la vista.

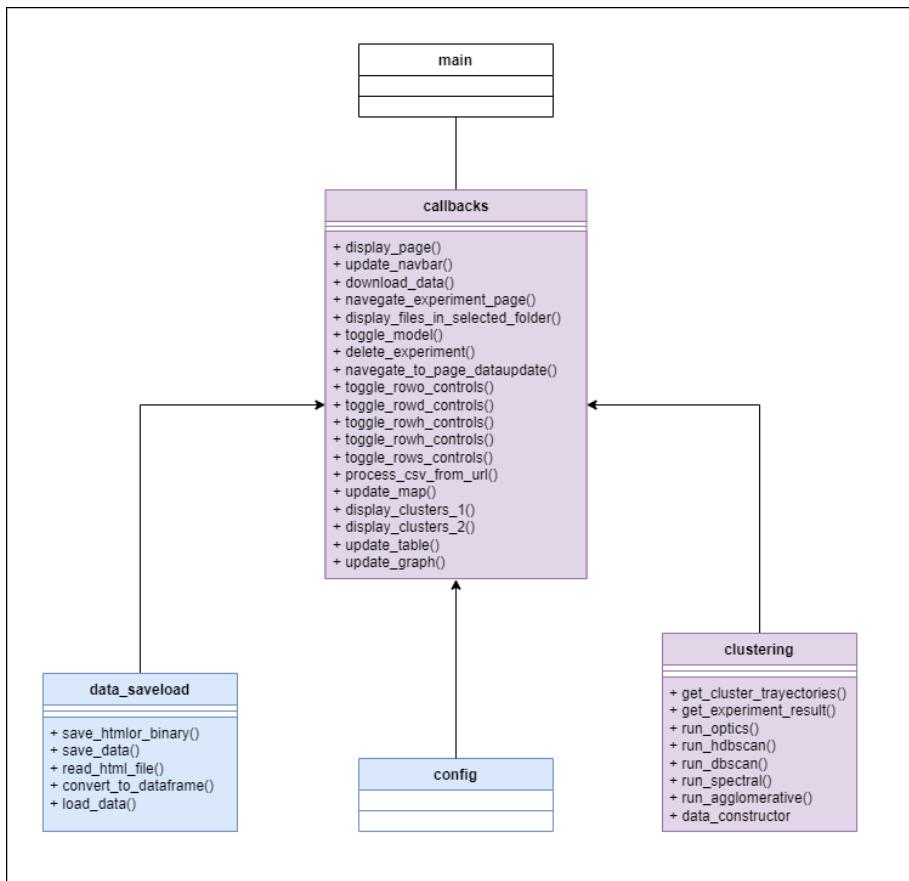


Figura C.4: Diagrama de clases, relaciones controladores.

- **Funciones complementarias (Utils):** Localizado en la carpeta `utils`, proporciona herramientas auxiliares y funciones adicionales para el controlador y la vista.

Conclusión

La implementación del modelo MVC en este proyecto asegura una separación clara de responsabilidades entre los componentes, permitiendo que cada uno se desarrolle, pruebe y modifique de manera independiente. Esta arquitectura no solo mejora la mantenibilidad del sistema, sino que también facilita su escalabilidad, permitiendo agregar nuevas funcionalidades sin comprometer la estabilidad general del proyecto.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apéndice se presenta la documentación técnica del proyecto, proporcionando una descripción detallada de la estructura del código y sus directorios. Este apartado tiene como objetivo facilitar la comprensión del proyecto para desarrolladores que deseen trabajar en él, analizarlo o realizar modificaciones futuras.

D.2. Estructura de directorios

Todo el código de este proyecto se encuentra dentro de la carpeta `code`, que se divide en dos subcarpetas principales:

- `app/` – Contiene toda la aplicación web desarrollada para el proyecto, organizada siguiendo el patrón Modelo-Vista-Controlador (MVC). Incluye el código relacionado con los controladores, modelos, vistas y utilidades, así como los recursos estáticos necesarios para la interfaz de usuario.
- `Research and experiments/` – Agrupa todas las pruebas y experimentos realizados durante el desarrollo. En esta carpeta se incluyen tanto los prototipos iniciales como los análisis realizados con diferentes componentes del proyecto. Solo se han conservado aquellas pruebas que se consideran claras y relevantes.

