UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DEL SOFTWARE

ENTORNO PARA LA SIMULACIÓN Y PREVENCIÓN DE DELITOS
ENVIRONMENT FOR CRIME SIMULATION AND PREVENTION

Realizado por
VICTORIA CARACUEL CASTRO

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJE Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2021

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

ENTORNO PARA LA SIMULACIÓN Y PREVENCIÓN DE DELITOS

ENVIRONMENT FOR CRIME SIMULATION AND PREVENTION

Realizado por
VICTORIA CARACUEL CASTRO

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJE Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2021

Fecha defensa: julio de 2021

Resumen

En este Trabajo Final de Grado se utilizan el Modelado y la Simulación basados en Agentes, para construir un entorno web que permite realizar simulaciones con el objetivo estudiar y analizar los delitos que se pueden cometer en una zona urbana. La aplicación permite modelar diferentes zonas policiales, dentro de ese escenario de simulación, así como simular el comportamiento de delincuentes y víctimas potenciales. La simulación se lleva a cabo a partir de diversos parámetros de entrada, introducidos por el usuario a través de un archivo JSON, en los que se modelan diversos aspectos como el nivel de delincuencia, la población y agentes policiales relativos a cada una de las zonas mencionadas anteriormente.

Los resultados de las simulaciones se almacenan en una base de datos para que el usuario que las genera pueda realizar estudios a posteriori de los resultados. La aplicación permite al usuario extraer los datos de los delitos (potenciales y cometidos) en formato CSV, así como visualizarlo mediante gráficos, a través de la propia interfaz de la aplicación, lo que podría ayudar a la toma de decisiones sobre aspectos tan importantes como la distribución de efectivos policiales en una zona urbana.

Para el desarrollo del entorno web de simulación, construido en este trabajo, se ha diseñado una arquitectura basada en microservicios. El backend que ofrece estos

microservicios ha sido implementado en MESA, un framework Python para el desarrollo de aplicaciones basadas en agentes, combinado con la tecnología Flask para la construcción de los microservicios. Para el almacenamiento y gestión de las diferentes simulaciones se ha utilizado la base de datos no relacional, MongoDB. Finalmente se ha implementado un frontend en Angular que facilita a los usuarios la construcción y parametrización de los escenarios de simulación, así como el acceso y visualización de los resultados de las simulaciones.

Palabras clave:
Modelado y Simulación basada en Agentes; Prevención de Delitos; Microservicios; Flask; Angular;

Abstract

In this Final Degree Project, Agent-based Modeling and Simulation Is used to build a web environment that allows doing simulations to study and analyze the crimes that can be committed in an urban area. The application allows modelling different police zones, within this simulation scenario, as well as simulate the behavior of criminals and potential victims. The simulation is carried out with some input parameters, entered by the user through a JSON file, in which various aspects are modeled such as the level of crime, the population and police officers relative to each of the areas mentioned above.

The results of the simulations are stored in a database so that the user who generates them can carry out and later study the results. The application allows the user to extract the crime data (potential and committed) in CSV format, as well as visualize it through plots, through the application's interface, which could help decision-making on aspects as significant as distribution of police officers in an urban area.

For the development of the simulation web environment, built in this work, an architecture based on microservices has been designed. The backend that offers these microservices has been implemented in MESA, a Python framework for the development of agent-based applications, combined with Flask technology for the construction of the microservices. For the storage and management of the different simulations, the non-relational database, MongoDB, has been used. Finally, the

frontend has been implemented in Angular (that makes it easier for users to build and parameterize the simulation scenarios, as well as access and display the results of the simulations).

Keywords:
Agent-based Modeling and Simulation; Crime Prevention; Microservices; Flask; Angular;

Índice

1

Introducción

1.1 Motivación y objetivos

Hoy en día, se utlizan grandes cantidades de datos para su estudio a través de técnicas tan diversas como el Aprendizaje Automático (*Machine Learning*) o el Modelado y Simulación basada en Agentes. Gracias a esto, podemos obtener conclusiones y predicciones recreando escenarios complejos, de una forma que los humanos no seríamos capaces de inferir directamente.

Mediante el uso de estas predicciones, podemos adelantarnos a los sucesos y tomar mejores decisiones. Estos avances tecnológicos nos pueden permitir progresar en ámbitos tan importantes como la medicina.

Uno de los ámbitos de aplicación del modelado y simulación basada en agentes es recrear escenarios urbanos a través de los cuales se pueden prevenir delitos. El objetivo de este Trabajo Final de Grado es implementar un sistema que sea capaz de simular los delitos que se producen en las ciudades, para así poder prevenir desgracias,

anticipándonos a los posibles conflictos y minimizando el número de delitos según la zona. Estas simulaciones pueden contribuir a decidir cuál podría ser la gestión óptima de los recursos policiales.

Para poder lograr esto, trabajaremos con parámetros relativos a la zona de actuación, como la población, los delincuentes, el número de policías y su forma de actuar, etc.

Una vez realizado el proceso de simulación, tendremos acceso a los datos de los delitos generados en el sistema (tanto los potenciales como los ejecutados), así como a gráficos que nos permiten ver de forma más visual el efecto de la delincuencia en cada una de las zonas policiales.

1.2 Estructura de la memoria

La memoria fruto de este Trabajo Final de Grado está compuesta por 8 capítulos:

- **Capítulo 1: Introducción.**

  En el capítulo de introducción definimos la motivación y objetivos del proyecto, así como el estudio del arte del Modelado y Simulación basada en Agentes

- **Capítulo 2: Tecnologías utilizadas**

  En este capítulo se expondrán los principales lenguajes y *frameworks* utilizados para hacer el trabajo, así como la justificación de la elección de cada uno de ellos.

- **Capítulo 3: Metodología de desarrollo y herramientas utilizadas**

  En el capítulo 3 se definirán los métodos que se han seguido para organizar el trabajo a realizar, la forma en la que se han organizado los repositorios de código

(incluida la gestión de ramas durante su desarrollo) y cómo se han gestionado los despliegues del sistema a un entorno de producción, explicando las herramientas utilizadas para ello. Otro de los temas a tratar en este punto es cómo se han organizado las fases de desarrollo del proyecto.

- **Capítulo 4: Análisis y especificación de Requisitos.**

  El capítulo 4 enumera los requisitos del sistema desarrollado, tanto funcionales como no funcionales y expone la forma en la que se han elaborado los casos de uso y prueba de los requisitos funcionales.

- **Capítulo 5: Modelado y diseño del sistema**

  En este capítulo, se narra la arquitectura del sistema, tras realizar una comparación de las arquitecturas microservicios y monolíticas. Además, indicamos cuál ha sido la estructura que se ha seguido para almacenar los datos y por último, explicamos cómo se ha modelado del sistema que simula los delitos.

- **Capítulo 6: Implementación**

  En el apartado 6, explicamos los aspectos de la implementación que pueden generar más intereses y exponemos el software y librerías utilizadas para el desarrollo del proyecto.

- **Capítulo 7: Mantenimiento y Pruebas**

  En el capítulo 7 comentaremos cómo se ha llevado a cabo la fase de pruebas en cada una de las iteraciones del proyecto e indicaremos cómo podríamos afrontar la fase de mantenimiento de cara al futuro.

- **Capítulo 8: Conclusiones**

  En el capítulo 8, el último de la memoria, se explicarán cuáles han sido los resultados obtenidos al final el proyecto, los conocimientos adquiridos y dificultades encontradas durante su desarrollo, así como, las mejoras y líneas futuras que se pueden seguir.

- **Apéndice A: Manual de usuario**

  En este apéndice, incluimos el manual de usuario realizado.

- **Apéndice B: Bocetos**

  En el apéndice B hemos añadido los bocetos iniciales de las interfaces de la aplicación, realizados a mano con la herramienta GoodNotes 5.

- **Apéndice C: Diseño de Interfaces con Figma**

  En el apéndice C están los diseños finales realizados con Figma.

- **Apéndice D: Casos de uso y Casos de Prueba**

  En este apéndice D se pueden encontrar los casos de uso y casos de prueba de los requisitos, redactados durante la fase de Análisis y Especificación de Requisitos.

- **Apéndice E: Pruebas Realizadas**

  En el apéndice E aparecen las tablas que hemos utilizado para llevar un control de las pruebas realizadas que están asociadas a los casos de pruebas del "Apéndice D".

- **Apéndice F: Horas de Tareas Realizadas**

  En el apéndice F están incluidas las tablas sobre las que se recogen las horas que se han asignado a cada tarea de GitLab.

1.3 Estudio del arte

El modelado basado en agentes (*Agent-based Modeling*, *ABM*), es una técnica que nos permite realizar simulaciones de entornos complejos. Para conseguir esto, debemos modelar los diferentes agentes que forman el sistema y las relaciones entre ellos, además de cómo se comportan e interactúan con el entorno. Su comportamiento podría estar definido incluso por algoritmos genéticos o redes neuronales.

Gracias a esta técnica, podemos estudiar el comportamiento global del sistema a partir de ciertos datos de entrada (viendo su evolución a través del paso de las distintas iteraciones o pasos), pudiendo capturar las diferentes interacciones entre los agentes y las consecuencias y efectos de ello. Esto nos permite poder analizar con posterioridad estos datos y tomar decisiones teniendo en cuenta los resultados.

El uso de ABM nos proporciona una flexibilidad que no la podríamos tener si usamos los métodos tradicionales de programación, ya que el sistema siempre permitirá añadir más agentes sobre el modelo. Los agentes serán autónomos y tendrán un estado que irá variando con el tiempo (éste está compuesto por las diferentes variables que lo forman).

ABM resulta muy útil en el caso de que las relaciones entre las entidades que forman el entorno de simulación sean complejas o no lineales, debido a que programar de forma tradicionales todas las interacciones, reglas de comportamientos, toma de decisiones o estados no sería viable, y una vez desarrollado el sistema, apenas sería escalable.

Actualmente, gracias a los grandes avances que se han realizado en el campo de la ciencia de datos, el ABM se puede combinar con el uso de grandes cantidades de datos para poder validar el modelo y que así se adapte más a la realidad que pretendemos simular, y por lo tanto, tendremos resultados más fiables.

2

Tecnologías utilizadas

En este apartado, vamos a explicar y tratar las principales tecnologías que se han usado para desarrollar el trabajo, argumentando su uso respecto al resto de opciones que ofrece el mercado.

2.1 MESA

Como el proyecto se centra en realizar la programación de la simulación de los delitos en un entorno urbano, a través del ABM, una de las decisiones más importante era seleccionar la herramienta para poder realizarla.

Durante el periodo de investigación y toma de decisiones sobre las tecnologías, se encontraron varias herramientas que permiten realizar esta tarea, como puede ser Netlogo, JADE, MASON y Repast, pero había una de ellas que se adaptaba

perfectamente a lo que estábamos buscando: MESA. Se trata de un framework que está construido sobre Python y que facilita al programador la labor de desarrollar un sistema multiagente.

A día de hoy Python es un lenguaje muy demandado en diferentes campos de la ingeniería del software e informática, por lo tanto, se vio que podía ser una buena opción aprovechar este trabajo para poder adentrarnos en él.

Las principales ventajas que nos ofrece MESA son:

- Gran cantidad de documentación en su página oficial, así como de tutoriales. Esto permitió poder adquirir conocimiento de una forma rápida, pudiendo realizar modelos sencillos en un corto periodo de tiempo (de hecho, antes de tomar la decisión definitiva sobre su uso, se pudo realizar la simulación del tutorial, lo que ayudó en gran parte a decantarse por él).
- Muchos ejemplos en su repositorio de Github, además actualizados con frecuencia (incluso en el tiempo que se ha desarrollado este proyecto se ha podido ver cómo esta biblioteca de ejemplo iba en aumento).
- Una gran comunidad en los foros, que resuelven las dudas.
- Es un framework nuevo que está en desarrollo y crecimiento, y en poco tiempo ha conseguido ser usado en muchos desarrollos. Se ha considerado que dentro de unos años va a tener mucha más importancia.
- Ofrece las herramientas necesarias para poder visualizar la simulación en un navegador. En nuestro caso, esto es un punto muy a favor, ya que consideramos que era importante poder visualizar cómo interactuaban los diferentes agentes en el "mapa" de la ciudad. Además, gracias a que está muy bien estructurada esta parte dentro del código del framework, ha sido muy sencillo poder adaptar la visualización del entorno de la simulación a nuestro proyecto.

- Permite utilizar planificadores para determinar el orden en el que realizarán las acciones cada uno de los agentes.
- Durante los "pasos" de la simulación, permite almacenar datos, para posteriormente analizarlos usando librerías como Pandas.

Otro de los frameworks que nos encontramos en Python para realizar simulaciones se trata de SPADE, pero éste está centrado en la simulación de mensajes.

2.2 Angular

A pesar de que la curva de aprendizaje de este framework se considera mayor que para aprender Vuejs o la librería React, se ha decidido utilizarlo porque ya se tenía algo de experiencia previa en su uso en los meses previos al inicio del proyecto, y por lo tanto lo que quedaba por aprender eran aspectos de optimización. Esto, ha permitido poder realizar funcionalidades que hubiera sido imposible hacerlo con las otras dos opciones, ya que se partía de cero.

Además, aunque en muchas ocasiones los desarrolladores se decanten por las otras opciones por la poca flexibilidad que puede ofrecer Angular a la hora de desarrollar, esto lo vemos como un punto a favor, ya que en muchas ocasiones un exceso de flexibilidad puede llevar a tener un código bastante sucio e insostenible y más aún cuando se está aprendiendo la tecnología a medida que se está desarrollando el proyecto. Respecto al lenguaje, preferimos el uso de TypeScript frente a Javascript, ya que nos permite detectar o evitar muchos errores durante la fase de la programación y no al compilar y aunque en Vuejs y React se puede utilizar TypeScript, hay que realizar procesos extras de instalación, mientras que en Angular ya lo trae por defecto.

2.3 Flask

Aunque se ha utilizado una arquitectura microservicios, lo cual ofrece flexibilidad a la hora de elegir diferentes tecnologías en cada uno de éstos, ya que son independientes, se ha optado por usar Python en todos, ya que, como se ha comentado con anterioridad, la librería con la que se ha planteado el proyecto para hacer la simulación se trata de MESA, y está hecha para Python. Por lo tanto, para no tener que lidiar con varios lenguajes (además de los usados en el lado del cliente) y poder estandarizar la creación de microservicios, se ha utilizado el mismo. Pero esto no impide que, en el futuro, al desarrollar nuevas funcionalidades, se decida abarcar otros lenguajes.

Una vez que se toma la decisión de utilizar Python, el debate se centra en decidir cuál de los dos frameworks más utilizados para el desarrollo web puede adaptarse mejor a la solución que planteamos: Flask o Django. Django es más complejo y podría ser la mejor opción en el caso de que backend y frontend estén juntos (como, por ejemplo, en las aplicaciones desarrolladas en otras tecnologías como Java Server Faces), pero como para este proyecto sólo lo vamos a usar para desarrollar APIs RESTful, podemos aprovechar que Flask es más ligero y rápido. Además, otro punto a favor es que Flask está de moda en el ámbito de los microservicios y lo están los gigantes tecnológicos, como puede ser Netflix y su servicio de streaming.

2.4 MongoDB: MongoEngine

Se ha elegido utilizar una base de datos no relacional, MongoDB, para almacenar los datos atendiendo a variar razones:

- La mayoría de los servicios de alojamiento en la nube obligan a usar bases de datos no relacionales. En este caso, como queríamos que la aplicación pudiese ser accesible desde cualquier ordenador, era un punto imprescindible. Además, MongoDB Atlas ofrece un gran servicio y tiene un nivel gratuito con bastantes

ventajas (a día de hoy podemos tener una capacidad de 500MB de forma completamente gratuita, algo que es suficiente para lo que queremos realizar).

- Flexibilidad. A la hora de desarrollar una aplicación de forma iterativa y que va sumando funcionalidades y requisitos a lo largo del tiempo, es muy importante que la base de datos sea flexible y que no sea dependiente de esquemas que hemos definido en los primeros ciclos de su vida.
- Escalabilidad: nos permite realizar un escalado horizontal. Ya que dentro de un clúster, podemos desplegar un numero variable de nodos según las necesidades del sistema.
- Velocidad: ofrecen una gran velocidad a la hora de realizar consultas en grandes cantidades de datos.

Para poder interactuar con nuestra base de datos, había que buscar la librería que mejor se adaptase a nuestras necesidades.

PyMongo es lo que nos recomienda la página oficial de MongoDB. Pero al no ser un ORM, no se pueden generar modelos, por lo tanto, se complica la validación de cada uno de los campos introducidos en la base de datos. Además, que, a la hora de hacer consultas o modificaciones en la base de datos, podemos sufrir "no sql injection" si no tenemos cuidado, ya que estamos utilizando una forma más primitiva de realizar este proceso. Al utilizar ORMs evitamos este problema.

Por otro lado, tenemos Flask-MongoAlchemy y Flask-MongoEngine. Al realizar búsquedas de información, nos encontramos que el primero de ellos, Flask-MongoAlchemy, es el que menos recursos presenta, además de que se trata de un ORM menos completo. Por otro lado, Flask-MongoEngine nos permite introducir una gran variedad de validaciones de los campos a la hora de generar los modelos. Aun así, cabe destacar que ambos son muy similares y que por lo tanto las diferencias son mínimas.

Por lo expuesto anteriormente, se eligió MongoEngine.

3

Metodología de desarrollo y herramientas utilizadas

En este apartado, vamos a explicar la metodología que se ha utilizada para abordar el desarrollo del proyecto, así como las herramientas que se han usado con este fin.

3.1 Metodología

3.1.1 Gitlab

Para poder desarrollar el proyecto llevando un control de las tareas realizadas y de versiones del código, se ha utilizado el servicio web GitLab. Se ha elegido GitLab frente a otros servicios porque la capa gratuita que ofrece nos permite:

- Tener en cada proyecto, de forma independiente, un control de versiones, tareas, etc.
- Tener una gran flexibilidad a la hora de crear "issues" o tareas, pudiendo asignarle a cada una de ellas las etiquetas que queramos, documentación, e incluso controlar el tiempo que se ha tardado en completarla. Además de eso, podrá tener asociados elementos como otras tareas, Pull Request o *commits* (en el caso de los *commits*, lo podremos hacer añadiendo *#idIssue* al inicio del mensaje del *commit* realizado).
- Tener un control sobre las distintas ramas y las *Pull Request* que se realizan a ellas.
- Añadir procesos de CI/CD.
- Organizar los distintos proyectos en un grupo, por lo que podemos tener en un solo tablero todas las tareas creadas en los distintos proyectos, lo que permite trabajar de un modo mucho más visual a la hora de cambiar el estado de las tareas.

