



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Plataforma para la gestión de sistemas multiagentes y su entorno virtual

Autor
Santiago Miquel García Santamaría

Tutor
Manuel Jesús Cobo Martín



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, 14 de junio de 2024



Plataforma para la gestión de sistemas multiagentes y su entorno virtual

Autor

Santiago Miquel García Santamaría

Tutor

Manuel Jesús Cobo Martín

Plataforma para la gestión de sistemas multiagentes y su entorno virtual.

Santiago Miquel García Santamaría

Resumen

Palabras clave: *sistemas multiagentes, microservicios, API REST, laboratorio virtual*

Este Trabajo Fin de Grado se enfoca en el diseño y desarrollo de un entorno virtual para sistemas multiagentes, basado en microservicios con una API REST. La plataforma proporciona un espacio bidimensional donde los agentes pueden interactuar con su entorno, recibir percepciones y realizar movimientos.

Los agentes están equipados con sensores que les proveen información vital sobre su entorno, como la distancia al objetivo y la disponibilidad de posiciones adyacentes. Además, poseen un conjunto de habilidades que les permiten ejecutar diversas acciones dentro del entorno.

La implementación del entorno se realiza mediante microservicios, lo que garantiza una arquitectura modular y escalable. Un visualizador gráfico permite observar el estado de los agentes en el mapa bidimensional. Además, al ser una API REST, el entorno está preparado para ser consumido de diversas maneras, ofreciendo flexibilidad en su integración con otras plataformas y sistemas.

Plataforma para la gestión de sistemas multiagentes y su entorno virtual.

Santiago Miquel García Santamaría

Abstract

Keywords: *multi-agent systems, microservices, REST API, virtual laboratory*

This Final Degree Project focuses on the design and development of a virtual environment for multi-agent systems, based on microservices with a REST API. The platform provides a two-dimensional space where agents can interact with their environment, receive perceptions, and make movements.

The agents are equipped with sensors that provide vital information about their environment, such as distance to the goal and availability of adjacent positions. Additionally, they have a set of skills that allow them to execute various actions within the environment.

The implementation of the environment is done through microservices, ensuring a modular and scalable architecture. A graphical visualizer allows observing the state of the agents on the two-dimensional map. Furthermore, being a REST API, the environment is prepared to be consumed in various ways, offering flexibility in its integration with other platforms and systems.

Yo, **Santiago Miquel García Santamaría**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI 78588307K, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Santiago Miquel García Santamaría

Granada a 14 de junio de 2024.

D. Manuel Jesús Cobo Martín, profesor del Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Plataforma para la gestión de sistemas multiagentes y su entorno virtual*, ha sido realizado bajo su supervisión por **Santiago Miquel García Santamaría**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 14 de junio de 2024.

El tutor:

Manuel Jesús Cobo Martín

Agradecimientos

A mi familia, quienes han sacrificado tanto para asegurarme el mejor futuro posible. A Marina, por creer en mí incluso cuando yo tenía dudas, su confianza ha sido un motor fundamental en mi vida. A Sergito y su familia, quienes han sido mi segunda casa y me han brindado un apoyo incondicional en momentos clave. A mis amigos, David, Carlos, Javi, Dani, Josemi, Anto, Noah y Rubén por su constante respaldo y alegría compartida en cada paso del camino. A mi tutor, cuya dedicación y compromiso han sido imprescindibles para el éxito de este proyecto. Y finalmente, a mí mismo, por seguir mis sueños.

Índice general

1. Introducción	21
1.1. Motivación	21
1.2. Objetivos del proyecto	23
1.3. Estructura del documento	24
2. Planificación	27
2.1. Metodología	27
2.2. Fases	28
2.2.1. Pila de Producto	28
2.2.2. Iteraciones	29
2.2.3. Diagrama de Gantt	30
2.3. Organización	30
2.3.1. Roles	30
2.3.2. Recursos	31
2.4. Presupuesto	32
2.5. Riesgos	33
3. Análisis	35
3.1. Implicados	35
3.2. Especificación de requisitos	35
3.2.1. Requisitos funcionales	36
3.2.2. Requisitos no funcionales	45
3.2.3. Requisitos de información	47
3.3. Casos de uso	48
3.3.1. Actores	48
3.3.2. Diagramas de los casos de uso	49
3.3.3. Descripciones de los casos de uso	53
3.4. Diagramas de secuencia	74
3.5. Modelo conceptual de datos	76
4. Diseño	77
4.1. Arquitectura del sistema	77
4.1.1. Arquitectura lógica	77

4.1.2. Arquitectura física	79
4.2. Patrones de diseño	80
4.2.1. Patrones creacionales	80
4.2.2. Patrones estructurales	80
4.2.3. Patrones de comportamiento	80
4.2.4. Otros patrones	81
4.3. Modelo lógico de datos	81
4.3.1. Diagrama lógico relacional	82
4.3.2. Paso a tablas	82
4.3.3. Normalización	83
4.3.4. Diagramas lógicos no relacionales	83
4.4. Modelo físico de datos	84
4.4.1. Diagrama físico relacional	85
4.4.2. Diagrama físico no relacional	86
4.5. Modelo de clases	87
4.6. API	89
4.7. Eventos Enviados por el Servidor	95
4.8. Interfaz de usuario	96
5. Implementación	97
5.0.1. Lenguajes	97
5.0.2. Frameworks	98
5.0.3. Sistemas Gestores de Bases de Datos	98
5.0.4. Infraestructura	98
5.0.5. Entorno de desarrollo	99
5.0.6. Otras herramientas	99
6. Pruebas	101
6.1. Entorno de evaluación	101
6.2. Estrategia	101
6.3. Ejecución y resultados	102
7. Manual	117
7.1. Instalación	117
7.1.1. Requisitos previos	117
7.1.2. Inventario de componentes	118
7.1.3. Procedimiento de configuración	118
7.2. Uso	119
7.2.1. Arranque del sistema	119
7.2.2. Gestión de Agentes, Habilidades y Sensores	120
7.2.3. Gestión de Mapas	123
7.2.4. Gestión de Usuarios	127
7.2.5. Gestión de Sesiones	129
7.2.6. Interacción práctica	131

8. Conclusiones	137
Bibliografía	139

Índice de figuras

1.1. Agentes y su entorno	22
2.1. Diagrama de Gantt	30
3.1. Diagramas de casos de uso - Gestión de usuarios	49
3.2. Diagramas de casos de uso - Gestión de agentes	50
3.3. Diagramas de casos de uso - Gestión de sensores	51
3.4. Diagramas de casos de uso - Gestión de habilidades	51
3.5. Diagramas de casos de uso - Gestión de mapas	52
3.6. Diagramas de casos de uso - Sesiones	52
3.7. Diagramas de secuencia - Flujo de una Sesión	74
3.8. Diagramas de secuencia - Flujo del Visualizador	75
3.9. Modelo conceptual de datos	76
4.1. Diagrama de microservicios	78
4.2. Diagrama lógico de datos - Almacén de Agentes	82
4.3. Diagrama lógico de datos - Mapas	83
4.4. Diagrama lógico de datos - Sesiones y Usuarios	84
4.5. Diagrama físico de datos - Agentes	85
4.6. Diagrama físico de datos - Mapas	86
4.7. Diagrama físico de datos - Sesiones y Usuarios	86
4.8. Diagrama de clases - Almacén	87
4.9. Diagrama de clases - Entorno	88
4.10. SSE - Funcionamiento	95
4.11. Interfaz de usuario - Visualizador	96
7.1. Manual - Crear Agente	120
7.2. Manual - Obtener Agente	121
7.3. Manual - Modificar Agente	121
7.4. Manual - Listar Agentes	122
7.5. Manual - Añadir Habilidad o Sensor al Agente	122
7.6. Manual - Eliminar Agente	123
7.7. Manual - Crear Mapa	123
7.8. Manual - Obtener Mapa	124
7.9. Manual - Obtener información del Mapa	124

7.10. Manual - Obtener obstáculos del Mapa	125
7.11. Manual - Obtener obstáculos de un Mapa según posición	125
7.12. Manual - Listar Mapas	126
7.13. Manual - Eliminar Mapa	126
7.14. Manual - Crear Usuario	127
7.15. Manual - Obtener Usuario	127
7.16. Manual - Modificar Usuario	128
7.17. Manual - Eliminar Usuario	128
7.18. Manual - Crear sesión	129
7.19. Manual - Obtener sesión	129
7.20. Manual - Obtener datos de los sensores	130
7.21. Manual - Realizar habilidad	130
7.22. Interacción práctica - Crear sesión	131
7.23. Interacción práctica - Configurar Visualizador	132
7.24. Interacción práctica - Visualizador	132
7.25. Interacción práctica - Observando obstáculos	133
7.26. Interacción práctica - Petición de movimiento	134
7.27. Interacción práctica - Actualización del Visualizador	134
7.28. Interacción práctica - Objetivo alcanzado	135
7.29. Interacción práctica - Visualizador con objetivo alcanzado . .	135

Índice de tablas

1.1.	Objetivo 1	23
1.2.	Objetivo 2	23
1.3.	Objetivo 3	23
1.4.	Objetivo 4	23
1.5.	Objetivo 5	24
1.6.	Objetivo 6	24
2.1.	Product Backlog	28
2.2.	Sprints - Programación temporal	29
2.3.	Costes del proyecto	32
2.4.	Riesgos del proyecto	33
3.1.	RF-1 (Crear agente)	36
3.2.	RF-2 (Eliminar agente)	36
3.3.	RF-3 (Modificar agente)	36
3.4.	RF-4 (Consultar agente)	37
3.5.	RF-5 (Añadir sensor a un agente)	37
3.6.	RF-6 (Añadir habilidad a un agente)	37
3.7.	RF-7 (Eliminar sensor de un agente)	37
3.8.	RF-8 (Eliminar habilidad de un agente)	38
3.9.	RF-9 (Listar agentes)	38
3.10.	RF-10 (Crear sensor)	38
3.11.	RF-11 (Obtener sensor)	38
3.12.	RF-12 (Modificar sensor)	39
3.13.	RF-13 (Eliminar sensor)	39
3.14.	RF-14 (Listar sensores)	39
3.15.	RF-15 (Crear habilidad)	39
3.16.	RF-16 (Obtener habilidad)	40
3.17.	RF-17 (Modificar habilidad)	40
3.18.	RF-18 (Eliminar habilidad)	40
3.19.	RF-19 (Listar habilidades)	40
3.20.	RF-20 (Crear mapa)	41
3.21.	RF-21 (Obtener mapa)	41
3.22.	RF-22 (Obtener información del mapa)	41

3.23. RF-23 (Listar nombre de los mapas)	41
3.24. RF-24 (Obtener obstáculos)	42
3.25. RF-25 (Obtener obstáculos según la posición)	42
3.26. RF-26 (Crear usuario)	42
3.27. RF-27 (Obtener usuario)	42
3.28. RF-28 (Modificar usuario)	43
3.29. RF-29 (Eliminar usuario)	43
3.30. RF-30 (Crear sesión)	43
3.31. RF-31 (Eliminar sesión)	43
3.32. RF-32 (Consultar sesión)	44
3.33. RF-33 (Modificar sesión)	44
3.34. RF-34 (Realizar habilidad)	44
3.35. RF-35 (Mostrar actualizaciones de los eventos)	44
3.36. RF-36 (Eliminar mapa)	45
3.37. RF-37 (Finalizar sesión)	45
3.38. RNF-1 (Rendimiento)	46
3.39. RNF-2 (Seguridad)	46
3.40. RNF-3 (Usabilidad)	46
3.41. RNF-4 (Disponibilidad)	46
3.42. RNF-5 (Fiabilidad)	46
3.43. RNF-6 (Mantenibilidad)	46
3.44. RNF-7 (Portabilidad)	47
3.45. RNF-8 (Escalabilidad)	47
3.46. RI-1 (Sesión)	47
3.47. RI-2 (Usuario)	47
3.48. RI-3 (Mapa)	47
3.49. RI-4 (Agente)	48
3.50. RI-5 (Sensor)	48
3.51. RI-6 (Habilidad)	48
3.52. CU-1 (Crear usuario)	53
3.53. CU-2 (Obtener usuario)	53
3.54. CU-3 (Modificar usuario)	54
3.55. CU-4 (Eliminar usuario)	54
3.56. CU-5 (Crear agente)	55
3.57. CU-6 (Eliminar agente)	55
3.58. CU-7 (Modificar agente)	56
3.59. CU-8 (Obtener agente)	56
3.60. CU-9 (Añadir sensor a un agente)	57
3.61. CU-10 (Añadir habilidad a un agente)	57
3.62. CU-11 (Eliminar sensor de un agente)	58
3.63. CU-12 (Eliminar habilidad de un agente)	58
3.64. CU-13 (Listar agentes)	59
3.65. CU-14 (Crear sensor)	59
3.66. CU-15 (Obtener sensor)	60

3.67. CU-16 (Modificar sensor)	60
3.68. CU-17 (Eliminar sensor)	61
3.69. CU-18 (Listar sensores)	61
3.70. CU-19 (Crear habilidad)	62
3.71. CU-20 (Obtener habilidad)	62
3.72. CU-21 (Modificar habilidad)	63
3.73. CU-22 (Eliminar habilidad)	63
3.74. CU-23 (Listar habilidades)	64
3.75. CU-24 (Crear mapa)	64
3.76. CU-25 (Eliminar mapa)	65
3.77. CU-26 (Obtener mapa)	65
3.78. CU-27 (Obtener información del mapa)	66
3.79. CU-28 (Obtener obstáculos)	66
3.80. CU-29 (Obtener información del mapa)	67
3.81. CU-30 (Crear sesión)	67
3.82. CU-31 (Eliminar sesión)	68
3.83. CU-32 (Obtener datos de la sesión)	68
3.84. CU-33 (Modificar sesión)	69
3.85. CU-34 (Realizar habilidad)	70
3.86. CU-35 (Mostrar actualizaciones)	71
3.87. CU-36 (Obtener obstáculos según posición)	72
3.88. CU-37 (Finalizar sesión)	72
3.89. CU-38 (Obtener datos de los sensores)	73
4.1. Endpoints - Agentes	90
4.2. Endpoints - Sensores	91
4.3. Endpoints - Habilidades	91
4.4. Endpoints - Mapas	92
4.5. Endpoints - Sesiones	93
4.6. Endpoints - Eventos	94
4.7. Endpoints - Usuarios	94
6.1. PM-1 (Prueba de crear agente)	102
6.2. PM-2 (Prueba de eliminar agente)	102
6.3. PM-3 (Prueba de error al eliminar agente)	102
6.4. PM-4 (Prueba de modificar agente)	102
6.5. PM-5 (Prueba de error al modificar agente)	103
6.6. PM-6 (Prueba de consultar agente)	103
6.7. PM-7 (Prueba de error al consultar agente)	103
6.8. PM-8 (Prueba de añadir sensor a un agente)	104
6.9. PM-9 (Prueba de error al añadir sensor a un agente)	104
6.10. PM-10 (Prueba de añadir habilidad a un agente)	104
6.11. PM-11 (Prueba de error al añadir habilidad a un agente)	105
6.12. PM-12 (Prueba de eliminar sensor de un agente)	105

6.13. PM-13 (Prueba de error al eliminar sensor de un agente) . . .	105
6.14. PM-14 (Prueba de eliminar habilidad de un agente)	106
6.15. PM-15 (Prueba de error al eliminar habilidad de un agente) .	106
6.16. PM-16 (Prueba de listar agentes)	106
6.17. PM-17 (Prueba de crear sensor)	106
6.18. PM-18 (Prueba de obtener sensor)	106
6.19. PM-19 (Prueba de error al obtener sensor)	107
6.20. PM-20 (Prueba de modificar sensor)	107
6.21. PM-21 (Prueba de error al modificar sensor)	107
6.22. PM-22 (Prueba de eliminar sensor)	107
6.23. PM-23 (Prueba de error al eliminar sensor)	107
6.24. PM-24 (Prueba de listar sensores)	108
6.25. PM-25 (Prueba de crear habilidad)	108
6.26. PM-26 (Prueba de obtener habilidad)	108
6.27. PM-27 (Prueba de error al obtener habilidad)	108
6.28. PM-28 (Prueba de modificar habilidad)	108
6.29. PM-29 (Prueba de error al modificar habilidad)	109
6.30. PM-30 (Prueba de eliminar habilidad)	109
6.31. PM-31 (Prueba de error al eliminar habilidad)	109
6.32. PM-32 (Prueba de listar habilidades)	109
6.33. PM-33 (Prueba de crear mapa)	109
6.34. PM-34 (Prueba de obtener mapa)	110
6.35. PM-35 (Prueba de error al obtener mapa)	110
6.36. PM-36 (Prueba de obtener información del mapa)	110
6.37. PM-37 (Prueba de error al obtener información de un mapa)	110
6.38. PM-38 (Prueba de listar nombre de los mapas)	111
6.39. PM-39 (Prueba de listar obstáculos de un mapa)	111
6.40. PM-40 (Prueba de obtener obstáculos de mapa según posición)	111
6.41. PM-41 (Prueba de error al obtener obstáculos de mapa según posición)	111
6.42. PM-42 (Prueba de crear usuario)	112
6.43. PM-43 (Prueba de obtener usuario)	112
6.44. PM-44 (Prueba de error al obtener usuario)	112
6.45. PM-45 (Prueba de modificar usuario)	112
6.46. PM-46 (Prueba de error al modificar usuario)	113
6.47. PM-47 (Prueba de eliminar usuario)	113
6.48. PM-48 (Prueba de error al eliminar usuario)	113
6.49. PM-49 (Prueba de crear sesión)	113
6.50. PM-50 (Prueba de eliminar sesión)	114
6.51. PM-51 (Prueba de error al eliminar sesión)	114
6.52. PM-52 (Prueba de consultar sesión)	114
6.53. PM-53 (Prueba de error al consultar sesión)	114
6.54. PM-54 (Prueba de modificar sesión)	114
6.55. PM-55 (Prueba de error al modificar sesión)	115

6.56. PM-56 (Prueba de realizar habilidad)	115
6.57. PM-57 (Prueba de error al realizar habilidad)	115
6.58. PM-58 (Prueba de eliminar mapa)	115
6.59. PM-59 (Prueba de error al eliminar mapa)	116
6.60. PM-60 (Prueba de finalizar sesión)	116
6.61. PM-61 (Prueba de mostrar actualizaciones de los eventos) . .	116

Capítulo 1

Introducción

1.1. Motivación

En las últimas décadas, los sistemas multiagentes [7] han emergido como una de las áreas más prometedoras y dinámicas en el campo de la informática, y, sobretodo, de la inteligencia artificial [8]. Con el fin de lograr objetivos específicos, un sistema multiagente está compuesto de múltiples agentes que interactúan entre sí y con su entorno.

Un agente, aunque no tiene una única definición, es una entidad que opera de manera independiente, tomando la iniciativa para realizar tareas y comunicándose con otros agentes o con el entorno para alcanzar sus metas. Es decir, es una entidad autónoma, proactiva y con capacidad de comunicación, que puede ser tanto un ente físico como de software. Además, puede ser reactivo y veraz, dado que puede responder a cambios en su entorno, evitando comunicar información falsa y siempre buscando cumplir las solicitudes que se le hace.

Una de las componentes esenciales de los sistemas multiagente es la capacidad de los agentes para percibir su entorno mediante sensores. Estos sensores son fundamentales para la recopilación de datos críticos sobre el entorno, los cuales son indispensables para la toma de decisiones y la ejecución de acciones de manera eficaz conforme a los objetivos establecidos. Los sensores pueden proporcionar diversos tipos de información, como la distancia al objetivo, la localización de obstáculos circundantes o la posición actual del propio agente.

Asimismo, es igualmente crucial destacar la importancia de los actuadores, que son los encargados de ejecutar las acciones una vez que se han cumplido ciertas condiciones predefinidas.

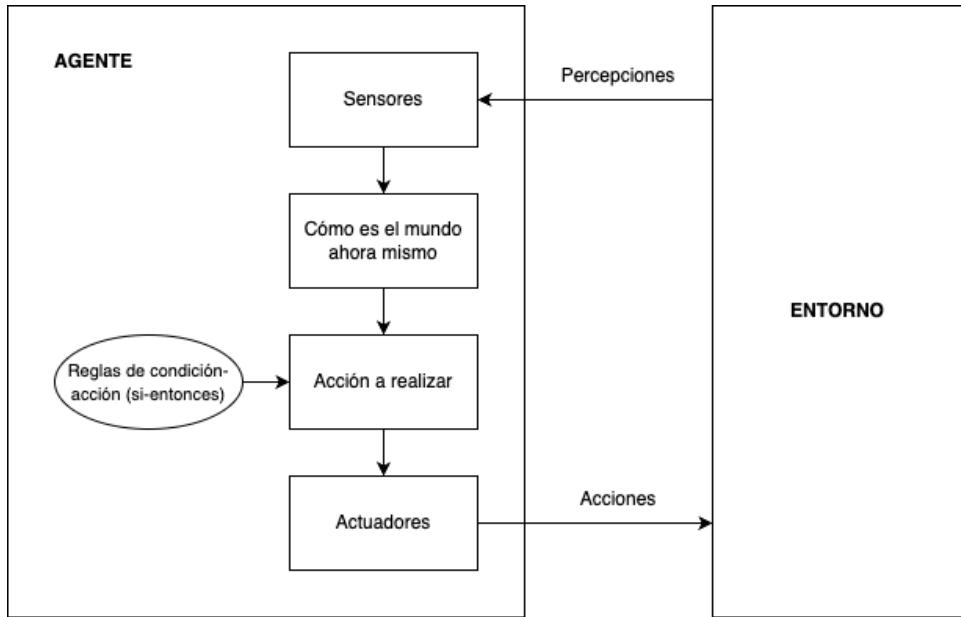


Figura 1.1: Agentes y su entorno

Cuando adoptamos un enfoque de programación orientada a agentes, para evitar que los desarrolladores creen infraestructuras para resolver problemas comunes como la comunicación entre agentes, se utilizan middlewares especializados que ofrecen estas funcionalidades, como JADE (Java Agent DEvelopment Framework) [1] .

En este contexto, surge la motivación principal de este Trabajo de Fin de Grado (TFG), que se centra en el desarrollo de un laboratorio dedicado a la ejecución y experimentación de sistemas multiagentes. Este laboratorio ofrecerá un entorno virtual controlado y medible donde los agentes, creados utilizando plataformas como JADE, podrán interactuar con el entorno, recibir percepciones a través de sus sensores y ejecutar movimientos para alcanzar sus objetivos. La representación del agente en un espacio virtual bidimensional, modelado como un mapa, permitirá una visualización clara y precisa de las interacciones entre los agentes y su entorno. Para facilitar la integración y la interoperabilidad, el entorno se expondrá mediante una API RESTful, siguiendo una arquitectura de microservicios que promueva la escalabilidad y la flexibilidad en el desarrollo y la experimentación de sistemas multiagentes. Este enfoque integrado entre el desarrollo de infraestructuras especializadas y la creación de entornos de experimentación contribuirá significativamente al avance y la comprensión de los sistemas multiagentes en diversos campos de aplicación, como en contextos educativos.

1.2. Objetivos del proyecto

El propósito principal de este Trabajo de Fin de Grado (TFG) es crear un entorno de laboratorio destinado a la ejecución y experimentación de sistemas multiagentes. Para lograr este propósito global, se han definido una serie de objetivos específicos.

OBJ-1	Comprender el funcionamiento de los sistemas multiagentes y su interacción con el entorno
Descripción	Estudiar cómo los agentes perciben y responden a su entorno y cómo interactúan entre sí para lograr sus objetivos

Tabla 1.1: Objetivo 1

OBJ-2	Analizar las necesidades de un laboratorio virtual de sistemas multiagentes
Descripción	Identificar y evaluar los requisitos y características esenciales para desarrollar un entorno de experimentación eficaz para sistemas multiagentes

Tabla 1.2: Objetivo 2

OBJ-3	Diseñar el entorno de laboratorio
Descripción	Crear un diseño detallado del entorno virtual, incluyendo la representación bidimensional del espacio, la integración de sensores y actuadores, y los componentes necesarios para la interacción de los agentes

Tabla 1.3: Objetivo 3

OBJ-4	Implementar y validar la plataforma de laboratorio
Descripción	Desarrollar los agentes y el entorno utilizando tecnologías adecuadas, asegurando la funcionalidad y la capacidad de los agentes para percibir y actuar en el entorno

Tabla 1.4: Objetivo 4

OBJ-5	Desarrollar una API RESTful mediante microservicios
Descripción	Crear una API RESTful que permita la interacción con el entorno virtual, utilizando una arquitectura de microservicios para facilitar la escalabilidad y la flexibilidad del sistema

Tabla 1.5: Objetivo 5

OBJ-6	Facilitar el uso del laboratorio en contextos educativos
Descripción	Asegurar que el entorno de laboratorio pueda ser utilizado como herramienta educativa, proporcionando documentación y ejemplos prácticos para su uso en la enseñanza de sistemas multiagentes

Tabla 1.6: Objetivo 6

1.3. Estructura del documento

Este documento está dividido en las siguientes partes:

- **Capítulo 1. Introducción:** motivación detrás del proyecto, objetivos y visión general de la memoria.
- **Capítulo 2. Planificación:** describe el plan de trabajo, con la metodología utilizada para llevar a cabo el proyecto, el presupuesto y los posibles riesgos a los que está expuesto.
- **Capítulo 3. Análisis:** examina los requisitos del sistema e identifica los actores involucrados y las necesidades del usuario.
- **Capítulo 4. Diseño:** aborda la arquitectura del sistema, los patrones de diseño utilizados y el modelado de los componentes que constituyen el proyecto.
- **Capítulo 5. Implementación:** explica cómo se llevó a cabo el desarrollo del sistema, las tecnologías utilizadas y las decisiones de implementación.
- **Capítulo 6. Pruebas:** describe los tipos de pruebas realizadas, los casos de prueba, los resultados obtenidos y cómo se solucionaron los problemas encontrados.