A continuación, se describe en detalle la estructura interna, incluyendo las funcionalidades principales de cada archivo o directorio.

- **code/**
 - **app/** – Directorio principal de la aplicación web.
 - **Research and experiments/** – Directorio que contiene experimentos y pruebas realizadas durante el desarrollo.
 - **app/**
 - **controllers/** – Contiene la lógica principal de control que conecta los modelos con las vistas.
 - **__init__.py** – Archivo de inicialización del módulo.
 - **callbacks.py** – Define los callbacks utilizados en la aplicación, gestionando la interacción entre la interfaz de usuario y la lógica del servidor.
 - **clustering.py** – Implementa funciones relacionadas con la ejecución de algoritmos de agrupamiento, incluyendo TRACLUS.
 - **models/** – Agrupa la lógica de procesamiento de datos y las funciones principales del algoritmo TRACLUS.
 - **__init__.py** – Archivo de inicialización del módulo.
 - **data_processing.py** – Contiene funciones para la limpieza, transformación y preparación de los datos.
 - **mapping.py** – Proporciona herramientas para la visualización de datos geoespaciales.
 - **TRACLUS.py** – Implementa el núcleo del algoritmo TRACLUS.
 - **saved_results/** – Almacena resultados generados y guardados por el usuario.
 - **test/** – Contiene pruebas unitarias y funcionales.
 - **__init__.py** – Archivo de inicialización del módulo de pruebas.
 - **text_callbacks.py** – Pruebas específicas para los callbacks de la aplicación.
 - **utils/** – Incluye utilidades generales y configuraciones globales.

- `__init__.py` – Archivo de inicialización del módulo.
- `config.py` – Configuraciones generales del proyecto, como rutas y parámetros predeterminados.
- `data_utils.py` – Funciones auxiliares para manipulación de datos.
- `views/` – Directorio que agrupa las vistas y la estructura visual de la aplicación.
 - `assets/` – Contiene recursos estáticos como hojas de estilo y archivos multimedia.
 - ◊ `style.css` – Archivo CSS para la personalización del diseño de la interfaz de usuario.
 - `layout/` – Define las vistas de cada sección de la aplicación.
 - ◊ `__init__.py` – Archivo de inicialización del módulo.
 - ◊ `dataupdate_page.py` – Página para la carga y actualización de datos.
 - ◊ `experiment_page.py` – Página para seleccionar algoritmos de clustering.
 - ◊ `map_page.py` – Página que muestra mapas con los datos sin tratar.
 - ◊ `select_page.py` – Página para la selección del experimento o la creación de uno nuevo.
 - ◊ `table_page.py` – Página para mostrar tablas con los resultados del análisis.
 - ◊ `TRACLUSmap_page.py` – Página que muestra mapas con los resultados del algoritmo TRACLUS.
- `__init__.py` – Archivo de inicialización del módulo principal.
- `main.py` – Archivo principal que ejecuta la aplicación web.
- `requirements.txt` – Archivo que especifica las dependencias necesarias para ejecutar el proyecto.

D.3. Manual del programador

Este apartado proporciona información sobre las herramientas clave que un programador necesita para trabajar en el proyecto de manera eficiente, desde la instalación hasta la configuración del entorno de desarrollo.

Herramientas clave

A continuación, se detallan las herramientas y su configuración:

Git

Git es esencial para gestionar el control de versiones del proyecto.

1. Descargue e instale Git desde la página oficial: <https://git-scm.com/>.
2. Configure su nombre de usuario y correo electrónico:

```
git config --global user.name "TuNombre"  
git config --global user.email "TuCorreo@example.com"
```

3. Clone el repositorio del proyecto:

```
git clone https://github.com/agd1017/TFG_TRACLUS.git
```

Visual Studio Code

Visual Studio Code es el editor recomendado.

1. Descargue VS Code desde: <https://code.visualstudio.com/>.
2. Instale las extensiones sugeridas:
 - Python.
 - Pylance.
 - GitLens.
3. Configure el intérprete de Python para que apunte a su entorno virtual.

Python

El proyecto requiere Python 3.11.2 o superior.

1. Descargue Python desde: <https://www.python.org/>.

2. Durante la instalación, habilite la opción *Add Python to PATH*.

3. Verifique la instalación con:

```
python --version
```

Librerías Python

El archivo `requirements.txt` especifica las dependencias necesarias.