Dentro del grupo creado para la realización de este Trabajo de Fin de Grado, denominado "TFG-Victoria", hay cinco proyectos. En cada uno de ellos, exceptuando el designado a las tareas de diseño, se combinan tareas de configuración, cursos o formación, despliegues, investigación, programación, etc. Además del propio repositorio para el control de versiones del código. En la siguiente imagen podemos ver los proyectos generados:

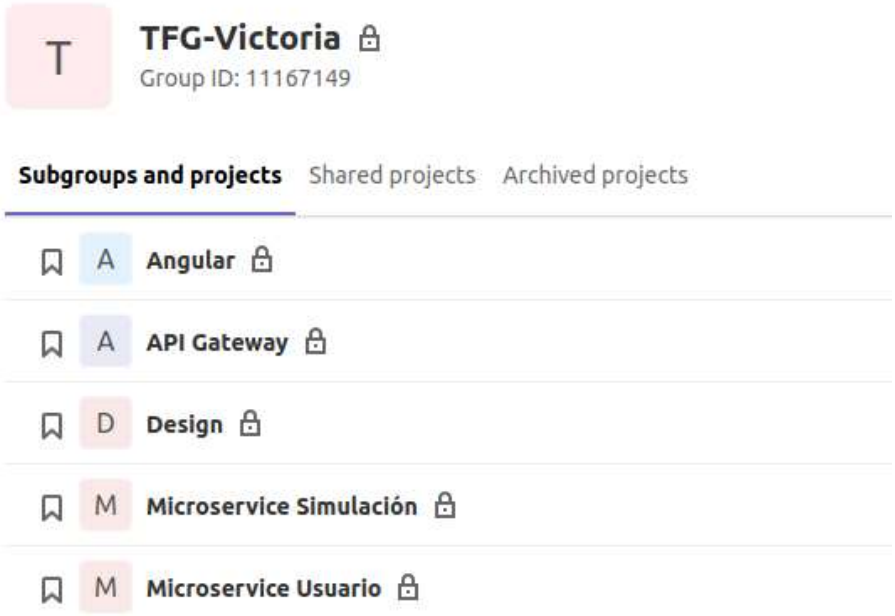


Figura 3.1 Proyectos que forman el grupo TFG-Victoria.

- Angular: se desarrolla la aplicación web (parte del cliente) y todo lo relativo a su despliegue.
- API Gateway: proyecto encargado del desarrollo del microservicio con el que se comunica el proyecto de Angular.
- Design: este proyecto se utiliza para tener organizada la documentación, como puede ser el listado de requisitos, bocetos, diseño de interfaces, casos de uso, casos de prueba, modelos, etc.
- Microservice Simulación: en él se realiza el proceso de la simulación y están los endpoints relativos a la gestión de éstas, como puede ser el borrado, creación, obtención de documentos, etc.
- Microservice Usuario: funcionalidades como el inicio de sesión o la gestión de los datos del perfil del usuario.

Para gestionar el estado de las tareas, se ha creado un tablero en el grupo "TFG-Victoria" que incluye las tareas generadas en los cinco proyectos:

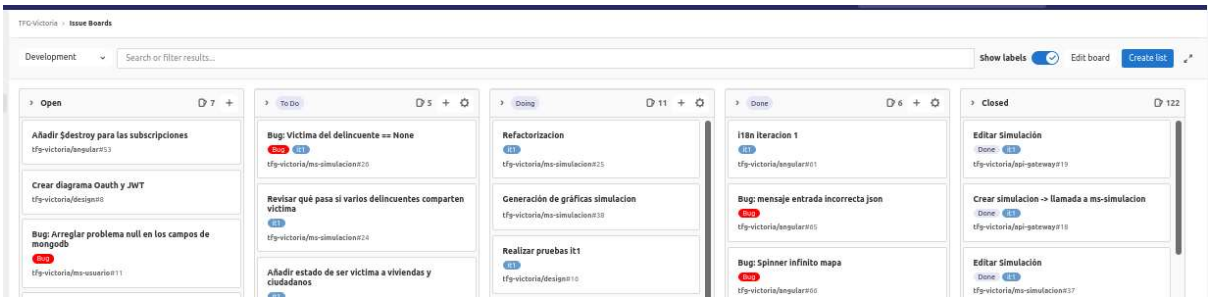


Figura 3.1 Tablero principal de los proyectos en GitLab.

Además, como se puede observar en las imágenes, también se han creado varias etiquetas que nos permiten identificar el estado que tienen:

- *To Do*: son tareas que aún no han comenzado a realizarse y que deberían ser empezadas en un corto periodo de tiempo, es decir, las candidatas a pasar al estado *Doing*.
- *Doing*: tareas en ejecución.
- *Done*: pasamos las tareas a este estado cuando consideramos que ya están completamente acabadas (en el caso de que sea una tarea que requiera programación, el código debe estar subido a la rama correspondiente del desarrollo de las funcionalidades que se están desarrollando y que además funcione correctamente).

Además de estos estados, Gitlab incluye otros dos, y también se les ha dado un uso:

- *Open*: simplemente quiere decir que la tarea ha sido creada.
- *Closed*: cuando cerramos una tarea, podemos decir que definitivamente está terminada. Si cuando esté en *Done*, se considera que realmente no es lo que se esperaba, se pasará de *Done* a *Doing*, en lugar de cerrarla.

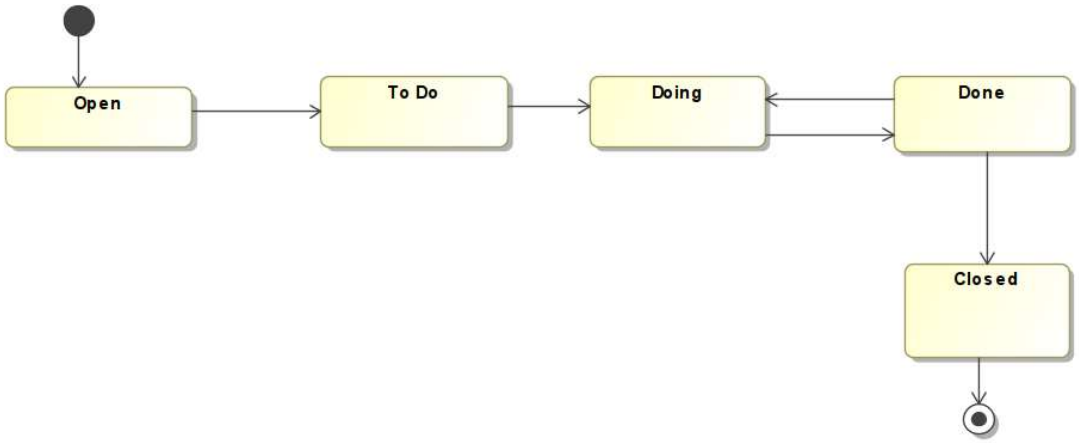


Figura 3.3 Diagrama de estados de las tareas.

Además de las etiquetas que nos indican el estado actual de las tareas, se han creado otras que también han sido bastantes útiles:

- Bug: gracias a ella, mirando el tablero, podemos ver rápidamente qué tareas tienen como fin resolver un bug.
- it0, it1, it2: nos indican la iteración en la que se ha realizado. Con esto podremos analizar cómo se han distribuido las tareas una vez que el proyecto ha finalizado.

Cada vez que se dedica tiempo a una tarea, al acabar ese periodo de tiempo, éste se añadirá a la tarea (añadiendo un comentario desde la interfaz web con el siguiente texto entrecomillado: "/spend Xm", siendo X el número de minutos). Gracias a esto, nos aparecerá a la derecha el tiempo acumulado que le hemos dedicado.

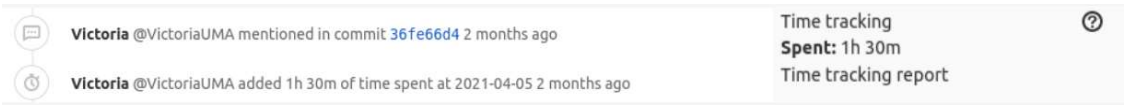


Figura 3.4 Control del tiempo en las tareas de GitLab.

Esto se ha aprovechado para llevar un control de las horas de trabajo realizadas. Como GitLab no nos permite ver de forma global las horas dedicadas a un proyecto y hay que entrar en cada una de las tareas, la solución ofrecida para ello ha sido que, al pasar

a Done cada una de las tareas, como hay que entrar a ella y revisar que todo sea correcto, se anota en un archivo Excel el tiempo total. Esto se puede visualizar en el Apéndice F.

Gracias a la generación de este documento, durante el desarrollo se han podido realizar estimaciones sobre el tiempo que quedaba de trabajo, ya que teníamos que adaptarnos a las 296 horas asignadas a este proyecto, y así hemos podido adaptar el trabajo de las distintas iteraciones. Además, esta información puede ser muy útil a la hora de plantear un proyecto similar, debido a que podremos realizar estimaciones partiendo de datos reales (esto no se ha podido utilizar como beneficio para el trabajo actual, ya que no disponíamos de registros previos), por lo que serán más fiables.

3.1.2 Estrategia de ramas

Durante el desarrollo del proyecto, en los repositorios que se han utilizado para construir el código que forma parte de la solución, se ha seguido una estrategia específica de ramas, para tener control sobre el código y sus distintas versiones. Aunque este proyecto está realizado por una sola persona, es una buena práctica utilizar diferentes ramas a la hora de desarrollar.

Procedemos a explicar los diferentes tipos de ramas utilizados:

- master: en ella se encuentra el código que está desplegado en el entorno de producción. Esta rama está bloqueada, por lo que la única forma de introducir código en ella es a través de la aceptación de *Pull Requests*.
- release: en esta rama se está el código que próximamente va a subirse a la rama master. Además, de aquí se sacan ramas bug_<idTareaBug> y hotfix_<idTareaBug>.

- feature_X: rama específica de la implementación de una funcionalidad o característica. Un ejemplo sería feature_perfil. Estas ramas se crean a partir de lo que haya en la rama release.
- develop: rama auxiliar que se usa para hacer *merge* de varias ramas *feature* antes de integrarlas con lo que haya en *release*. Haciendo esto, evitamos generar problemas en *release* a la hora de fusionar el código de varias ramas, ya que *release* se encuentra muy próxima a master.
- bug_<idTareaBug>: rama temporal creada para solucionar los bugs que se hayan detectado cuando ya se ha integrado la rama feature_x y no estén en el entorno de producción.
- hotfix_<idTareaBug>: rama temporal creada para solucionar los bugs que se encuentren desplegados en el entorno de producción. Al solucionar el bug e integrarlo en *release*, se procederá a hacer un *merge* de *release* en master para subir la corrección al fallo que existía en producción.

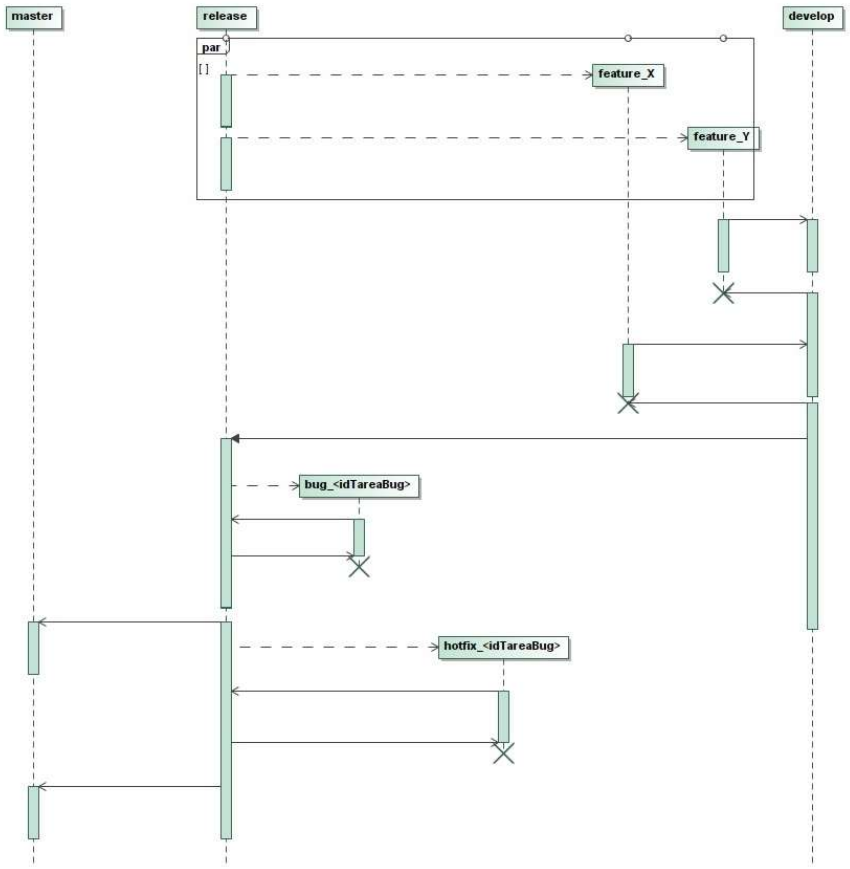


Figura 3.5 Diagrama de secuencias que nos permite observar la estrategia de ramas.

3.1.3 Gestión de despliegues

Una de las desventajas del uso de la arquitectura microservicios es la cantidad de servicios que hay que desplegar para tener el sistema accesible desde cualquier lugar. Por esta razón se ha optado por automatizar este proceso. De esta forma , cada vez que se acepte una *Pull Request* que tenga como destino la rama master (en nuestro caso el origen siempre será release), se producirá el despliegue de la nueva versión del software. Para ello, se ha hecho uso del CD/CI de Gitlab.

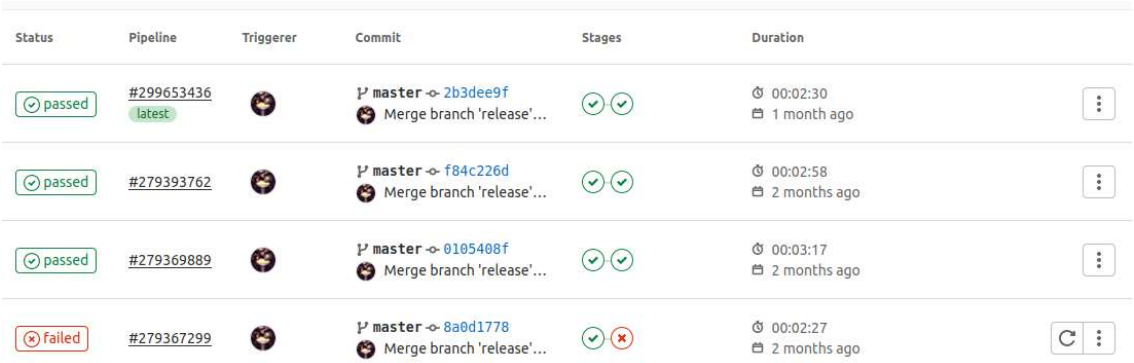


Figura 3.6 Ejemplos de algunos de los despliegues realizados usando GitLab.

Heroku:

Se ha utilizado Heroku para desplegar los servicios desarrollados usando Flask. Se he elegido este proveedor PaaS (*Platform as a Service*) porque ofrece una capa gratuita que es suficiente para el proyecto que estamos realizando. Algunas de las características que obtenemos de con esta cuota gratuita:

- 550 horas de "dynos"(contenedor Linux donde se despliega la aplicación que hayamos indicado) por mes, pudiendo repartirlas entre todas las aplicaciones que tengamos. Si se añade la tarjeta de crédito, aunque sigamos en la tarifa gratuita, aumenta a 1000 horas.
- Dominios personalizados
- Permite despliegue usando Git y Docker (además del básico que es usando la terminal con la cuenta de Heroku)

Existen varios tipos de "dynos", en nuestro caso, como se trata de una aplicación web, es el denominado "web". Hay otros, como puede ser el caso del tipo "worker", que mantiene la aplicación despierta todo el tiempo (esto se usa, por ejemplo, para desarrollar Chat Bots). Al usar los del tipo "web", nuestras aplicaciones tendrán los siguientes estados:

- Despierta: podemos acceder a la aplicación e interactuar con ella. Tras una hora sin interaccionar, pasará al estado “Durmiendo”.
- Durmiendo: la aplicación se encuentra inactiva, y al usarla tardará unos 10 -15 segundos en responder la primera vez y pasará al estado “Despierta”.
- Apagada: la aplicación no va a responder (aunque pasen los 10-15 segundos mencionados anteriormente). Para estar en este estado, o está desactivada o no tiene código.

Para poder realizar el despliegue con éxito ha sido necesario incluir varios archivos en el repositorio de GitLab:

- Procfile: debemos indicar, por cada tipo de “dyno”, el comando que debe ejecutar para iniciar la aplicación. Por ejemplo, si tuviéramos un bot realizado con Node.js, en el archivo aparecería “worker: npm start”. Para las aplicaciones que estamos desarrollando en Flask es “web: gunicorn app:app”
- Runtime.txt: en este archivo se indica la versión de Python a utilizar (debe ser una de las soportadas por el stack que estemos utilizando de Heroku). Esto se puede ver en la web https://devcenter.heroku.com/articles/stack#stack-support-details
- Requirements.txt: este archivo es necesario porque estamos ejecutando aplicaciones desarrolladas en Python, y por lo tanto, debemos indicar las dependencias que se deben instalar para poder ejecutar el código.
- .gitlab-ci.yml: se trata de un archivo en el que indicamos la forma en la que se debe instalar y desplegar el software, cuáles son los ficheros de dependencias, la rama que va a actuar como “trigger” para el despliegue, es decir, la que va a activar el despliegue al aceptar una Pull Request sobre ella y los datos de la aplicación de Heroku sobre la que vamos a actuar.

.gitlab-ci.yml 384 Bytes Edit Web IDE

```yaml
build:
  stage: build
  script:
    - apt-get update -qy
    - apt-get install -y python3-dev python3-pip
    - pip3 install -r requirements.txt
  only:
    - master

deploy:
  stage: deploy
  script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
    - dpl --provider=heroku --app=$HEROKU_APP_NAME --api-key=$HEROKU_SECRET_KEY
  only:
    - master
```

Figura 3.7 Archivo con extensión .yaml utilizado para desplegar los microservicios Flask.

En la imagen anterior, podemos ver cómo aparece las variables $HEROKU_APP_NAME y $HEROKU_SECRET_KEY, que son variables que hemos introducido en el repositorio de GitLab (en Ajustes → CI/CD).

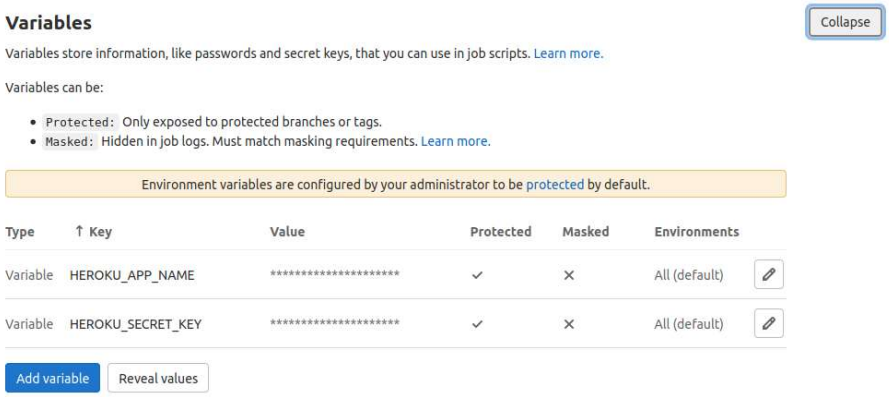


Figura 3.8 Localización de las variables introducidas en el archivo .yaml

Netlify:

Usamos sus servicios de alojamiento para desplegar la aplicación de Angular.

Para establecer la conexión entre el repositorio de GitLab y Netlify, simplemente habrá que crear el sitio web usando el botón “New site from Git” y seguir los pasos. En el

campo “Build command”, debemos introducir: ng build –prod y para saber qué hay que poner en “Publish directory”, debemos ejecutar el comando que hemos indicado antes en nuestra máquina y ver en qué ruta se generan los archivos.