- **Capítulo 7. Manual:** proporciona instrucciones para el uso del sistema, incluyendo instalación, configuración y operativa básica.
- **Capítulo 8. Conclusiones:** resume los éxitos obtenidos en el proyecto, las dificultades encontradas durante su desarrollo, el cumplimiento de los objetivos y las sugerencias para futuros trabajos o mejoras.

Capítulo 2

Planificación

Este capítulo es fundamental puesto que se establecen las bases y el camino a seguir para la realización del proyecto.

2.1. Metodología

Para garantizar el éxito y la eficacia de un proyecto de desarrollo de software, es vital la elección de la metodología de trabajo adecuada. Aunque existen diversos enfoques tradicionales, para los proyectos donde los requisitos cambian con frecuencia y donde la retroalimentación con el cliente final es fundamental, son apropiadas las **metodologías ágiles** [5].

Las metodologías ágiles se caracterizan por aplicar procedimientos en ciclos de trabajo cortos. Así, se consigue ir entregando avances del trabajo final sin necesidad de esperar hasta que está terminado. De entre las metodologías ágiles que existen, para la elaboración de nuestro proyecto se ha optado por **Scrum** [6].

Scrum se sustenta en ciclos de trabajo conocidos como *sprints* o *iteraciones*, que pueden tener una duración variable, desde una hasta cuatro semanas, durante los cuales se avanza en la entrega del proyecto. En cada iteración, se produce una nueva versión del producto, mejorando y superando la anterior. Además, se realizan reuniones periódicas en busca de realizar los ajustes necesarios.

Antes que nada, debemos planear la Pila de Producto, también conocida como *Product Backlog* donde se registran las tareas requeridas en el proyecto. Luego, se seleccionan tareas de esta lista para cada iteración, estimando el esfuerzo (tiempo) necesario y finalmente se comienza a implementar cada *sprint*, realizando revisiones de los mismos, repitiendo el ciclo hasta finalizar la lista.

2.2. Fases

Nos encargaremos en esta sección de diseñar la Pila de Producto con las tareas a realizar y de mostrar cada iteración del proyecto.

2.2.1. Pila de Producto

[Apr] Aprendizaje de Java y Spring Boot (1)
[Apr] Aprendizaje de TypeScript y Angular (2)
[Apr] Aprendizaje de MongoDB y bases de datos espaciales (3)
[Apr] Aprendizaje de MySQL (4)
[Apr] Aprendizaje de Eventos Enviados por el Servidor (5)
[Apr] Aprendizaje de Docker (6)
[Fro] Diseño del frontend (7)
[Fro] Implementación del frontend (8)
[Fro] Añadir complejidad frontend (9)
[Bac] Creación de BBDD Agentes (10)
[Bac] Creación de BBDD Mapas (11)
[Bac] Creación de BBDD Sesiones y Usuarios (12)
[Bac] Adición de índices espaciales (13)
[Bac] Creación de backend de Agentes y Mapas (14)
[Bac] Adición de complejidad backend de Agentes y Mapas (15)
[Bac] Creación de backend de Sesiones y Usuarios (16)
[Bac] Adición de complejidad backend de Sesiones y Usuarios (17)
[Bac] Creación de Eventos Enviados por el Servidor (18)
[Doc] Desarrollo de introducción y objetivos (19)
[Doc] Desarrollo de análisis (20)
[Doc] Desarrollo de diseño e implementación (21)
[Doc] Desarrollo de pruebas (22)
[Doc] Desarrollo de conclusiones y bibliografía (23)
[Doc] Preparar presentación para el tribunal (24)

Tabla 2.1: Product Backlog

Las etiquetas indican el tipo de tarea: [Apr] son tareas de aprendizaje, [Fro] las implementaciones en el frontend, [Bac] las implementaciones en el backend y [Doc] las tareas de documentación.

2.2.2. Iteraciones

Para el desarrollo del proyecto, se hemos dividido las tareas en las siguientes iteraciones o *sprints*:

- **Iteración 1:** establecer las bases del backend, comenzar con la documentación y aprendizajes iniciales. Tareas de la pila: (1), (3), (4), (6), (10), (11) y (19).
- **Iteración 2:** seguimos con la implementación básica del backend. Tareas de la pila: (12), (14) y (20).
- **Iteración 3:** añadiendo complejidad al backend. Tareas de la pila: (5), (13), (15) y (21).
- **Iteración 4:** comenzamos a avanzar en el frontend mientras realizamos una de las tareas de backend más importantes. Tareas de la pila: (2), (16) y (17).
- **Iteración 5:** completar tareas del frontend y asegurarse de la integración del sistema. Tareas de la pila: (7), (8), (9) y (18).
- **Iteración 6:** pulir detalles, preparar la documentación final y la presentación. Tareas de la pila: (22), (23) y (24)

Las horas dedicadas a cada *sprint* quedan reflejadas en la Tabla 2.2.

Iteración	Tareas	Horas
1	7	69
2	3	57
3	4	65
4	3	70
5	4	66
6	3	35
Total	24	362

Tabla 2.2: Sprints - Programación temporal

2.2.3. Diagrama de Gantt

El desarrollo de nuestro proyecto se dividió en 21 semanas. Para representar gráficamente la cronología del proyecto, utilizamos la herramienta conocida como Diagrama de Gantt.

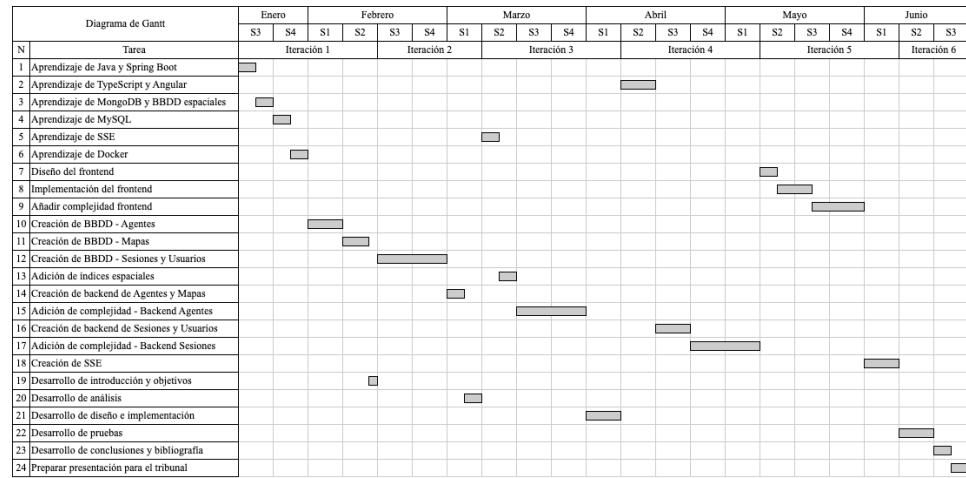


Figura 2.1: Diagrama de Gantt

2.3. Organización

En el proyecto, se identifican diversos roles esenciales. En esta sección, se detallan las responsabilidades de cada rol y sus interrelaciones. Además, se especifican los recursos de hardware y software utilizados.

2.3.1. Roles

Encontramos dos roles:

- **Director:** encargado de proporcionar directrices y orientación para el proyecto. Revisa, corrige y brinda retroalimentación del trabajo.
- **Estudiante:** responsable de llevar a cabo todas las tareas relacionadas con el proyecto, desde la elaboración del código hasta la documentación.

2.3.2. Recursos

Enumeraremos todos los recursos de hardware y software utilizados.

Si hablamos del **hardware**, se ha utilizado un ordenador portátil MacBook Air 2020 de 13'3 pulgadas, con una memoria RAM de 8GB, chip de procesamiento M1 y 256GB de almacenamiento SSD.

En el caso del **software**, tenemos lo siguiente:

- Sistema operativo: macOS Sonoma.
- IDEs: IntelliJ IDEA y Visual Studio Code.
- Lenguajes de programación: Java y JavaScript.
- Frameworks: Spring Boot y Angular.
- Lenguajes de etiquetado: HTML.
- Lenguajes de estilo: CSS.
- Elaboración de diagramas: draw.io.
- Procesador de texto: LaTeX.
- Gestión de dependencias: Maven y npm.
- Infraestructura: Docker.
- Bases de datos: MySQL y MongoDB.
- Comunicación entre roles: Telegram y Google Meet.
- Almacenamiento del proyecto: Google Drive.
- Control de versiones: GitHub.
- Otros recursos: O'Reilly y Biblioteca UGR.

2.4. Presupuesto

En esta sección se presenta una estimación detallada de los gastos incurridos durante la realización del proyecto, abarcando tanto los costes humanos como los materiales.

El coste asociado a las **licencias** es de 0 euros, ya que se ha utilizado la cuenta asociada a la Universidad de Granada para la utilización de las mismas. Sin esta cuenta, se debería considerar, por ejemplo, el IDE IntelliJ IDEA, cuyo precio oscila entre 169 y casi 300 euros.

Considerando que los ordenadores suelen tener un período de amortización de unos 3 años y sabiendo que el **portátil del desarrollador** vale 900 euros, eso son 300 euros al año que en 21 semanas se convierten en 131,25 euros.

Si hablamos del coste de la **mano de obra**, el salario promedio en España para un ingeniero informático junior es de aproximadamente 25 euros por hora contando con los impuestos.

Los **costes indirectos** como materiales de oficina, luz e internet suelen rondar el 10 % de la mano de obra.

Con todo esto en mente, obtenemos lo siguiente:

Coste	Horas	Precio	Total
Licencias	362	0 €	0 €
Portátil	-	-	131,25 €
Desarrollador	362	25 €/h	9050 €
Otros	-	10 % Desarrollador	905 €
			10086,25 €

Tabla 2.3: Costes del proyecto

Por tanto, el desarrollo de este proyecto ha costado **10086,25€**.

2.5. Riesgos

La evaluación de riesgos es un componente crítico en la planificación del proyecto, ya que permite anticipar posibles problemas y desarrollar estrategias para mitigarlos.

Para cada riesgo que hemos identificado, detallaremos su impacto en el proyecto, la probabilidad de que el mismo ocurra y sus soluciones.

Riesgo	Impacto	Probabilidad	Solución
Dificultad de aprendizaje de las tecnologías utilizadas	Se produciría un retraso en la planificación del proyecto	Alta	Redefinir la planificación
Enfermedad o accidente del desarrollador	Se produciría una demora en la planificación del proyecto	Baja	Replanificar el proyecto
Caída de los servicios utilizados	Se produciría una demora en la planificación del proyecto	Baja	Redefinir la planificación
Dificultades con ciertas funcionalidades del sistema	Se produciría un retraso en la planificación del proyecto	Media	Redefinir la planificación y/o el proyecto
Fallos en el hardware	Podría acarrear atrasos graves en la planificación	Baja	Replanificar el proyecto, además de arreglar el ordenador o buscarle sustitución

Tabla 2.4: Riesgos del proyecto

Capítulo 3

Análisis

3.1. Implicados

Los implicados en el desarrollo de software son aquellos que toman un rol determinado en el sistema. Centrándonos en los usuarios directos de la aplicación, tenemos lo siguiente:

- **Administrador:** encargado de supervisar y controlar todos los aspectos esenciales del sistema para asegurar su óptimo desempeño.
- **Usuario:** con un conocimiento moderado de la aplicación, podrá crear y eliminar sesiones utilizando distinto tipo de agentes y mapas, que le brindarán la información del entorno gracias a la lectura de sus sensores, pudiendo utilizar dichos datos para mover su agente por el mapa y alcanzar objetivos.
- **Visualizador:** Interfaz gráfica que muestra de manera amigable los cambios en una sesión.

3.2. Especificación de requisitos

En esta sección, detallaremos las diversas condiciones y necesidades que nuestro producto debe satisfacer. Para ello, seguiremos un modelo que nos permita definir y describir los distintos requisitos del sistema.

Para evitar malentendidos durante el proyecto, cada requisito se identificará inequívocamente mediante un código formado por una etiqueta indicando el tipo de requisito y el número de orden. Este código se utilizará como referencia en todo momento durante el desarrollo del proyecto.

3.2.1. Requisitos funcionales

El sistema debe cumplir una serie de requisitos para satisfacer las necesidades del usuario. Describiremos los servicios y funciones que proveerá el programa. Por tanto, en esta sección detallamos qué hace dicho sistema.

RF-1	Crear agente
Descripción	El sistema deberá permitir la creación de un agente
Prioridad	Alta
Datos de entrada	El nombre y la descripción del agente
Datos de salida	El identificador del agente
Observaciones	Ninguna

Tabla 3.1: RF-1 (Crear agente)

RF-2	Eliminar agente
Descripción	El sistema deberá permitir la eliminación de un agente
Prioridad	Alta
Datos de entrada	El identificador del agente
Datos de salida	Ninguno
Observaciones	El agente deberá existir

Tabla 3.2: RF-2 (Eliminar agente)

RF-3	Modificar agente
Descripción	El sistema deberá facilitar la modificación de un agente
Prioridad	Media
Datos de entrada	El identificador del agente junto con los datos a modificar
Datos de salida	Ninguno
Observaciones	El agente deberá existir

Tabla 3.3: RF-3 (Modificar agente)

RF-4	Consultar agente
Descripción	El sistema deberá facilitar los datos de un agente
Prioridad	Alta
Datos de entrada	El identificador del agente
Datos de salida	Identificador, nombre, descripción y habilidades
Observaciones	El agente deberá existir

Tabla 3.4: RF-4 (Consultar agente)

RF-5	Añadir sensor a un agente
Descripción	El sistema deberá brindar la adición de un sensor a un agente
Prioridad	Alta
Datos de entrada	El identificador del agente y del sensor
Datos de salida	Ninguno
Observaciones	El agente y el sensor deberán existir

Tabla 3.5: RF-5 (Añadir sensor a un agente)

RF-6	Añadir habilidad a un agente
Descripción	El sistema deberá facilitar la adición de una habilidad a un agente
Prioridad	Alta
Datos de entrada	El identificador del agente y de la habilidad
Datos de salida	Ninguno
Observaciones	El agente y la habilidad deberán existir

Tabla 3.6: RF-6 (Añadir habilidad a un agente)

RF-7	Eliminar sensor de un agente
Descripción	El sistema deberá brindar la eliminación de un sensor de un agente
Prioridad	Alta
Datos de entrada	El identificador del agente y del sensor
Datos de salida	Ninguno
Observaciones	El agente y el sensor deberán existir

Tabla 3.7: RF-7 (Eliminar sensor de un agente)

RF-8	Eliminar habilidad de un agente
Descripción	El sistema deberá facilitar la eliminación de una habilidad de un agente
Prioridad	Alta
Datos de entrada	El identificador del agente y de la habilidad
Datos de salida	Ninguno
Observaciones	El agente y la habilidad deberán existir

Tabla 3.8: RF-8 (Eliminar habilidad de un agente)

RF-9	Listar agentes
Descripción	El sistema deberá mostrar un listado de los agentes disponibles
Prioridad	Alta
Datos de entrada	Ninguno
Datos de salida	Los agentes con sus datos: nombre, descripción, sensores y habilidades
Observaciones	Ninguna

Tabla 3.9: RF-9 (Listar agentes)

RF-10	Crear sensor
Descripción	El sistema deberá facilitar la creación de un sensor
Prioridad	Alta
Datos de entrada	El nombre y descripción del sensor
Datos de salida	El identificador del sensor
Observaciones	Ninguna

Tabla 3.10: RF-10 (Crear sensor)

RF-11	Obtener sensor
Descripción	El sistema deberá facilitar la obtención de un sensor
Prioridad	Alta
Datos de entrada	El identificador del sensor
Datos de salida	El nombre y descripción del sensor
Observaciones	El sensor deberá existir

Tabla 3.11: RF-11 (Obtener sensor)

RF-12	Modificar sensor
Descripción	El sistema deberá permitir la modificación de un sensor
Prioridad	Media
Datos de entrada	El identificador del sensor, el nombre y la descripción
Datos de salida	Ninguno
Observaciones	El sensor deberá existir

Tabla 3.12: RF-12 (Modificar sensor)

RF-13	Eliminar sensor
Descripción	El sistema deberá permitir la eliminación de un sensor
Prioridad	Alta
Datos de entrada	El identificador del sensor
Datos de salida	Ninguno
Observaciones	El sensor deberá existir y no estar conectado con un agente

Tabla 3.13: RF-13 (Eliminar sensor)

RF-14	Listar sensores
Descripción	El sistema deberá mostrar un listado de los sensores disponibles
Prioridad	Alta
Datos de entrada	Ninguno
Datos de salida	Cada sensor con su id, nombre y descripción
Observaciones	Ninguna

Tabla 3.14: RF-14 (Listar sensores)

RF-15	Crear habilidad
Descripción	El sistema deberá facilitar la creación de una habilidad
Prioridad	Alta
Datos de entrada	El nombre y descripción de la habilidad
Datos de salida	El identificador de la habilidad
Observaciones	Ninguna

Tabla 3.15: RF-15 (Crear habilidad)

RF-16	Obtener habilidad
Descripción	El sistema deberá facilitar la obtención de una habilidad
Prioridad	Alta
Datos de entrada	El identificador de la habilidad
Datos de salida	El nombre y descripción e identificación de la habilidad
Observaciones	La habilidad deberá existir

Tabla 3.16: RF-16 (Obtener habilidad)

RF-17	Modificar habilidad
Descripción	El sistema deberá permitir la modificación de una habilidad
Prioridad	Media
Datos de entrada	El identificador de la habilidad, el nombre y la descripción
Datos de salida	Ninguno
Observaciones	La habilidad deberá existir

Tabla 3.17: RF-17 (Modificar habilidad)

RF-18	Eliminar habilidad
Descripción	El sistema deberá permitir la eliminación de una habilidad
Prioridad	Alta
Datos de entrada	El identificador de la habilidad
Datos de salida	Ninguno
Observaciones	El sensor deberá existir y no estar conectado con un agente

Tabla 3.18: RF-18 (Eliminar habilidad)

RF-19	Listar habilidades
Descripción	El sistema deberá mostrar un listado de las habilidades disponibles
Prioridad	Alta
Datos de entrada	Ninguno
Datos de salida	Listado con las habilidades del sistema
Observaciones	Ninguna

Tabla 3.19: RF-19 (Listar habilidades)

RF-20	Crear mapa
Descripción	El sistema deberá facilitar la creación de un mapa
Prioridad	Alta
Datos de entrada	El nombre, tamaño, obstáculos y sensores ligados a la partida como: energía, posición u objetivos
Datos de salida	El identificador del mapa
Observaciones	Ninguna

Tabla 3.20: RF-20 (Crear mapa)

RF-21	Obtener mapa
Descripción	El sistema deberá permitir la obtención del mapa
Prioridad	Alta
Datos de entrada	El identificador del mapa
Datos de salida	El nombre, tamaño, sensores ligados al mapa y obstáculos
Observaciones	El mapa deberá existir

Tabla 3.21: RF-21 (Obtener mapa)

RF-22	Obtener información del mapa
Descripción	El sistema deberá permitir la obtención de las características del mapa
Prioridad	Media
Datos de entrada	El identificador del mapa
Datos de salida	El nombre, tamaño y sensores ligados al mapa
Observaciones	El mapa deberá existir

Tabla 3.22: RF-22 (Obtener información del mapa)

RF-23	Listar nombre de los mapas
Descripción	El sistema deberá facilitar una lista con todos los mapas
Prioridad	Baja
Datos de entrada	Ninguno
Datos de salida	Una lista con los nombres de los mapas y sus identificadores
Observaciones	Ninguna

Tabla 3.23: RF-23 (Listar nombre de los mapas)

RF-24	Obtener obstáculos
Descripción	El sistema facilitará la obtención de los obstáculos de un mapa
Prioridad	Media
Datos de entrada	El identificador del mapa
Datos de salida	Una lista con los obstáculos
Observaciones	El mapa deberá existir

Tabla 3.24: RF-24 (Obtener obstáculos)

RF-25	Obtener obstáculos según la posición
Descripción	El sistema facilitará la obtención de los obstáculos de un mapa según la posición y un campo de visión dado
Prioridad	Alta
Datos de entrada	El identificador del mapa, la posición del mapa y el campo de visión
Datos de salida	Una lista con los obstáculos
Observaciones	El mapa deberá existir, la posición tendrá que estar dentro del mapa y la distancia no podrá ser negativa

Tabla 3.25: RF-25 (Obtener obstáculos según la posición)

RF-26	Crear usuario
Descripción	El sistema deberá permitir la creación de un usuario
Prioridad	Alta
Datos de entrada	Los datos de usuario como su DNI, nombre, apellidos y subgrupo de la asignatura
Datos de salida	Ninguno
Observaciones	Ninguna

Tabla 3.26: RF-26 (Crear usuario)

RF-27	Obtener usuario
Descripción	El sistema deberá permitir la obtención de los datos de un usuario
Prioridad	Alta
Datos de entrada	El identificador de usuario
Datos de salida	Los datos asociados a un usuario
Observaciones	El usuario deberá existir

Tabla 3.27: RF-27 (Obtener usuario)

RF-28	Modificar usuario
Descripción	El sistema deberá permitir la modificación de un usuario
Prioridad	Media
Datos de entrada	El identificador de usuario y los datos a modificar
Datos de salida	Ninguno
Observaciones	El usuario deberá existir

Tabla 3.28: RF-28 (Modificar usuario)

RF-29	Eliminar usuario
Descripción	El sistema deberá permitir la eliminación de un usuario
Prioridad	Media
Datos de entrada	El identificador de usuario
Datos de salida	Ninguno
Observaciones	El usuario deberá existir

Tabla 3.29: RF-29 (Eliminar usuario)

RF-30	Crear sesión
Descripción	El sistema deberá facilitar la instauración de una sesión
Prioridad	Alta
Datos de entrada	El identificador del usuario, del mapa y del agente a utilizar
Datos de salida	El identificador de sesión
Observaciones	Los datos de entrada deben existir

Tabla 3.30: RF-30 (Crear sesión)

RF-31	Eliminar sesión
Descripción	El sistema deberá permitir la eliminación de la sesión de usuario
Prioridad	Media
Datos de entrada	El identificador de sesión
Datos de salida	Ninguno
Observaciones	La sesión deberá existir y no estar en uso

Tabla 3.31: RF-31 (Eliminar sesión)

RF-32	Consultar sesión
Descripción	El sistema deberá facilitar la obtención de la sesión
Prioridad	Alta
Datos de entrada	El identificador de sesión
Datos de salida	El identificador de sesión, del usuario, del agente y del mapa, además de otros datos como los sensores del entorno actual
Observaciones	La sesión deberá existir

Tabla 3.32: RF-32 (Consultar sesión)

RF-33	Modificar sesión
Descripción	El sistema deberá facilitar la actualización de la sesión
Prioridad	Alta
Datos de entrada	El identificador de sesión
Datos de salida	Los datos a actualizar
Observaciones	La sesión deberá existir

Tabla 3.33: RF-33 (Modificar sesión)

RF-34	Realizar habilidad
Descripción	El sistema deberá permitir las acciones del agente generadas por el usuario
Prioridad	Alta
Datos de entrada	El identificador de sesión y la habilidad a realizar
Datos de salida	El estado tras realizar la acción y su éxito
Observaciones	La sesión deberá existir y la habilidad deberá poder ser realizada por ese agente

Tabla 3.34: RF-34 (Realizar habilidad)

RF-35	Mostrar actualizaciones de los eventos
Descripción	El sistema deberá permitir la visualización de los eventos de una sesión
Prioridad	Alta
Datos de entrada	Ninguno
Datos de salida	Los datos de actualización de los eventos
Observaciones	La sesión deberá existir

Tabla 3.35: RF-35 (Mostrar actualizaciones de los eventos)

RF-36	Eliminar mapa
Descripción	El sistema deberá permitir la eliminación del mapa
Prioridad	Baja
Datos de entrada	Identificador del mapa
Datos de salida	Ninguno
Observaciones	El mapa deberá existir

Tabla 3.36: RF-36 (Eliminar mapa)

RF-37	Finalizar sesión
Descripción	El sistema deberá permitir la finalización de una sesión, debido a un error o por misión completada
Prioridad	Alta
Datos de entrada	Identificador de sesión
Datos de salida	Ninguno
Observaciones	La sesión debe existir

Tabla 3.37: RF-37 (Finalizar sesión)

3.2.2. Requisitos no funcionales

Los requisitos no funcionales o requisitos de calidad denotan las características y propiedades que debe satisfacer el sistema, relacionadas con el desempeño, calidad y éxito. Es decir, establecen cómo debe ser el sistema. Nos centraremos en:

- Rendimiento: describe las expectativas de velocidad y capacidad del programa.
- Seguridad: define las medidas de seguridad para proteger el sistema.
- Usabilidad: define la experiencia de usuario al interactuar con el sistema.
- Disponibilidad: establece el tiempo de operatividad del sistema.
- Fiabilidad: garantiza el funcionamiento del sistema.
- Mantenibilidad: describe cómo mantener el sistema con el tiempo.
- Portabilidad: define la capacidad del programa de adaptarse a distintos sistemas.
- Escalabilidad: describe cómo maneja el sistema el aumento de carga sin comprometer el rendimiento.

RNF-1	Rendimiento
Descripción	Las acciones del usuario deberán tener un tiempo máximo de respuesta de 2 o 3 segundos

Tabla 3.38: RNF-1 (Rendimiento)

RNF-2	Seguridad
Descripción	Cada usuario deberá tener un identificador único certificando que este mismo es el que usa la plataforma

Tabla 3.39: RNF-2 (Seguridad)

RNF-3	Usabilidad
Descripción	EL usuario deberá ser informado sobre las acciones que realiza, utilizando protocolos conocidos como HTTP, aunque la interfaz de sesión deberá ser más amigable y minimalista para obtener una respuesta del estado del agente visualmente más atractiva y evitar cargas de memoria en el cliente

Tabla 3.40: RNF-3 (Usabilidad)

RNF-4	Disponibilidad
Descripción	El sistema estará disponible el tiempo que el administrador vea necesario para evaluar a los usuario. Esto podrías ser ciertas horas del día o incluso un día entero

Tabla 3.41: RNF-4 (Disponibilidad)

RNF-5	Fiabilidad
Descripción	Se deberán realizar copias de seguridad para prevenir la pérdida de información

Tabla 3.42: RNF-5 (Fiabilidad)

RNF-6	Mantenibilidad
Descripción	El código deberá estar bien documentado y seguir buenas prácticas de programación para poder mejorar y añadir actualizaciones fácilmente al sistema

Tabla 3.43: RNF-6 (Mantenibilidad)

RNF-7	Portabilidad
Descripción	El sistema podrá funcionar independientemente del sistema operativo utilizado gracias al uso de contenedores y máquinas virtuales de compilación

Tabla 3.44: RNF-7 (Portabilidad)

RNF-8	Escalabilidad
Descripción	El sistema deberá estar preparado para tener conectados desde 1 a casi 100 usuarios simultáneamente

Tabla 3.45: RNF-8 (Escalabilidad)

3.2.3. Requisitos de información

Los requisitos de información definen qué información almacenaremos en el sistema para ofrecer las funcionalidades del mismo.