1. Cree un entorno virtual:

```
python -m venv venv
```

2. Active el entorno virtual:

- En Windows:

```
venv\Scripts\activate
```

- En macOS/Linux:

```
source venv/bin/activate
```

3. Instale las dependencias:

```
pip install -r requirements.txt
```

D.4. Compilación, instalación y ejecución del proyecto

En este apartado se detallan los pasos para ejecutar el proyecto en un entorno local desde el código fuente.

Preparación del entorno

1. Asegúrese de haber instalado todas las herramientas clave mencionadas en el *Manual del programador*.
2. Clone el repositorio del proyecto en su máquina local.
3. Navegue hasta la carpeta code/app.
4. Cree y active el entorno virtual como se describe en la sección de **Librerías Python**.

Ejecución del proyecto

1. Ejecute el archivo principal de la aplicación:

```
python main.py
```

2. Abra un navegador web y acceda a:

```
http://127.0.0.1:8050/
```

Modificaciones y control de versiones

Para realizar cambios en el código:

1. Edite los archivos necesarios utilizando Visual Studio Code.
2. Guarde los cambios y verifique que la aplicación funcione correctamente.
3. Utilice Git para gestionar los cambios:

```
git add .
git commit -m "Descripción de los cambios"
git push origin rama-principal
```

Con estas instrucciones, el programador podrá compilar, instalar y ejecutar el proyecto localmente de manera efectiva.

D.5. Pruebas del sistema

En esta sección se describirá cómo realizar pruebas en el sistema para comprobar que los cambios realizados se han implementado correctamente.

Prueba manual

Las pruebas más utilizadas en este proyecto, especialmente para verificar la ejecución del algoritmo y la funcionalidad general de la aplicación, son de tipo manual.

Para realizar las pruebas de manera correcta y en un entorno local, se recomienda modificar la función `app.run_server()` ubicada al final del archivo `main.py`.

Se deben realizar los siguientes ajustes:

- Añadir el parámetro `debug=True`, lo que permitirá visualizar en la consola los errores o advertencias durante la ejecución.
- Configurar el servidor local especificando el host y el puerto. Por ejemplo:

```
app.run_server(debug=True, host='127.0.0.1', port=8050)
```

Después de realizar esta configuración, ejecute el archivo `main.py`. Se deberá acceder en un navegador web a la dirección `http://127.0.0.1:8050`. Si la aplicación se ha cargado correctamente, se mostrará la página inicial, desde la cual podrá navegar y utilizarla normalmente.

Para realizar cualquier prueba, siga el flujo de trabajo estándar de la aplicación para llegar al recurso que desea verificar. Por ejemplo:

- Si el cambio realizado afecta a la visualización de los resultados tras cargar un experimento, intente acceder a uno de los experimentos ya creados.
- Si el cambio afecta a la carga de datos, deberá crear nuevos experimentos, ya que los experimentos guardados anteriormente no reflejarán los cambios en la lógica de carga.

En caso de que no sea posible acceder a los experimentos guardados, asegúrese de que la configuración de rutas en el archivo `config.py` sea válida para su dispositivo. Si es necesario, actualice las rutas para que apunten correctamente a las ubicaciones de los datos y recursos requeridos.

Este método permite identificar errores directamente en el flujo de trabajo de la aplicación y verificar que los cambios realizados se comporten de manera esperada.

Testing

El proyecto también incluye pruebas automáticas que pueden ejecutarse para verificar la funcionalidad de diversos componentes. Estas pruebas están ubicadas en el directorio `test/`.

Ejecución de pruebas

Para ejecutar las pruebas existentes, siga estos pasos:

1. Asegúrese de que todas las dependencias necesarias estén instaladas. Puede verificar esto ejecutando:

```
pip install -r requirements.txt
```

2. Desde la terminal, navegue hasta el directorio raíz del proyecto donde se encuentra el archivo `main.py`.
3. Ejecute las pruebas utilizando el siguiente comando:

```
pytest -v {nombre test}
```

4. Si este no funciona use el siguiente:

```
PYTHONPATH=code/app pytest -v code/app/test/{nombre test}
```

5. Revise los resultados en la terminal. Si alguna prueba falla, se mostrará el motivo del fallo y la ubicación correspondiente en el código.