Vamos a ver el procedimiento que hay que seguir para que funcione correctamente el proyecto desplegado, ya que si no seguimos estos pasos, cuando el usuario refresque la ventana del navegador, no podrá acceder a la ruta en la que se encontraba anteriormente, por lo tanto, tampoco será posible acceder mediante URL a cualquiera de las páginas que no sean la principal.

En el archivo angular.json, debemos añadir la ruta del archivo que vamos a crear (“src/_redirects”) en la lista de la propiedad “assets”.

Tras hacer esto, creamos ese archivo en la raíz de la carpeta “src”. Simplemente debemos escribir esta línea: /* /index.html 200

3.2 Fases del trabajo

Para desarrollar este proyecto, se ha usado un ciclo de vida iterativo e incremental, realizando tres iteraciones. Además, cada 2-3 semanas, se han realizado reuniones con el tutor, para mostrarle lo que se había desarrollado hasta el momento y poder así decidir con qué se iba a continuar en las próximas semanas.

Sólo las dos primeras iteraciones han sido de desarrollo y en ellas se han seguido distintas fases, de forma metodológica, que se adaptan a los procesos realizado durante la ingeniería del software.

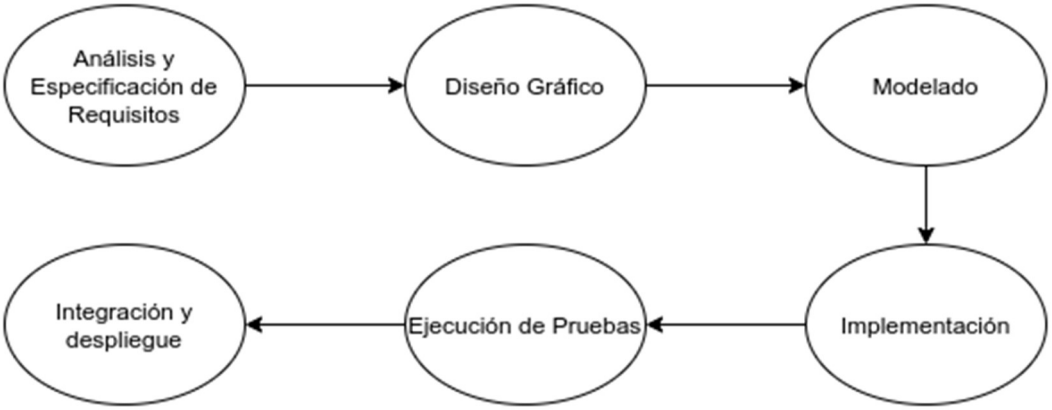


Figura 3.9 Fases realizadas en las iteraciones de desarrollo.

1. **Análisis y especificación de requisitos:**

   Toda la documentación generada en esta fase se ha realizado usando el formato markdown (.md) y se ha subido al repositorio del proyecto de diseño. Esto nos permite poder llevar una trazabilidad de cómo han ido evolucionando los requisitos a lo largo del desarrollo, porque, como ya sabemos, los requisitos no son inmutables y evolucionan a lo largo del tiempo. En esta fase, se han realizado distintos procesos:

   - Definir el listado de requisitos funcionales que se van a implementar durante la iteración. Para seleccionar estos requisitos, se han tenido en cuenta las reuniones realizadas con el tutor.
   - Generar los casos de uso de cada uno de los requisitos.
   - Generar los casos de prueba de cada uno de los requisitos. El formato en el que se han definido los casos de prueba ha sido mediante el uso del lenguaje gherkin, que es el que usa la herramienta Cucumber (utilizada por grandes empresas como PayPal, Canon o Roche). Esto permitiría en un futuro poder automatizar los test definidos durante el análisis de requisitos.

2. **Diseño gráfico:**

Partiendo de los casos de uso de cada uno de los requisitos funcionales, se han desarrollado los bocetos (están incluidos en el Apéndice B) y finalmente diseños finales de la interfaz gráfica (éstos se pueden ver en el Apéndice C). Realizar una fase de diseño gráfico en cada iteración, ha permitido agilizar el proceso de desarrollo, ya que mientras se realizaba en diseño, se han podido detectar fallos en la futura interfaz y realizar cambios cuando se trata de un dibujo o una maqueta, es mucho más sencillo que hacerlo una vez que ya está desarrollado el código HTML y CSS.

- Bocetos: al realizar los bocetos a mano, lo que más se estaba buscando era que se incluyesen todas las funcionalidades en la aplicación y determinar la forma en la que se va a distribuir el contenido.
- Maquetas finales usando Figma: partiendo de los bocetos, gracias a esta herramienta y al uso de un "Kit" de componentes de Boostrap 4, se han podido realizar las maquetas de forma casi exacta a la realidad. Durante esta fase, se han tomado decisiones de colores, y todo lo que se puede acercar más al nivel estético. Al ser muy cercano a la interfaz del producto final, en muchas ocasiones, al desarrollar las maquetas en Figma, se ha podido observar cómo el diseño que se había planteado en el boceto presentaba ciertos problemas y se ha tenido que volver a atrás para redefinir la interfaz.

3. **Modelado:**

Es muy importante modelar la solución que vamos a ejecutar, antes de pasar a la fase de implementación, ya que normalmente la idea que tenemos no se plasma tal y como queremos a la primera, y debemos revisar varias veces este modelo.

En nuestro caso, se ha realizado un modelado de los datos que se van a almacenar en la base de datos, de los agentes de la simulación y de la propia estructura de componentes o capas dentro de cada uno de los proyectos, así como de la arquitectura que va a seguir la aplicación completa.

4. Implementación:

Durante la implementación, como había que hacer la parte del cliente y del servidor, se han seguido varias metodologías de trabajo, para poder avanzar en las dos de forma independiente y finalmente integrarlas.

- Angular: en primer lugar, utilizando los archivos PDF de las maquetas generadas usando Figma, se implementa lo relativo a la interfaz gráfica, utilizando datos *fake* o *mocks*. Para los datos *fake*, como vamos a interactuar con una API RESTful, creamos objetos JSON, simulando que éstos nos llegan desde la API. Gracias a esto, no dependemos de que funcione el backend para probar y desarrollar la aplicación web.

  Una vez que están los endpoints del backend acabados, podemos pasar a integrar ambas partes, que será simplemente ajustar las diferencias que haya en los modelos (si es que las hay).

- Microservicios Backend: se desarrollan de forma independiente y se prueban usando Swagger o Postman.

5. Ejecución de Pruebas:

Al acabar de desarrollar cada una de las iteraciones, se procede a realizar las pruebas correspondientes a los casos de prueba. Para llevar un control de las pruebas realizadas, se ha hecho uso de un archivo Excel, el que se indican las características principales de cada una de las pruebas realizadas (id, caso de prueba, requisito al que corresponde, fecha, entorno, etc.)

6. **Integración de ramas y despliegue**

   Cuando se ha terminado de desarrollar una característica, se procede a integrar su desarrollo con la rama release y posteriormente a la rama master (teniendo que realizar los *merge* que corresponda). Al desplegar en master, como hemos indicado anteriormente, la aplicación web se subirá de forma automática a Netlify y el resto de los servicios a Heroku.

Una vez que sabemos las fases que se ejecutan en cada una de las iteraciones de desarrollo, vamos a proceder a describir, de forma breve, lo que se ha abordado en cada una de las iteraciones:

Iteración 0:

En esta fase, lo primero que se realizó fue investigar sobre las tecnologías que se iban a usar en el proyecto, para decidir cuáles eran las mejores o más apropiadas en cada caso.

Una vez que se decidieron las tecnologías, hubo que realizar un proceso de formación, mediante el uso de la documentación oficial de cada una de ellas, tutoriales de la plataforma Youtube y, sobre todo lo que fue de más ayuda, la realización de cursos mediante la plataforma Pluralsight. Aunque esté incluido en la iteración 0, porque fue

el momento en el que se hizo con mayor hincapié, se han seguido realizando actividades formativas durante todo el desarrollo del proyecto, a medida que surgían las necesidades.

Tras obtener cierta base en las tecnologías que se iban a usar, se procedió a inicializar los repositorios de los proyectos "Angular", "Design", "Microservice Usuario" y "API Gateway" en GitLab, teniendo que realizar las configuraciones pertinentes, como puede ser la instalación de librerías, la configuración de los despliegues de forma automática y seleccionar las estructuras de capas y componentes que éstos van a tener durante su desarrollo. Además, de esto, hubo que añadir a GitLab todo lo relativo a tableros, tareas, etiquetas y ramas.

Otra de las tareas distintivas de esta iteración ha sido la creación de las plantillas necesarias para poder generar los casos de uso y de prueba.

En cuanto a funcionalidades, se ha desarrollado lo relativo al usuario, es decir, el proceso de registro e inicio de sesión usando Oauth 2.0 y los tokens JWT, y todo lo relativo al perfil del usuario, incluyendo el soporte multilenguaje.

Iteración 1:

Esta iteración se centra en lo relativo a la simulación. Por lo tanto, lo primero que se hizo fue investigar, mediante el uso de artículos de investigación, acerca de la programación basada en agentes y ABM. Una vez conocidas las bases de esto, se procedió al estudio de la herramienta MESA.

Tras conseguir desarrollar pequeños modelos de simulación, se planteaba el reto de cómo podríamos mostrar en la web el mapa de la simulación. En un principio estuvimos considerando usar mapas reales para generar el mapa de una ciudad (utilizando

herramientas como Google Maps, Mapbox u OpenLayers), pero vimos que esto complicaba mucho la simulación, además de que, para poder simular, lo importante era que los parámetros de entrada se pudiesen extraer de ciudades reales, como, por ejemplo, índices de delincuencia respecto a la población, porcentaje de terreno ocupado por viviendas, etc. Por lo tanto, se decidió usar el elemento HTML <canvas>.

Para probar todo esto, antes de empezar con el modelo de la simulación de delitos, se procedió a realizar un prototipo usando uno de los ejemplos que incluía MESA en su repositorio de Github. Una vez que este prototipo funcionaba sin errores, empezamos a modelar el sistema objeto de este trabajo, así como la parte relativa los requisitos funcionales, como puede ser listar simulaciones, editarlas, etc.

Iteración 2:

Cuando ya habíamos finalizado la aplicación, se ha procedido a recopilar toda la documentación generada durante el desarrollo del proyecto. Para así poder revisarla, y desarrollar esta memoria.

4

Análisis y especificación de Requisitos

4.1 Requisitos Funcionales

En primer lugar, vamos a mostrar el listado completo de requisitos del sistema. Éstos se dividen en tres categorías funcionales: “Registro y acceso”, “Perfil de los usuarios” y “Gestión de las simulaciones”.

1. **Registro y acceso**
   - RF01: Los usuarios podrán registrarse usando su cuenta de Google.
   - RF02: Los usuarios podrán iniciar sesión usando su cuenta de Google.
   - RF03: Los usuarios podrán cerrar sesión.

2. Perfil de los usuarios:

- RF04: Los usuarios podrán gestionar sus datos en el sistema.
    - RF04.1: Los usuarios podrán asociar un avatar a su cuenta.
    - RF04.2: Los usuarios podrán eliminar el avatar asociado a su cuenta.
    - RF04.3: Los usuarios podrán editar su información principal.
    - RF04.4: Los usuarios podrán modificar sus preferencias.
- RF05: Los usuarios podrán eliminar su cuenta.

3. Gestión de las simulaciones:

- RF06: Los usuarios podrán crear simulaciones.
- RF07: Los usuarios podrán visualizar las simulaciones.
    - RF07.1: Los usuarios podrán visualizar el listado de simulaciones generadas.
    - RF07.2: Los usuarios podrán visualizar los parámetros de entrada y datos básicos de una simulación.
    - RF07.3: Los usuarios podrán visualizar el mapa del entorno de una simulación.
- RF08: Los usuarios podrán editar el nombre y la descripción de las simulaciones.
- RF09: Los usuarios podrán eliminar las simulaciones.
- RF10: Los usuarios podrán descargar los parámetros de entrada de una simulación.
- RF11: Los usuarios podrán descargar los datos de salida de los delitos generados en una simulación.
- RF12: Los usuarios podrán filtrar las simulaciones del listado de simulaciones.

En cada uno de los requisitos, se han definido sus respectivos casos de uso y casos de prueba. Lo cual nos permite tener una idea más clara de lo que se quiere conseguir con

cada uno de ellos y cómo debe actuar el sistema. Debido a la extensión que éstos tienen, se ha decidido incluir en este apartado de la memoria un ejemplo de uno de ellos, y para ver el resto se puede acudir al Apéndice D.

RF05: Los usuarios podrán eliminar su cuenta.
Caso de uso

CASO DE USO	**RF05:** Los usuarios podrán eliminar su cuenta.
ÁMBITO	Perfil de los usuarios
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán eliminar su cuenta.	
PRECONDICIÓN	
El usuario ha iniciado sesión y se encuentra en "Ajustes Cuenta", en la pestaña "Perfil".	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha podido eliminar la cuenta.	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá eliminado la cuenta y dejará de estar identificado.	
ESCENARIO PRINCIPAL DE ÉXITO	
1. El usuario pulsa el botón "Eliminar esta Cuenta" 2. El sistema muestra un modal de confirmación para realizar la acción. 3. El usuario pulsa el botón de confirmación. 4. El sistema elimina los datos de la cuenta 5. El sistema cierra la sesión del usuario. 6. El sistema muestra al usuario a la pantalla de inicio. 7. El sistema muestra un mensaje de éxito	
ESCENARIOS ALTERNATIVOS	
3.a. El usuario cancela la acción 3.a.1. Fin del caso de uso 4.a. Se produce un error al eliminar la cuenta 4.a.1. El sistema informa al usuario de la situación 4.a.2. Fin del caso de uso	

Tabla 4.1: Caso de uso RF05.

Caso de prueba

```
Feature: (RF05) Eliminar cuenta

  Background:
   Given un usuario que está registrado
   And está en "Ajustes Cuenta"
   And está en la pestaña "Perfil"
   And pulsa el botón "Eliminar esta cuenta" en el perfil
   And pulsa el botón "Eliminar Cuenta" en el modal de confirmación


  Scenario: (01) Cuenta eliminada correctamente
   When el sistema elimina los datos de la cuenta correctamente
   Then el sistema cierra la sesión del usuario
   And el sistema muestra al usuario la pantalla de inicio
   And el sistema muestra un mensaje de éxito "Cuenta eliminada correctamente"


  Scenario: (02) Error al eliminar la cuenta
   When hay un error al eliminar la cuenta en el sistema
   Then el sistema cierra el modal de confirmación
   And el sistema muestra un mensaje de error "No se ha podido eliminar la cuenta"
```

Tabla 4.2: Caso de prueba RF05.

4.2 Requisitos No Funcionales

- RNF01: El sistema deberá ser una aplicación web accesible a través de internet.
- RNF02: El sistema deberá tener soporte de múltiples idiomas, siendo obligatorios el español y el inglés.

Eficiencia:

- RNF03: El sistema deberá almacenar en una caché las simulaciones del usuario, para minimizar los tiempos de respuesta.

Usabilidad:

- RNF04: El sistema deberá tener un tiempo de aprendizaje inferior a 2 horas.
- RNF05: El sistema deberá poseer manual de usuario.
- RNF06: El sistema deberá tener un diseño "Responsive".
- RNF07: El sistema deberá proporcionar al usuario realimentación sobre el resultado de sus acciones, tanto mensajes de error como de éxito.

Seguridad:

- RNF08: El sistema deberá usar el estándar Oauth 2.0, usando Google Oauth.
- RNF09: Los datos del sistema sólo serán accesibles a través de tokens JWT.
- RNF10: El sistema deberá pasar todos los casos de pruebas del plan de prueba antes de su despliegue en el entorno de producción.

5

Modelado y diseño del sistema

5.1 Modelado Arquitectónico

5.1.1 Arquitectura Microservicios vs Monolítica

Cada vez se usan, con más frecuencia, las arquitecturas de microservicios para desarrollar proyectos, y es que, ofrece muchas ventajas con respecto a las aplicaciones monolíticas. Aun así, también debemos tener claro los puntos que contra que puede presentar y hacer un balance, en cada proyecto, para ver qué es lo que mejor se adapta en cada caso.

Para empezar, deberíamos definir qué son los microservicios. Los microservicios son servicios autónomos e independientemente desplegables, que colaboran para formar una aplicación. El tamaño que éstos deben tener no está definido, pues, dependerá de cada caso.

Los sistemas monolíticos están desarrollados de forma íntegra en un único repositorio, utilizando un solo proceso, base de datos y servidor, por lo tanto, a la hora de poder desplegar nuestro producto en los diferentes entornos, solo debemos preocuparnos de una fuente de código (aunque esto conlleva un riesgo, ya que, si falla el despliegue, se cae toda la aplicación).

Todo esto, hace que este tipo de sistemas tengan una menor capacidad de escalado y dificultades en términos de mantenibilidad, complicando su crecimiento a medida que aumentan otros factores o variables como el número de usuarios, de desarrollares, módulos, etc. Se producen problemas a la hora de escalar de forma horizontal (a nivel de software) y de forma vertical (suele aumentar mucho el presupuesto).

Además, es muy difícil cambiar de tecnología en mitad del desarrollo, ya que, si se decide hacer el cambio, hay que aplicarlo a todo el código, por lo que se tiende a acabar usando tecnologías desfasadas por no poder realizar este proceso.

Sin embargo, si nos decantamos por una arquitectura microservicios, tendremos la libertad para elegir la tecnología más apropiada en cada uno de los servicios, tanto en términos de lenguajes, como en bases de datos. Podremos hacer que los servicios escalen de manera independiente y cada uno de ellos pueden pertenecer a equipos diferentes, ya que lo importante es la interfaz que proporcionan de cara a la comunicación con el resto de los servicios (por lo tanto, es muy importante usar ésta como si fuera un contrato y producir versiones cuando vayamos a aplicar cambios en éstas, para que los clientes del servicio puedan seguir utilizándolo, aunque se realicen cambios).

Además, no tendremos el problema de que, si un servicio falla, falla todo el sistema, ya que como hemos dicho anteriormente, se ejecutan de forma independiente, e incluso, gracias al uso de cachés, podemos tener acceso a los datos que ofrece un servicio en los momentos en el que éste no se encuentre en funcionamiento.

Pero a pesar de que solventa muchas de las desventajas que presentan los sistemas monolíticos, añade varias dificultades a la hora de realizar el desarrollo. Se complica la interacción entre los distintos componentes que forman la aplicación, ya que hay muchas llamadas entre microservicios y éstas en ocasiones pueden ser ineficientes. También podemos ver como un punto en contra, que, para desplegar todos los servicios es casi imprescindible automatizar el proceso, ya que hacerlo de forma manual puede llevar mucho tiempo.

Otra de las limitaciones que nos encontramos a la hora de trabajar con esta arquitectura, es que como cada servicio almacena los datos de forma independiente, no podemos realizar "joins" de los datos, por lo que en la mayoría de los casos se recurrirá a la duplicación de datos, y esto puede generar problemas de inconsistencia, ya que, por lo general, al editar un campo, no se realizará la modificación en todas las bases de datos al mismo tiempo.