RI-1	Sesión
Contenido	Información de partida: id, estado, id del usuario, id del agente, id del mapa, posición actual, objetivo actual, objetivos, energía, altitud, sensores, habilidades, log, nombre del agente y nombre del mapa

Tabla 3.46: RI-1 (Sesión)

RI-2	Usuario
Contenido	Datos personales del usuario: id, nombre, apellidos, dni y grupo

Tabla 3.47: RI-2 (Usuario)

RI-3	Mapa
Contenido	Datos del plano: id, nombre, tamaño, energía, altitud, spawn, objetivos y obstáculos

Tabla 3.48: RI-3 (Mapa)

RI-4	Agente
Contenido	Información del agente: id, nombre y descripción

Tabla 3.49: RI-4 (Agente)

RI-5	Sensor
Contenido	Información del sensor: id, nombre y descripción

Tabla 3.50: RI-5 (Sensor)

RI-6	Habilidad
Contenido	Información de la habilidad: id, nombre y descripción

Tabla 3.51: RI-6 (Habilidad)

3.3. Casos de uso

A través del modelo de casos de uso, se puede definir claramente el sistema, establecer su contexto de uso y ofrecer una perspectiva sobre cómo lo utilizan los usuarios. Es decir, se utilizan para capturar y representar los requisitos funcionales de una manera más contextualizada y orientada al usuario.

Este modelo se compone de usuarios o sistemas (actores), acciones que el sistema realizará (casos de uso) y las relaciones entre ellos. Para describir y representar estos elementos, se emplean diagramas UML.

3.3.1. Actores

En la aplicación se cuenta con tres actores:

- **Administrador:** actúa como "superusuario", tiene acceso a todo el sistema.
- **Usuario:** es capaz de crear y terminar sesiones, además de poder pedir a los gentes que usa los valores de sus sensores para poder aplicar dicho conocimiento a sus algoritmos de búsqueda.
- **Visualizador:** permite ver de una forma más amigable el movimiento del agente durante una sesión.

3.3.2. Diagramas de los casos de uso

Representaremos de manera gráfica la interacción entre los usuarios y el sistema. Destacamos los ya nombrados actores, identificando el rol del usuario en el sistema, y los casos de uso, las acciones que lleva a cabo el sistema en respuesta a las interacciones con los actores.

Para ser más claros, encapsularemos los casos de uso relacionados es distintos subsistemas.

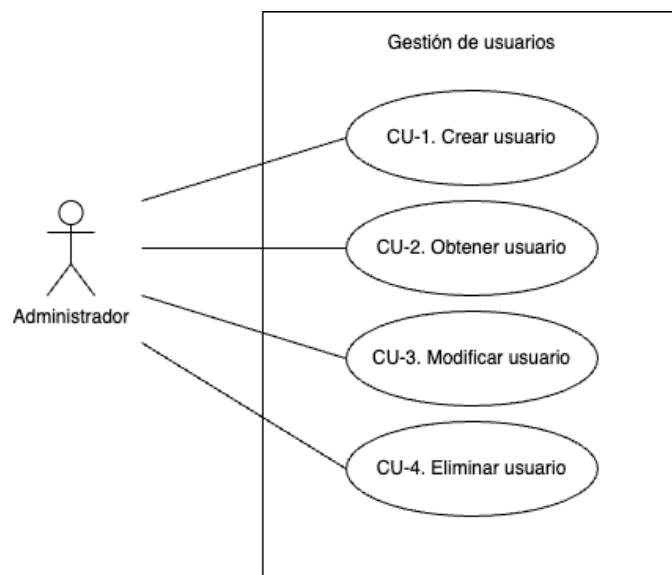


Figura 3.1: Diagramas de casos de uso - Gestión de usuarios

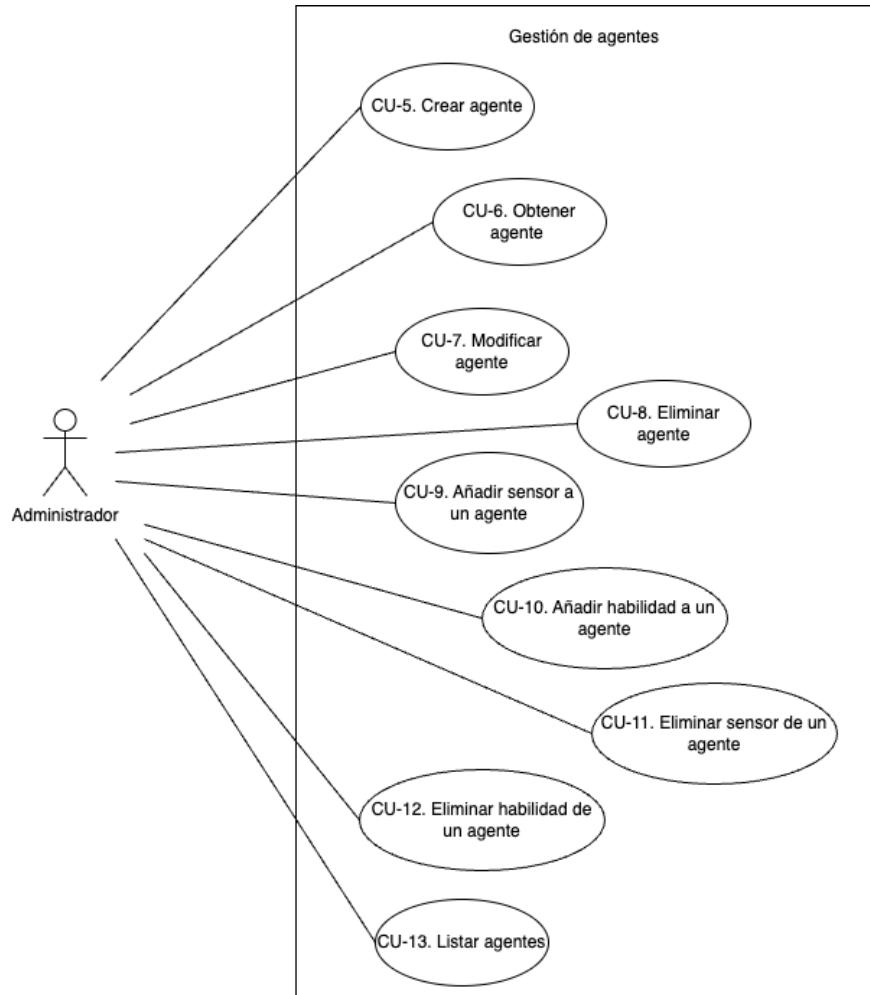


Figura 3.2: Diagramas de casos de uso - Gestión de agentes

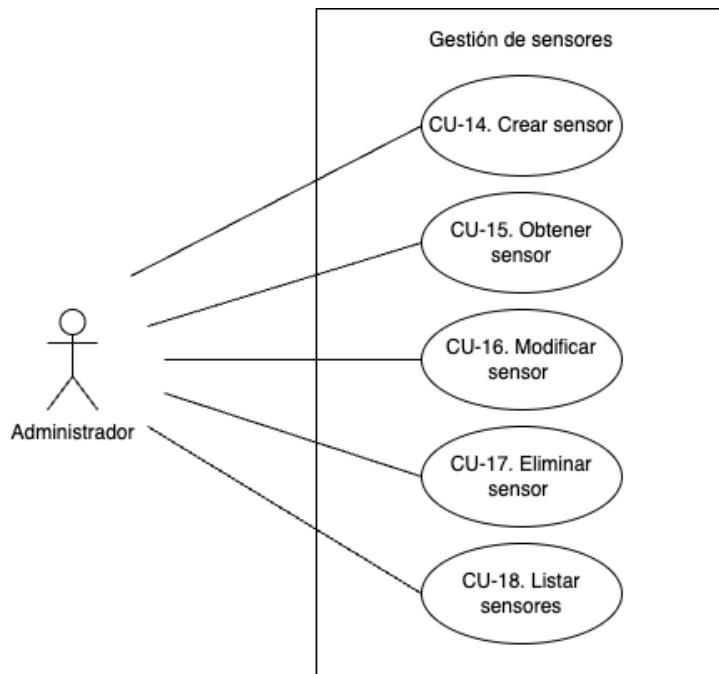


Figura 3.3: Diagramas de casos de uso - Gestión de sensores

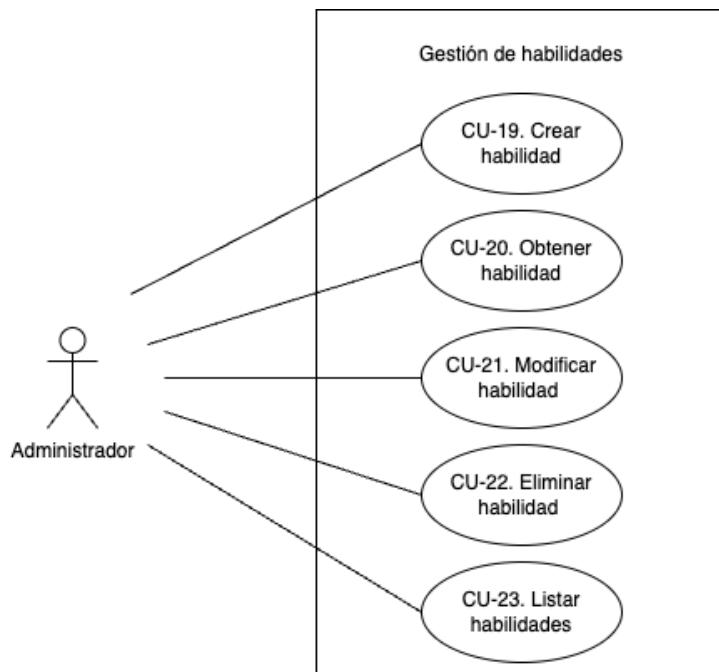


Figura 3.4: Diagramas de casos de uso - Gestión de habilidades

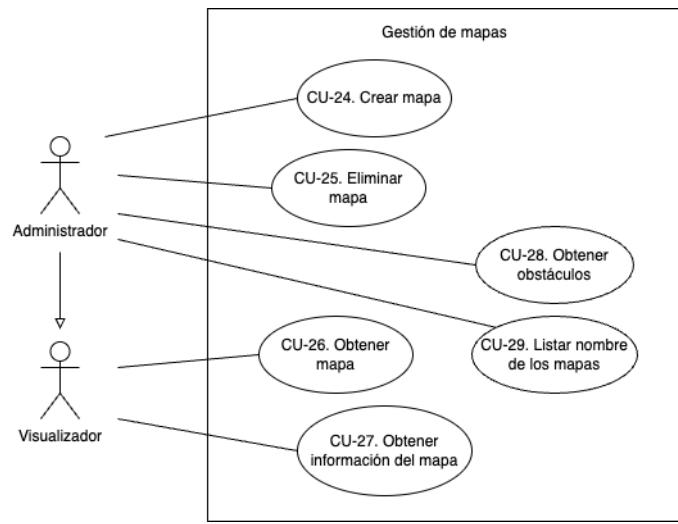


Figura 3.5: Diagramas de casos de uso - Gestión de mapas

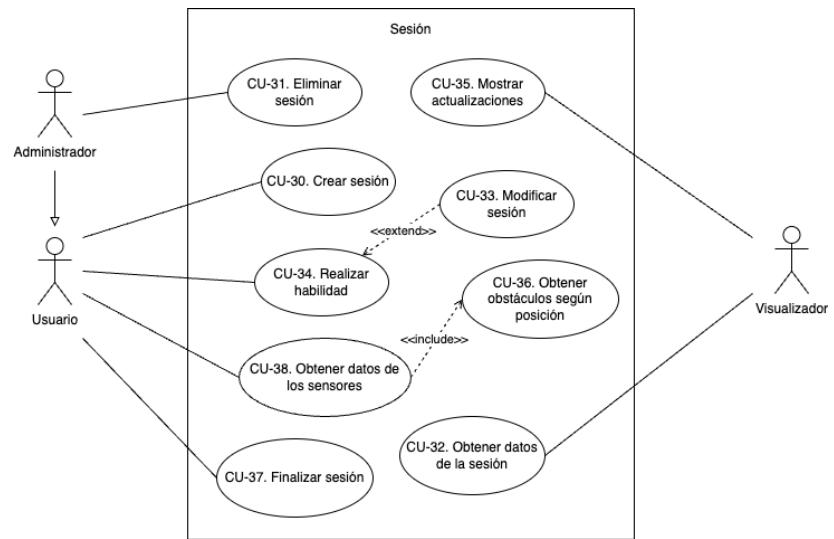


Figura 3.6: Diagramas de casos de uso - Sesiones

3.3.3. Descripciones de los casos de uso

En este apartado desglosaremos cada caso de uso con el fin de obtener información más detallada como la descripción, las condiciones que se deben cumplir antes y después de cada uno o la secuencia de pasos que siguen.

CU-1	Crear usuario
Descripción	Dar de alta un usuario en el sistema
Actores	Administrador
Referencias	RF-26
Precondiciones	Ninguna
Postcondiciones	El usuario queda registrado en el sistema
Flujo normal	<ol style="list-style-type: none"> 1. Administrador: proporciona los datos del usuario al sistema 2. Sistema: comprueba los datos 3. Sistema: crea el usuario
Flujo alternativo	3b. Sistema: no crea el usuario y presenta notificación de fallo

Tabla 3.52: CU-1 (Crear usuario)

CU-2	Obtener usuario
Descripción	Obtener los datos de un usuario en el sistema
Actores	Administrador
Referencias	RF-27
Precondiciones	El usuario debe existir
Postcondiciones	Se muestran los datos del usuario
Flujo normal	<ol style="list-style-type: none"> 1. Administrador: proporciona el identificador del usuario al sistema 2. Sistema: comprueba los datos 3. Sistema: envía los datos del usuario
Flujo alternativo	3b. Sistema: no conoce el usuario y presenta notificación de fallo

Tabla 3.53: CU-2 (Obtener usuario)

CU-3	Modificar usuario
Descripción	Modificar los datos de un usuario en el sistema
Actores	Administrador
Referencias	RF-28
Precondiciones	El usuario debe existir
Postcondiciones	Se modifican los datos del usuario
Flujo normal	1. Administrador: asigna los nuevos datos del usuario al identificador de usuario 2. Sistema: comprueba los datos 3. Sistema: actualiza los datos
Flujo alternativo	3b. Sistema: notifica fallo por identificador de usuario incorrecto 3c. Sistema: notifica fallo por datos introducidos incorrectos

Tabla 3.54: CU-3 (Modificar usuario)

CU-4	Eliminar usuario
Descripción	Dar de baja un usuario en el sistema
Actores	Administrador
Referencias	RF-29
Precondiciones	Ninguna
Postcondiciones	El usuario queda eliminado del sistema
Flujo normal	1. Administrador: proporciona el identificador del usuario al sistema 2. Sistema: comprueba el identificador 3. Sistema: borra el usuario
Flujo alternativo	3b. Sistema: notifica fallo por usuario incorrecto

Tabla 3.55: CU-4 (Eliminar usuario)

CU-5	Crear agente
Descripción	Crear un agente en el sistema
Actores	Administrador
Referencias	RF-1
Precondiciones	Ninguna
Postcondiciones	El agente queda registrado en el sistema
Flujo normal	1. Administrador: envía los datos del agente 2. Sistema: comprueba los datos 3. Sistema: almacena el agente
Flujo alternativo	3b. Sistema: notifica error por datos incorrectos

Tabla 3.56: CU-5 (Crear agente)

CU-6	Eliminar agente
Descripción	Eliminar un agente en el sistema
Actores	Administrador
Referencias	RF-2
Precondiciones	El agente debe existir
Postcondiciones	El agente queda eliminado del sistema
Flujo normal	1. Administrador: proporciona el identificador del agente 2. Sistema: comprueba los datos 3. Sistema: elimina el agente
Flujo alternativo	3b. Sistema: notifica error por identificador incorrecto

Tabla 3.57: CU-6 (Eliminar agente)

CU-7	Modificar agente
Descripción	Modifica un agente en el sistema
Actores	Administrador
Referencias	RF-3
Precondiciones	El agente debe existir
Postcondiciones	Se modifican los datos del agente en el sistema
Flujo normal	1. Administrador: proporciona el identificador del agente y los datos que quiere modificar al sistema 2. Sistema: comprueba los datos y el identificador 3. Sistema: modifica el agente
Flujo alternativo	3b. Sistema: notifica error por identificador incorrecto 3c. Sistema: notifica error por datos incorrectos

Tabla 3.58: CU-7 (Modificar agente)

CU-8	Obtener agente
Descripción	Obtiene los datos del agente almacenados en el sistema
Actores	Administrador
Referencias	RF-4
Precondiciones	El agente debe existir
Postcondiciones	Se muestran los datos del agente
Flujo normal	1. Administrador: proporciona el identificador del agente al sistema 2. Sistema: comprueba el identificador 3. Sistema: muestra los datos asociados al agente
Flujo alternativo	3b. Sistema: notifica error por identificador incorrecto

Tabla 3.59: CU-8 (Obtener agente)

CU-9	Añadir sensor a un agente
Descripción	Añade un sensor al agente seleccionado
Actores	Administrador
Referencias	RF-5
Precondiciones	El agente y el sensor deben existir
Postcondiciones	El agente tiene un nuevo sensor
Flujo normal	1. Administrador: proporciona los identificadores del agente y del sensor al sistema 2. Sistema: comprueba si existen tanto el agente como el sensor 3. Sistema: añade el sensor al agente y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por identificadores incorrectos 3b. Sistema: notifica error por agente con el sensor ya instalado

Tabla 3.60: CU-9 (Añadir sensor a un agente)

CU-10	Añadir habilidad a un agente
Descripción	Añade una habilidad al agente seleccionado
Actores	Administrador
Referencias	RF-6
Precondiciones	El agente y la habilidad deben existir
Postcondiciones	El agente tiene una nueva habilidad
Flujo normal	1. Administrador: proporciona ids del agente y la habilidad al sistema 2. Sistema: comprueba ids 3. Sistema: añade habilidad al agente y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por identificadores incorrectos 3c. Sistema: notifica error por habilidad ya asignada

Tabla 3.61: CU-10 (Añadir habilidad a un agente)

CU-11	Eliminar sensor de un agente
Descripción	Elimina un sensor del agente seleccionado
Actores	Administrador
Referencias	RF-7
Precondiciones	El agente y el sensor deben existir
Postcondiciones	El agente tiene un sensor menos
Flujo normal	1. Administrador: proporciona ids del agente y del sensor al sistema 2. Sistema: comprueba ids 3. Sistema: elimina sensor al agente y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por identificadores incorrectos 3c. Sistema: notifica error por sensor no acoplado al agente

Tabla 3.62: CU-11 (Eliminar sensor de un agente)

CU-12	Eliminar habilidad de un agente
Descripción	Elimina una habilidad del agente seleccionado
Actores	Administrador
Referencias	RF-8
Precondiciones	El agente y la habilidad deben existir
Postcondiciones	El agente tiene una habilidad menos
Flujo normal	1. Administrador: proporciona ids del agente y de la habilidad al sistema 2. Sistema: comprueba ids 3. Sistema: elimina habilidad al agente y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por identificadores incorrectos 3c. Sistema: notifica error por habilidad no acoplado al agente

Tabla 3.63: CU-12 (Eliminar habilidad de un agente)

CU-13	Listar agentes
Descripción	Lista los agentes disponibles en el sistema
Actores	Administrador
Referencias	RF-9
Precondiciones	Ninguna
Postcondiciones	Se muestran todos los agentes
Flujo normal	1. Administrador: realiza petición para listar agentes 2. Sistema: envía una lista con los agentes disponibles
Flujo alternativo	2b. Sistema: envía una lista vacía dado que no hay agentes

Tabla 3.64: CU-13 (Listar agentes)

CU-14	Crear sensor
Descripción	Crea un nuevo sensor en el sistema
Actores	Administrador
Referencias	RF-10
Precondiciones	Ninguna
Postcondiciones	Sensor registrado en el sistema
Flujo normal	1. Administrador: envía los datos del sensor 2. Sistema: comprueba los datos 3. Sistema: almacena al sensor y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por datos incorrectos

Tabla 3.65: CU-14 (Crear sensor)

CU-15	Obtener sensor
Descripción	Obtiene los datos del sensor almacenados en el sistema
Actores	Administrador
Referencias	RF-11
Precondiciones	El ensor debe existir
Postcondiciones	Se muestran los datos del agente
Flujo normal	1. Administrador: proporciona id del sensor al sistema 2. Sistema: comprueba id 3. Sistema: muestra los datos asociados al sensor
Flujo alternativo	3b. Sistema: notifica error por id incorrecto

Tabla 3.66: CU-15 (Obtener sensor)

CU-16	Modificar sensor
Descripción	Modifica un sensor en el sistema
Actores	Administrador
Referencias	RF-12
Precondiciones	El sensor debe existir
Postcondiciones	Se muestran los datos del agente
Flujo normal	1. Administrador: proporciona id del sensor y los datos a modificar al sistema 2. Sistema: comprueba id y datos 3. Sistema: modifica el sensor y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por id incorrecto 3c. Sistema: notifica error por datos incorrectos

Tabla 3.67: CU-16 (Modificar sensor)

CU-17	Eliminar sensor
Descripción	Elimina sensor en el sistema
Actores	Administrador
Referencias	RF-13
Precondiciones	El sensor debe existir
Postcondiciones	El sensor deja de existir en el sistema
Flujo normal	1. Administrador: proporciona id del sensor y pide eliminar 2. Sistema: comprueba id 3. Sistema: elimina el sensor y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por id incorrecto

Tabla 3.68: CU-17 (Eliminar sensor)

CU-18	Listar sensores
Descripción	Lista los sensores disponibles en el sistema
Actores	Administrador
Referencias	RF-14
Precondiciones	Ninguna
Postcondiciones	Se muestra la lista de los sensores
Flujo normal	1. Administrador: realiza petición para listar sensores 2. Sistema: envía una lista de los sensores disponibles
Flujo alternativo	3b. Sistema: envía una lista vacía por inexistencia de sensores

Tabla 3.69: CU-18 (Listar sensores)

CU-19	Crear habilidad
Descripción	Crea una nueva habilidad en el sistema
Actores	Administrador
Referencias	RF-15
Precondiciones	Ninguna
Postcondiciones	Nueva habilidad registrada en el sistema
Flujo normal	1. Administrador: envía datos de la habilidad y pide crearla 2. Sistema: comprueba los datos 3. Sistema: almacena habilidad y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por datos incorrectos

Tabla 3.70: CU-19 (Crear habilidad)

CU-20	Obtener habilidad
Descripción	Obtiene los datos de la habilidad almacenada en el sistema
Actores	Administrador
Referencias	RF-16
Precondiciones	La habilidad debe existir
Postcondiciones	Muestra los datos de la habilidad
Flujo normal	1. Administrador: proporciona id de la habilidad y pide obtener 2. Sistema: comprueba id 3. Sistema: envía los datos a sociados al id
Flujo alternativo	3b. notifica error por id incorrecto

Tabla 3.71: CU-20 (Obtener habilidad)

CU-21	Modificar habilidad
Descripción	Modifica habilidad en el sistema
Actores	Administrador
Referencias	RF-17
Precondiciones	La habilidad debe existir
Postcondiciones	Se modifican los datos de la habilidad
Flujo normal	1. Administrador: proporciona id de la habilidad y pide modificación 2. Sistema: comprueba id 3. Sistema: modifica los datos a sociados al id
Flujo alternativo	3b. notifica error por id incorrecto

Tabla 3.72: CU-21 (Modificar habilidad)

CU-22	Eliminar habilidad
Descripción	Elimina habilidad del sistema
Actores	Administrador
Referencias	RF-18
Precondiciones	La habilidad debe existir
Postcondiciones	Deja de existir la habilidad en el sistema
Flujo normal	1. Administrador: proporciona id de la habilidad y pide eliminación 2. Sistema: comprueba id 3. Sistema: elimina la habilidad y notifica éxito
Flujo alternativo	3b. notifica error por id incorrecto

Tabla 3.73: CU-22 (Eliminar habilidad)

CU-23	Listar habilidades
Descripción	Lista las habilidades disponibles en el sistema
Actores	Administrador
Referencias	RF-19
Precondiciones	Ninguna
Postcondiciones	Muestra todas las habilidades
Flujo normal	1. Administrador: pide al sistema la lista de habilidades 2. Sistema: muestra todas las habilidades
Flujo alternativo	2b. Sistema: muestra lista vacía por inexistencia de habilidades

Tabla 3.74: CU-23 (Listar habilidades)

CU-24	Crear mapa
Descripción	Crea un mapa en el sistema
Actores	Administrador
Referencias	RF-20
Precondiciones	Ninguna
Postcondiciones	Un nuevo mapa se establece en el sistema
Flujo normal	1. Administrador: pide al sistema crear un mapa y le envía los datos del mismo 2. Sistema: comprueba los datos 3. Sistema: genera el nuevo mapa y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por datos incorrectos

Tabla 3.75: CU-24 (Crear mapa)