Creación de nuevas pruebas

Si necesita añadir nuevas pruebas al sistema, siga estos pasos:

1. Cree un nuevo archivo de prueba en el directorio `test/`. Por convención, los archivos de prueba deben comenzar con el prefijo `test_`, por ejemplo: `test_nueva_funcionalidad.py`.
2. Dentro del archivo, defina funciones que comiencen con el prefijo `test_`, como en el siguiente ejemplo:

```
def test_nueva_funcionalidad():
    resultado = funcion_a_probar(parametros)
    assert resultado == valor Esperado
```

3. Utilice las funciones de prueba para validar las salidas de las funciones o métodos que haya implementado. Asegúrese de cubrir casos de uso válidos, así como posibles errores o entradas no válidas.
4. Ejecute nuevamente `pytest` para confirmar que todas las pruebas, incluidas las nuevas, se ejecutan correctamente.

Al incluir nuevas pruebas, asegúrese de documentar el propósito de cada prueba dentro del archivo correspondiente. Esto ayudará a otros desarrolladores a comprender el objetivo y alcance de cada conjunto de pruebas.

Apéndice E

Documentación de usuario

E.1. Introducción

En este apartado se detallan los requisitos necesarios para utilizar correctamente la aplicación web, así como el curso completo que debe seguir el usuario para usarla adecuadamente.

E.2. Requisitos de usuarios

Para el uso normal de la aplicación, el usuario solo necesita lo siguiente:

- Un navegador web compatible como Firefox, Google Chrome, etc.
- Acceso a Internet.
- Acceder a la página web <https://tfg-traclus-2.onrender.com>.
- Tener un archivo de trayectorias válido, en formato Excel o CSV.

E.3. Instalación

En este caso, no debería ser necesaria la instalación de ningún software adicional, ya que la aplicación está diseñada para ejecutarse en un entorno web accesible a través del navegador. Solo es necesario contar con los requisitos mencionados anteriormente para usarla correctamente.

E.4. Manual del usuario

Esta aplicación fue creada con el objetivo de realizar pruebas en el algoritmo TRACLUS utilizando diferentes configuraciones de clústeres. El flujo de acciones en la aplicación está diseñado para ser sencillo y se divide en dos posibles caminos principales.

Pantalla de inicio

Nada más acceder a la aplicación, el usuario encontrará la página inicial.

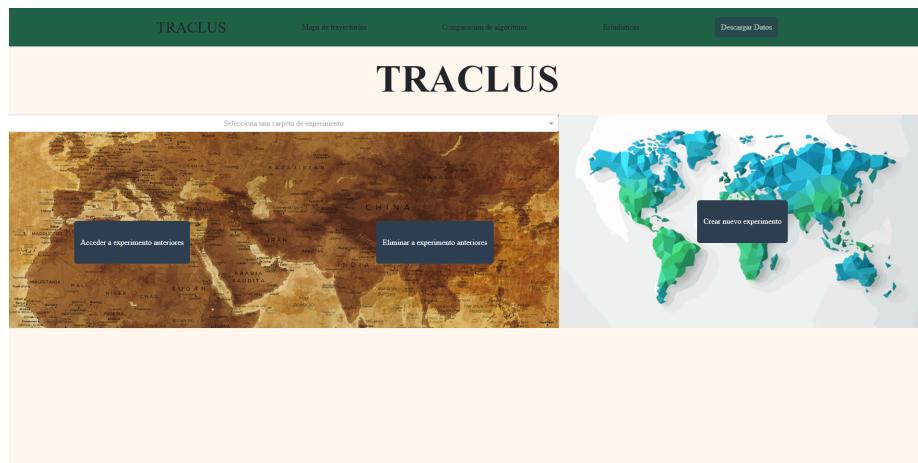


Figura E.1: Pantalla inicial de la aplicación

La barra de navegación en esta pantalla tiene todos los botones desactivados (indicados en color negro). Las opciones disponibles para el usuario dependen del estado de los experimentos existentes:

- Si no existen experimentos, la única opción disponible será crear un nuevo experimento.
- Si ya hay experimentos creados, las opciones adicionales serán cargar o eliminar un experimento existente.

Crear un nuevo experimento

Si el usuario decide crear un nuevo experimento, será redirigido a una página donde podrá seleccionar los algoritmos y configurarlos.