5.1.2 Arquitectura de la aplicación

Analizando las ventajas y desventajas de las arquitecturas expuestas en el apartado anterior, se ha decidido utilizar un sistema basado en microservicios, ya que nos interesa que sea una aplicación escalable para poder seguir trabajando en ella en el futuro (más allá de este Trabajo Final de Grado).

Durante el desarrollo, se han desarrollado varios servicios, usando diferentes tecnologías.

- Aplicación web: se trata un SPA (*Single Page Application*) desarrollado usando el *framework* Angular y utilizando *Typescript* como lenguaje de programación (éste se trata de un superconjunto del muy conocido *Javascript*)

- Microservicios Backend: todos ellos han sido codificados usando el lenguaje de programación Python, y más concretamente el *framework Flask*. Para poder manejar los datos, se ha hecho uso del sistema de base de datos *MongoDB* a través del framework *MongoEngine*.
- Api Gateway: al igual que los microservicios backend, se ha utilizado *Flask* para su desarrollo.

Como se puede observar, además de la aplicación web y los microservicios backend que se encargan de persistir los datos y realizar las operaciones de la lógica de negocio, tenemos un microservicio denominado Api Gateway. Esto es debido a que se ha utilizado el patrón arquitectónico conocido por el mismo nombre.

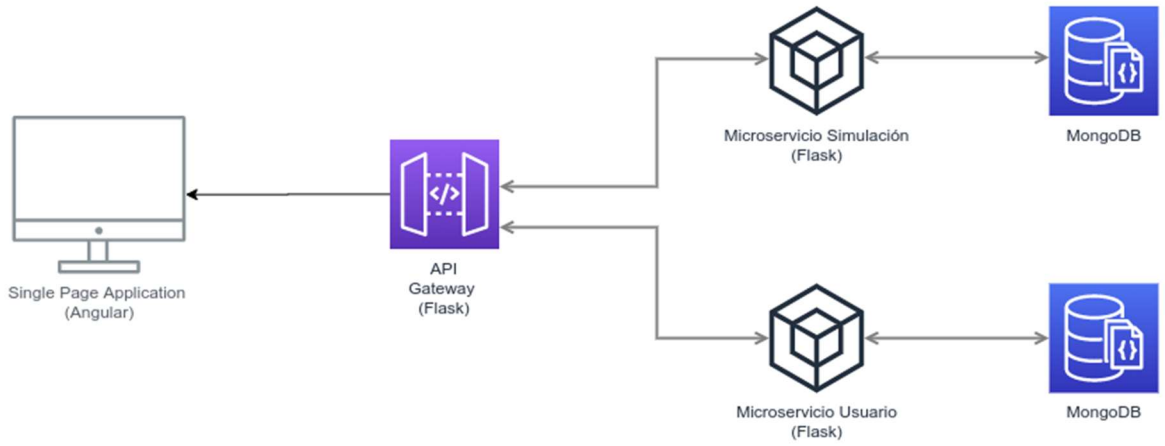


Figura 5.1 Diagrama de la arquitectura de la aplicación.

Gracias a esto, podemos lidiar con algunas de las desventajas de utilizar una arquitectura basada en microservicios, como puede ser la desorganización de llamadas que se producen entre los diferentes microservicios (hecho que ocurre a pesar de que respetemos el principio de que estos deben funcionar de forma independiente), y más aún, desde la aplicación cliente (si no se realizara esto, tendríamos en ella un listado con cada uno de los servicios y sus endpoints). Al tener que estar adaptándose a la interfaz de cada uno de los microservicios, se puede generar un gran desorden en él, ya

que incluso, cada microservicio, podría estar usando diferentes sistemas de envío de mensajes. Por lo tanto, la mejor opción es crear una pasarela que permita la comunicación entre la aplicación cliente (en este caso el frontend desarrollado en Angular) y el resto de los microservicios.

Para comunicar unos servicios con otros, se está haciendo uso de una comunicación síncrona, más concretamente de HTTP, ya que está estandarizado, se puede usar de forma muy sencilla casi desde cualquier lenguaje y nos permite enviar o recibir datos usando JSON o XML (cada vez más en desuso).

5.2 Modelo de datos

En las bases de datos de los microservicios, hemos almacenado dos entidades principales:

Usuario:

- _id: ObjectId. Se trata de la clave autogenerada por MongoDB que sirve como identificador.
- idGoogle: String. Campo que proporciona la API de Google al completar el protocolo de Oauth.
- imagen: String. En este campo almacenamos la URL generada por Imgur tras subir la imagen a su servicio a través de la interfaz que ofrece en su API.
- nombre: String. Es el nombre completo del usuario. Al realizar el registro, lo obtenemos de la API de Google
- apodo: String. Es el apodo asociado a la cuenta de Google.
- email: String. Es el correo asociado a la cuenta de Google
- preferencias: EmbeddedDocument. Se trata de una estructura compuesta por más atributos, ya que está previsto que en el futuro pueda haber más opciones.

A día de hoy, sólo está incluido el campo “idioma”, que se trata de un String que podrá tener los valores ‘es’ y ‘en’.

Simulación:

- _id: ObjectId. Clave primaria.
- nombre: String
- descripcion: String
- idAutor: ObjectId. Este campo se obtendrá del token JWT que va incluido en la cabecera ‘Authorization’ al hacer la petición al servidor.
- fecha: Date. Introducimos de forma automática la fecha en la que se ha creado la simulación.
- datosEntrada: File. Almacenaremos en la base de datos el archivo JSON con todos los parámetros de entrada.
- datosMapa: String. Guardaremos un código que identifica el documento guardado en un servidor.
- datosSalida: File. Se trata de un archivo CSV con los datos de los delitos cometidos durante la simulación.
- imagenesGraficas: List<String>. Consiste en listado donde estarán las URL (obtenida mediante el servicio Imgur) de cada una de la imágenes obtenidas al generar las gráficas a partir de los datos de salida.

En el capítulo “7. Implementación” hablaremos de la toma de decisiones de algunos de estos campos, como pueden ser los de datosEntrada, datosMapa y datosSalida.

5.3 Modelado de la Simulación

Antes de definir el modelo utilizado para realizar la simulación, es necesario hacer una aclaración sobre cómo funciona ésta. Para realizar una simulación, los componentes

básicos que se necesitan son el modelo y los agentes. El modelo se encarga de dirigir toda la simulación, inicializando los agentes a través de los datos de entrada, y encargándose de la ejecución de los pasos. En los pasos de una simulación, por cada agente, se ejecuta el comportamiento que tenga definido como "paso".

Vamos a poner un ejemplo en código Python de la estructura que deberían seguir los sistemas basados en agentes. Si tuviéramos un modelo que lo único que hace es que haya personas que tienen una probabilidad del 50% de saludar, el código podría ser el siguiente:

```
class Persona():
    def step(self):
        quiere_saludar = random.choice([True, False])
        if quiere_saludar:
            print("Hola")

class Modelo():
    def __init__(self, num_personas, num_pasos):
        self.__listado_personas = self.__crear_personas(num_personas)
        self.__num_pasos = num_pasos

    def step(self):
        for agente in self.__listado_personas:
            agente.step()

    def ejecutar_simulacion(self):
        for indice_pasos in range(self.__num_pasos):
            print("Paso " + str(indice_pasos))
            self.step()

    def __crear_personas(self, num_personas) -> List[Persona]:
        '''
        :return: lista de agentes creados
        '''
        lista = []
```

```
        for i in range(num_personas):
            lista.append(Persona())
        return lista

modelo = Modelo(3, 5)
modelo.ejecutar_simulacion()
```

Tabla 5.1: Ejemplo agentes.

Al tener un factor de aleatoriedad, cada simulación tendrá resultados diferentes, pero un ejemplo de salida podría ser este:

```
Paso 0
Hola
Hola
Paso 1
Paso 2
Hola
Hola
Paso 3
Hola
Paso 4
Hola
Hola
```

Tabla 5.2: Salida ejemplo agentes.

El framework MESA nos aporta las herramientas necesarias para poder hacer este proceso teniendo los agentes distribuidos por una cuadrícula (también podría usarse un grafo), encargándose él de la gestión de los identificadores de cada uno de los agentes.

Además del modelo y los agentes, contiene otro componente muy útil, denominado planificador ("scheduler" si consultamos en su documentación). Éste se encarga de definir el orden en el que se van a activar los pasos de los agentes. En el ejemplo anterior, con la forma en la que estamos iterando en la lista, siempre se ejecutan los pasos de las personas en el mismo orden. MESA nos ofrece diferentes estrategias de planificación, por lo tanto, para que un agente ejecute sus pasos, habrá que añadirlo al

planificador. En la aplicación desarrollada, estamos ejecutando los agentes del planificador en un orden aleatorio (utilizando la clase RandomActivation proporcionada por el framework).

5.3.1 Agentes

Los agentes se encontrarán distribuidos en un grid o cuadrícula. Para poder asemejar el sistema a cómo se gestionan los recursos policiales en la realidad, se ha decidido dividir este mapa en zonas, donde cada una corresponde a una zona policial. Esto permite poder definir los parámetros de entrada relativos a los delitos, policías, % de edificación y población, en cada una de las zonas. Por lo tanto, podremos simular una ciudad en la que hay distintos distritos policiales.

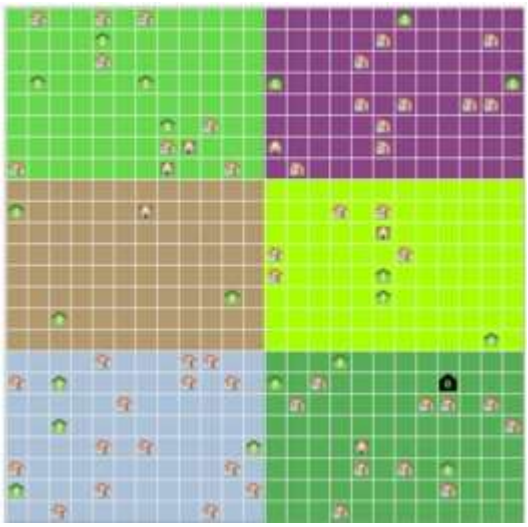


Figura 5.2 Ejemplo de división en zonas (cada zona posee un color diferente).

El diagrama de clases correspondiente a los agentes que se han implementado, para la ejecución del sistema, es el siguiente:

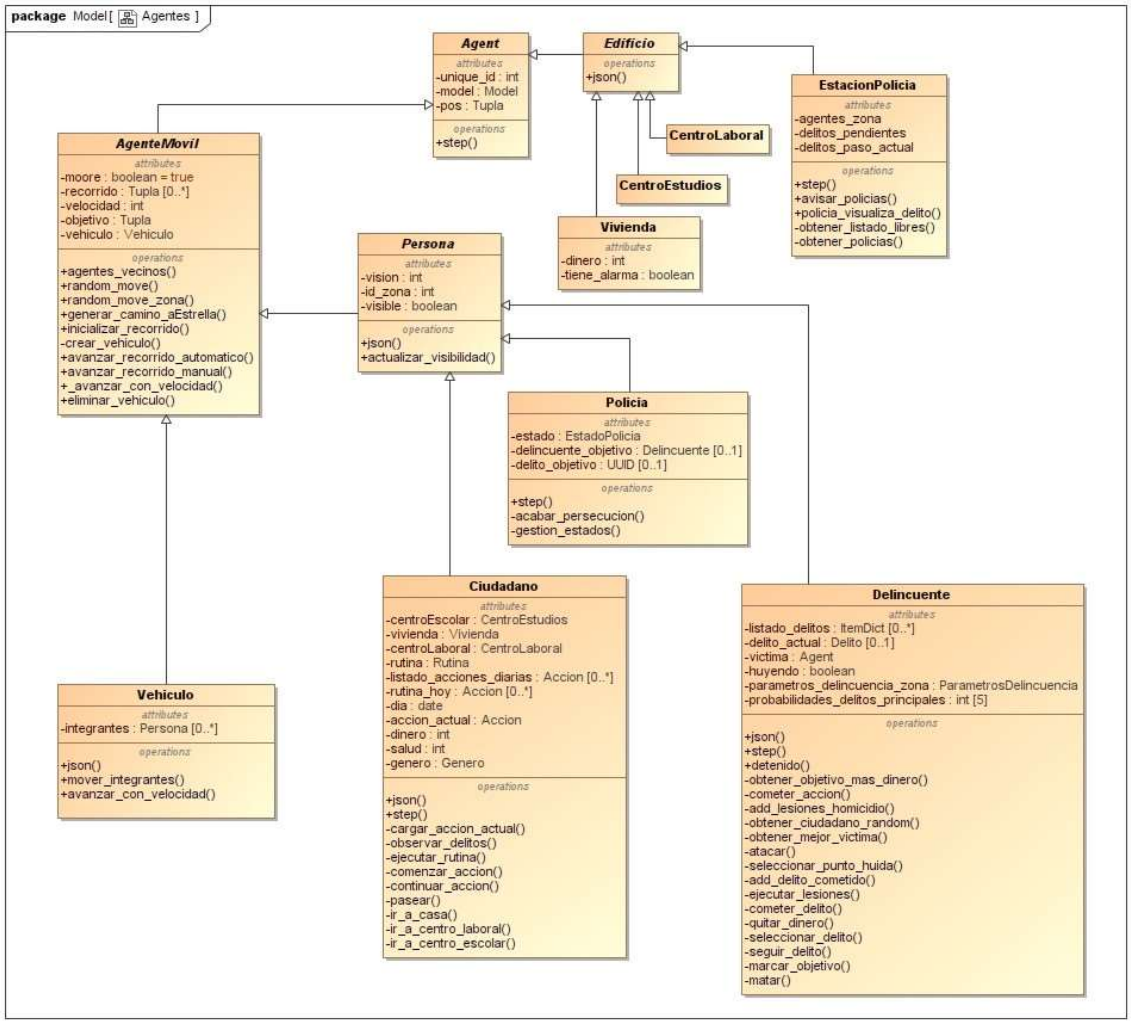


Figura 5.3 Diagrama de clases de los agentes que forman el modelo de la simulación de delitos.

Agentes auxiliares:

Agent:

Es la clase base que nos aporta MESA y que aprovechamos en nuestros agentes gracias a los mecanismos de herencia.

AgenteMovil:

Este agente tampoco lo vamos a usar para crear instancias, sino que actúa de base del resto de agentes. Nos permite agrupar los atributos y métodos propios de un agente que realiza viajes en el mapa fijando distintos puntos. Para ello, marcamos un punto objetivo, y mediante el algoritmo A*, calculamos la ruta hasta él (evitando obstáculos

como pueden ser los edificios situados por la ciudad). Los agentes móviles, podrán desplazarse utilizando diferentes velocidades, y, además, podrán tener o no un vehículo, lo cual hará que se desplacen con mayor velocidad.

Se ha añadido un sistema para que se frene poco a poco antes de llegar al objetivo. Por lo tanto, si hemos indicado que se avance con la opción de frenado, y vamos a velocidad 5 (recorrerá 5 celdas por paso), pero solo quedan 4, se reducirá la velocidad. Esto no interesaba en el caso de los agentes del tipo "Policia", ya que querrán llegar cuando antes al lugar del delito, y, por lo tanto, en su caso, se realizará el recorrido sin ello. Al poner el campo "moore" como verdadero, estamos indicando que se puede mover usando 8 direcciones, en vez de 4. En nuestro sistema siempre lo estamos usando a "True", ya que se asemeja más a la realidad.

Vehiculo:

Los vehículos serán generados de forma automática según la distancia a la que se encuentre la persona de su punto de destino a la hora de calcular el recorrido. Este agente no forma parte del planificador que hemos comentado anteriormente, ya que se genera y elimina de forma dinámica, y es controlado por el resto de los tipos de AgentesMoviles. Por lo tanto, accediendo al atributo "vehiculo", llamarán al método avanzar_con_velocidad() cuando se considere necesario. Mover_integrantes() se utiliza para que las personas que se encuentren en el vehículo se desplacen con él al avanzar por el grid.

Persona:

Se trata de una clase base que sirve, además de para añadir ciertos atributos, para compartir del método actualizar_visibilidad() con sus subclases. Cuando una persona esté en la misma celda que un edificio o de su vehículo, el atributo visible está como falso, para así poder ocultarlo a la hora de mostrar la simulación de forma gráfica.

Edificio:

Nos permite agrupar los agentes que representan edificios en la realidad.

Agentes principales:

Vivienda, CentroEstudio y CentroLaboral:

A día de hoy, la distinción entre estos agentes nos sirve para poder determinar las rutinas horarias que van a ejecutar los ciudadanos. Además, las viviendas también pueden ser víctimas de delitos, y por ellos tiene los atributos "dinero" y "tiene_alarma".

EstacionPolicia:

Representa una comisaría de policía. Posee el atributo "agentes_zona", que es un diccionario cuyas claves son los id de las zonas y el valor es el conjunto de policías asignados a esa zona (en el caso de que un agente no tenga una zona fijada, estará en el conjunto cuya clave es "None").

La estación de policía puede recibir dos tipos de llamada:

- Por parte de los ciudadanos: cuando un ciudadano presencia un delito y llama a la estación, ésta se encarga de mandar a un agente de la zona si aún no hay agentes asignados al delito. En el caso de que en esa zona no haya agentes disponibles, se mandará a uno de los que no tiene zona asignada. Si no hubiese ninguno disponible (ni en su zona y de los que no tienen zona), se procederá a guardar el delito para asignar agentes cuando haya libres.
- Por parte de la policía: cuando un agente, durante su labor de patrulla, visualiza un delito, avisa a la estación de que va a realizar una actuación policial. Así la estación anota que ese delito ya tiene un agente encargado.

Antes hemos mencionado que puede ocurrir que no haya ningún agente disponible. Cuando el planificador le dé el turno a la estación de policía, intentará asignar los delitos pendientes usando el mismo procedimiento descrito anteriormente.

Policia:

Podemos definir su comportamiento con el diagrama de estados.

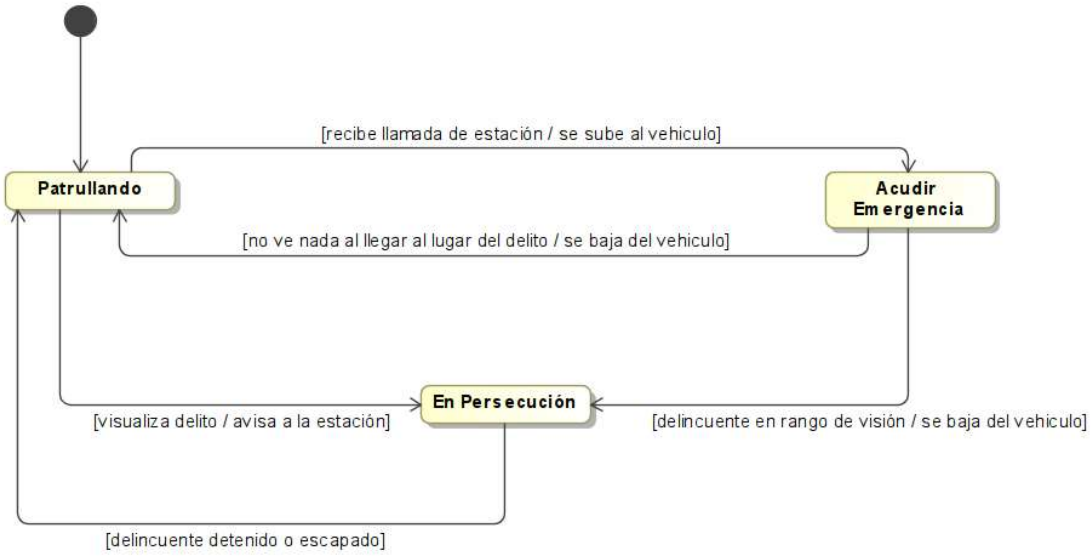


Figura 5.4 Diagrama de estados del agente "Policia".