CU-25	Eliminar mapa
Descripción	Elimina el mapa del sistema
Actores	Administrador
Referencias	RF-36
Precondiciones	El mapa debe existir
Postcondiciones	El mapa se ha eliminado
Flujo normal	1. Administrador: proporciona id del mapa al sistema 2. Sistema: comprueba el id 3. Sistema: elimina el mapa que contiene ese id
Flujo alternativo	3b. Sistema: notifica error por id incorrecto

Tabla 3.76: CU-25 (Eliminar mapa)

CU-26	Obtener mapa
Descripción	Obtiene el mapa del sistema
Actores	Administrador, Visualizador
Referencias	RF-21
Precondiciones	El mapa debe existir
Postcondiciones	Se envían los datos del mapa
Flujo normal	1. Usuario: proporciona id del mapa al sistema 2. Sistema: comprueba el id 3. Sistema: envía los datos asociados al id
Flujo alternativo	3b. Sistema: notifica error por id incorrecto

Tabla 3.77: CU-26 (Obtener mapa)

CU-27	Obtener información del mapa
Descripción	Obtiene ciertos datos del mapa
Actores	Administrador, Visualizador
Referencias	RF-22
Precondiciones	El mapa debe existir
Postcondiciones	El sistema envía los datos del mapa
Flujo normal	1. Usuario: proporciona id del mapa al sistema 2. Sistema: comprueba el id 3. Sistema: envía los datos informativos (no envía obstáculos) asociados al id
Flujo alternativo	3b. Sistema: notifica error por id incorrecto

Tabla 3.78: CU-27 (Obtener información del mapa)

CU-28	Obtener obstáculos
Descripción	Obtiene obstáculos asociados a un mapa
Actores	Administrador
Referencias	RF-24
Precondiciones	El mapa debe existir
Postcondiciones	El sistema muestra los obstáculos
Flujo normal	1. Administrador: proporciona id del mapa al sistema 2. Sistema: comprueba el id 3. Sistema: envía los obstáculos del mapa
Flujo alternativo	3b. Sistema: notifica error por id incorrecto

Tabla 3.79: CU-28 (Obtener obstáculos)

CU-29	Listar nombre de los mapas
Descripción	Lista todos los nombres de los mapas del sistema
Actores	Administrador
Referencias	RF-23
Precondiciones	Ninguna
Postcondiciones	El sistema lista los nombres de los mapas
Flujo normal	<ol style="list-style-type: none"> 1. Administrador: pide listado de mapas 2. Sistema: envía el listado con los nombres
Flujo alternativo	2b. Sistema: envía el listado vacío por inexistencia de mapas

Tabla 3.80: CU-29 (Obtener información del mapa)

CU-30	Crear sesión
Descripción	Crea la sesión para comenzar a utilizar al agente en el entorno
Actores	Administrador, Usuario
Referencias	RF-30
Precondiciones	Deben existir el usuario, el mapa y el agente a utilizar
Postcondiciones	Se crea una sesión con un identificador único
Flujo normal	<ol style="list-style-type: none"> 1. Usuario: El usuario pide al sistema crear la sesión enviando los identificadores de mapa, usuario y agente 2. Sistema: comprueba los datos en sus distintos subsistemas asociados 3. Sistema: envía al usuario un identificador único de sesión
Flujo alternativo	<ol style="list-style-type: none"> 3b. Sistema: notifica error por datos incorrectos 3c. Sistema: notifica error por fallos internos

Tabla 3.81: CU-30 (Crear sesión)

CU-31	Eliminar sesión
Descripción	Elimina una sesión del sistema
Actores	Administrador
Referencias	RF-31
Precondiciones	La sesión debe existir y estar activa
Postcondiciones	Se elimina la sesión del sistema
Flujo normal	1. Administrador: proporciona id de la sesión a eliminar 2. Sistema: comprueba el id y si esta sesión está activa 3. Sistema: elimina la sesión y notifica éxito
Flujo alternativo	3b. Sistema: notifica error por id inválido 3c. Sistema: notifica error por sesión activa

Tabla 3.82: CU-31 (Eliminar sesión)

CU-32	Obtener datos de la sesión
Descripción	Obtiene datos de partida
Actores	Administrador, Visualizador
Referencias	RF-32
Precondiciones	La sesión debe existir
Postcondiciones	Se muestran los datos
Flujo normal	1. Administrador: proporciona id de sesión 2. Sistema: comprueba id y sus datos pertinentes 3. Sistema: envía los datos
Flujo alternativo	3b. Sistema: notifica error por id incorrecto

Tabla 3.83: CU-32 (Obtener datos de la sesión)

CU-33	Modificar sesión
Descripción	Actualiza la sesión
Actores	Administrador, Usuario
Referencias	RF-33
Precondiciones	La sesión debe existir y estar activa
Postcondiciones	Se modifica la sesión pedida
Flujo normal	1. Usuario: proporciona id de sesión y datos a modificar 2. Sistema: comprueba datos 3. Sistema: modifica la sesión
Flujo alternativo	3b. Sistema: notifica error por id incorrecto 3c. Sistema: notifica error por datos incorrectos

Tabla 3.84: CU-33 (Modificar sesión)

CU-34	Realizar habilidad
Descripción	Se pide al sistema que se actualice la posición del agente en la sesión actual
Actores	Usuario, Administrador
Referencias	RF-34
Precondiciones	La sesión y la habilidad deben existir
Postcondiciones	Se modifica la sesión según lo pedido
Flujo normal	<p>1. Usuario: proporciona el id de sesión y la habilidad a realizar</p> <p>2. Sistema: comprueba datos en sus subsistemas (sesión existe y activa, habilidad correcta, energía no vacía, obstáculos)</p> <p>3. Sistema: siguiendo el CU-33 modifica la sesión</p> <p>4. Sistema: notifica habilidad realizada</p>
Flujo alternativo	<p>4b. Sistema: notifica error por id incorrecto</p> <p>4c. Sistema: notifica error por habilidad incorrecta o no realizada</p> <p>4d. Sistema: notifica error por energía vacía o "muerte"</p> <p>4e. Sistema: notifica objetivo completado o misión completada</p>

Tabla 3.85: CU-34 (Realizar habilidad)

CU-35	Mostrar actualizaciones
Descripción	Cada evento que ocurra durante una sesión es transmitido
Actores	Administrador, Visualizador
Referencias	RF-35
Precondiciones	La sesión debe estar iniciada
Postcondiciones	Se muestra información de los sensores de partida con cada cambio
Flujo normal	<ol style="list-style-type: none"> 1. Usuario: proporciona id de sesión y pide registrarse en la cola de eventos 2. Sistema: registra el usuario 3. Sistema: con cada nuevo evento, envía la información
Flujo alternativo	<ol style="list-style-type: none"> 2b. Sistema: notifica error por sesión inexistente 3b. Sistema: no envía información por sesión expirada

Tabla 3.86: CU-35 (Mostrar actualizaciones)

CU-36	Obtener obstáculos según posición
Descripción	Dada una posición del agente y una distancia, obtiene los obstáculos a su alrededor
Actores	Administrador, Usuario
Referencias	RF-25
Precondiciones	El mapa debe existir
Postcondiciones	Se muestra la lista de obstáculos
Flujo normal	<ol style="list-style-type: none"> 1. Usuario: proporciona id del mapa, la distancia visual y la posición del agente 2. Sistema: mediante sus subsistemas, comprueba datos 3. Sistema: genera la lista de obstáculos cercanos
Flujo alternativo	<ol style="list-style-type: none"> 3b. Sistema: notifica error por id incorrectos 3c. Sistema: notifica error por datos incorrectos (posición no encontrada, distancia inválida)

Tabla 3.87: CU-36 (Obtener obstáculos según posición)

CU-37	Finalizar sesión
Descripción	Finaliza la sesión por conseguir objetivo o por errores
Actores	Administrador, Usuario
Referencias	RF-37
Precondiciones	La sesión debe existir y ser válida
Postcondiciones	La sesión finaliza y es inválida
Flujo normal	<ol style="list-style-type: none"> 1. Usuario: proporciona id y pide cerrar sesión 2. Sistema: comprueba si la partida sigue en curso 3. Sistema: inactiva la sesión para que no se pueda usar
Flujo alternativo	3b. Sistema: notifica error por sesión ya inactiva

Tabla 3.88: CU-37 (Finalizar sesión)

CU-38	Obtener datos de los sensores
Descripción	Según el tipo de sensores que tiene el agente, se devuelve el valor de los sensores
Actores	Administrador, Usuario
Referencias	RF-25, CU-36
Precondiciones	El agente debe tener sensores y estos deben existir
Postcondiciones	Se listan los obstáculos detectados por los sensores
Flujo normal	<p>1. Usuario: pide obtener los datos de sus sensores en un mapa</p> <p>2. Sistema: mediante CU-36 obtiene los datos de los obstáculos en un rango</p> <p>3. Sistema: según el tipo de sensores del agente, lista únicamente los obstáculos en posiciones concretas de ese rango</p>
Flujo alternativo	<p>3b. Sistema: si no hay obstáculos la lista es vacía</p> <p>3c. Sistema: notifica error por id incorrecto</p>

Tabla 3.89: CU-38 (Obtener datos de los sensores)

3.4. Diagramas de secuencia

Un diagrama de secuencia es un tipo de diagrama de interacción UML que muestra cómo los componentes del sistema interactúan en una secuencia temporal. Es útil para visualizar y comprender el flujo de mensajes entre los diferentes objetos durante la ejecución de un proceso o función específica.

En un diagrama de secuencia, los objetos se representan como cajas, y las interacciones entre ellos se muestran como flechas que indican el flujo de mensajes.

Al crear un diagrama de secuencia, se busca capturar la secuencia de interacciones basada, por ejemplo, en los casos de uso previamente analizados. Como muchos de ellos son triviales, estudiaremos únicamente la interacción en un flujo normal de una sesión.

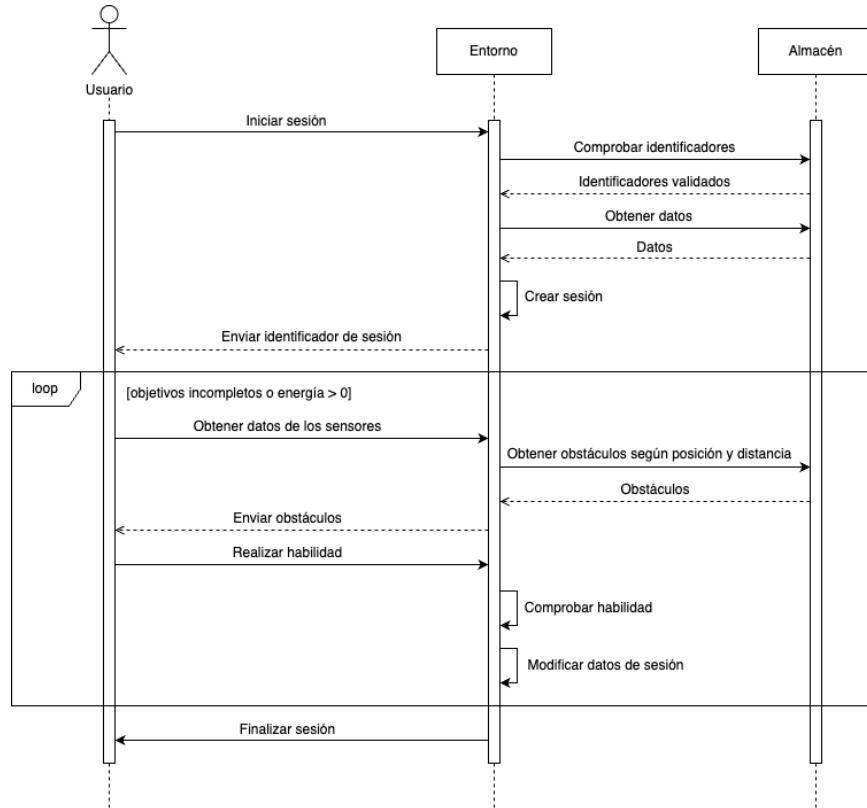


Figura 3.7: Diagramas de secuencia - Flujo de una Sesión

En el diagrama de secuencia anterior, podemos ver el uso principal del sistema para un usuario cualquiera. Se observan objetos como el Entorno, que controla la sesión, y el Almacén, que administra los datos del sistema. En capítulos siguientes se detallará esta disposición.

El otro diagrama de secuencia que cabe destacar es el relacionado con el Visualizador, actor que interactúa con el sistema para mostrar en tiempo real las actualizaciones de una sesión.

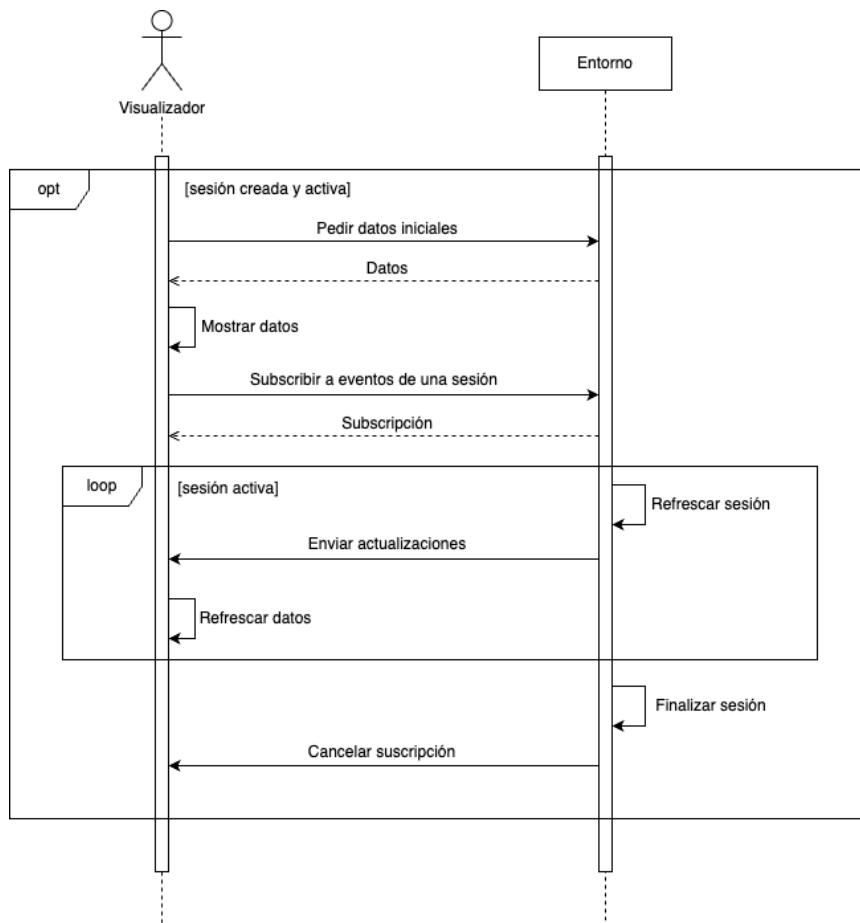


Figura 3.8: Diagramas de secuencia - Flujo del Visualizador

Es importante comentar la simbología utilizada, como el uso de *opt* para definir secuencias que se dan gracias a una condición, o el uso de *loop* para definir que se repiten secuencias hasta que una condición se deja de cumplir.

3.5. Modelo conceptual de datos

El modelo conceptual de datos es una representación abstracta de la estructura y las relaciones de los datos dentro del sistema. Su propósito es capturar de manera comprensible los requisitos de información de los usuarios, sin adentrarse en detalles de implementación.

Por consiguiente, este modelo incluye las entidades principales y sus relaciones, prescindiendo de atributos o claves específicas. De esta manera, se obtiene un esquema abstracto de alto nivel, independiente de plataformas o tecnologías específicas de datos.

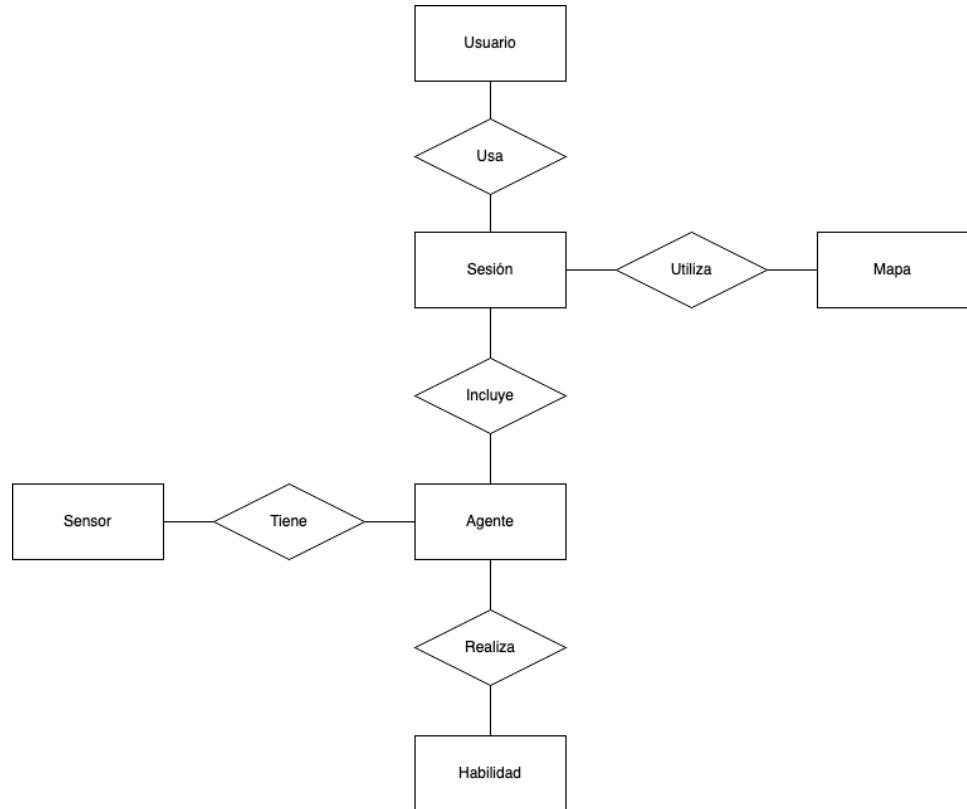


Figura 3.9: Modelo conceptual de datos

Capítulo 4

Diseño

4.1. Arquitectura del sistema

4.1.1. Arquitectura lógica

El diseño arquitectónico que utilizaremos para elaborar nuestro sistema se basa en microservicios [2]. A diferencia de una arquitectura monolítica donde una aplicación se construye como una unidad, la arquitectura de microservicios divide la aplicación en un conjunto de servicios independientes y más pequeños, que se comunican mediante protocolos ligeros como HTTP.

Utilizar este diseño, nos provee de ciertas ventajas que se consideran de alta importancia, entre ellas:

- **Facilidad de despliegue:** los problemas de dependencias se solucionan más rápidamente. En contraste con una aplicación monolítica, donde las dependencias pueden ser más difíciles de gestionar durante la migración de sistemas, la modularidad facilita la identificación y resolución de fallos. Además, hay diversas formas de desplegar estos componentes, como en contenedores, lo que reduce las posibilidades de fallos relacionados con las dependencias.
- **Facilidad de mantenimiento:** el mantenimiento se vuelve más manejable al dividir la aplicación en servicios más especializados y, por tanto, más simples para la comprensión.
- **Mejora de escalabilidad y de rendimiento:** permite aumentar el paralelismo del sistema, asignando los recursos de manera más eficiente y permitiendo que los servicios que experimentan una alta demanda pueden escalar horizontalmente sin afectar a los demás.

Nuestras APIs (abstracción de funciones y procedimientos) las desarrollaremos mediante la lógica de restricciones **REST**, que se adapta a nuestra arquitectura de microservicios.

En el siguiente esquema se observa la fragmentación realizada en nuestro sistema.

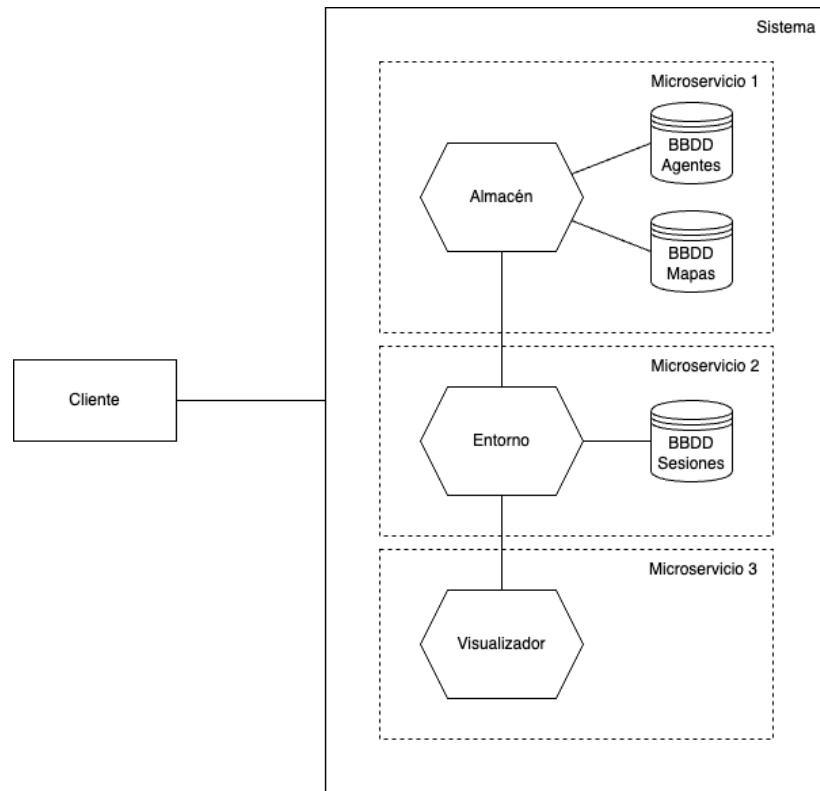


Figura 4.1: Diagrama de microservicios

En la Figura 4.1 se puede observar la división de nuestro sistema en tres microservicios, los cuales realizan las siguientes labores:

- **Almacén:** es esencial para almacenar y gestionar toda la información relacionada con los agentes, incluyendo sus habilidades y sensores, así como los mapas utilizados en el juego. Actúa como un repositorio centralizado que facilita el acceso rápido y eficiente a los datos necesarios para la ejecución de las partidas y la estrategia de los agentes.
- **Entorno:** es fundamental para la gestión de las sesiones de juego y juega un papel crucial en la orquestación de las interacciones entre

los agentes programados por los usuarios. Este microservicio se encarga de iniciar y monitorear cada sesión, controlando el movimiento de los agentes, verificando su estado y determinando el final de la partida. Además, administra todos los parámetros y configuraciones de la sesión, asegurando que el juego funcione correctamente y de manera eficiente para múltiples sesiones simultáneas.

- **Visualizador:** mejora la experiencia del usuario final al mostrar datos en tiempo real como la energía de cada agente, su posición y movimiento a lo largo del mapa, entre otros aspectos relevantes de la partida.

4.1.2. Arquitectura física

Cuando hemos hablado acerca de la arquitectura lógica, se ha definido en alto nivel cómo se organiza el sistema mediante distintos componentes de software que se conectan entre sí.

Ahora, detallaremos cómo los componentes de software de la arquitectura lógica se mapean en el hardware:

- **Servidores web dedicados:** los microservicios de Almacén, Entorno y Visualizador se despliegan en su propio servidor para garantizar el aislamiento y la escalabilidad del sistema. Estos servidores están configurados con puertos específicos para facilitar la comunicación tanto con el cliente como entre microservicios.
- **Contenedores para bases de datos:** cada modelo de datos se aloja en un contenedor permitiendo su portabilidad y fácil instalación. Se necesita un servidor para levantar contenedores y puertos específicos para cada contenedor.
- **Interfaz de usuario:** el cliente interactúa con el sistema mediante un navegador web, lo que requiere que el Visualizador esté optimizado para ser compatible con los principales navegadores.
- **Independencia del sistema operativo:** el sistema está diseñado para ser independiente del sistema operativo, garantizando la compatibilidad y facilitando la implementación en diversas infraestructuras.
- **Acceso a internet:** Es esencial que el cliente disponga de acceso a internet para conectar con los servidores. Sin embargo, el sistema está diseñado para permitir también una implementación local.

4.2. Patrones de diseño

Los patrones de diseño [3] representan soluciones consolidadas para los desafíos recurrentes en el diseño de software, extraídas de la experiencia acumulada de los desarrolladores a lo largo del tiempo.

En nuestro proyecto, hemos adoptado una variedad de estos patrones y los hemos clasificados en distintos grupos según su naturaleza.

4.2.1. Patrones creacionales

Se centran en la creación de objetos de manera flexible y eficiente.

- **Inyección de Dependencias:** desacopla las clases de una capa de otras, facilitando su comunicación. Utilizado entre las distintas clases del sistema que se van a definir.
- **Builder:** permite construir objetos complejos paso a paso. Produce distintos tipos y representaciones de un objeto empleando el mismo código de construcción. Utilizado cuando creamos DTOs mediante solicitudes HTTP.

4.2.2. Patrones estructurales

Ofrecen un enfoque para organizar clases y objetos de manera que se formen estructuras más complejas, manteniendo al mismo tiempo la flexibilidad y eficiencia de estas estructuras.

- **DAO (Objeto de Acceso a Datos):** separa la lógica de negocio del acceso a los datos. Utilizado para acceder a los datos de las distintas BBDD.

4.2.3. Patrones de comportamiento

Se encargan de la interacción entre objetos y la distribución de responsabilidades entre ellos.

- **Observer:** define que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente. Es útil para actualizar el estado del Visualizador.

4.2.4. Otros patrones

- **Inversión de Control:** implica confiar la tarea de crear y administrar objetos a un framework externo, en lugar de gestionarlo directamente dentro del código del programa.
- **DTO** (Objeto de Transferencia de Datos): abstrae los datos subyacentes, permitiendo que las diferentes partes de la aplicación se comuniquen sin necesidad de conocer la estructura interna de los datos. Utilizado para la entrada y salida de datos de los distintos microservicios.

4.3. Modelo lógico de datos

En la Sección 3.5 tenemos un primer acercamiento de alto nivel sobre el modelo de datos utilizado en este proyecto, que nos presenta las entidades principales del sistema y las relaciones construidas entre sí.