Figura E.2: Pantalla para la creación de un nuevo experimento

En esta página:

- Solo estará habilitado el botón **Home** en la barra de navegación, permitiendo regresar a la pantalla inicial.
- El usuario debe seleccionar los algoritmos que desea utilizar y completar los parámetros requeridos para cada uno.

Al completar estos pasos, el botón inferior permitirá avanzar a la siguiente pantalla. Es importante destacar que este botón no estará activo hasta que se hayan configurado correctamente todos los algoritmos.

Figura E.3: Pantalla para la creación de un nuevo experimento con datos

Cargar datos

En la pantalla de carga de datos, el usuario debe completar la información necesaria para iniciar el experimento.



Figura E.4: Pantalla de carga de datos

En esta sección, se solicita:

- Nombre del experimento.
- Archivo de datos en formato CSV o Excel.
- Número de datos a procesar del archivo seleccionado.

Al hacer clic en el botón de carga, el sistema procesará los datos. En caso de error, se mostrará un mensaje indicando la causa (datos incompatibles, parámetros incorrectos o fallos de conexión).



Figura E.5: Pantalla de carga de datos cargando

Visualización de resultados

Si el procesamiento es exitoso, el usuario será redirigido a la página de mapas, donde se muestran los datos sin tratar en un mapa.

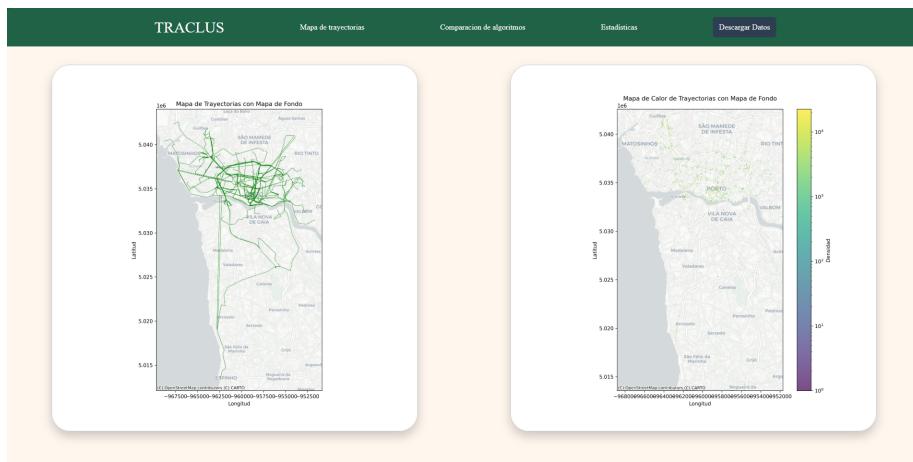


Figura E.6: Pantalla de visualización de datos sin tratar

En esta etapa, la barra de navegación desbloqueará todos los botones:

- **Comparación de algoritmos:** Muestra mapas con los resultados de los algoritmos.

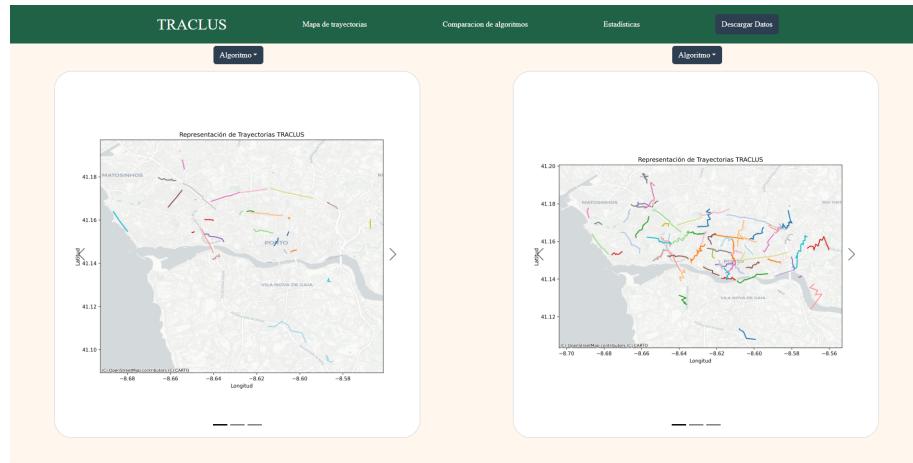


Figura E.7: Pantalla de visualización de mapas con resultados representados

- **Estadísticas:** Proporciona una tabla con datos analíticos.