Cuando se encuentre en la celda vecina del lugar donde está el delincuente, procede a detenerlo. En el caso de que el delincuente haya terminado su proceso de huida, se tomará como que ya se ha escapado.

Como se puede ver en el diagrama, a la hora de acudir a una emergencia, lo hará en un coche de policía y una vez que detecte al delincuente o se llegue al lugar del delito y no esté, se bajará de éste.

Ciudadano:

Los ciudadanos tendrán asociada una rutina, y dependiendo de ésta, realizarán diferentes actividades. Cuando terminan de realizar una actividad (ya sea porque ha llegado a su hora de fin o porque se ha realizado el tiempo indicado), se pasará a la

siguiente que le corresponda según la rutina. En la imagen de la figura 5.5, podemos ver las rutinas existentes en el sistema. Nótese que los tiempos de traslado de la figura se han establecido en una hora para facilitar su representación en el diagrama, aunque en la implementación cada actividad comienza inmediatamente después de finalizar la anterior, sin tener en cuenta ese tiempo de traslado.

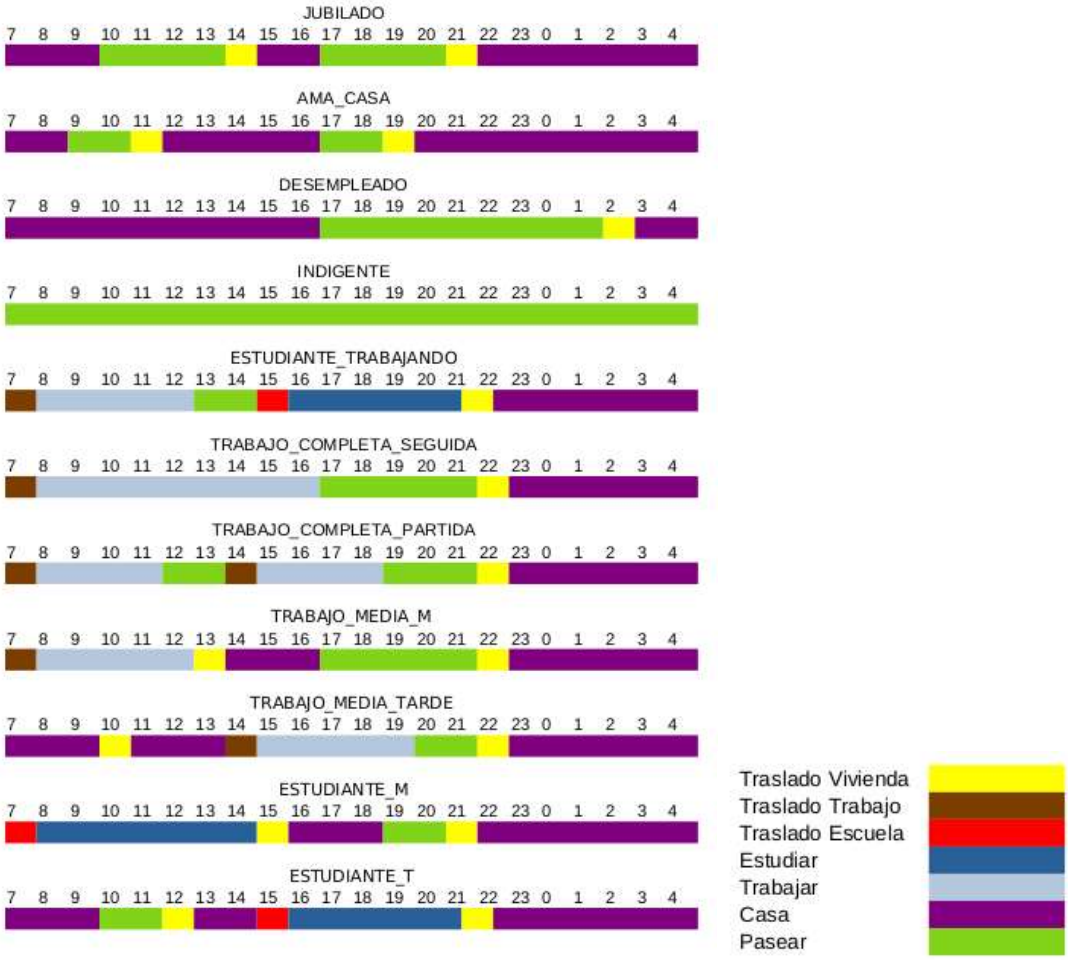


Figura 5.5 Rutinas de los ciudadanos.

Los pasos de los agentes del tipo ciudadano consistirán en lo siguiente:

- Observar si hay delito: en el caso de que en su rango de visión haya un delito, avisará a la estación de policía.

- Ejecutar la actividad que le corresponda (si toca cambiar de actividad, empezará a realizar la siguiente).

- Actualizar la visibilidad (como hemos mencionado antes, puede encontrarse en su vehículo o en un edificio).

Delincuente:

Para definir el comportamiento de los agentes Delincuente, mostramos el diagrama de la figura 5.6, que nos muestra el ciclo de actividad.

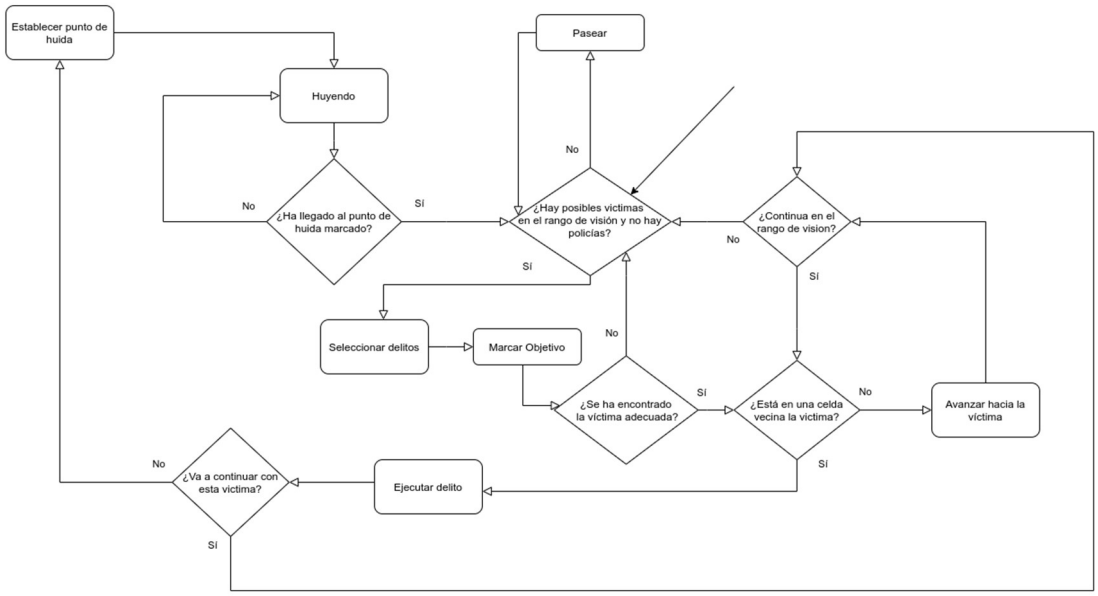


Figura 5.6 Diagrama que muestra las actividades que realiza un delincuente.

Si durante cualquier punto de la ejecución, un policía lo detiene, volverá al nodo inicial.

Como podemos observar, el delincuente lo que hace primero es seleccionar los delitos que va a ejecutar (si hay agentes que puedan ser víctimas y no hay policías en su rango de visión), y entonces es cuando busca la víctima más adecuada. Si no la encuentra, en el siguiente paso de la simulación realizará el mismo proceso.

Si encontró una víctima adecuada, procederá a iniciar la persecución. Los delitos tienen varios estados: sin_empezar, en_proceso, acabado y huida. En el momento en el que

una de las celdas vecinas tenga a la víctima procederá a ejecutar el delito, y pasará del estado sin_empezar a en_proceso.

Un delito puede estar en proceso varios pasos. Esto dependerá de factores como los parámetros de entrada. Por ejemplo, si se ha determinado que los delitos ejecutados van a ser robo, lesiones y homicidio, hasta que no cometa el asesinato, va a continuar con el delito mientras siga en el rango de visión o no haya sido detenido.

Los delitos que puede cometer un delincuente en el sistema los dividimos en delitos principales y delitos extras. A partir del delito principal, se elegirán los extras.

Delito Principal	**Posibles extras**
Robo	Lesiones, Homicidio
Asesinato	Nada
Violación	Lesiones, Homicidio
Hurto	Nada
Lesiones	Homicidio

Tabla 5.3: Delitos del sistema.

Como los ciudadanos tienen un atributo relacionado con la salud, al cometer un delito de lesiones ésta les bajará. En el caso de que llegue a cero, será homicidio.

Durante la ejecución de los pasos de los delincuentes, se irá almacenando la información relativa a los delitos y sus estados, para que el usuario puede obtener un archivo CSV con toda esta información y también realizar la generación de gráficas.

6

Implementación

6.1 Aspectos a destacar

6.1.1 Oauth 2.0 y tokens JWT

En primer lugar, se quiere destacar el uso del estándar Oauth 2.0, ya que esto es un aspecto que implica tanto al frontend como a todo el backend.

Cuando el usuario inicia sesión a través de su cuenta de Google en el frontend, Google como proveedor Oauth nos proporciona ciertos datos, entre ellos el id de un token firmado.

Una vez que hemos recibido este token, debemos enviárselo al servidor, y éste se encargará de verificarlo, haciendo otra llamada a Google Oauth. Al verificarlo, también nos devuelve la expiración que tendrá ese token.

En nuestro caso, una vez que se ha verificado el token de Google, procedemos a generar un token JWT interno, para poder almacenar en él los datos que nos interesen de cara

a la sesión de los usuarios y la verificación de permisos. El servidor le indicará al cliente el token que se ha generado y la aplicación que hemos desarrollado en Angular lo almacena en el sistema de almacenamiento propio del navegador ("session storage").

Cada vez que se realice una llamada a las APIs que requiera estar registrado (en nuestros proyectos van a ser todas las llamadas menos la que nos sirve para realizar el registro/inicio de sesión), se comprobará ese token, y se tendrá acceso a los datos que incluye.

Actualmente sólo lo estamos usando para indicar el id del usuario, pero en el futuro, si se quisiera implementar el acceso a las rutas por roles, sería muy sencillo, ya que solo tendríamos que introducir esos campos de nuestros tokens, y añadir un middleware/wrapper/decorador que compruebe el rol según la ruta a la que se quiere acceder.

Para mandar el token, debemos introducirlo en la cabecera "Authorization" siguiendo el siguiente formato: "Bearer <tokenJWT>". Donde tokenJWT sería una estructura compuesta por tres partes:

- Header (cabecera)
- Payload: datos almacenados
- Verify Signature: esto sirve para verificar que los datos son auténticos.

Los datos están encriptados usando el algoritmo HS254. Mediante el uso de páginas web como jwt.io, podemos ver el contenido de los token.

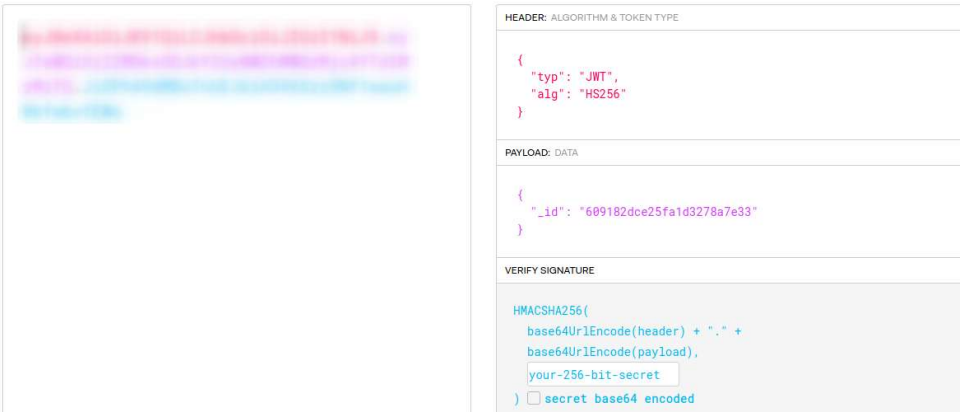


Figura 6.1 Ejemplo de datos que incluyen los tokens JWT de la aplicación.

Para implementar el envío y recepción de los tokens JWT, en nuestros microservicios backend hemos desarrollado un *wrapper* (nos permite ejecutar funciones antes de la ejecución de otras, pudiendo transformar datos antes de la ejecución de la función principal) que se encarga de verificar el token recibido en la cabecera y obtener el id del usuario.

En la parte del cliente, hemos generado lo que se conoce como "Interceptor". Gracias a esto, indicamos que cada vez que vamos a realizar la llamada http al nuestra API, si tenemos el token almacenado en el navegador, que lo incluya en la cabecera "Authorization", de esta forma nos aseguramos de que se manda siempre que sea necesario.

6.1.2 Cliente

Lazy Loading:

Para realizar el proyecto, se ha usado lo que se denomina estructura basada en *features*.

Al realizar la configuración inicial del proyecto, se han desarrollado dos módulos principales:

- Core: para los componentes que solo se tengan que instanciar una vez en toda la aplicación (modal, barra de navegación, services, etc.).
- Shared: para compartir componentes, pipelines, etc. entre varios módulos.

Se ha usado la técnica de "lazy loading", que consiste en que solo se cargarán los módulos necesarios; esto permite mejorar bastante la aplicación en términos de rendimiento, ya que evitaremos cargas innecesarias.

Para no incumplir esta característica, es muy importante respetar la separación de los módulos y no realizar llamadas a servicios o componentes que se encuentran en otro módulo. En el caso de que un componente o servicio se utilice en más de un módulo, se pasará al módulo Shared. De esta forma evitaremos cargar un módulo completo cuando solo queremos tener acceso a cierta funcionalidad limitada.

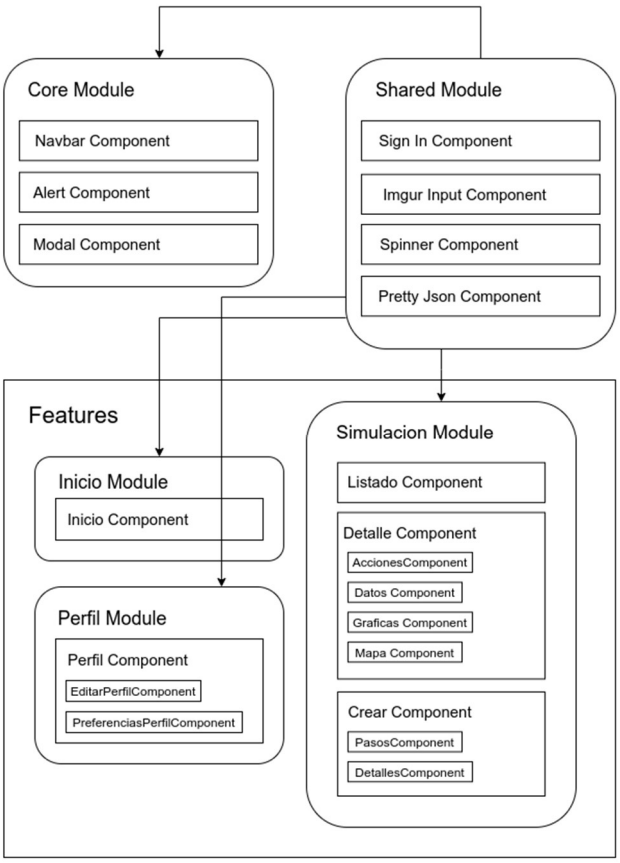


Figura 6.2 Diagrama de componentes generados en el proyecto.

Depurar código desde Visual Studio Code:

Para poder depurar el código a medida que se estaba desarrollando, se ha configurado el proyecto de forma que podemos hacer este proceso desde la propia IDE de desarrollo, en nuestro caso Visual Studio Code, ya que por defecto no podemos realizarlo. Cuando decidimos depurar desde el editor, se lanzará una nueva ventana del navegador, la cual está como si fuese modo incógnito, ya que no tiene guardadas las cuentas, contraseñas, marcadores, etc.

Modal Service:

En el módulo central (*core*) se ha generado un servicio que nos permite invocar un modal, con simplemente el título, la descripción, etc. Además, como algunos de ellos tendrán botones, aparte de indicarles el texto, de forma asíncrona podremos saber si el usuario ha decidido realizar la acción principal o si por el contrario ha pulsado el botón de cancelar o lo ha cerrado. Esto es muy útil, ya que, a la hora de desarrollar modales con distintas funcionalidades, simplemente habrá que realizar la acción en función del dato que nos llegue desde el servicio.

Alert Service:

Si utilizamos este servicio, podremos indicarle a la aplicación que genere un mensaje del similar a una notificación en la esquina superior derecha, que desaparecerá a los 3 segundos si el usuario no ha pulsado el botón de cerrarlo. Esto lo usaremos para indicar cuando una operación se ha realizado con éxito o si ha sucedido algún error. Usando el enumerado *TipoAlert*, indicaremos los colores que tendrá esta notificación.

Logger Service:

Cuando queramos mostrar algo en la consola del navegador, en lugar de mostrarlo usando el método console.log() se utiliza este servicio. Gracias a él, podemos determinar cómo mostramos los mensajes de forma generalizada (pudiendo elegir entre distintos tipos de logs gracias al enumerado TipoLog). Esto también permite desactivar y activarlos cuando queramos, actualmente se han dejado desactivados la aplicación se ejecuta en modo de producción.

Formularios:

En Angular hay dos tipos de formularios, los reactivos (Reactive) y los basados en plantillas (Template). Para esta aplicación se ha usado los segundos ya que son más sencillos y en este caso no necesitamos realizar validaciones muy complejas.

Guards:

Para poder proteger las rutas, hemos introducido un Guard que se encarga de mostrar un modal al acceder a una ruta protegida (actualmente los son todas menos la ventana de inicio).

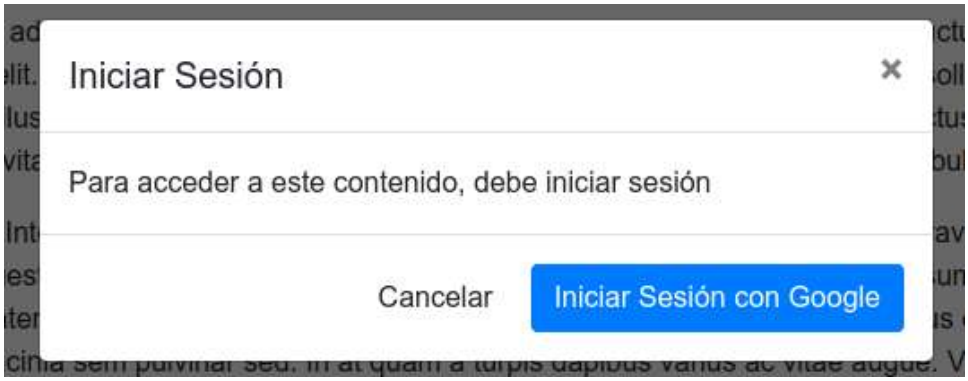


Figura 6.3 Modal que permite controlar el acceso a las rutas protegidas.