Nuestra misión en este apartado es describir, con el mayor detalle posible, las entidades del sistema y sus relaciones.

Para adaptarnos a las diversas necesidades del modelado de datos que surgen en el contexto del proyecto necesitamos una **capa de almacenamiento** que tenga un esquema rígido para los datos de los agentes, como el modelo relacional, mientras que para los mapas o las sesiones se requiere cierta flexibilidad o un esquema dinámico y se opta por un modelo no relacional.

En nuestro modelo de datos, gestionamos información geoespacial que es crítica para el funcionamiento eficiente de la aplicación, como las coordenadas en los mapas. Utilizaremos **índices espaciales** para asegurar que se realicen las consultas de manera eficiente.

El uso de índices espaciales es crucial porque sin ellos, las consultas que involucran datos geoespaciales serían extremadamente lentas.

4.3.1. Diagrama lógico relacional

El almacén de agentes, consta de tres entidades fundamentales: los agentes, las habilidades y los sensores. Cada una de estas entidades tiene tres atributos comunes, incluyendo un identificador único, un nombre y una descripción. Cada agente puede poseer múltiples habilidades y sensores y, a su vez, una habilidad o un sensor puede estar asociado a múltiples agentes.

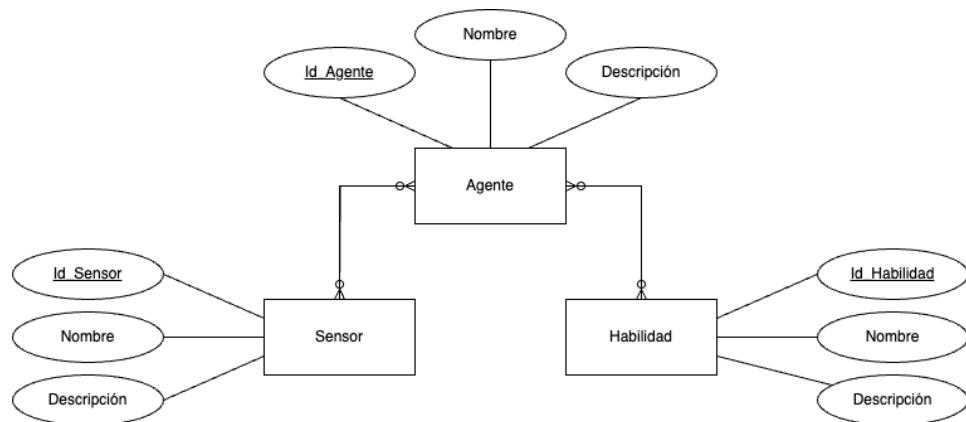


Figura 4.2: Diagrama lógico de datos - Almacén de Agentes

4.3.2. Paso a tablas

Tras haber definido el modelo relacional, formalizaremos el paso a tablas. Como las relaciones representadas son "muchos a muchos", se generarán tablas para cada entidad y además tablas para representar estas relaciones.

- Agente(Id_Agente (CP), Nombre, Descripción).
- Sensor(Id_Sensor (CP), Nombre, Descripción).
- Habilidad(Id_Habilidad (CP), Nombre, Descripción).
- Agente_Sensor(Id_Agente (CP, CF a Agente.Id_Agente), Id_Sensor (CP, CF a Sensor.Id_Sensor)).
- Agente_Habilidad(Id_Agente (CP, CF a Agente.Id_Agente), Id_Habilidad (CP, CF a Habilidad.Id_Habilidad)).

Vemos que CP representa la clave primaria de cada tabla (atributo que identifica de manera única a cada tupla) y CF la clave foránea o externa (apunta a la clave primaria de otra tabla y depende de ella).

4.3.3. Normalización

Con el paso a tablas representando las necesidades de nuestro sistema, debemos garantizar la integridad de la información y la no redundancia de los datos. Para ello, pasaremos a la etapa de normalización, cuyo propósito es el mencionado.

Para normalizar, se hace uso de las formas normales (FN), que resuelven problemas específicos de diseño. Existen diversas, aunque se suelen aplicar las tres primeras, como haremos en este caso.

Nuestro modelo cumple la 1FN dado que existe atomicidad en los datos, estos no están repetidos ni multivaluados. También cumple la 2FN ya que, además de cumplir la primera, todos los atributos que no forman parte de una clave dependen de forma completa de la clave principal. Y finalmente, se encuentra en 3FN puesto que cumple la 2FN y los atributos no clave, no dependen de otros atributos no clave.

4.3.4. Diagramas lógicos no relacionales

En el caso de almacenamiento de los mapas, las sesiones y los usuarios, tenemos un modelo no relacional basado en documentos, que aunque tiene diferente enfoque, podemos representar como un diagrama entidad-relación teniendo en cuenta que las colecciones son entidades y los atributos referenciales son relaciones.

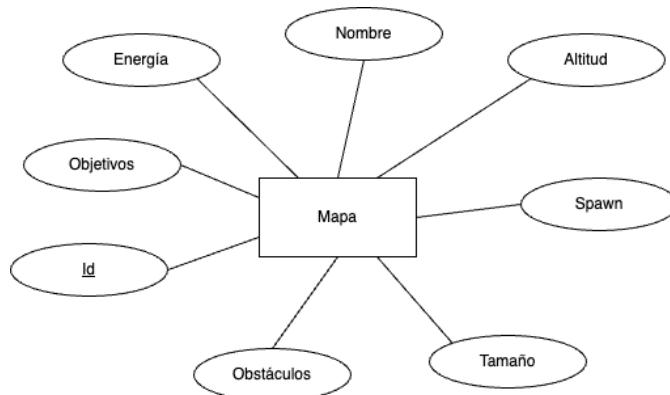


Figura 4.3: Diagrama lógico de datos - Mapas

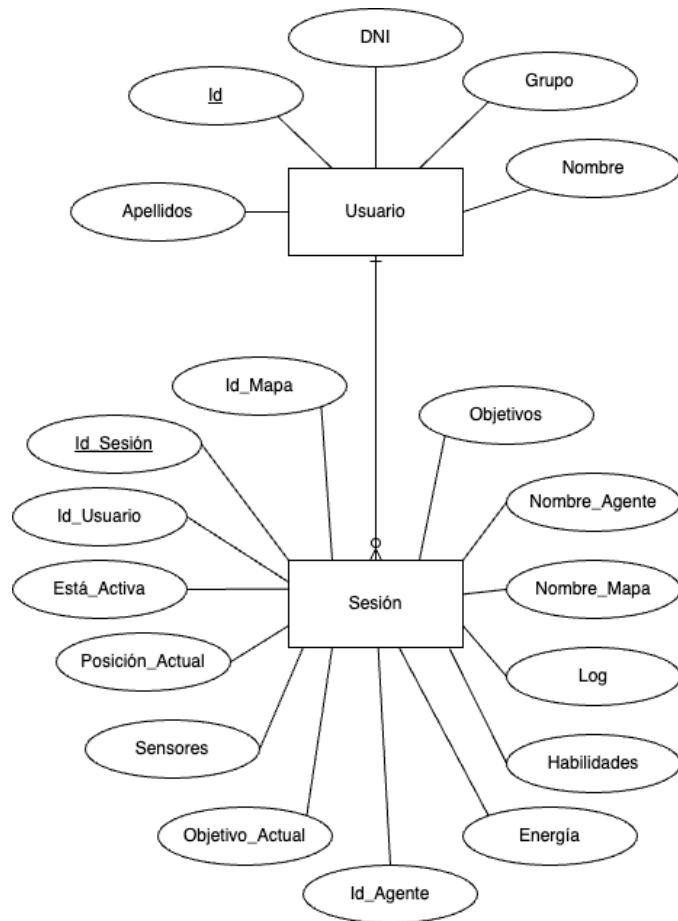


Figura 4.4: Diagrama lógico de datos - Sesiones y Usuarios

En lugar de normalizar los datos en tablas separadas como se hace en el modelo relacional anteriormente visto, en modelos no relacionales, se tiende a denormalizar los datos para mejorar el rendimiento de las consultas y simplificar el acceso a los datos. Esto puede implicar almacenar datos redundantes en un solo documento.

4.4. Modelo físico de datos

En el desarrollo del modelo lógico de la base de datos para el sistema en discusión, se ha procedido con una definición detallada de los distintos modelos de datos que se integrarán en la estructura de almacenamiento de información.

Esta sección se dedicará a la elaboración de las representaciones específicas.

cas de estos modelos en dos formatos principales: relacional y no relacional orientado a documentos. La especificación exhaustiva incluirá las tablas, columnas, claves primarias y claves foráneas en el modelo relacional, y la estructura de los documentos y las colecciones en el modelo no relacional.

Es fundamental subrayar que, en este punto del trabajo, no se discutirán las particularidades de los sistemas gestores de bases de datos que eventualmente podrían ser utilizados para implementar ambos modelos.

4.4.1. Diagrama físico relacional

En primer lugar, veamos el modelo físico de datos relacional:

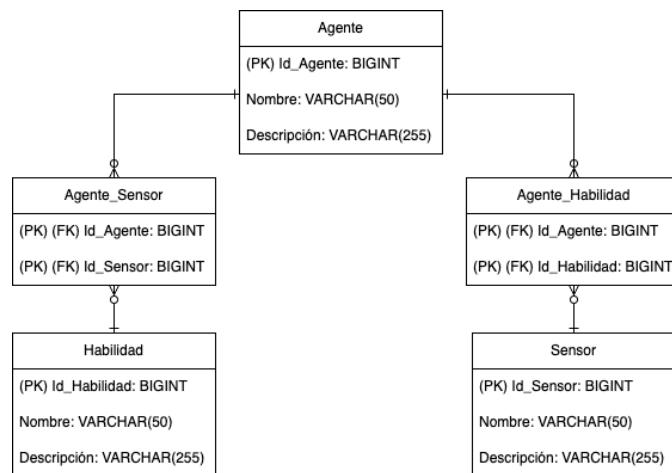


Figura 4.5: Diagrama físico de datos - Agentes

En la Figura 4.5 observamos las tablas que habíamos normalizado en la Subsección 4.3.3 junto con el tipo de valor de cada atributo, las claves primarias y las relaciones existentes entre tablas mediante claves foráneas.

4.4.2. Diagrama físico no relacional

Veamos de forma específica los distintos modelo de datos no relacionales de los mapas, usuarios y sesiones.

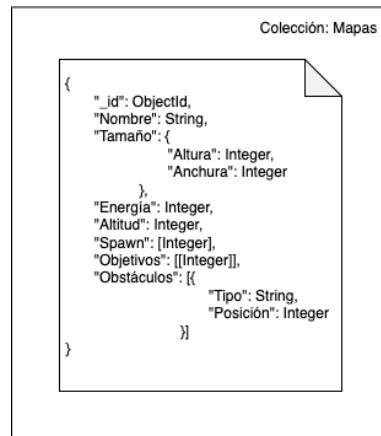


Figura 4.6: Diagrama físico de datos - Mapas

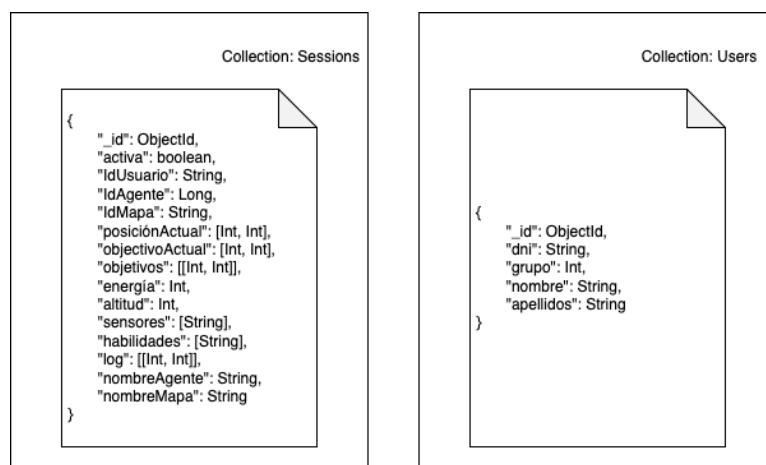


Figura 4.7: Diagrama físico de datos - Sesiones y Usuarios

Son visibles colecciones de listas de pares clave-valor (clave-tipo), ya que en este caso, representamos el diagrama físico utilizando un formato **JSON/BSON** que es el utilizado en las bases de datos orientadas a documentos.

4.5. Modelo de clases

Los diagramas de clases son esenciales en la fase de diseño puesto que proporcionan una representación visual de las clases que componen el sistema.

Vamos a representar los distintos microservicios enfocados en el backend de nuestro sistema.

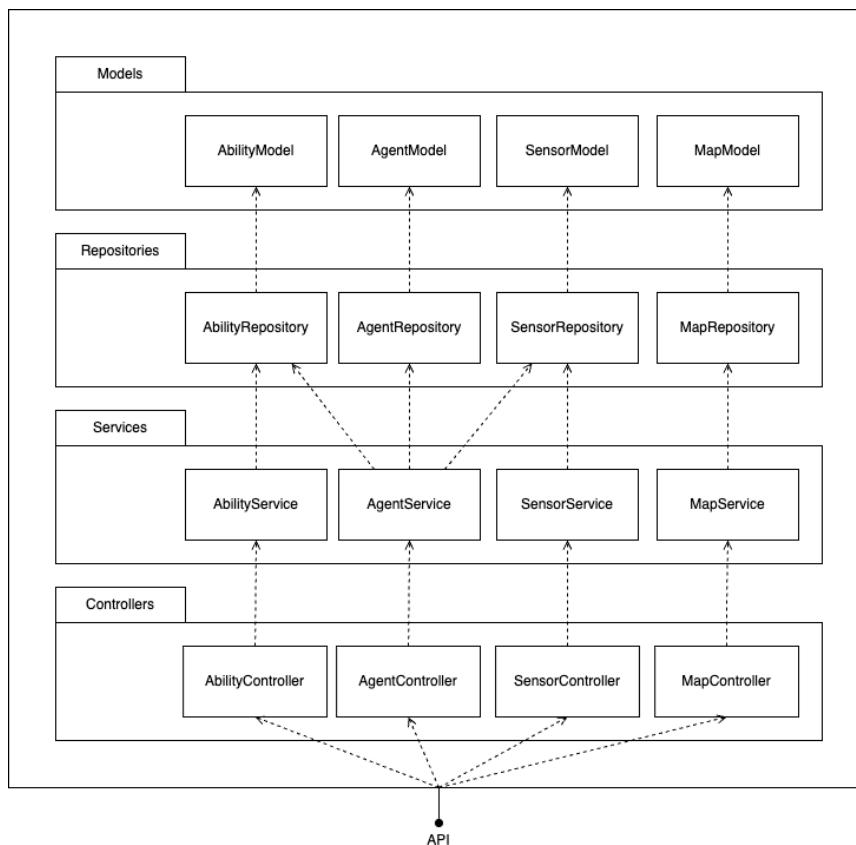


Figura 4.8: Diagrama de clases - Almacén

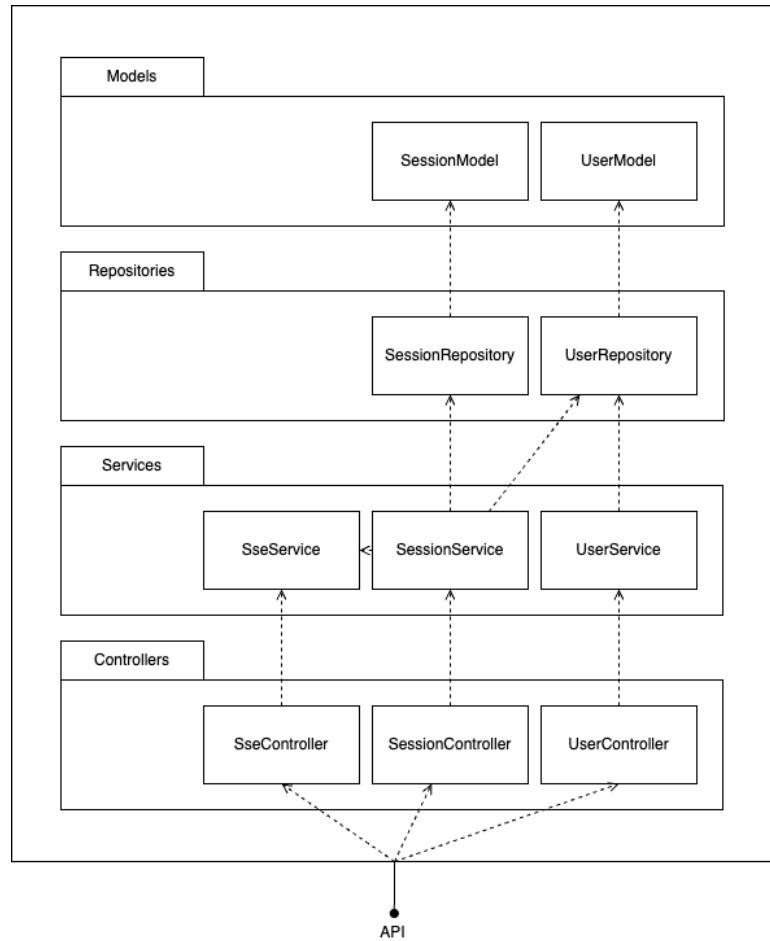


Figura 4.9: Diagrama de clases - Entorno

Los diagramas de clases se han organizado en distintos paquetes que reflejan un diseño modular y claro.

Los **modelos** definen la estructura de datos del sistema, correspondiendo a tablas en las bases de datos relacionales o documentos en las no relacionales. Los **repositorios**, en cambio, proporcionan una interfaz más limpia para interactuar con los datos. Además tenemos los **servicios** que definen la lógica de negocio del sistema y finalmente, los **controladores**, que gestionan las interacciones con el usuario, manejando las peticiones y devolviendo las respuestas adecuadas.

Representamos las relaciones de dependencia, es decir, que una clase use otra, con flechas discontinuas que van desde la clase utilizadora a la clase utilizada.

4.6. API

Una API [4] (Interfaz de Programación de Aplicaciones) no es más que un intermediario entre componentes que permite que se comuniquen entre sí. Lo que buscan concretamente es ofrecer acceso a ciertos servicios de una aplicación mediante la abstracción de la misma, evitando que nos tengamos que adentrar en cómo funciona exactamente dicha aplicación.

El sistema que estamos modelando utiliza, como se ha comentado, límites de arquitectura basados en REST (Transferencia de Estado Representacional), por lo que se considera una **RESTful API**, y tiene como característica principal el uso del protocolo de transferencia de hipertexto (HTTP) para ejecutar operaciones sobre datos en diversos formatos, como JSON o XML.

Las RESTful APIs utilizan **endpoints** para dar respuestas a las peticiones al sistema, exponen las funciones de la aplicación.

En esta sección buscamos documentar las distintas APIs que ofrecen nuestros microservicios para su uso de manera interna, ya sea por el Visualizador, o por parte de usuarios mediante otros programas.

En primer lugar, detallaremos el microservicio de Almacén, utilizando como referencia los distintos controladores, que manejan las interacciones entrantes entre los consumidores de la API.

* Si el campo *parámetros* se encuentra vacío, la petición HTTP no requiere datos en su cuerpo o ruta. En cambio, si el campo *salida* está vacío, el cuerpo de la respuesta no tiene contenido y mediante la cabecera informa si la solicitud ha sido procesada exitosamente o si, por el contrario, ha ocurrido un error. Los códigos de estado HTTP utilizados son: 200 OK (éxito en la solicitud), 404 Not Found (recurso no encontrado) y 500 Internal Server Error (error interno del servidor).

Controlador de Agentes

Verbo	Ruta	Descripción	Parámetros	Salida
GET	/agents	Obtener agentes	*	Lista agentes: id, nombre, descripción, sensores y habilidades
POST	/agents	Crear agente	Nombre y descripción	Id
GET	/agents/{id}	Obtener agente	Id	Id, nombre, descripción, sensores y habilidades
DELETE	/agents/{id}	Eliminar agente	Id	*
PATCH	/agents/{id}	Actualizar agente	Id, nombre y descripción	*
POST	/agents/sensor	Añadir sensor	Id agente e id sensor	*
POST	/agents/ability	Añadir habilidad	Id agente e id habilidad	*
DELETE	/agents/sensor	Eliminar sensor	Id agente e id sensor	*
DELETE	/agents/ability	Eliminar habilidad	Id agente e id habilidad	*

Tabla 4.1: Endpoints - Agentes

Controlador de Sensores

Verbo	Ruta	Descripción	Parámetros	Salida
GET	/sensors	Obtener sensores	*	Lista sensores: id, nombre y descripción
POST	/sensors	Crear sensor	Nombre y descripción	Id
GET	/sensors/{id}	Obtener sensor	Id	Id, nombre y descripción
DELETE	/sensors/{id}	Eliminar sensor	Id	*
PATCH	/sensors/{id}	Actualizar sensor	Id, nombre y descripción	*

Tabla 4.2: Endpoints - Sensores

Controlador de Habilidades

Verbo	Ruta	Descripción	Parámetros	Salida
GET	/abilities	Obtener habilidades	*	Lista habilidades: id, nombre y descripción
POST	/abilites	Crear habilidad	Nombre y descripción	Id
GET	/abilities/{id}	Obtener habilidad	Id	Id, nombre y descripción
DELETE	/abilities/{id}	Eliminar habilidad	Id	*
PATCH	/abilities/{id}	Actualizar habilidad	Id, nombre y descripción	*

Tabla 4.3: Endpoints - Habilidades

Controlador de Mapas

Verbo	Ruta	Descripción	Parámetros	Salida
GET	/maps/{id}	Obtener mapa	Id	Nombre, tamaño, energía, altitud, spawn, objetivos, obstáculos
GET	/maps	Listar nombre mapas	*	Lista: nombre e id de mapas
POST	/maps	Crear mapa	Nombre, tamaño (filas, columnas), obstáculos (tipo, coordenadas), energía, altitud, spawn y objetivos	Id
GET	/maps/{id}-/info	Obtener información del mapa	Id	Nombre, tamaño (filas, columnas), objetivos, altitud y energía
GET	/maps/{id}/-obstacles	Obtener obstáculos mapa	Id	Lista de obstáculos: tipo y coordenadas
GET	/maps/{id}/-obstacles/find?-{x}&{y}&{distance}	Obstáculos cerca posición	Id, coordenada y distancia	Obstáculos
DELETE	/maps/{id}	Eliminar mapa	Id	*

Tabla 4.4: Endpoints - Mapas

Tras haber definido cada endpoint de los controladores del microservicio de Almacén, veamos los de los controladores del Entorno.

Controlador de Sesiones

Verbo	Ruta	Descripción	Parámetros	Salida
GET	/session/{id}	Obtener sesión	Id	Nombre del usuario, del mapa y del agente, id agente, id mapa, energía y posición actual
GET	/session/{id} /sensors	Obtener datos de los sensores (obstáculos cercanos)	Id	Lista de obstáculos (tipo y coordenadas)
POST	/session/{id} /move	Realizar habilidad (mover agente)	Id y habilidad	Mensaje con estado del agente y éxito de la habilidad)
POST	/session	Crear sesión	Ids de agente, usuario y mapa	Id de sesión
DELETE	/session/{id}	Eliminar sesión	Id de sesión	*

Tabla 4.5: Endpoints - Sesiones

Controlador de Eventos

Verbo	Ruta	Descripción	Parámetros	Salida
GET	/sse/connect/-{id}	Suscripción a la cola de eventos en una partida	Id de la sesión	Receptor de eventos

Tabla 4.6: Endpoints - Eventos

Controlador de Usuarios

Verbo	Ruta	Descripción	Parámetros	Salida
POST	/users	Crear usuario	Nombre, apellidos, DNI y grupo	Id
GET	/users/{id}	Obtener usuario	Id	Nombre, apellidos, DNI y grupo
PATCH	/users/{id}	Actualizar usuario	Id y datos a actualizar (Nombre, apellidos, DNI y/o grupo)	*
DELETE	/users/{id}	Borrar usuario	Id	*

Tabla 4.7: Endpoints - Usuarios

4.7. Eventos Enviados por el Servidor

En el capítulo de análisis, se expone la Tabla 3.86 que define un caso de uso para mostrar las actualizaciones de los eventos que ocurren en el servidor. Esto es de vital importancia para el Visualizador puesto que necesita la información actualizada acerca de cada sesión.

A diferencia de las solicitudes HTTP tradicionales, donde el cliente solicita información al servidor y este le responde, nos hemos enfocado en que el servidor pueda enviar datos sin que el cliente lo solicite de forma explícita. Para ello, usaremos los **SSE** (Eventos Enviados por el Servidor), una tecnología que permite enviar actualizaciones a un cliente de manera asíncrona mediante una conexión HTTP estándar, lo que la hace idónea para implementarlo en nuestra RESTful API.



Figura 4.10: SSE - Funcionamiento

En primer lugar, el cliente establece una conexión HTTP persistente con el servidor. Una vez establecida la conexión, el servidor puede enviar los eventos en cualquier momento. Si el servidor no recibe actualizaciones, posee un *timeout* para cerrar la conexión.

Un mecanismo similar es el *long polling*, donde el cliente envía una solicitud al servidor y este la retiene abierta hasta que tiene nuevos datos para enviar. Luego, responde con esos datos al cliente, quien debe iniciar una nueva solicitud para futuras actualizaciones. Este proceso repetido puede ser intensivo para el servidor y dificulta la escalabilidad.

Además, existen otros protocolos como *WebSockets*, pero hemos optado por SSE debido a su eficiencia en el manejo de actualizaciones de estado.

4.8. Interfaz de usuario

Nuestro sistema se caracteriza por su enfoque centrado en la lógica de funcionamiento, priorizando un backend robusto que garantiza un procesamiento eficiente y preciso de los datos. Aún así, reconocemos la importancia del frontend y por ello el Visualizador se ha diseñado con atención para asegurar una presentación clara y efectiva de la información.