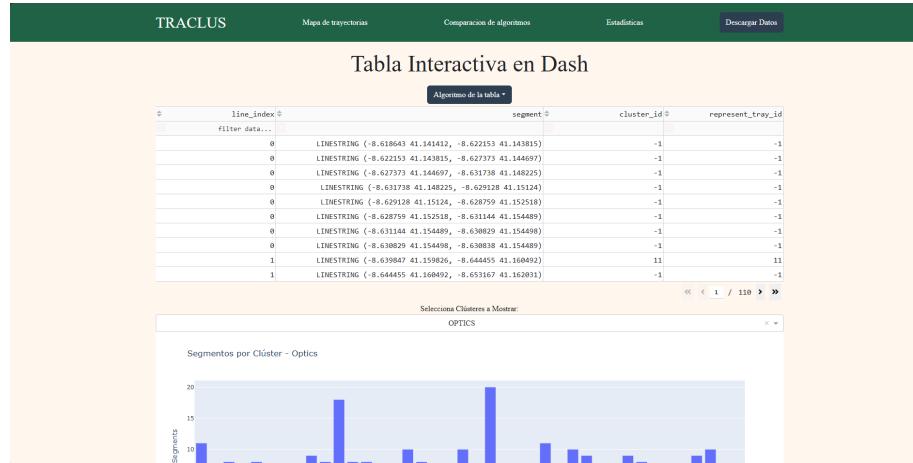


Figura E.8: Pantalla de en bruto

- **Descargar:** Permite descargar un archivo ZIP con todos los datos procesados del experimento.

Gestión de experimentos existentes

Tras la creación de un experimento, el usuario puede volver a la pantalla inicial, donde tendrá nuevas opciones habilitadas:

- **Cargar experimento existente:** Redirige directamente a la pantalla de mapas, con acceso a todas las funcionalidades.
- **Eliminar experimento:** Despliega un menú para confirmar la eliminación del experimento seleccionado.

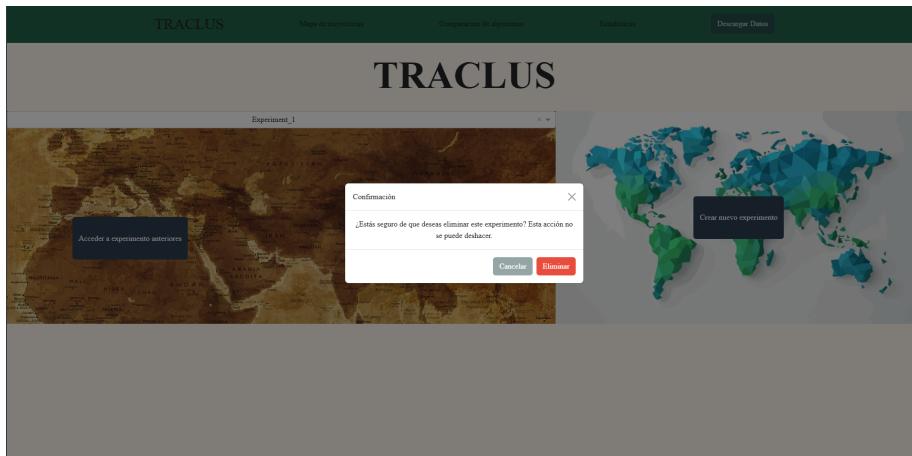


Figura E.9: Modal de confirmación para borrar experimento

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

Este anexo incluirá una reflexión personal del alumnado sobre los aspectos de la sostenibilidad que se abordan en el trabajo. Se pueden incluir tantas subsecciones como sean necesarias con la intención de explicar las competencias de sostenibilidad adquiridas durante el alumnado y aplicadas al Trabajo de Fin de Grado.

Más información en el documento de la CRUE [https://www.crue.org/
wp-content/uploads/2020/02/Directrices_Sostenibilidad_Crue2012.pdf](https://www.crue.org/wp-content/uploads/2020/02/Directrices_Sostenibilidad_Crue2012.pdf).

Este anexo tendrá una extensión comprendida entre 600 y 800 palabras.