Si intentamos acceder a una de esas rutas, aparece el modal de la imagen. En el caso de que se cancele la acción, se irá a la ventana de inicio. Al darle a iniciar sesión, se realizará el proceso de Oauth con Google (igual que si lo hacemos desde la barra de navegación, ya que ese botón es un componente, para poder reutilizar la lógica del inicio de sesión en cualquier sitio), y en el caso de que vaya todo bien, se accederá directamente a la ruta elegida.

Caché:

Los datos relativos a las simulaciones se han cacheado para evitar llamadas innecesarias al servidor. Esta caché se limpiará en el caso de que se cierre sesión, o de que se realicen operaciones con las simulaciones que afecten a la base de datos (crear, editar o eliminar simulaciones).

Asistente o Wizard:

Para facilitar la introducción de datos de la simulación por parte del usuario, hemos dividido ésta tarea en varias fases o pasos. Una vez que el usuario haya rellenado todos los campos de forma correcta, se activará el botón “Ir a Resumen”. Desde el resumen, se podrán ver todos los datos introducidos y volver a atrás o empezar el proceso de generación de la simulación.

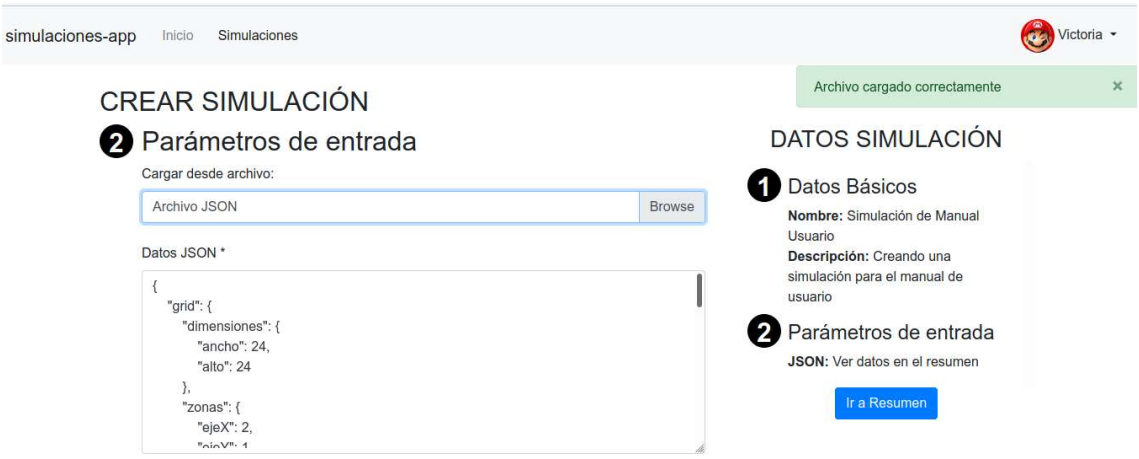


Figura 6.4 Asistente para crear simulaciones

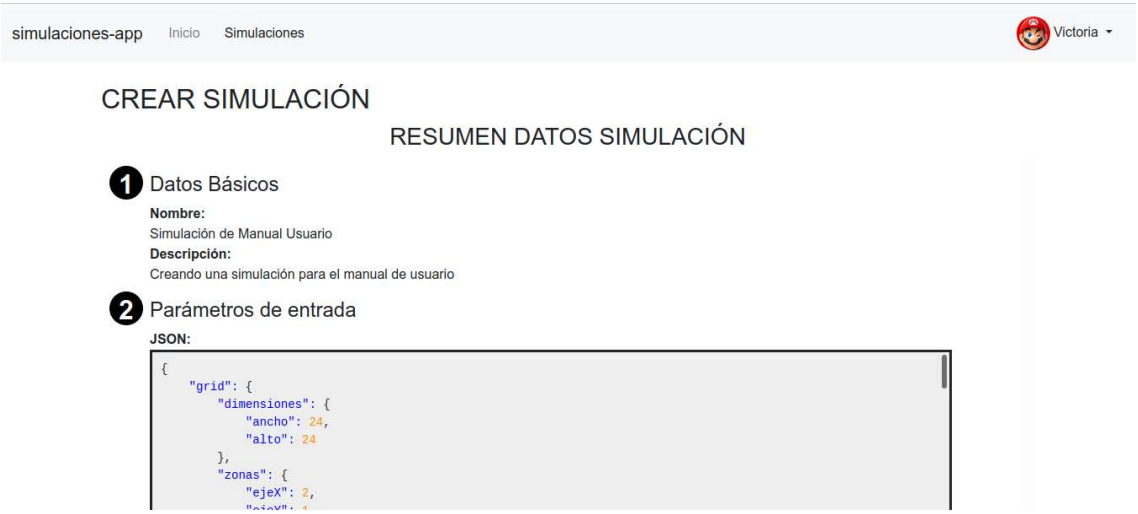


Figura 6.5 Ejemplo de resumen para crear simulaciones

Cargar de Nuevo y spinners:

Al realizar las llamadas al servidor, indicamos que se está realizando la llamada mediante el uso de spinners. En el caso de falle, indicamos al usuario que se ha producido un error, y si se trataba de una llamada para mostrar datos por pantalla, aparecerá el botón “Cargar de nuevo”.

Figura 6.6 Ejemplo de spinner al acceder a los datos.

Figura 6.7 Ejemplo de botón "Cargar de nuevo"

6.1.3 Servidor

Capas:

A la hora de desarrollar el código en los microservicios, se ha divido en varias capas:

- Base de datos: aquí se incluye lo relativo a los modelos de la base de datos (en la API Gateway no existe esta capa).
- Features: es una capa que agrupa los modelos, controladores y manejadores de los *endpoints* de la API RESTful.
- Controllers: se encargan de definir las rutas de la aplicación, las peticiones (*request*) y las respuestas (*response*) que se deben usar, así como de verificarlas. Si los datos son correctos (y siendo una ruta protegida está el token JWT), se llamará al gestor (*handler*) que tiene asignado.
- Handlers: se encarga de implementar la lógica del endpoint.

- Utils: en esta capa tenemos clases y métodos que nos ayudan al desarrollo, como por ejemplo ,el *wrapper* que se encarga de verificar el *token*, un sistema de logs, etc.

Además de estas capas, en el microservicio de la simulación tenemos una dedicada a la implementación de ésta.

Swagger:

Para que otros desarrolladores puedan beneficiarse del trabajo realizado, se ha generado lo conocido como Swagger UI. Esto permite obtener la documentación de cada uno de los *endpoints* de las APIs, pudiendo incluso realizar las llamadas desde la propia interfaz.

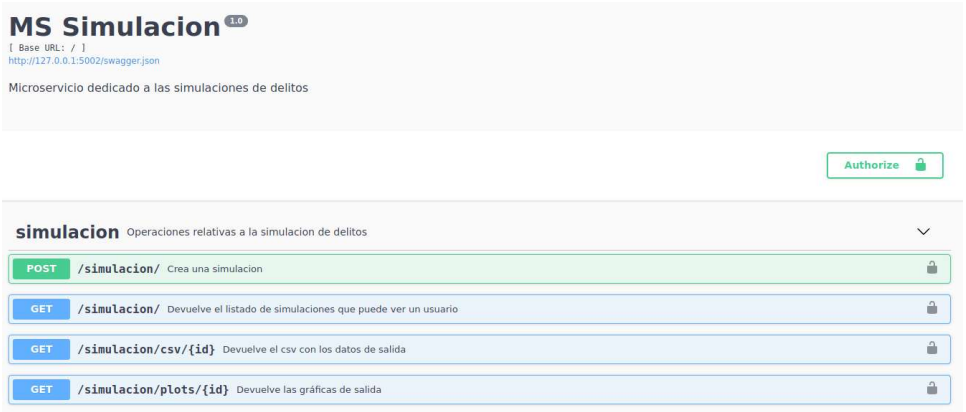


Figura 6.8 Swagger UI generado con la librería flask-restplus.

En la zona de la derecha podemos ver que aparece un candado sobre los *endpoints*, eso quiere decir que necesitamos realizar un proceso de autenticación para poder utilizarlos. En nuestro caso, como antes hemos comentado que estamos usando *tokens* JWT, nos aparece esta ventana al pinchar sobre el botón "Authorize" o cualquiera de los candados.

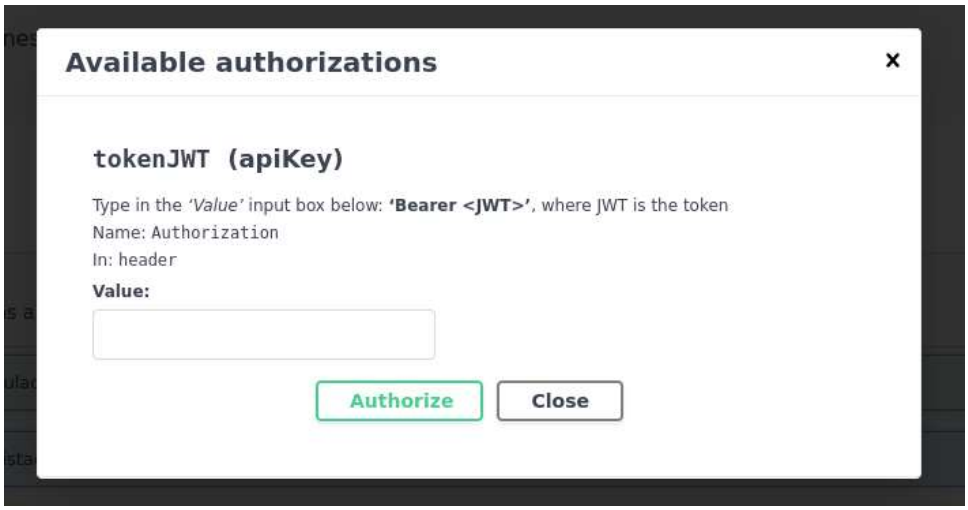


Figura 6.9 Autorización en Swagger UI

Algoritmo A*:

Para que los agentes de la simulación pudiesen ir de un lugar a otro del mapa empleando la ruta más corta posible, se ha procedido a utilizar el popular algoritmo de *pathfinding* conocido como A*.

Para desarrollar este algoritmo, lo primero que hemos tenido que hacer es generar la estructura de los nodos que vamos a almacenar en el árbol. En nuestro caso, tienen los siguientes atributos:

- posicion: nos indica la posición en el eje x e y.
- padre: nodo padre del actual (vacío en el caso del nodo raíz)
- coste: coste de ir desde la posición inicial hasta la posición actual
- cerrado: booleano que indica si el nodo está o no cerrado
- estimacion: estimación del coste de ir desde la posición actual hasta la posición final (calculada mediante el heurístico seleccionado)

Tendremos una cola ordenada de nodos “abiertos”, teniendo en cuenta la suma de los valores de “g” (coste) y “h” (estimación), en el orden de menor a mayor. En el caso de que coincidan, irá antes el que menos coste tenga.

El nodo que se encuentre en la primera posición de la cola será el siguiente en ser expandido.

Al empezar el algoritmo, introducimos el nodo raíz en la cola (la posición inicial). Mientras haya nodos en la cola de abiertos, cogemos el primer elemento y lo expandimos, es decir, lo quitamos de la cola, lo ponemos como cerrado, modificamos sus datos en el árbol (en el caso del nodo raíz hay que añadirlo) y si no es la posición final, tendremos que buscar los nodos sucesores. En el caso de que sea la posición final, ya hemos encontrado el camino y sólo habrá que volver hacia atrás usando los nodos padres de cada uno de los nodos.

Para encontrar los sucesores, tendremos que ver las celdas vecinas sobre las que sean válidas (que estén dentro de los límites del mapa y, si no se trata de la posición final, que no haya ningún edificio en ella).

Por cada uno de los sucesores:

- si no estaba en el árbol, lo añadimos al árbol y a la cola de abiertos
- si ya estaba en el árbol, vemos si es un mejor sucesor que el que había en el árbol (que "g" + "h" sea menor, o que sean iguales pero el coste es menor). Si es mejor, habrá que sustituirlo en el árbol y añadir el nuevo nodo a la cola de abiertos (en el caso de que estuviera abierto el nodo que hemos sustituido, lo quitamos de la cola de abiertos)

Como los agentes se pueden mover usando 8 direcciones, en lugar de utilizar como heurístico ("h" o estimación) la distancia Manhattan, usamos la distancia diagonal, penalizando las celdas diagonales con raíz cuadrada de 2 (esto se ve reflejado tanto en el heurístico como en la asignación de costes al realizar los pasos).

6.2 Librerías principales

Entre las dependencias instaladas en el proyecto de Angular, podemos destacar las siguientes:

- ng-boostrap: gracias a esta dependencia, hemos podido utilizar el framework Boostrap para el desarrollo de las interfaces de la web, y por lo tanto, poder

tener un diseño “Responsive”. El uso de esta dependencia ha influido bastante en la forma en la que se han desarrollado los ficheros HTML.

- ngx-translate: librería que nos permite aplicar el soporte de varios idiomas.
  La forma oficial que indica la documentación de Angular añade mucho código y hay que introducir muchos parámetros en los archivos de traducción por cada una de ellas. Por esta razón, tras analizar ambas opciones (la forma que ofrece angular por defecto y ésta librería), para una aplicación sencilla como esta, es mejor usar ngx-translate, ya que permite añadir traducciones de forma rápida (una vez que están puestas las etiquetas solo hay que crear archivos .json para añadir nuevos idiomas).

- font-awesome: nos permite añadir de forma muy sencilla iconos de la página con el nombre “Font Awesome”, sin tener que descargar de forma individual cada una de ellas, ya que utilizamos un componente en el que simplemente debemos indicar el identificador del icono, y las características de éste (tamaño, color, etc.).
- angularx-social-login: dependencia que nos permite interactuar con varios proveedores de Oauth. En nuestro caso lo hemos utilizado para hacerlo con Google, pero permite otros como Facebook, Twitter o Amazon.
- rxjs: es una librería que nos permite trabajar de forma asíncrona mediante el uso de “Observables”. Gracias a ella, hemos podido realizar gran parte de las funcionalidades.

En los diferentes microservicios de Flask, hemos instalado (aparte del propio framework Flask), las siguientes librerías:

- flask-restplus: es la librería que nos ha permitido poder generar la interfaz de swagger a medida que íbamos desarrollando el código. Sin el uso de ella, el código

de nuestros servicios hubiese sido muy diferente a como es, ya que ha influido de manera radical en la forma de desarrollar las APIs.

- matplotlib: se ha utilizado para desarrollar las gráficas a partir de los datos de los delitos generados durante la simulación.
- Mesa: framework utilizado para desarrollar la simulación
- mongoengine: ORM utilizado para interactuar con la base de datos MongoDB
- Flask-Cors: librería que se ha utilizado para permitir CORS a todos los dominios. Esto se ha aplicado a todas las rutas, pero esta librería nos permite definir para qué rutas se va a permitir.
- pandas: utilizado para el análisis de los datos de la simulación, y así posteriormente poder generar las gráficas.
- requests: nos ha permitido realizar las peticiones HTTP de unos microservicios a otros (también a las APIs externas utilizadas).
- google-auth: librería para realizar la autenticación a través de Google. En nuestro caso, nos sirve para verificar el token Oauth 2.0 y obtener los datos básicos del usuario.
- python-dotenv: gracias a ella hemos podido cargar los datos que se encuentran en los archivos .env, y así hemos evitado subir credenciales a los repositorios.
- PyJWT: nos ha permitido generar y decodificar los tokens JWT utilizados para las sesiones.

6.3 Recursos Software Utilizados

6.3.1 Aplicaciones

- Visual Studio Code: es un editor de código que permite desarrollar en cualquier lenguaje de programación, y además ofrece soporte a ellos gracias a las extensiones que se pueden instalar visitando el mercado que ofrece. En nuestro

caso, se ha utilizado para desarrollar el código de la aplicación web (Angular). Vamos a comentar de forma breve las principales extensiones que nos han sido de gran ayuda:

  - Angular Language Service (v12.0.5): aporta ayuda a la hora de desarrollar con angular (lista de autocompletado, mensajes de error, información de las funciones/métodos y la funcionalidad de ir a la definición del método/función).
  - Auto Import (v1.5.4): permite añadir automáticamente la ruta de los archivos utilizados al importar elementos usando TypeScript.
  - Debugger for Firefox (v2.9.4): extensión utilizada para poder configurar lo necesario para poder hacer "debug" desde VSCode mientras desarrollamos, en vez de tener que hacerlo desde la ventana del navegador.
  - GitLens – Git supercharged (v11.5.1): al tener el cursor en una línea de código, permite ver qué *commit* se ha modificado por última vez (teniendo acceso de forma rápida a los cambios que se hicieron), además de que al final de la línea, tras el código, aparece el autor, la fecha, el mensaje del *commit*.
  - Todo Tree (v0.0.213): permite visualizar los TODOs del proyecto y tener un listado con ellos.
  - TypeScript Hero (v3.0.0): ofrece ayuda a la hora de desarrollar usando TypeScript.
  - vscode-icons (v11.5.0): añade iconos a las carpetas y archivos, según su nombre y extensión. Gracias a esto, es mucho más sencillo navegar a través del sistema de ficheros.

- PyCharm: se trata de un IDE (Entorno de Desarrollo Integrado), usado para desarrollar programas que usen código Python y que pertenece a la compañía

JetBrains. Permite desplegar las aplicaciones desde el propio entorno y ofrece herramientas para el uso de los repositorios Git.

- Typora: editor utilizado para escribir archivos en formato markdown.
- Sublime Text 3: editor de texto muy ligero que permite escribir en cualquier lenguaje. En este proyecto se ha utilizado sobre todo para la edición de archivos con extensión .txt o cuando hacía falta analizar datos JSON, ya que, aunque se cierre el editor, mantiene los archivos que no se hayan guardado.
- MongoDB Compass: programa que nos permite tener una interfaz gráfica a la hora de trabajar con las bases de datos MongoDB. Podemos hacer conexiones a las bases de datos locales y a las que se encuentran en MongoDB Atlas.
- GoodNotes 5: aplicación móvil usada desde un Ipad que ha permitido realizar a mano los bocetos de las interfaces gráficas y posteriormente exportarlos a PDF.
- Postman: programa que permite realizar llamadas a las APIs. Usado sobre todo para interactuar con APIs externas como puede ser la de Imgur.
- MagicDraw: software utilizado para realizar los diagramas UML.
- Draw.io: aplicación que ha permitido realizar gran parte de los diagramas o gráficos debido a la facilidad de uso que presenta.

6.3.2 Herramientas web

- Heroku: se trata de una plataforma como servicios (PaaS), que nos permite desplegar los servicios de forma gratuita. Utilizada para desplegar los microservicios desarrollados en Flask.
- Netlify: aplicación web que ofrece servicios de alojamiento, utilizada para desplegar la web desarrollada con Angular.
- MongoDB Atlas: plataforma que nos permite tener nuestra base de datos MongoDB en la nube.