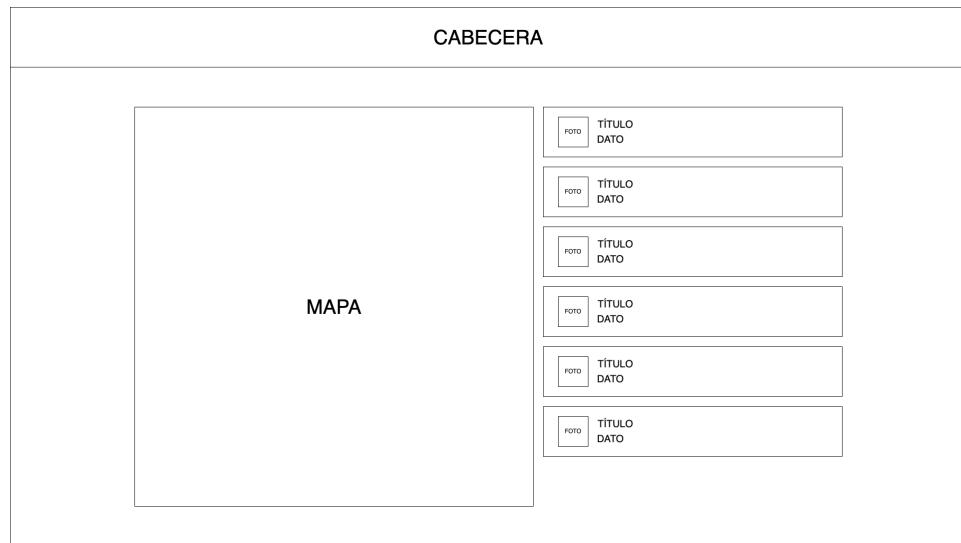


Figura 4.11: Interfaz de usuario - Visualizador

Como podemos observar en la Figura 4.11, el Visualizador está formado por distintos atributos que representan los datos de la sesión, junto con el mapa que muestra la posición de los obstáculos, el agente y los objetivos.

Capítulo 5

Implementación

Este capítulo se centra en proporcionar detalles sobre cómo se desarrolla y construye el sistema de software propuesto.

5.0.1. Lenguajes

Como lenguaje de programación principal tenemos a **Java** [9]. Este es un de alto nivel, orientado a objetos y multiplataforma, que se ha elegido para construir el backend principalmente por su madurez y estabilidad, ya que ha sido ampliamente utilizado durante años en el desarrollo de software empresarial. Además, cuenta con un amplio ecosistema que simplifica el proceso de desarrollo.

Para la creación del Visualizador, dado que se utiliza un navegador, hemos optado por **TypeScript** [10], un superconjunto de **JavaScript**, un lenguaje de programación de alto nivel, interpretado, orientado a objetos que se ha convertido en el lenguaje de facto del desarrollo web. La elección de TypeScript se debe a su capacidad para mejorar la productividad y la calidad del código mediante características como el tipado estático y las interfaces. Además, TypeScript se integra perfectamente con el ecosistema de JavaScript, lo que facilita la transición y la colaboración con otras bibliotecas y frameworks. La elección de TypeScript/JavaScript no se basa únicamente en su ubicuidad, sino también en la extraordinaria experiencia de usuario que nos permite construir y en su facilidad de desarrollo.

Además de JavaScript/TypeScript, que nos ofrece la funcionalidad, para dar estructura al contenido del Visualizador utilizamos **HTML** y para darle la apariencia **CSS** como lenguaje de estilos.

5.0.2. Frameworks

Nos hemos decantado **Spring Boot** para desarrollar el backend. Spring Boot simplifica el proceso de desarrollo de aplicaciones basadas en Spring [11], uno de los frameworks (conjunto de herramientas y componentes predefinidos que proporcionan una estructura y funcionalidades comunes para el desarrollo de software) de propósito general más utilizados.

Elimina la necesidad de configuraciones manuales tediosas al ofrecer características como autoconfiguración, donde la mayoría de las configuraciones se realizan automáticamente. También proporciona un servidor integrado, lo que significa que puedes ejecutar tu aplicación sin configuraciones adicionales.

Gracias a Spring Boot, nos centramos en la lógica del negocio en lugar de en la configuración técnica.

Por otro lado, en el frontend, se ha optado por **Angular** [12]. Angular es un framework de desarrollo de aplicaciones web frontend basado en TypeScript que utiliza un enfoque basado en componentes, donde los diferentes aspectos de la interfaz de usuario se dividen en componentes reutilizables.

Hemos elegido Angular como framework frontend porque comparte la misma filosofía de robustez y calidad que Spring Boot. Al utilizar Angular junto con Spring Boot, nos aseguramos de seguir buenas prácticas de diseño de software en todos los niveles, desde el frontend hasta el backend.

5.0.3. Sistemas Gestores de Bases de Datos

En el desarrollo de nuestro proyecto, hemos adoptado un enfoque híbrido al utilizar tanto bases de datos SQL como NoSQL como se explicó en la Sección 4.3.

Para manejar el modelo relacional de datos, optamos por **MySQL** [13], una base de datos SQL, debido a su reputación de robustez y confiabilidad. Para el modelo no relacional, incorporamos **MongoDB** [14], una base de datos NoSQL orientada a documentos en la que se guardan los datos en estructuras de datos BSON con un esquema dinámico, en lugar de en tablas como se hace en las bases de datos relacionales, lo que nos brinda flexibilidad.

5.0.4. Infraestructura

Se ha optado por la utilización de **Docker** [15] para la implementación de las bases de datos requeridas.

Docker simplifica la creación, implementación y ejecución de aplicaciones mediante contenedores que encapsulan todo su entorno, incluidas bibliotecas

y herramientas. Al utilizar contenedores para las bases de datos, evitamos problemas de compatibilidad y conflictos entre versiones, facilitando la gestión de entornos de desarrollo y producción.

5.0.5. Entorno de desarrollo

Con el desarrollo en Java en mente, se ha utilizado **IntelliJ IDEA** [16] como IDE (Entorno de Desarrollo Integrado) principal. Su integración perfecta con Java y más precisamente con Spring Boot, proporciona un flujo de trabajo fluido y productivo. Además, viene con funcionalidades avanzadas para interactuar con bases de datos y nos ha servido para manipular las nuestras.

Para el desarrollo del frontend con Angular, se ha elegido **Visual Studio Code** [17]. Este editor de código ofrece una amplia gama de extensiones específicas para Angular que facilitan el desarrollo como autocompletado de código o depuración integrada.

5.0.6. Otras herramientas

Para probar y validar las API desarrolladas, se utiliza **Postman** [18], una herramienta popular en el mundo del desarrollo de software. Postman permite enviar peticiones HTTP a las API y analizar las respuestas recibidas de manera eficiente.

Para la creación de diagramas necesarios en el desarrollo del proyecto, se ha empleado **draw.io** [19]. Esta herramienta web ofrece una interfaz intuitiva y opciones de diseño variadas, facilitando la creación rápida de diagramas.

Para almacenar el sistema, se ha optado por **Google Drive** [22], la herramienta de almacenamiento de archivos por excelencia de Google.

En este proyecto, se ha utilizado **Overleaf** [20], una plataforma en línea que permite escribir documentos **LaTeX** [21]. LaTeX permite escribir documentos con un formato profesional, gestionando automáticamente aspectos como el diseño o la numeración de secciones, entre otros.

Capítulo 6

Pruebas

Las pruebas detalladas en este capítulo se emplearán para garantizar el correcto funcionamiento y la calidad del sistema implementado.

6.1. Entorno de evaluación

El equipo que se ha utilizado para probar el sistema es un MacBook Air con las siguientes características:

- Chip: Apple M1.
- Memoria RAM: 8GB.
- Almacenamiento: 256GB SSD.
- Tarjeta gráfica: Integrada en chip.
- Sistema Operativo: MacOS Sonoma.

6.2. Estrategia

El proceso de pruebas adoptado se basa en **pruebas manuales**, es decir, pruebas realizadas por personas, con el objetivo de verificar el correcto funcionamiento del sistema.

6.3. Ejecución y resultados

Se detallan a continuación todas las pruebas realizadas.

PM-1	Prueba de crear agente
Descripción	Comprobamos que se crea un agente
Datos de entrada	Nombre y descripción
Salida esperada	Nuevo registro en la BBDD coincidente
Resultado	Nuevo registro en la BBDD con los datos

Tabla 6.1: PM-1 (Prueba de crear agente)

PM-2	Prueba de eliminar agente
Descripción	Comprobamos que se elimina un agente con su id
Datos de entrada	Identificador
Salida esperada	Identificador no encontrado en la BBDD
Resultado	Identificador inexistente en la BBDD

Tabla 6.2: PM-2 (Prueba de eliminar agente)

PM-3	Prueba de error al eliminar agente
Descripción	Comprobamos error de eliminación si el id del agente no existe
Datos de entrada	Identificador no existente
Salida esperada	Error de eliminación
Resultado	Error al no encontrar el id

Tabla 6.3: PM-3 (Prueba de error al eliminar agente)

PM-4	Prueba de modificar agente
Descripción	Comprobamos que se modifica un agente
Datos de entrada	Identificador, nombre y descripción
Salida esperada	Id coincidente con datos actualizados en la BBDD
Resultado	Id coincidente en la BBDD con los datos dados

Tabla 6.4: PM-4 (Prueba de modificar agente)

PM-5	Prueba de error al modificar agente
Descripción	Comprobamos error de modificación si el id del agente no existe
Datos de entrada	Identificador no existente, nombre y descripción
Salida esperada	Recurso no encontrado
Resultado	Error por registro no encontrado

Tabla 6.5: PM-5 (Prueba de error al modificar agente)

PM-6	Prueba de consultar agente
Descripción	Comprobamos que se muestran datos del agente
Datos de entrada	Identificador
Salida esperada	Datos asociados a ese id
Resultado	Datos exactos al registro de la BBDD con ese id

Tabla 6.6: PM-6 (Prueba de consultar agente)

PM-7	Prueba de error al consultar agente
Descripción	Comprobamos que no se muestran datos del agente
Datos de entrada	Identificador no existente
Salida esperada	Registro no encontrado
Resultado	Tupla no encontrada en BBDD

Tabla 6.7: PM-7 (Prueba de error al consultar agente)

PM-8	Prueba de añadir sensor a un agente
Descripción	Comprobamos que se añade un sensor al agente si los dos existen
Datos de entrada	Identificadores de agente y sensor
Salida esperada	Sensor visible en agente
Resultado	Agente en la BBDD con dicho sensor

Tabla 6.8: PM-8 (Prueba de añadir sensor a un agente)

PM-9	Prueba de error al añadir sensor a un agente
Descripción	Comprobamos que no se añade un sensor al agente si alguno no existe
Datos de entrada	Identificadores de agente y sensor con alguno o ambos erróneos
Salida esperada	Error de petición en la adición
Resultado	Error de petición

Tabla 6.9: PM-9 (Prueba de error al añadir sensor a un agente)

PM-10	Prueba de añadir habilidad a un agente
Descripción	Comprobamos que se añade una habilidad al agente si los dos existen
Datos de entrada	Identificadores de agente y habilidad
Salida esperada	Habilidad visible en agente
Resultado	Agente en la BBDD con dicha habilidad

Tabla 6.10: PM-10 (Prueba de añadir habilidad a un agente)

PM-11	Prueba de error al añadir habilidad a un agente
Descripción	Comprobamos que no se añade una habilidad al agente si alguno no existe
Datos de entrada	Identificadores de agente y habilidad con alguno o ambos erróneos
Salida esperada	Error de petición en la adición
Resultado	Error de petición

Tabla 6.11: PM-11 (Prueba de error al añadir habilidad a un agente)

PM-12	Prueba de eliminar sensor de un agente
Descripción	Comprobamos que se elimina un sensor de un agente si la relación entre ambos existe y si los identificadores son correctos
Datos de entrada	Identificadores del agente y sensor correctos
Salida esperada	Tupla no encontrada en la BBDD
Resultado	Relación no encontrada en la BBDD

Tabla 6.12: PM-12 (Prueba de eliminar sensor de un agente)

PM-13	Prueba de error al eliminar sensor de un agente
Descripción	Comprobamos error de eliminación si algún identificador o la relación entre ellos no existe
Datos de entrada	Identificadores erróneos o sin relación
Salida esperada	Error de eliminación
Resultado	Error al tratar de eliminar sensor del agente

Tabla 6.13: PM-13 (Prueba de error al eliminar sensor de un agente)

PM-14	Prueba de eliminar habilidad de un agente
Descripción	Comprobamos que se elimina una habilidad de un agente si la relación entre ambos existe y si los identificadores son correctos
Datos de entrada	Identificadores del agente y habilidad correctos y con relación entre sí
Salida esperada	Tupla no encontrada en la BBDD
Resultado	Relación no encontrada en la BBDD

Tabla 6.14: PM-14 (Prueba de eliminar habilidad de un agente)

PM-15	Prueba de error al eliminar habilidad de un agente
Descripción	Comprobamos error de eliminación si algún identificador o la relación entre ellos no existe
Datos de entrada	Identificadores erróneos o sin relación
Salida esperada	Error de eliminación
Resultado	Error al tratar de eliminar habilidad del agente

Tabla 6.15: PM-15 (Prueba de error al eliminar habilidad de un agente)

PM-16	Prueba de listar agentes
Descripción	Comprobamos que se listan todos los agentes guardados
Datos de entrada	Ninguno
Salida esperada	Lista de agentes con sus datos
Resultado	Se muestran todos los agentes y los datos individuales

Tabla 6.16: PM-16 (Prueba de listar agentes)

PM-17	Prueba de crear sensor
Descripción	Comprobamos que se crea un sensor
Datos de entrada	Nombre y descripción del sensor
Salida esperada	Nuevo registro en BBDD coincidente
Resultado	Se crea una nueva fila en la BBDD con los datos dados

Tabla 6.17: PM-17 (Prueba de crear sensor)

PM-18	Prueba de obtener sensor
Descripción	Comprobamos que se puede consultar un sensor
Datos de entrada	Identificador de sensor existente
Salida esperada	Nombre y descripción del sensor
Resultado	Datos mostrados coincidentes con la BBDD

Tabla 6.18: PM-18 (Prueba de obtener sensor)

PM-19	Prueba de error al obtener sensor
Descripción	Comprobamos que no se puede consultar un sensor inexistente
Datos de entrada	Identificador de sensor inexistente
Salida esperada	Error al consultar
Resultado	Error por datos asociados no encontrados

Tabla 6.19: PM-19 (Prueba de error al obtener sensor)

PM-20	Prueba de modificar sensor
Descripción	Comprobamos que se puede modificar un sensor
Datos de entrada	Identificador de sensor existente con su nombre y/o descripción
Salida esperada	Datos modificados en BBDD
Resultado	Registro con id coincidente en BBDD actualizado

Tabla 6.20: PM-20 (Prueba de modificar sensor)

PM-21	Prueba de error al modificar sensor
Descripción	Comprobamos que no se puede modificar un sensor inexistente
Datos de entrada	Identificador erróneo
Salida esperada	Error de petición
Resultado	Error al intentar modificar

Tabla 6.21: PM-21 (Prueba de error al modificar sensor)

PM-22	Prueba de eliminar sensor
Descripción	Comprobamos que se puede eliminar un sensor
Datos de entrada	Identificador existente
Salida esperada	Tupla inexistente en BBDD
Resultado	Registro eliminado de la BBDD

Tabla 6.22: PM-22 (Prueba de eliminar sensor)

PM-23	Prueba de error al eliminar sensor
Descripción	Comprobamos que no se puede eliminar un sensor inexistente
Datos de entrada	Identificador inexistente
Salida esperada	Error por tupla inexistente en BBDD
Resultado	Error por registro no encontrado de la BBDD

Tabla 6.23: PM-23 (Prueba de error al eliminar sensor)

PM-24	Prueba de listar sensores
Descripción	Comprobamos que se pueden listar todos los sensores
Datos de entrada	Ninguno
Salida esperada	Sensores con sus datos asociados
Resultado	Sensores y sus datos coincidentes con la BBDD

Tabla 6.24: PM-24 (Prueba de listar sensores)

PM-25	Prueba de crear habilidad
Descripción	Comprobamos que se crea una habilidad
Datos de entrada	Nombre y descripción de la habilidad
Salida esperada	Nuevo registro en BBDD coincidente
Resultado	Se crea una nueva fila en la BBDD con los datos dados

Tabla 6.25: PM-25 (Prueba de crear habilidad)

PM-26	Prueba de obtener habilidad
Descripción	Comprobamos que se puede consultar una habilidad
Datos de entrada	Identificador de habilidad existente
Salida esperada	Nombre y descripción de la habilidad
Resultado	Datos mostrados coincidentes con la BBDD

Tabla 6.26: PM-26 (Prueba de obtener habilidad)

PM-27	Prueba de error al obtener habilidad
Descripción	Comprobamos que no se puede consultar una habilidad inexistente
Datos de entrada	Identificador de habilidad inexistente
Salida esperada	Error al consultar
Resultado	Erro por datos asociados no encontrados

Tabla 6.27: PM-27 (Prueba de error al obtener habilidad)

PM-28	Prueba de modificar habilidad
Descripción	Comprobamos que se puede modificar una habilidad
Datos de entrada	Identificador de habilidad existente con su nombre y/o descripción
Salida esperada	Datos modificados en BBDD
Resultado	Registro con id coincidente en BBDD actualizado

Tabla 6.28: PM-28 (Prueba de modificar habilidad)

PM-29	Prueba de error al modificar habilidad
Descripción	Comprobamos que no se puede modificar una habilidad inexistente
Datos de entrada	Identificador erróneo
Salida esperada	Error de petición
Resultado	Error al intentar modificar

Tabla 6.29: PM-29 (Prueba de error al modificar habilidad)

PM-30	Prueba de eliminar habilidad
Descripción	Comprobamos que se puede eliminar una habilidad
Datos de entrada	Identificador existente
Salida esperada	Tupla inexistente en BBDD
Resultado	Registro eliminado de la BBDD

Tabla 6.30: PM-30 (Prueba de eliminar habilidad)

PM-31	Prueba de error al eliminar habilidad
Descripción	Comprobamos que no se puede eliminar una habilidad inexistente
Datos de entrada	Identificador inexistente
Salida esperada	Error por tupla inexistente en BBDD
Resultado	Error por registro no encontrado de la BBDD

Tabla 6.31: PM-31 (Prueba de error al eliminar habilidad)

PM-32	Prueba de listar habilidades
Descripción	Comprobamos que se pueden listar todas las habilidades
Datos de entrada	Ninguno
Salida esperada	Habilidades con sus datos asociados
Resultado	Habilidades y sus datos coincidentes con la BBDD

Tabla 6.32: PM-32 (Prueba de listar habilidades)

PM-33	Prueba de crear mapa
Descripción	Comprobamos que se puede crear un mapa
Datos de entrada	Nombre, tamaño, obstáculos y sensores ligados a la partida como la energía
Salida esperada	Nueva tupla con estos datos asociados en BBDD
Resultado	Nuevo registro con los datos en BBDD

Tabla 6.33: PM-33 (Prueba de crear mapa)

PM-34	Prueba de obtener mapa
Descripción	Comprobamos que se puede obtener un mapa
Datos de entrada	Identificador de mapa existente en BBDD
Salida esperada	NOMBRE, tamaño, obstáculos y sensores ligados al mapa coincidentes con la BBDD
Resultado	Datos del mapa coincidentes con los datos del identificador en la BBDD

Tabla 6.34: PM-34 (Prueba de obtener mapa)

PM-35	Prueba de error al obtener mapa
Descripción	Comprobamos que no se puede obtener un mapa con un id incorrecto
Datos de entrada	Identificador de mapa inexistente en BBDD
Salida esperada	Error al obtener mapa
Resultado	Petición errónea de mapa

Tabla 6.35: PM-35 (Prueba de error al obtener mapa)

PM-36	Prueba de obtener información de un mapa
Descripción	Comprobamos que se pueden obtener características de un mapa existente
Datos de entrada	Identificador de mapa existente en BBDD
Salida esperada	NOMBRE, tamaño y sensores ligados al mapa coincidentes con los datos del identificador en la BBDD
Resultado	Información del mapa coincidente con los datos del identificador en la BBDD

Tabla 6.36: PM-36 (Prueba de obtener información del mapa)

PM-37	Prueba de error al obtener información de un mapa
Descripción	Comprobamos que no se puede obtener información de un mapa con id incorrecto
Datos de entrada	Identificador de mapa inexistente en BBDD
Salida esperada	Error al obtener mapa
Resultado	Petición errónea de mapa

Tabla 6.37: PM-37 (Prueba de error al obtener información de un mapa)

PM-38	Prueba de listar nombre de los mapas
Descripción	Comprobamos que se listan los nombres de los mapas
Datos de entrada	Ninguno
Salida esperada	Lista con nombre e identificador asociado a cada mapa
Resultado	Se muestran todos los mapas con su nombre e id coincidentes con la BBDD

Tabla 6.38: PM-38 (Prueba de listar nombre de los mapas)

PM-39	Prueba de listar obstáculos de un mapa
Descripción	Comprobamos que se listan todos los obstáculos de un mapa
Datos de entrada	Identificador de mapa existente en BBDD
Salida esperada	Lista de obstáculos con su tipo y coordenadas
Resultado	Se muestran todos los obstáculos con su tipo y coordenadas coincidentes con la BBDD

Tabla 6.39: PM-39 (Prueba de listar obstáculos de un mapa)

PM-40	Prueba de obtener obstáculos de mapa según posición
Descripción	Comprobamos que se listan todos los obstáculos de un mapa según la posición y distancia dadas
Datos de entrada	Identificador de mapa existente en BBDD, posición y distancia correctas
Salida esperada	Lista de obstáculos con su tipo y coordenadas en esa distancia
Resultado	Se muestran todos los obstáculos con su tipo y coordenadas coincidentes con la BBDD en esa distancia

Tabla 6.40: PM-40 (Prueba de obtener obstáculos de mapa según posición)

PM-41	Prueba de error al obtener obstáculos de mapa según posición
Descripción	Comprobamos que no se listan obstáculos con identificador de mapa incorrecto, posición fuera del mapa o distancia negativa
Datos de entrada	Identificador de mapa inexistente en BBDD, posición fuera del mapa o distancia incorrecta
Salida esperada	Error al listar obstáculos del mapa
Resultado	No se muestran los obstáculos por información dada incorrectamente

Tabla 6.41: PM-41 (Prueba de error al obtener obstáculos de mapa según posición)

PM-42	Prueba de crear usuario
Descripción	Comprobamos que se puede crear un usuario con datos correctos
Datos de entrada	Datos de usuario como dni, nombre, apellidos y subgrupo de la asignatura
Salida esperada	Nueva tupla en BBDD con los datos dados
Resultado	Nuevo registro en BBDD con los datos proporcionados

Tabla 6.42: PM-42 (Prueba de crear usuario)

PM-43	Prueba de obtener usuario
Descripción	Comprobamos que se puede obtener un usuario
Datos de entrada	Identificador de usuario
Salida esperada	Datos del usuario con ese identificador asociado
Resultado	Datos coincidentes con la tupla que tiene ese identificador en la BBDD

Tabla 6.43: PM-43 (Prueba de obtener usuario)

PM-44	Prueba de error al obtener usuario
Descripción	Comprobamos que no se puede obtener un usuario con id incorrecto
Datos de entrada	Identificador de usuario incorrecto
Salida esperada	Error por usuario incorrecto
Resultado	Error al no encontrar usuario

Tabla 6.44: PM-44 (Prueba de error al obtener usuario)

PM-45	Prueba de modificar usuario
Descripción	Comprobamos que se puede modificar un usuario
Datos de entrada	Identificador de usuario y datos a modificar
Salida esperada	Tupla con identificador coincidente en BBDD modificada
Resultado	Antiguo registro en BBDD con ese id actualizada

Tabla 6.45: PM-45 (Prueba de modificar usuario)

PM-46	Prueba de error al modificar usuario
Descripción	Comprobamos que no se puede modificar un usuario con id incorrecto
Datos de entrada	Identificador de usuario incorrecto
Salida esperada	Error por usuario incorrecto
Resultado	Error al no encontrar usuario

Tabla 6.46: PM-46 (Prueba de error al modificar usuario)

PM-47	Prueba de eliminar usuario
Descripción	Comprobamos que se puede eliminar un usuario
Datos de entrada	Identificador de usuario existente
Salida esperada	Tupla con identificador proporcionado inexistente
Resultado	Se ha eliminado la tupla asociada a ese identificador en la BBDD

Tabla 6.47: PM-47 (Prueba de eliminar usuario)

PM-48	Prueba de error al eliminar usuario
Descripción	Comprobamos que no se puede eliminar un usuario con id incorrecto
Datos de entrada	Identificador de usuario incorrecto
Salida esperada	Error por usuario incorrecto
Resultado	Error al no encontrar usuario

Tabla 6.48: PM-48 (Prueba de error al eliminar usuario)

PM-49	Prueba de crear sesión
Descripción	Comprobamos que se puede crear una sesión brindando datos correctos
Datos de entrada	Identificadores existentes de usuario, mapa y agente
Salida esperada	Nueva sesión creada en BBDD con los datos proporcionados
Resultado	Nuevo registro en BBDD con los datos proporcionados

Tabla 6.49: PM-49 (Prueba de crear sesión)

PM-50	Prueba de eliminar sesión
Descripción	Comprobamos que se pueda eliminar una sesión si esta no está activa
Datos de entrada	Identificador de sesión finalizada
Salida esperada	Registro de sesión eliminado de la BBDD
Resultado	Tupla con identificador utilizado eliminada de la BBDD

Tabla 6.50: PM-50 (Prueba de eliminar sesión)

PM-51	Prueba de error al eliminar sesión
Descripción	Comprobamos que no se pueda eliminar una sesión si está activa o es inexistente
Datos de entrada	Identificador de sesión activa o inexistente
Salida esperada	Error al borrar sesión
Resultado	No se puede eliminar la sesión

Tabla 6.51: PM-51 (Prueba de error al eliminar sesión)

PM-52	Prueba de consultar sesión
Descripción	Comprobamos que se pueda obtener una sesión
Datos de entrada	Identificador de sesión
Salida esperada	Los datos adjuntos a la sesión como el usuario, agente o sensores de partida
Resultado	Se muestran los datos de la sesión

Tabla 6.52: PM-52 (Prueba de consultar sesión)

PM-53	Prueba de error al consultar sesión
Descripción	Comprobamos que no se pueda obtener una sesión si el identificador es incorrecto
Datos de entrada	Identificador de sesión inválido
Salida esperada	Error de petición
Resultado	Error por identificador incorrecto

Tabla 6.53: PM-53 (Prueba de error al consultar sesión)

PM-54	Prueba de modificar sesión
Descripción	Comprobamos que se pueda modificar una sesión
Datos de entrada	Identificador de sesión y los datos a actualizar
Salida esperada	La tabla en la BBDD se actualiza
Resultado	Los datos en la BBDD son modificados

Tabla 6.54: PM-54 (Prueba de modificar sesión)

PM-55	Prueba de error al modificar sesión
Descripción	Comprobamos que no se pueda modificar una sesión si el identificador es incorrecto
Datos de entrada	Identificador de sesión inválido
Salida esperada	Error de petición
Resultado	Error por identificador incorrecto

Tabla 6.55: PM-55 (Prueba de error al modificar sesión)

PM-56	Prueba de realizar habilidad
Descripción	Comprobamos se puedan realizar acciones con el agente
Datos de entrada	Identificador de sesión y habilidad a realizar
Salida esperada	Se informa de si la habilidad a tenido éxito o no y en caso positivo se modifica la tupla de la sesión en la BBDD
Resultado	Si la habilidad tiene éxito se modifica el registro de la BBDD de la sesión y sino se nos informa de que la habilidad es inválida

Tabla 6.56: PM-56 (Prueba de realizar habilidad)

PM-57	Prueba de error al realizar habilidad
Descripción	Comprobamos no se puedan realizar acciones que no tiene el agente
Datos de entrada	Identificador de sesión y habilidad a realizar no válida
Salida esperada	Error de petición
Resultado	Se genera un error al realizar la habilidad

Tabla 6.57: PM-57 (Prueba de error al realizar habilidad)

PM-58	Prueba de eliminar mapa
Descripción	Comprobamos que se puede borrar un mapa
Datos de entrada	Identificador de mapa
Salida esperada	Mapa eliminado de la BBDD
Resultado	Registro eliminado de la BBDD

Tabla 6.58: PM-58 (Prueba de eliminar mapa)

PM-59	Prueba de error al eliminar mapa
Descripción	Comprobamos que no se pueda borrar un mapa inexistente
Datos de entrada	Identificador de mapa
Salida esperada	Error al realizar la petición
Resultado	Error en la petición

Tabla 6.59: PM-59 (Prueba de error al eliminar mapa)

PM-60	Prueba de finalizar sesión
Descripción	Comprobamos que la sesión finaliza tras conseguir objetivos o perder
Datos de entrada	Identificador de sesión
Salida esperada	Variable de actividad puesta como falsa en la BBDD
Resultado	Encontramos la sesión inactiva

Tabla 6.60: PM-60 (Prueba de finalizar sesión)

PM-61	Prueba de mostrar actualizaciones de los eventos
Descripción	Comprobamos que al actualizar una sesión esto se ve reflejado en el Visualizador
Datos de entrada	Acciones a tomar en una sesión
Salida esperada	Datos modificados en el Visualizador como la posición del agente o la energía coincidentes con la BBDD
Resultado	Los datos mostrados son idénticos a los de la BBDD

Tabla 6.61: PM-61 (Prueba de mostrar actualizaciones de los eventos)

Capítulo 7

Manual

Este apartado es una guía detallada para la instalación, configuración, uso y funcionamiento del laboratorio virtual creado en este proyecto.

7.1. Instalación

7.1.1. Requisitos previos

Para que el sistema funcione correctamente, son necesarios unos requisitos mínimos tanto de hardware como de software.

En cuanto a los **requisitos de hardware**, necesitamos un ordenador con estas características:

- Conexión estable a Internet.
- 4GB de memoria RAM.

Si hablamos de los **requisitos de software**, podemos destacar los siguientes elementos:

- Java 17 y JDK (Java Development Kit) 21 o superiores.
- Node 21 y Angular 17 o superiores
- MacOS, Windows o Linux.
- Docker Desktop o Docker Engine para los contenedores.
- Navegador web compatible con SSE y JavaScript.

7.1.2. Inventario de componentes

El sistema se puede dividir en los siguientes componentes:

- **Frontend:** el Visualizador que permite ver gráficamente la sesión de cada usuario.
- **Backend:** basado en dos microservicios llamados Entorno y Almacén, cruciales para el funcionamiento del sistema.

7.1.3. Procedimiento de configuración

Para empezar a utilizar nuestro programa, se deben seguir unos pasos de instalación que están enumerados en la siguiente lista:

1. **Descargar el proyecto:** al descargar el proyecto, encontramos dos carpetas principales: Frontend y Backend.
2. **Ejecutar demonio de Docker:** nos aseguramos de que Docker Desktop está abierto o de que el demonio de Docker está activado para evitar errores posteriores.
3. **Configurar los contenedores:** Dentro de la carpeta Backend, hay dos subcarpetas importantes: Store y Environment. Cada una contiene configuraciones para contenedores Docker en carpetas interiores llamadas Mongo y MySQL. Entrando en cada carpeta interior, debemos ejecutar el comando `docker compose build` para construir las imágenes de cada contenedor (3 en total).
4. **Configurar los microservicios:** Utilizamos Maven Wrapper (script que permite ejecutar Maven* sin necesidad de tenerlo instalado globalmente en el sistema) para evitar problemas de dependencias en Java. Dentro de cada microservicio (Store y Environment), simplemente ejecutamos el comando `./mvnw clean install` (`./mvnw.cmd clean install` en Windows) para instalar todas las dependencias necesarias.

* Maven es una herramienta de gestión de proyectos y construcción de software para proyectos basados en Java.

7.2. Uso

La plataforma de este proyecto tiene como fin el proporcionar un laboratorio virtual para sistemas multiagentes. Dicho sistema tiene las siguientes características:

- Gestionar usuarios, agentes, habilidades, sensores, mapas y sesiones.
- Ejecutar movimientos o habilidades y obtener el entorno mediante sensores.
- Visualizar sesiones en un mapa bidimensional.

Necesitaremos un software que nos permita enviar peticiones HTTP para utilizar nuestro sistema. En este manual, se utilizará Postman con este fin.

7.2.1. Arranque del sistema

Para que la plataforma ofrezca sus características, debemos ponerla en marcha. Descargamos el proyecto desde Google Drive o GitHub:

- <https://tinyurl.com/23jae6t8>.
- <https://github.com/smgarsan/TFG.git>

Luego, debemos ubicarnos en la carpeta raíz del proyecto (donde se encuentran el Frontend y el Backend) y acceder a las siguientes rutas:

- *Backend/Store/mongo*.
- *Backend/Store/mysql*.
- *Backend/Environment/mongo*.

Dentro de cada una de estas carpetas, ejecutamos el comando `docker compose up -d` para levantar los contenedores de las bases de datos en segundo plano.

Tras ello, dentro de cada carpeta de los microservicios almacenados en Backend (tanto Store como Environment), ejecutamos el comando `./mvnw spring-boot:run` que ejecutará cada microservicio.

El **Entorno** del sistema se expondrá en el puerto **8001** y el **Almacén** en el puerto **8080**. La puesta en marcha del componente Visualizador, se explicará mas adelante.

7.2.2. Gestión de Agentes, Habilidades y Sensores

Debido a su similitud en el uso, hemos agrupado las funcionalidades relacionadas con Agentes, Habilidades y Sensores. Todos estos componentes pueden ser creados, obtenidos, modificados y eliminados de manera similar. La única diferencia es el endpoint que se utiliza para cada tipo de componente. Además, están estrechamente interrelacionados, ya que las Habilidades y los Sensores son elementos fundamentales de cada Agente.

Así, pondremos de ejemplo de uso la manipulación de un Agente. Para **crear un Agente**, debemos realizar la siguiente petición:

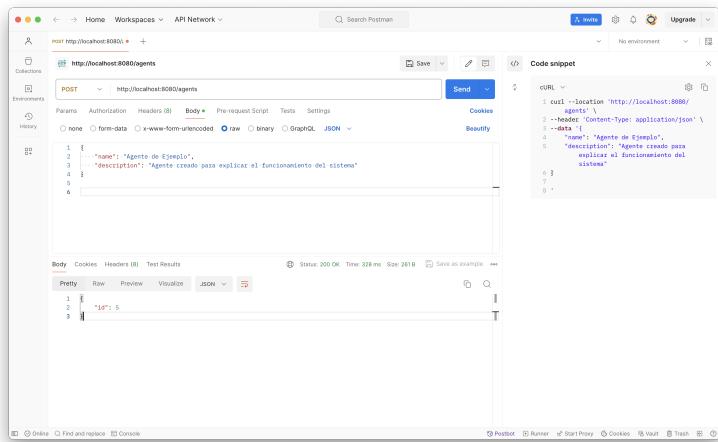
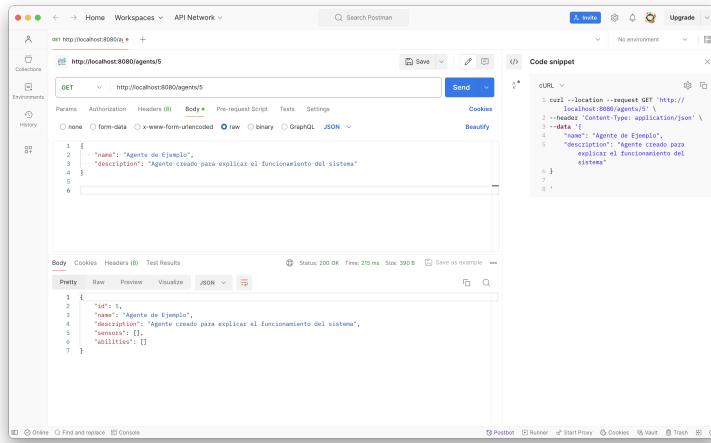


Figura 7.1: Manual - Crear Agente

Para entender Postman, vamos a ver la interfaz. En la parte superior nos encontramos con el tipo de solicitud, el endpoint utilizado y el cuerpo de la petición a enviar. Debajo, la respuesta del servidor a nuestra petición. Y, por último, vemos a la derecha la petición traducida a la biblioteca cURL de línea de comandos, que podemos utilizar desde la terminal, sin necesidad de descargar Postman.

Si queremos **obtener** los datos del agente, realizamos esta solicitud:



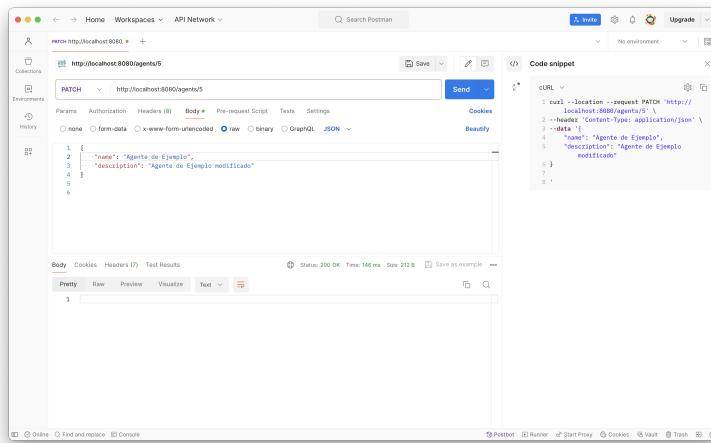
```
curl -X GET "http://localhost:8080/agents/5"
Content-Type: application/json
{
  "name": "Agente de Ejemplo",
  "description": "Agente creado para explicar el funcionamiento del sistema"
}
```

Body Headers Test Results Status: 200 OK Time: 275 ms Size: 390 B Save as example

```
{
  "id": 5,
  "name": "Agente de Ejemplo",
  "description": "Agente creado para explicar el funcionamiento del sistema",
  "secrets": [],
  "subtitles": []
}
```

Figura 7.2: Manual - Obtener Agente

Para **modificar** un Agente:



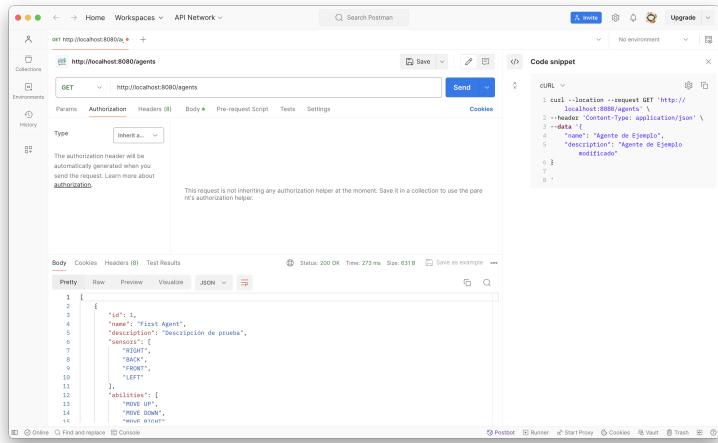
```
curl -X PATCH "http://localhost:8080/agents/5"
Content-Type: application/json
{
  "name": "Agente de Ejemplo",
  "description": "Agente de Ejemplo modificado"
}
```

Body Headers Test Results Status: 200 OK Time: 140 ms Size: 211 B Save as example

```
1
```

Figura 7.3: Manual - Modificar Agente

Podemos **listar** todos los agentes guardados:

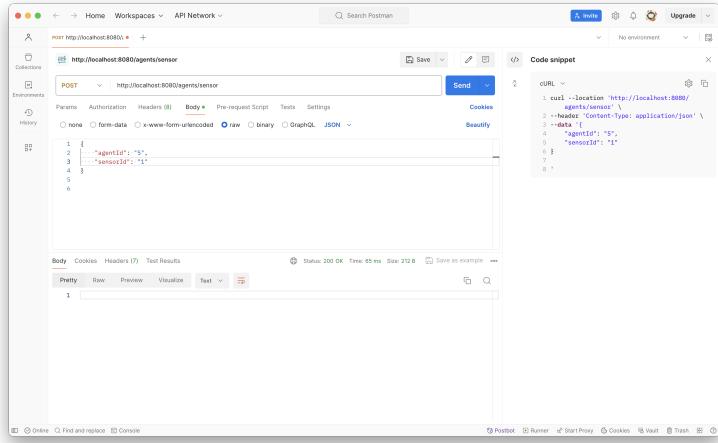


```
curl -X GET "http://localhost:8080/agents"
```

```
[{"id": 1, "name": "First Agent", "description": "Descripción de prueba", "sensors": [{"RIGHT": "RIGHT", "MIDDLE": "MIDDLE", "FRONT": "FRONT", "LEFT": "LEFT"}, {"abilities": [{"UP": "UP", "DOWN": "DOWN", "ARM": "ARMED"}]}]}
```

Figura 7.4: Manual - Listar Agentes

Además, es posible **añadir** a estos **sensores** o **habilidades** utilizando los identificadores:



```
curl -X POST "http://localhost:8080/agents/sensor"
```

```
{"agentId": "5", "sensorId": "1"}
```

Figura 7.5: Manual - Añadir Habilidad o Sensor al Agente

Por último, vemos cómo **eliminar** un Agente:

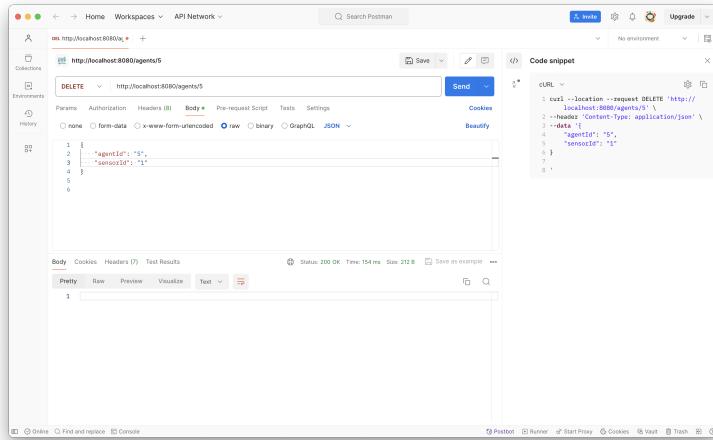


Figura 7.6: Manual - Eliminar Agente

7.2.3. Gestión de Mapas

Para establecer una sesión efectiva, resulta fundamental contar con un Mapa. Con el propósito de **crear** dicho mapa, procedemos de la siguiente manera:

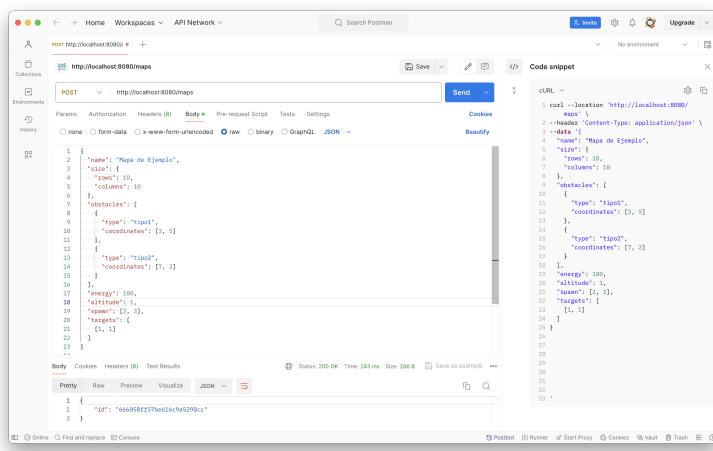
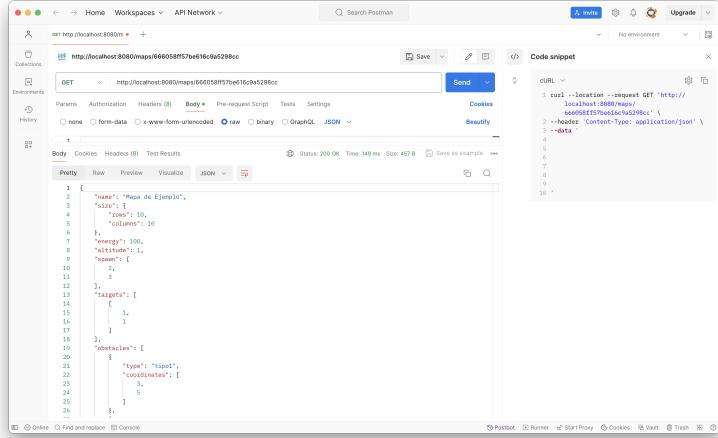


Figura 7.7: Manual - Crear Mapa

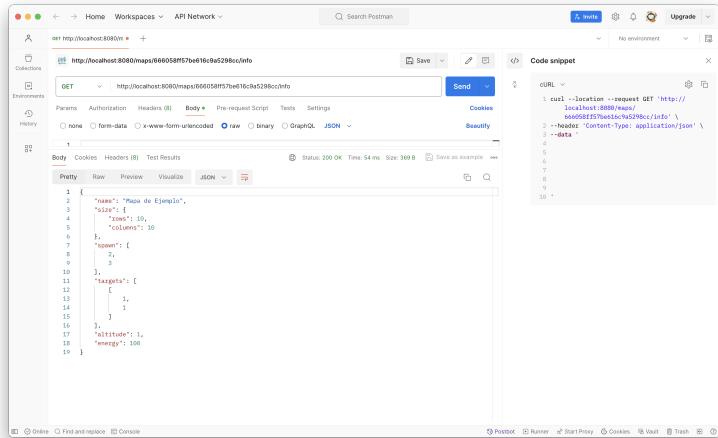
Con el identificador devuelto, podemos **obtener** el Mapa:



```
curl -X GET "http://localhost:8080/maps/66605ff57be616c9a5298cc" \n  -H "Content-Type: application/json" \n  -d "name=Mapa de Ejemplo"\n\n{\n    "name": "Mapa de Ejemplo",\n    "size": {\n        "x": 10,\n        "y": 10\n    },\n    "rows": 10,\n    "columns": 10,\n    "energy": 100,\n    "altitude": 1,\n    "spawn": {\n        "x": 0,\n        "y": 0\n    },\n    "targets": [\n        {\n            "x": 1,\n            "y": 1\n        }\n    ],\n    "obstacles": [\n        {\n            "type": "loot",\n            "coordinates": [\n                3,\n                9\n            ]\n        }\n    ]\n}
```

Figura 7.8: Manual - Obtener Mapa

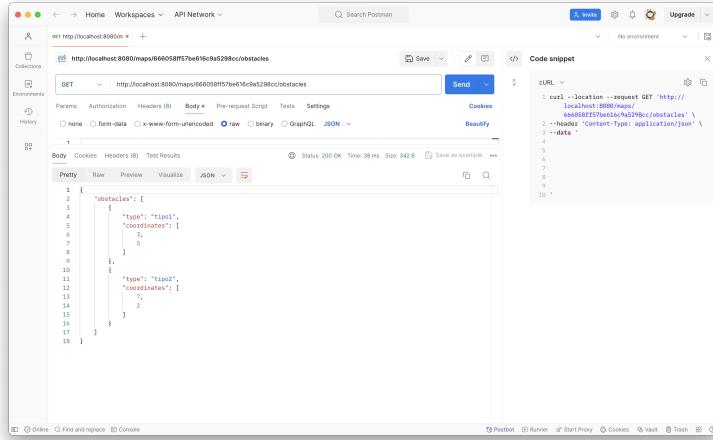
También es posible **obtener** únicamente la **información** del Mapa:



```
curl -X GET "http://localhost:8080/maps/66605ff57be616c9a5298cc/info" \n  -H "Content-Type: application/json" \n  -d "name=Mapa de Ejemplo"\n\n{\n    "name": "Mapa de Ejemplo",\n    "size": {\n        "x": 10,\n        "y": 10\n    },\n    "rows": 10,\n    "columns": 10,\n    "energy": 100,\n    "altitude": 1,\n    "spawn": {\n        "x": 0,\n        "y": 0\n    },\n    "targets": [\n        {\n            "x": 1,\n            "y": 1\n        }\n    ],\n    "obstacles": [\n        {\n            "type": "loot",\n            "coordinates": [\n                3,\n                9\n            ]\n        }\n    ]\n}
```

Figura 7.9: Manual - Obtener información del Mapa

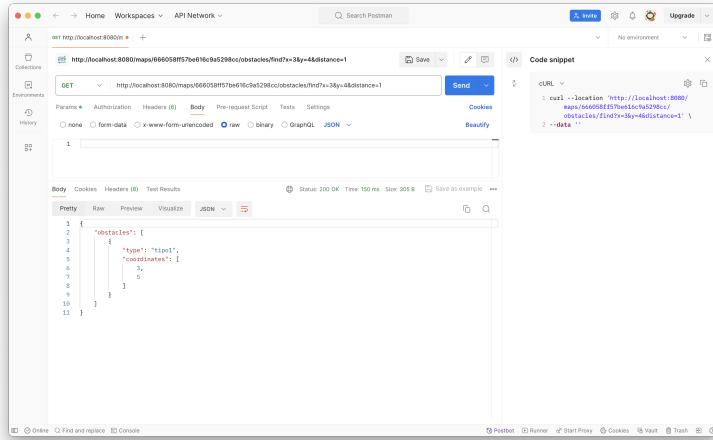
O por otro lado, **obtener los obstáculos** del mismo:



```
1 [ "obstacles": [ 2 { "type": "tipos1", "coordinates": [ 3 4 5 6 7 8 9 10 ], 11 }, { "type": "tipos2", "coordinates": [ 12 13 14 15 16 ] 17 } 18 ] ]
```

Figura 7.10: Manual - Obtener obstáculos del Mapa

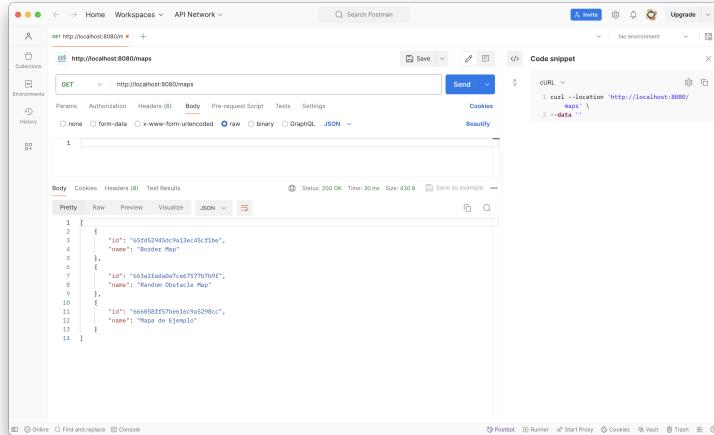
Dada una **posición** en el mapa, es posible **obtener los obstáculos** a cierta distancia:



```
1 [ "obstacles": [ 2 { "type": "tipos1", "coordinates": [ 3 4 5 6 7 8 9 10 ] 11 } 12 ] ]
```

Figura 7.11: Manual - Obtener obstáculos de un Mapa según posición

Se puede **listar** todos los **nombres** de los Mapas:

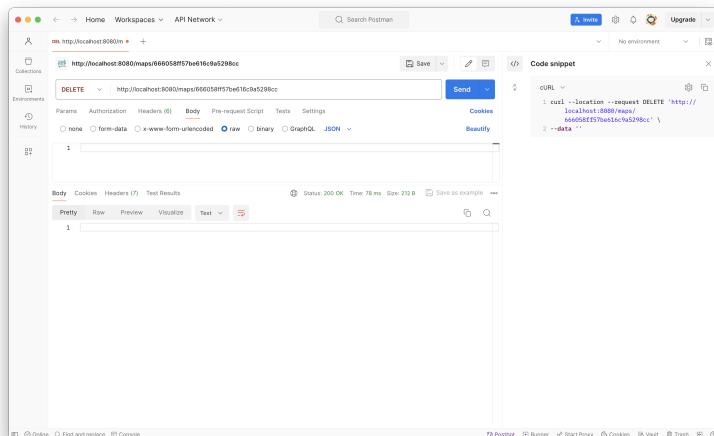


The screenshot shows the Postman application interface. A GET request is made to `http://localhost:8080/maps`. The response status is 200 OK, and the body is displayed in JSON format, listing several map names:

```
[{"id": "e5d032434569a13ec45f1be", "name": "Border Map"}, {"id": "e63b2f6a0a7c4e7579f9f", "name": "Random Objects Map"}, {"id": "e66d3f5757b0e11c9a5298cc", "name": "Mapa de Ejemplo"}]
```

Figura 7.12: Manual - Listar Mapas

Y, por último, **eliminar** el mapa deseado dado su identificador:



The screenshot shows the Postman application interface. A DELETE request is made to `http://localhost:8080/maps/e66d3f5757b0e11c9a5298cc`. The response status is 200 OK, indicating the map was successfully deleted.