- Microsoft Office 365: suite de Microsoft que incluye las herramientas de Office. En nuestro caso se ha hecho uso de: Excel, Word, OneDrive, PowerPoint y OneNote.
- GitLab: herramienta web utilizado para el control de tareas y versiones del código.
- Figma: aplicación web utilizado para desarrollar las maquetas gráficas de la aplicación web.

7

Mantenimiento y Pruebas

7.1 Pruebas

Al finalizar de desarrollar el código de cada una de las iteraciones (una vez que se integraba la rama feature_X con release), se procedía a probar el sistema de forma exhaustiva siguiendo los casos de prueba generados en la fase de análisis y especificación de requisitos.

Para ello, se ha usado el lenguaje gherkin, con el objetivo de poder automatizar estas pruebas en un futuro, combinando Cucumber con algún entorno de pruebas como puede ser Selenium.

La duración del proyecto ha impedido poder profundizar más en este aspecto, ya que realizar un plan de pruebas exhaustivo podría ser incluso el contenido principal de un

trabajo entero, debido a la dificultad que esto supone y la cantidad de tipos de pruebas que habría que hacer.

En el Apéndice E podemos ver las tablas de las pruebas realizadas correspondientes a los casos de prueba definidos.

Cuando se realizaba una prueba y el resultado obtenido no era igual al esperado se procedía a abrir una tarea de bug en Gitlab. Una vez resuelto, e integrado en la rama que correspondía, se procedía a probar de nuevo ese caso de prueba. Si observamos la tabla, podemos ver cómo se han hecho pruebas tanto en entornos locales como en entornos remotos accediendo a versiones del sistema desplegadas en Heroku y Netlify.

7.2 Mantenimiento

Respecto al mantenimiento, con la estrategia de ramas que hemos planteado en el apartado "4.1.2 Estrategia de ramas", vemos que podemos resolver los bugs que se descubran una vez desplegado el sistema. Además, para mantener el sistema a lo largo del tiempo, se procederá a generar versiones en los endpoints de los microservicios cuando cambien campos de las *requests* o *responses* (accediendo a las diferentes versiones utilizando la cabecera "x-api-version", siendo por defecto la 1.0 si no se incluye esta cabecera), para seguir dando soporte a los clientes que utilizasen las versiones antiguas, pero a la vez seguir mejorando el sistema.

8

Conclusiones

8.1 Resultados obtenidos

A través de este trabajo se ha conseguido desarrollar un entorno web que cumple con todo los objetivos planteados inicialmente, es decir, una aplicación web basada en microservicios, capaz de generar simulaciones del comportamiento de ciudadanos en una zona urbana, en base a diversos parámetros relacionados con el nivel de delincuencia, la población o el porcentaje de agentes policiales. Estos parámetros pueden particularizarse para las diferentes zonas (o regiones) del mapa global de la simulación. Además, el sistema permite que el usuario pueda ver la simulación, es decir, la interacción entre los diferentes agentes, en un mapa . Los resultados de la simulación puede almacenarse y descargarse en un archivo CSV, y generar gráficos asociados a estos datos. Esto permite poder realizar análisis más exhaustivos de los resultados.

Teniendo en cuenta la limitación temporal que marca el tiempo de desarrollo de un TFG, se ha conseguido realizar una aplicación bastante completa, ya que además de lo comentado, incluye aspectos tecnológicos como autenticación a través del Oauth con

Google y la gestión de tokens JWT para hacer más seguros los servicios, documentación de la API usando Swagger, un sistema de internacionalización para aplicar un soporte de varios idiomas o un asistente (*wizard*) para la entrada de los datos relativos a las simulaciones.

A todo esto hay que incluir que la aplicación tiene un diseño de interfaz adaptable (responsive), y que se ha desplegado en un entorno de producción, siendo accesible desde cualquier navegador a través de su URL (https://tfg-victoria.netlify.app). Además, gracias a haber usado un diseño arquitectónico basado en microservicios, podríamos continuar añadiendo nuevas funcionalidades a la aplicación, de forma relativamente sencilla.

8.2 Conocimientos adquiridos

Durante el desarrollo del proyecto, se han adquirido bastantes conocimientos y competencias que no se habían conseguido durante la etapa de estudio del grado.

A pesar de que contamos con la asignatura "Modelado y Diseño del Software", en ésta no se desarrollaron modelos para más adelante poderlos aplicarlos a un problema real. En este caso, al realizar una programación basada en agentes, la labor de modelar este sistema ha sido muy importante.

El lenguaje Python tampoco se había utilizado hasta ahora más allá que para hacer unas pequeñas prácticas de la asignatura "Seguridad en Servicios y Aplicaciones", por lo que se ha tenido que aprender las bases del lenguaje a medida que se iba desarrollando el código de la aplicación.

Respecto al cliente de la aplicación, se había estado usando Angular a un nivel muy básico en los meses previos, y esto ha servido para aumentar el conocimiento de forma

significativa, ya que se ha tenido que enfrentar problemas que surgen solo al desarrollar aplicaciones con un cierto tamaño.

Por otro lado, respecto a la gestión, se desconocía que gracias al uso de los archivos .yaml se podían desplegar aplicaciones de forma automática en servidores remotos simplemente integrando código en ciertas ramas.

Al realizar un proyecto así, se ha visto la importancia de realizar una buena planificación, llevar un control de tareas hechas, el tiempo dedicado a ellas, realizar una buen análisis y especificación de requisitos y, sobre todo, el uso de bocetos e interfaces gráficas para definirla antes de comenzar a programas, ya que gracias a esto se ha podido ahorrar mucho tiempo de programación. También la importancia de las pruebas, ya que al tener que escribir tantas líneas de código es muy fácil equivocarse en cualquier punto y no darnos cuenta de que hay algo que no funciona como se esperaba.

8.3 Dificultades

Como se ha mencionado antes, se partía de cero respecto al lenguaje Python, y esto en muchas ocasiones ha dificultado el desarrollo de los microservicios backend, ya que soluciones a problemas que ahora se ven con claridad debido al conocimiento adquirido, se tardaba mucho en conseguirlas e incluso había ocasiones en las que no ha sido posible y se ha visto más adelante, al avanzar el desarrollo, cuál era la mejor forma de solucionarlo.

Por otro lado, que Flask sea una tecnología relativamente nueva, hace que aún no se dispongan de ciertos mecanismos que son bastantes necesarios, como puede ser una capa para el manejo de la base de datos. Además, al buscar en la web ejemplos de aplicaciones desarrolladas en Flask, la mayoría eran ejemplos muy simples, por lo que

se ha tenido que desarrollar el código desde cero, sin poder partir de plantillas o estructuras creadas por gente con experiencia previa.

Otra gran dificultad fue descubrir cómo se iba a implementar la funcionalidad de visualizar el mapa de la simulación, ya que teníamos el backend y frontend separados, y, por lo tanto, debíamos ser capaces de trasladar los datos relativos al grid desde el servidor hasta el cliente.

8.3 Mejoras y Líneas Futuras

Una de las mejoras que se podría hacer es ejecutar las simulaciones en segundo plano, ya que éste es un proceso algo pesado y tarda bastante en hacerse. Otra de las opciones podría ser poder visualizar la simulación mientras se realiza, y que por lo tanto no haya tiempos de espera.

Respecto a las funcionalidades, se podría añadir la opción de poder comparar los datos y gráficas de varias simulaciones, ya que podría ser bastante interesante de cara a ver cómo cambian los resultados en función de los parámetros de entrada y añadir la generación de API Key para que otros desarrolladores puedan utilizar el servidor a través de ese método de autorización.

A pesar de que se han realizado pruebas siguiendo los casos de prueba generados, éstas podrían haber sido mucho más exhaustivas, realizando análisis relativos a los distintos tipos de pruebas y automatizándolas.

En cuanto a la mejorar del sistema de simulación, sería muy interesante poder poseer datos reales de los delitos que se han producido en una zona, y usar esto para poder validar el modelo y mejorarlo en base a los resultados. Es decir, combinar la ciencia de datos con la simulación.

Bibliografía

Silva, P. C., Batista, P. V., Lima, H. S., Alves, M. A., Guimarães, F. G., & Silva, R. C. (2020). COVID-ABS: An agent-based model of COVID-19 epidemic to simulate health and economic effects of social distancing interventions. *Chaos, Solitons & Fractals*, *139*, 110088.

Angione, C., Silverman, E., & Yaneske, E. (2020). Using Machine Learning to Emulate Agent-Based Simulations. *arXiv preprint arXiv:2005.02077*.

Agent-based modeling in Python 3. (n.d.). https://mesa.readthedocs.io/en/stable/ (accedido el 26 de junio de 2021)

Welcome to Flask. (n.d.). https://flask.palletsprojects.com/en/1.1.x/ (accedido el 26 de junio de 2021)

Welcome to Flask-REST Plus's documentation!. (n.d.). https://flask-restplus.readthedocs.io/en/stable/ (accedido el 26 de junio de 2021)

Angular - Introduction to the Angular Docs. (n.d.). https://angular.io/docs (accedido el 26 de junio de 2021)

Wahlin, D. (2019, January 15). Angular Architecture and Best Practices. https://app.pluralsight.com/library/courses/angular-architecture-best-practices (accedido el 26 de junio de 2021)

Heath, M. (2019, November 01). Microservices Fundamentals. https://app.pluralsight.com/courses/microservices-fundamentals/ (accedido el 26 de junio de 2021)

Apéndice A

Manual de Usuario

Iniciar Sesión

Para registrarse se sigue el mismo procedimiento que el que vamos a exponer a continuación.

Figura A.1 Página de inicio.

Si pulsamos sobre “Simulaciones” en la barra de navegación, aparecerá una ventana modal, donde se indica que hace falta iniciar sesión para acceder a ese contenido.

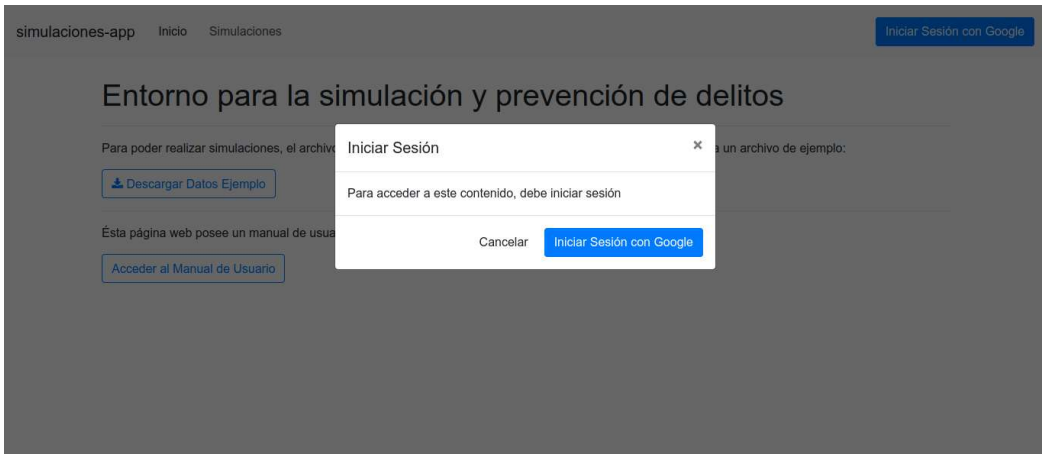


Figura A.2 Modal de inicio de sesión.

Si pulsamos el botón "Iniciar Sesión con Google", podremos acceder a la aplicación usando nuestra cuenta de Google.

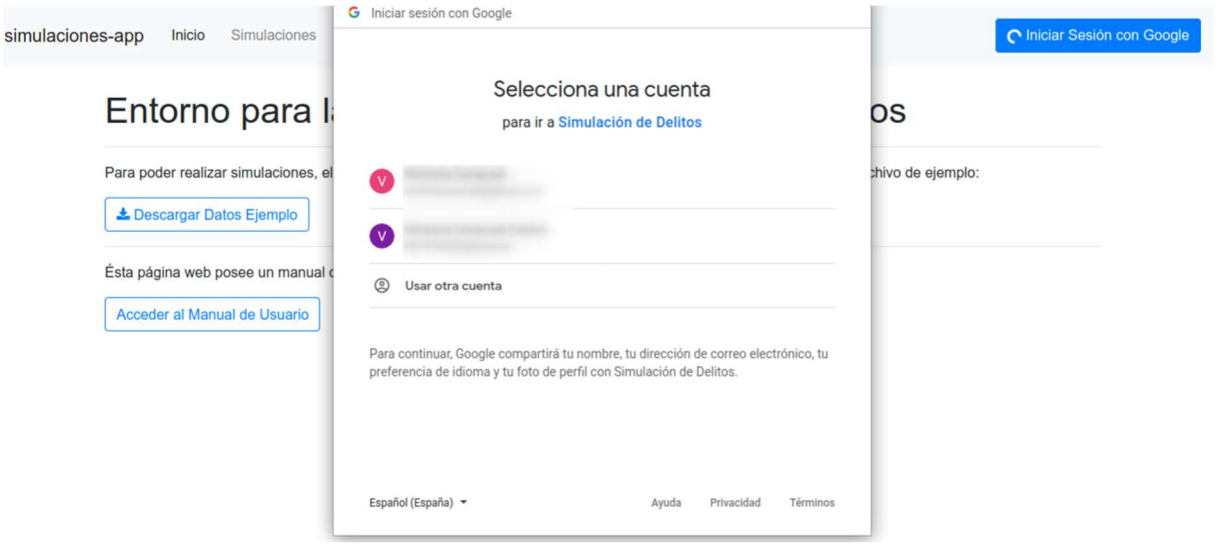


Figura A.3: Iniciar sesión con Google.

Al iniciar sesión, se cargarán nuestros datos en la aplicación

Figura A.4: Inicio de sesión con éxito

Cerrar Sesión

Tendremos que pulsar sobre el nombre o icono de usuario, y de desplegará un menú que nos permite cerrar sesión.

Figura A.5: Submenú de la barra de navegación

Acceder al perfil

Si en el submenú que aparece en la figura A.5 pulsamos sobre “Ajustes Cuenta”, podremos acceder al perfil del usuario.

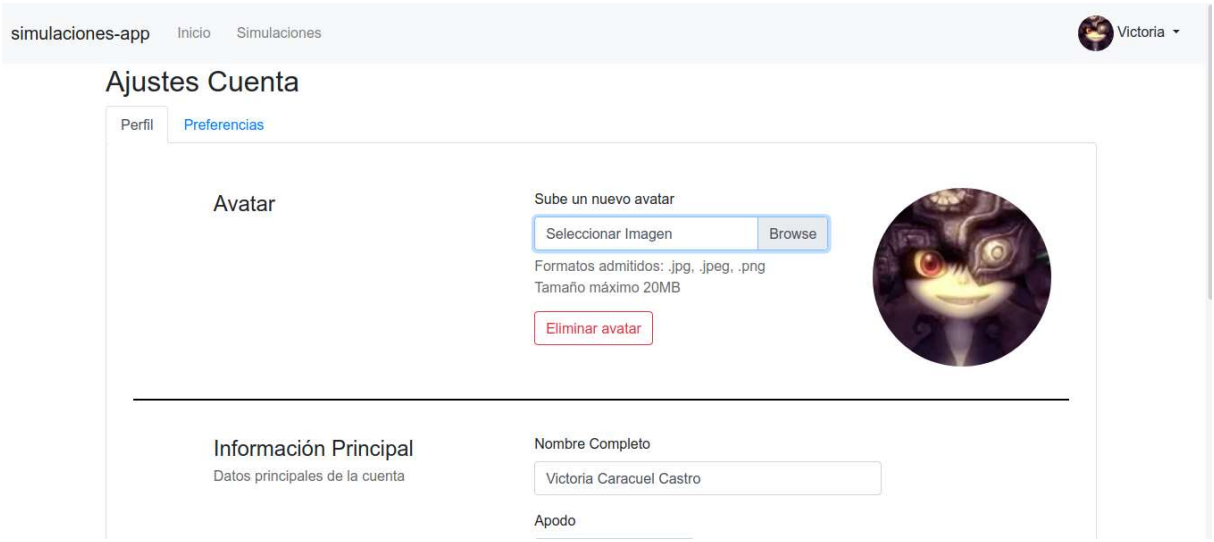


Figura A.6: Perfil del usuario.

Eliminar imagen de perfil

Si pulsamos en el botón "Eliminar avatar", aparecerá una ventana para confirmar la eliminación de la imagen del perfil del usuario.

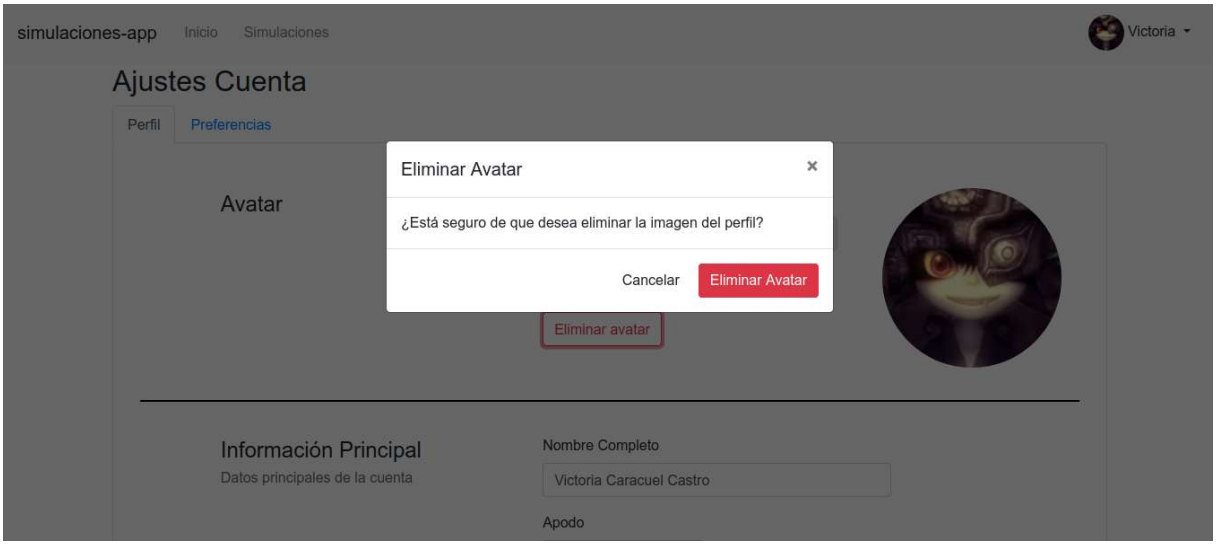


Figura A.7: Eliminar imagen perfil.