Figura 7.13: Manual - Eliminar Mapa

7.2.4. Gestión de Usuarios

Los usuarios se pueden **crear** de esta manera:

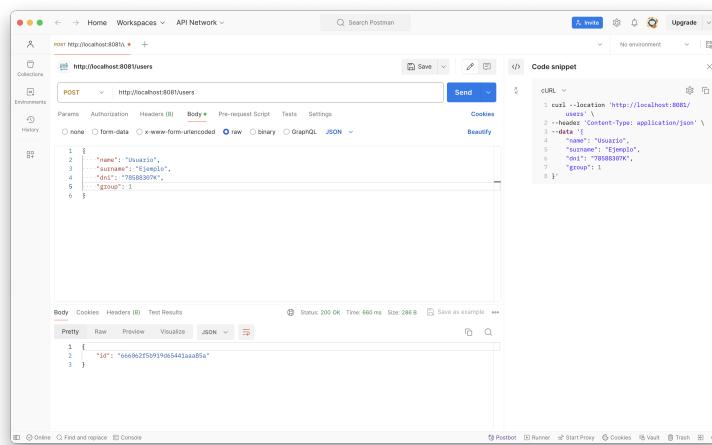


Figura 7.14: Manual - Crear Usuario

Se pueden **consultar** con su identificador:

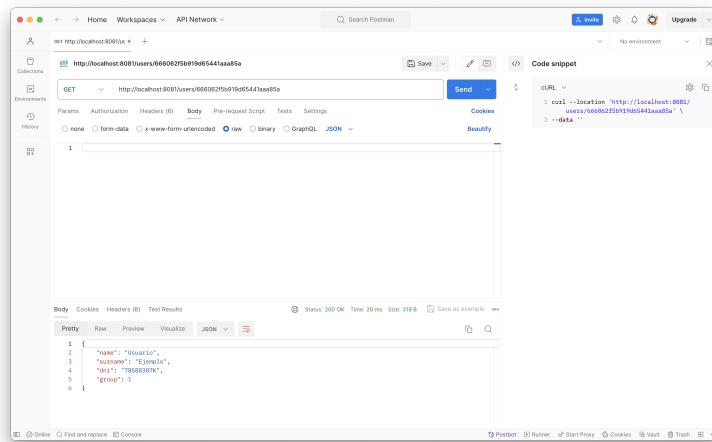
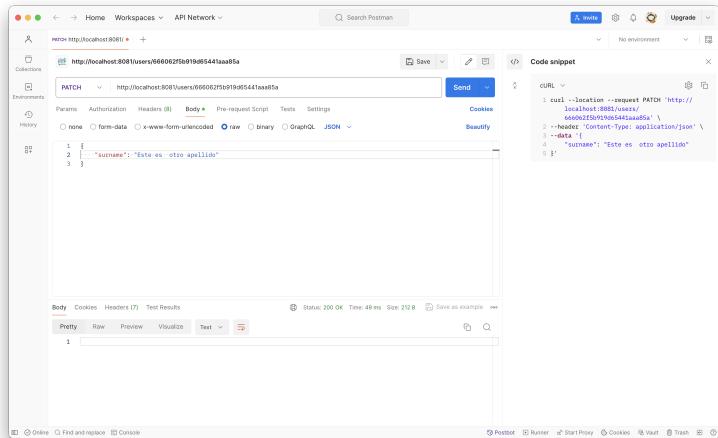


Figura 7.15: Manual - Obtener Usuario

Como en otros casos, se pueden **modificar**:



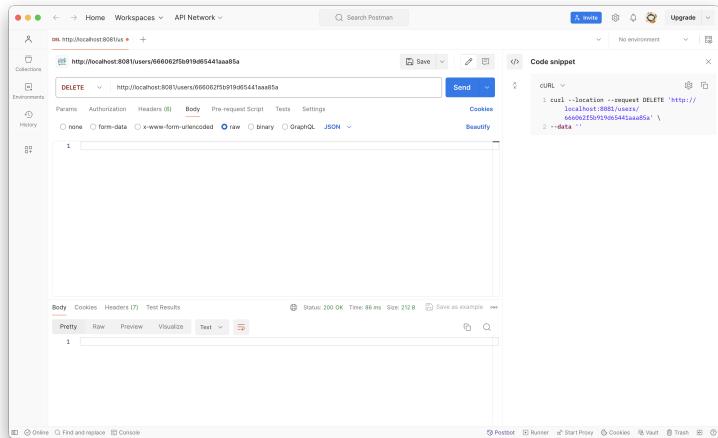
The screenshot shows the Postman application interface. A PATCH request is being made to the URL `http://localhost:8081/users/664062f0919d5441aaaf5a`. The request body is set to JSON and contains the following data:

```
1 {  
2   "surname": "Este es otro apellido"  
3 }
```

The response status is 200 OK, indicating successful modification.

Figura 7.16: Manual - Modificar Usuario

Y eliminar:



The screenshot shows the Postman application interface. A DELETE request is being made to the URL `http://localhost:8081/users/664062f0919d5441aaaf5a`. The request body is set to JSON and contains the following data:

```
1 {  
2 }
```

The response status is 200 OK, indicating successful deletion.

Figura 7.17: Manual - Eliminar Usuario

7.2.5. Gestión de Sesiones

Si lo que queremos es **crear** una Sesión, necesitaremos los identificadores del mapa, usuario y agente a utilizar:

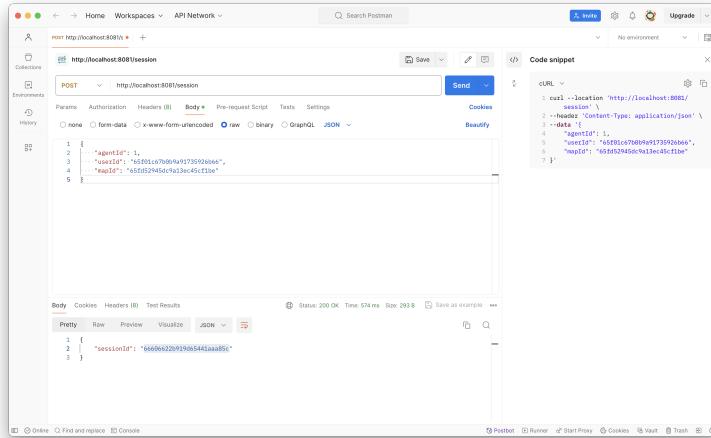


Figura 7.18: Manual - Crear sesión

Esta sesión puede ser **obtenida** de la siguiente manera:

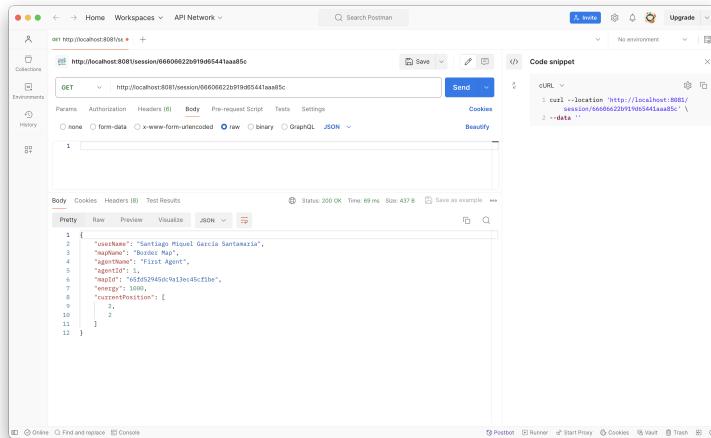


Figura 7.19: Manual - Obtener sesión

Teniendo en cuenta el tipo de Sensores con los que cuenta el Agente que hemos utilizado, podemos **obtener** datos de los **sensores** (obstáculos cercanos a la posición actual):

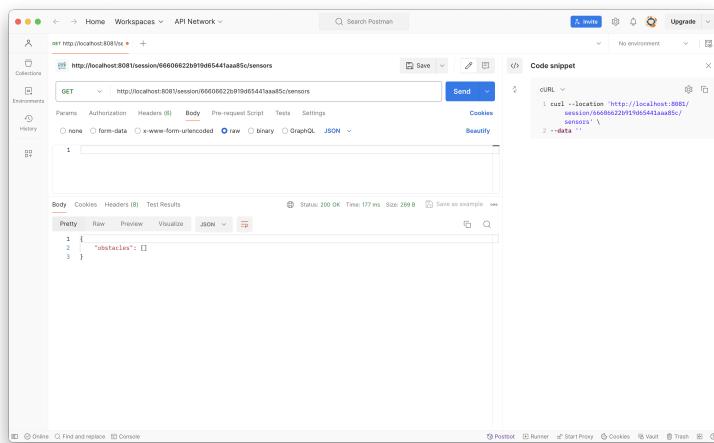


Figura 7.20: Manual - Obtener datos de los sensores

Y por último, el agente se puede **mover** en el Entorno según las habilidades que tenga:

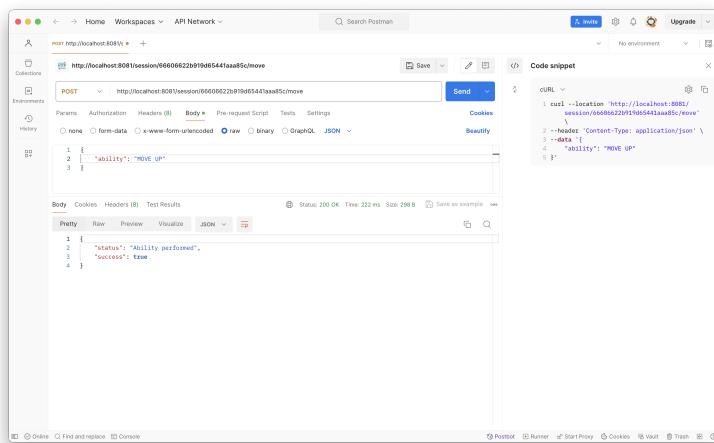


Figura 7.21: Manual - Realizar habilidad

7.2.6. Interacción práctica

A lo largo del manual, se han abordado múltiples aspectos de configuración de la aplicación, sin embargo, no se ha explorado la experiencia del usuario en su interacción práctica con la plataforma, especialmente desde la perspectiva de aquellos que buscan simplemente disfrutar del entorno de juego. En este apartado, veremos la rutina de uso del sistema.

En primer lugar, crearemos una sesión con nuestro usuario, el mapa que queremos utilizar y uno de los agentes proporcionados:

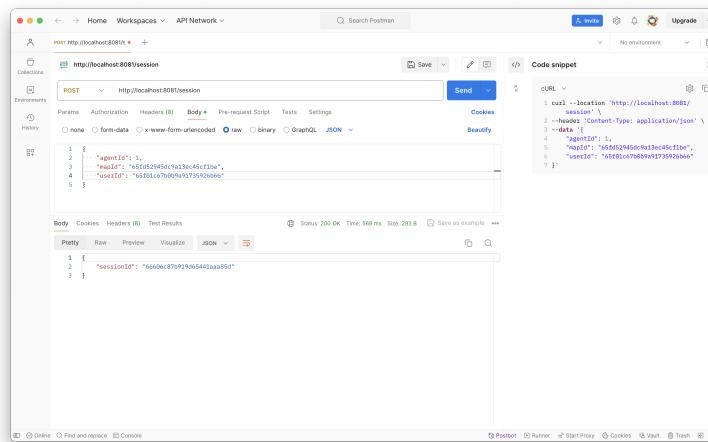


Figura 7.22: Interacción práctica - Crear sesión

Nos dirigiremos desde la raíz del proyecto a `Frontend/src/services/` y con un editor de código, modificamos el `sessionId` en el archivo `global.service.ts` con el valor del identificador devuelto.

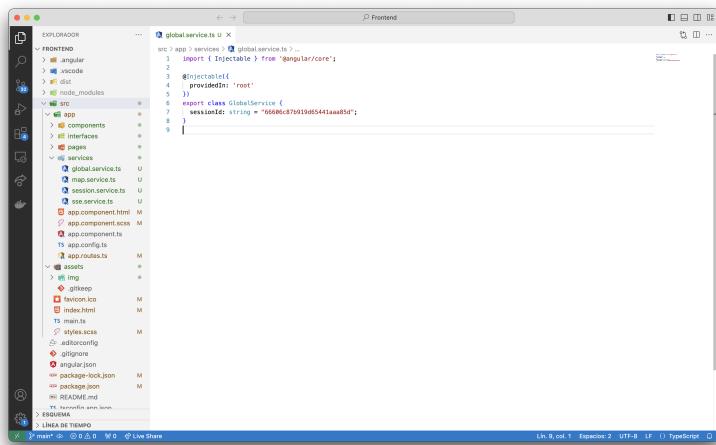


Figura 7.23: Interacción práctica - Configurar Visualizador

Desde una terminal en el interior de la carpeta Frontend, ejecutamos el comando `ng serve` para levantar el Visualizador en el puerto 4200. Accediendo a `http://localhost:4200/`: en un navegador veremos lo siguiente:

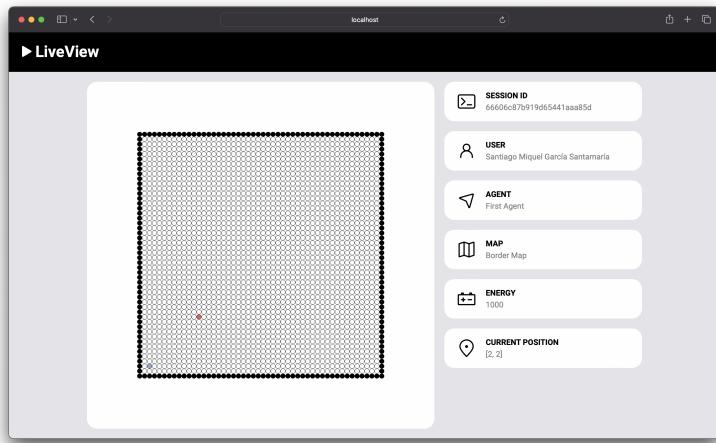


Figura 7.24: Interacción práctica - Visualizador

En el margen izquierdo se encuentra representado un mapa bidimensional, en el cual los objetivos se destacan en tonalidad roja, mientras que la ubicación del agente se muestra en un color más tenue. Asimismo, la presencia de obstáculos se indica mediante el color negro. En el margen derecho, se presentan datos relevantes como la posición actual del agente y el nivel de energía restante.

Ahora, podemos pedirle a los sensores que nos muestren los obstáculos que hay alrededor:

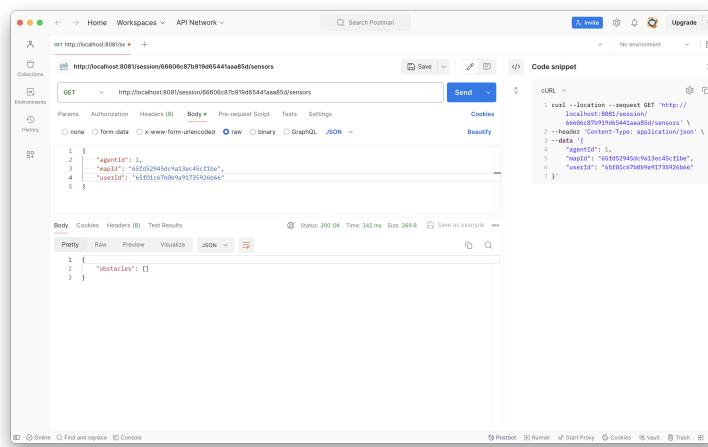


Figura 7.25: Interacción práctica - Observando obstáculos

El agente utilizado tiene sensores que muestra la información a una distancia de 1 unidad arriba, abajo y a ambos lados del agente, es por ello que la respuesta es una lista vacía.

Tenemos que movernos hacia el objetivo, así que realizamos una petición de movimiento:

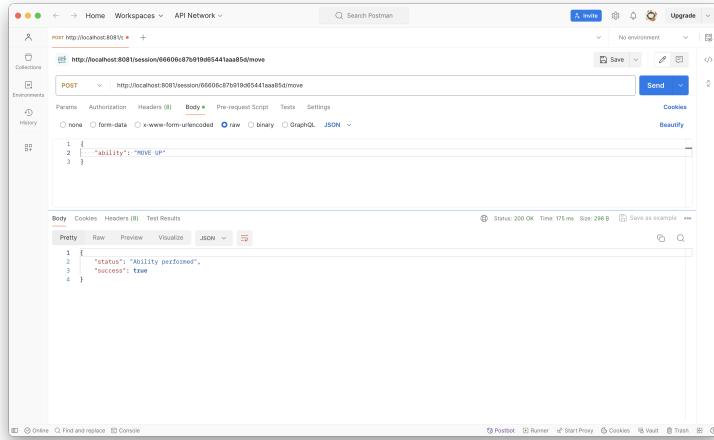


Figura 7.26: Interacción práctica - Petición de movimiento

Vemos que el Visualizador muestra el cambio en tiempo real:

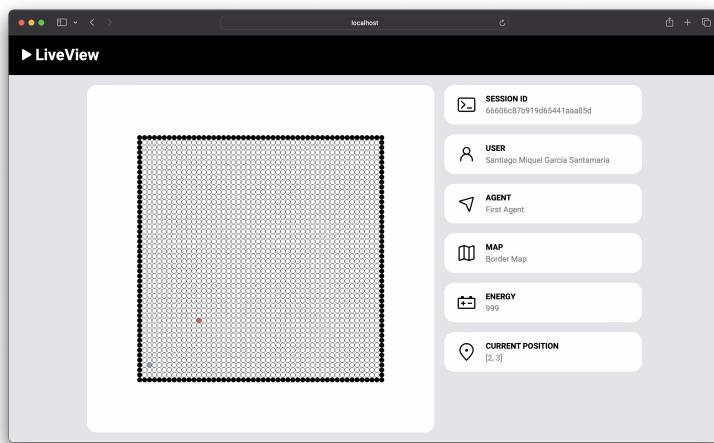


Figura 7.27: Interacción práctica - Actualización del Visualizador

Es notoria la disminución de la energía y el cambio de la posición actual del agente.

Cuando llegamos al objetivo, el servidor nos lo dirá y la sesión terminará:

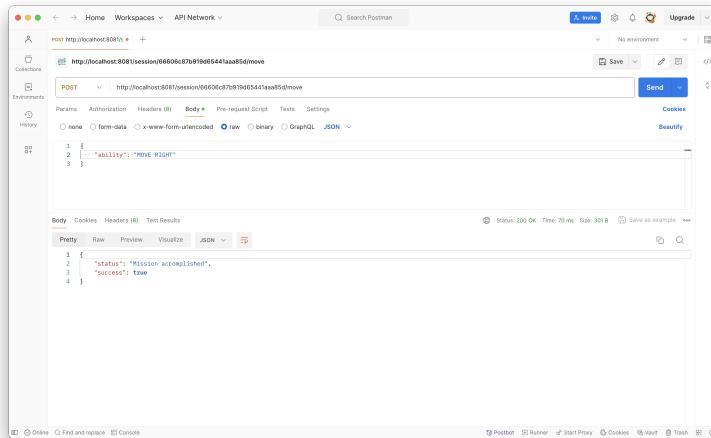


Figura 7.28: Interacción práctica - Objetivo alcanzado

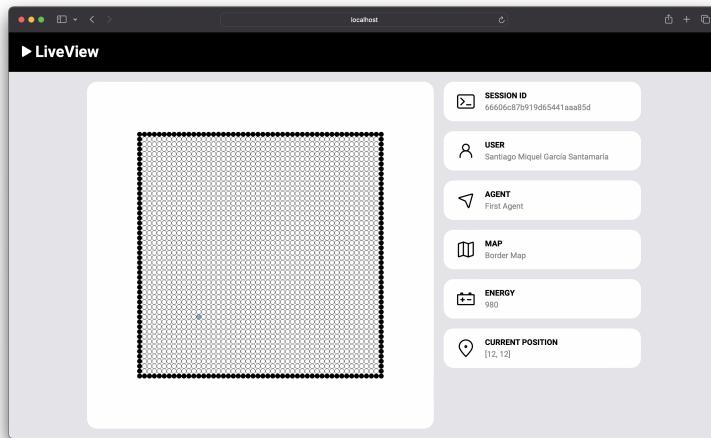


Figura 7.29: Interacción práctica - Visualizador con objetivo alcanzado

Capítulo 8

Conclusiones

Como análisis tras la finalización del proyecto, se pretende esbozar en este capítulo un resumen de los logros obtenidos y las experiencias acumuladas, sugiriendo además posibles vías futuras de investigación y mejora del proyecto.

Los objetivos mencionados en el primer capítulo han sido alcanzados en su totalidad. Se ha logrado una comprensión exhaustiva del funcionamiento de los sistemas multiagentes, lo cual ha sentado las bases para la ejecución exitosa del proyecto.

Además, se ha llevado a cabo un análisis profundo que ha servido de fundamento para el diseño e implementación del sistema, el cual se ha basado en microservicios exponiéndose mediante una API RESTful. El hecho de haber llevado a cabo todo el análisis y diseño del sistema ha representado un avance significativo en el conocimiento sobre el desarrollo de software. Los diagramas y patrones de diseño previamente conocidos se han fortalecido y se ha adquirido un dominio profundo de nuevos conceptos y técnicas.

Cada funcionalidad desarrollada en el laboratorio ha sido sometida a pruebas y validaciones, garantizando así su funcionamiento óptimo. Asimismo, a través de la elaboración de un manual detallado, se fomenta la utilización del sistema en contextos educativos relacionados con los sistemas multiagentes. Su fácil configuración y uso lo convierten en una herramienta idónea para la enseñanza en este campo.

El uso de tecnologías como Angular y Spring Boot ha proporcionado robustez, aunque con una curva de aprendizaje más pronunciada. Aun así, el potencial que ofrecen es inmenso. Aunque hoy en día pueden parecer alternativas anticuadas debido al auge de otras herramientas, Angular y Spring Boot siguen siendo ampliamente utilizados tanto para mantener sistemas ya existentes como para el desarrollo de nuevos proyectos.

Cabe destacar el uso de Docker, que debido a su capacidad para simplificar el proceso de desarrollo, se ha vuelto fundamental en el desarrollo de software moderno y en este trabajo.

En cuanto a posibles vías futuras de investigación y mejoras del proyecto, se destacan varias áreas claves. En primer lugar, se podría considerar la mejora en la creación de los mapas dentro del sistema, incrementando su complejidad. Otra área de mejora potencial sería la implementación de un sistema de notificaciones para informar a los usuarios sobre el estado del juego y otros eventos relevantes mediante plataformas de mensajería como Telegram. Además, se podría explorar la posibilidad de agregar soporte para múltiples jugadores en una misma partida. Por último, el desarrollo de un Visualizador más completo podría mejorar significativamente la experiencia de uso, permitiendo el uso de la mayoría de peticiones directamente desde una interfaz gráfica, convirtiendo la aplicación en una solución web integral.

En el contexto actual, con la inteligencia artificial en pleno crecimiento y convirtiéndose en una tendencia predominante, hemos desarrollado una herramienta que no solo cumple con los objetivos propuestos, sino que podría servir para entrenar modelos de IA. Este enfoque asegura que nuestro sistema no sea relevante únicamente hoy, sino que también tenga aplicaciones prácticas en el futuro.

Bibliografía

- [1] Bellifemine, Fabio; Caire, Giovanni; Greenwood, Dominic (2007). Developing Multi-Agent Systems with JADE. John Wiley & Sons, Ltd.
- [2] Roldán Martínez, David (2018). Microservicios: un enfoque integrado. Madrid: Ra-Ma.
- [3] Debrauwer, Laurent (2018). Patrones de diseño en Java. Los 23 modelos de diseño. Barcelona : ENI.
- [4] Gough, James; Bryant, Daniel; Auburn, Matthew (2022). Mastering API Architecture. O'Reilly Media, Inc.
- [5] Martin, Robert C. (2019). Clean Agile: Back to Basics. Pearson.
- [6] Rubin, Kenneth S. (2012). Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley Professional.
- [7] Wooldridge, Michael (2009). An Introduction to MultiAgent Systems, 2nd Edition. Wiley.
- [8] Russell, Stuart; Norvig, Peter (2021). Artificial Intelligence: A Modern Approach, 4th Edition. Pearson Series.
- [9] Oracle. Java. [En línea] <https://www.java.com/es/>
- [10] Microsoft. TypeScript. [En línea] <https://www.typescriptlang.org/>
- [11] VMWare Tanzu. Spring. [En línea] <https://spring.io/>
- [12] Google. Angular. [En línea] <https://angular.dev/>
- [13] Oracle. MySQL. [En línea] <https://www.mysql.com/>
- [14] Mongo DB, Inc. MongoDB. [En línea] <https://www.mongodb.com/>
- [15] Docker, Inc. Docker. [En línea] <https://www.docker.com/>
- [16] JetBrains. IntelliJ IDEA. [En línea] <https://www.jetbrains.com/idea/>

- [17] Microsoft. VS Code. [En línea] <https://code.visualstudio.com/>
- [18] Postman, Inc. Postman. [En línea] <https://www.postman.com/>
- [19] Atlassian. Draw.io [En línea] <https://app.diagrams.net/>
- [20] Overleaf. [En línea] <https://es.overleaf.com/>
- [21] LaTeX. [En línea] <https://www.latex-project.org/>
- [22] Google Drive. [En línea] <https://drive.google.com/>