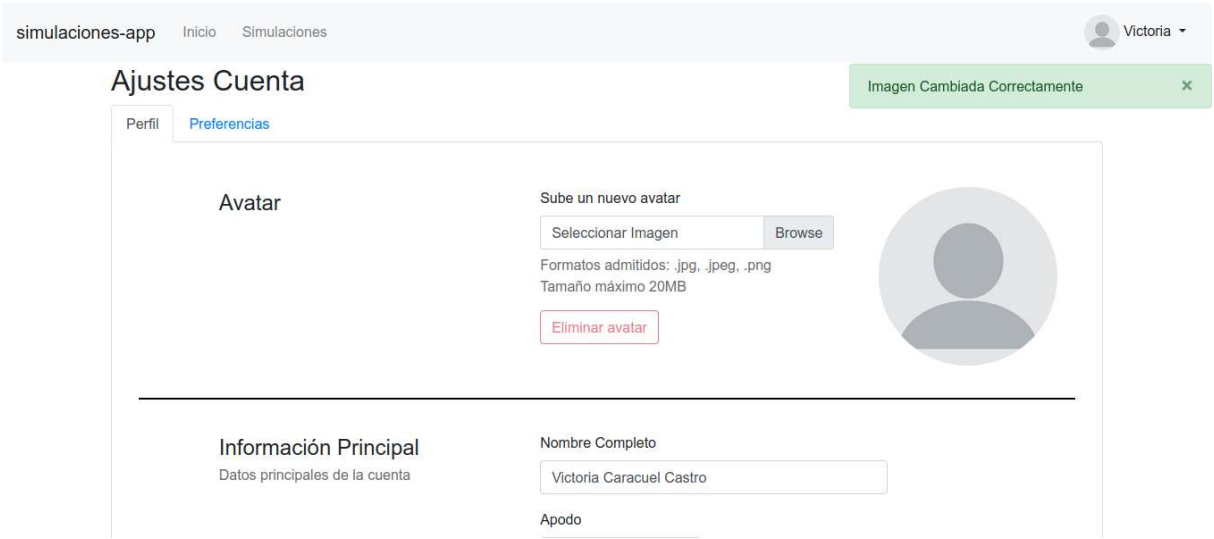


Figura A.8: Imagen del perfil eliminada correctamente.

Cambiar Imagen

Para cambiar la imagen, simplemente tendremos que pulsar en el botón “Browse” y seleccionar la imagen que queramos cargar.

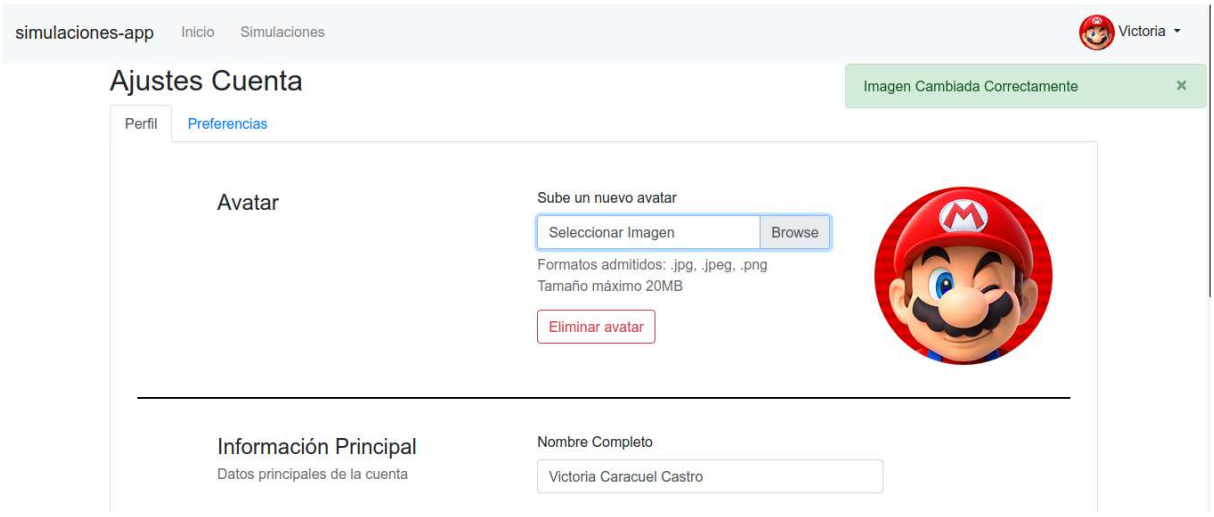


Figura A.8: Imagen del perfil cambiada correctamente.

Cambiar Datos

Para cambiar los datos asociados a la cuenta, simplemente deberemos cambiar la información que aparece en pantalla y pulsar el botón “Guardar Cambios”

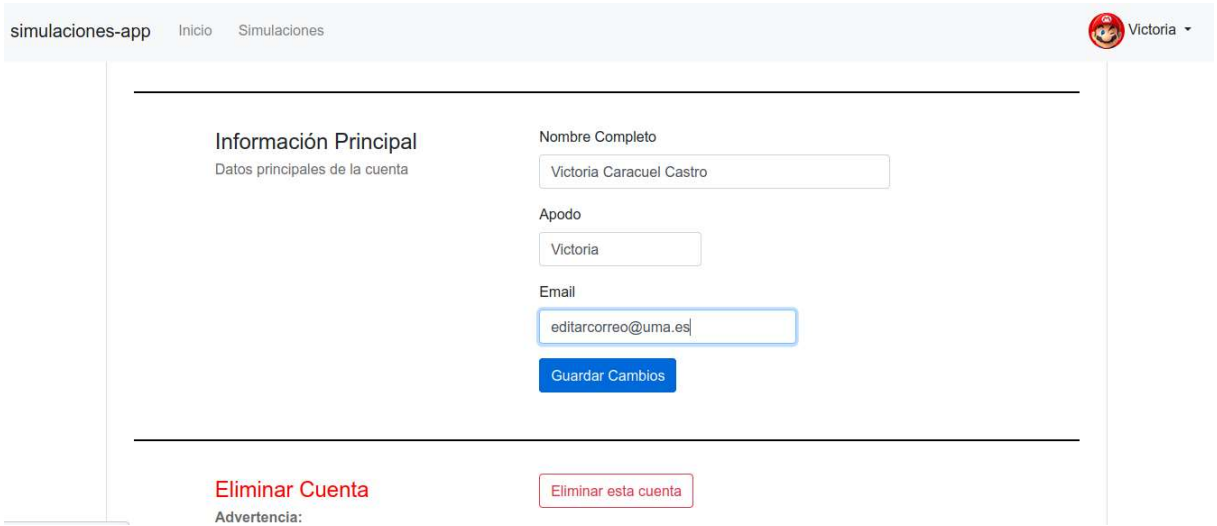


Figura A.9: Editar datos del perfil

Eliminar Cuenta

Para eliminar la cuenta, hay que pulsar el botón "Eliminar esta cuenta" que vemos en la figura A.9 y una vez que salga la ventana de confirmación, darle a "Eliminar Cuenta"

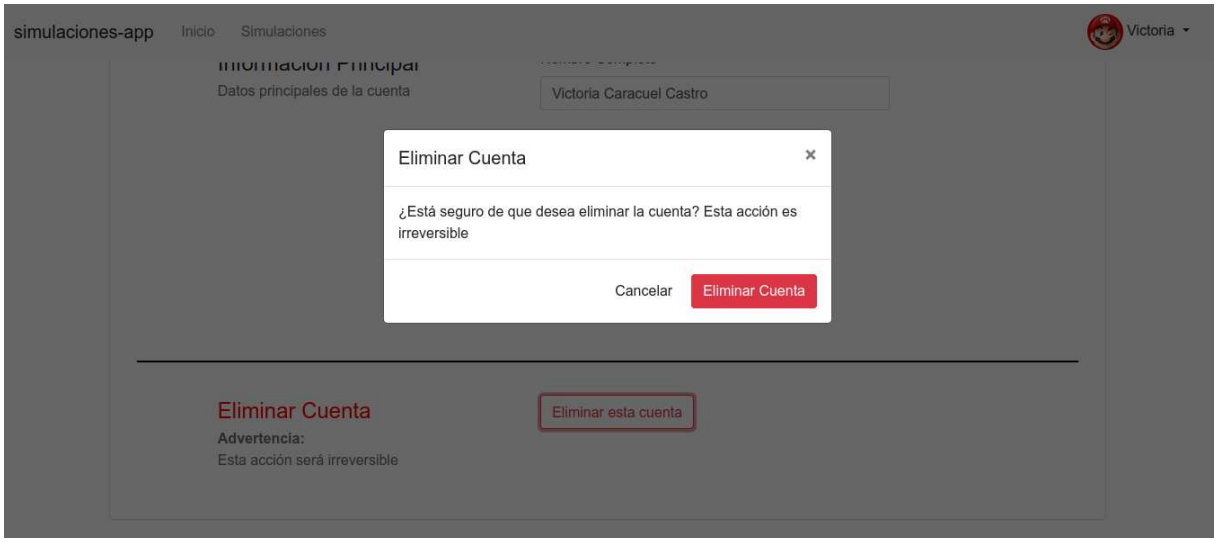


Figura A.10: Confirmación para eliminar cuenta.

Cambiar Idioma

Para cambiar el idioma, debemos pulsar en la pestaña “Preferencias” dentro del perfil del usuario. Una vez que estemos en la zona de preferencias, debemos seleccionar un idioma y darle a “Guardar Cambios”

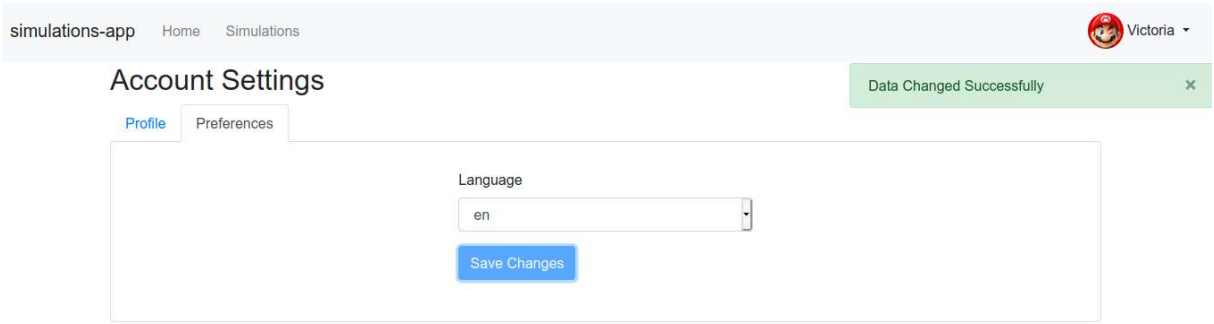


Figura A.11: Idioma cambiado correctamente.

Ver listado de Simulaciones

Dentro de la ventana de “Simulaciones” tenemos acceso al listado de simulaciones realizadas por el usuario.

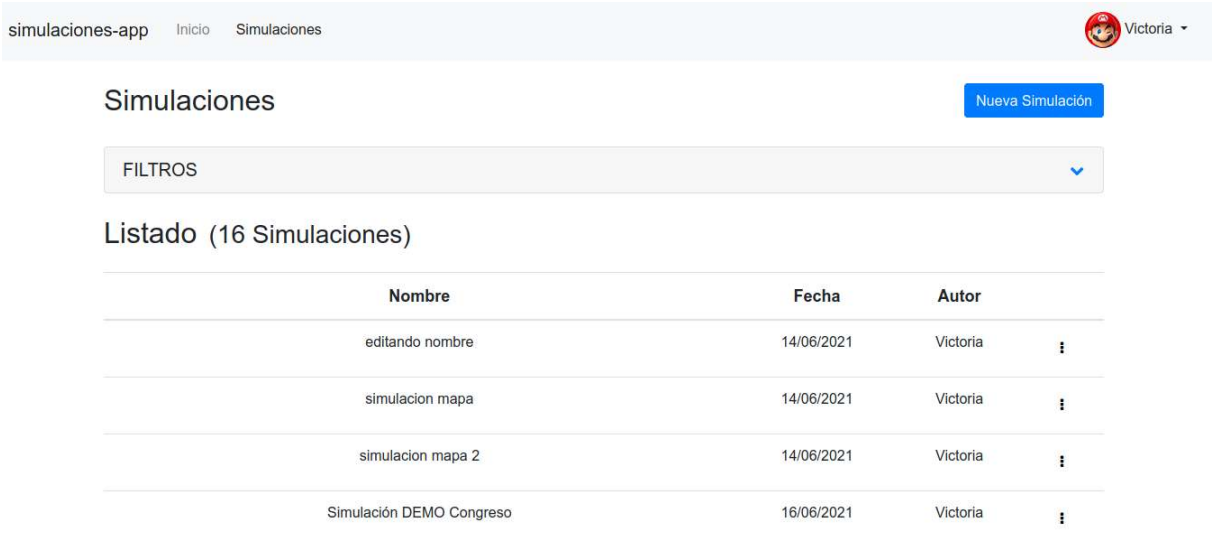


Figura A.12: Listado de simulaciones

Al estar en el listado, podremos filtras las simulaciones por nombre (se mostrarán las simulaciones que contengan en su nombre el texto introducido, sin distinguir entre letras mayúsculas y minúsculas).

Filtrar Simulaciones

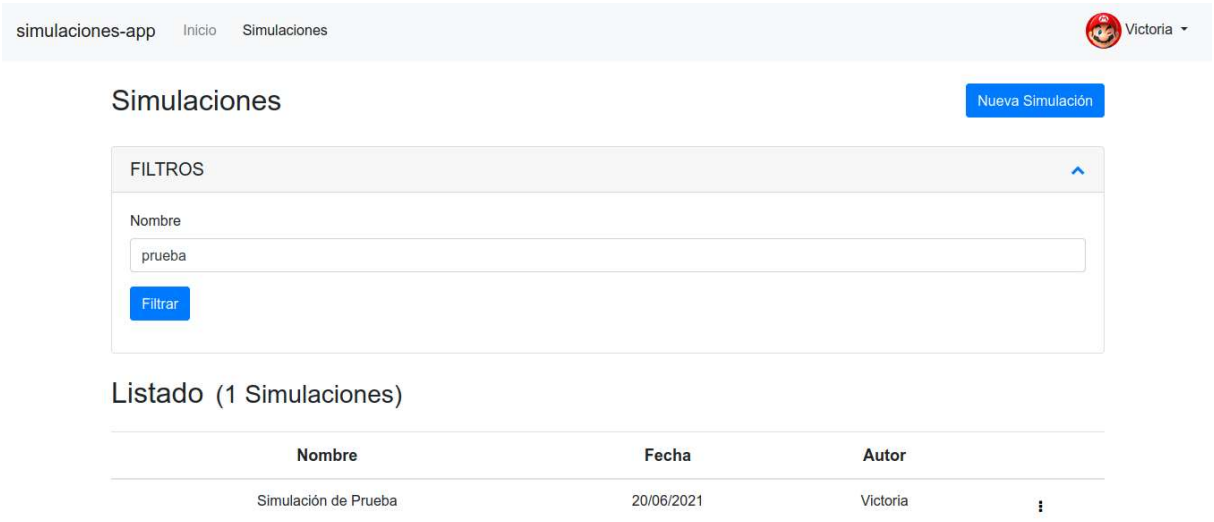


Figura A.13: Filtrar las simulaciones

Si pulsamos en los tres puntos que aparece en la zona de la derecha del listado, tendremos acceso a varias opciones: “Ver Detalle”, “Descargar CSV” y “Eliminar”

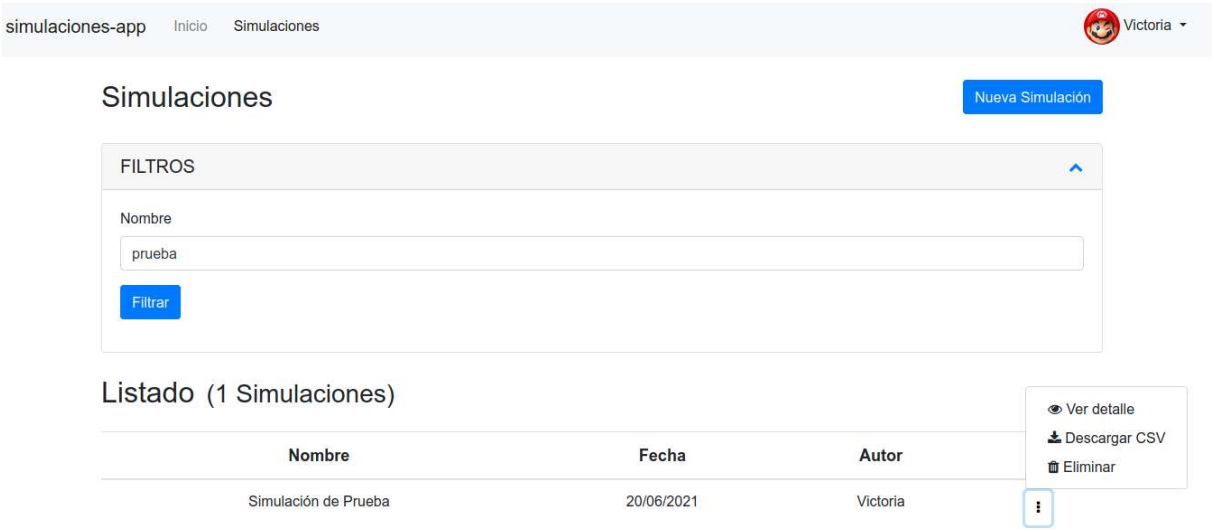


Figura A.14: Acciones Simulación.

Al darle a “Eliminar”, nos aparecerá una ventana de confirmación para borrar de forma definitiva.

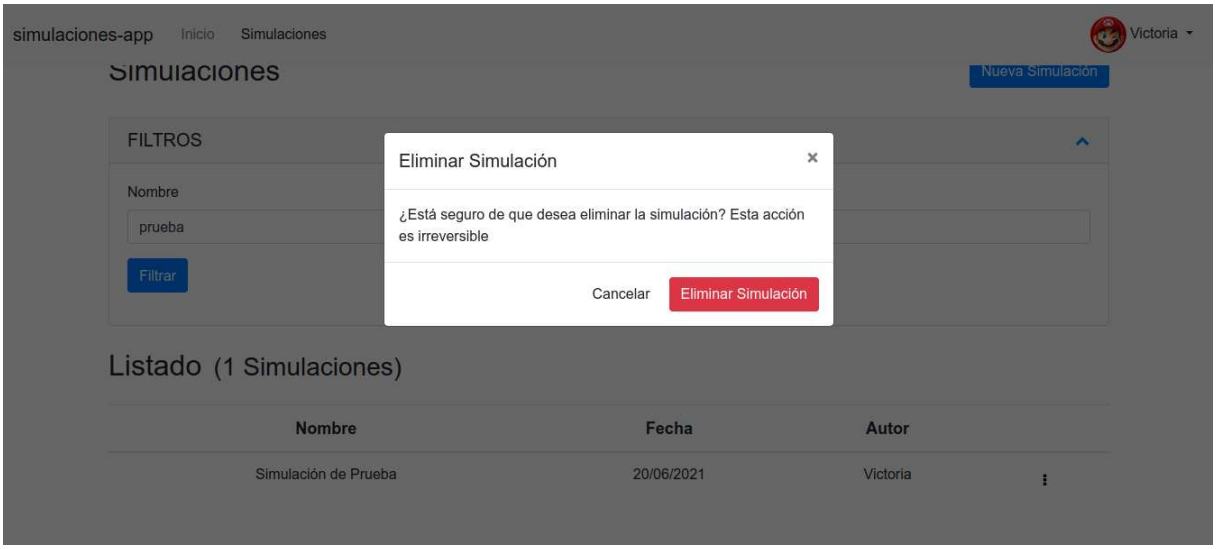


Figura A.15: Modal confirmación de eliminar Simulación.

Detalle Simulación

Si en la pantalla de la figura A.14 le damos a "Ver detalle", la aplicación mostrará el contenido de la figura A.16 y la figura A.17. Si pulsamos sobre “Guardar Cambios” tras modificar el nombre o la descripción, modificaremos éstos datos en la simulación.

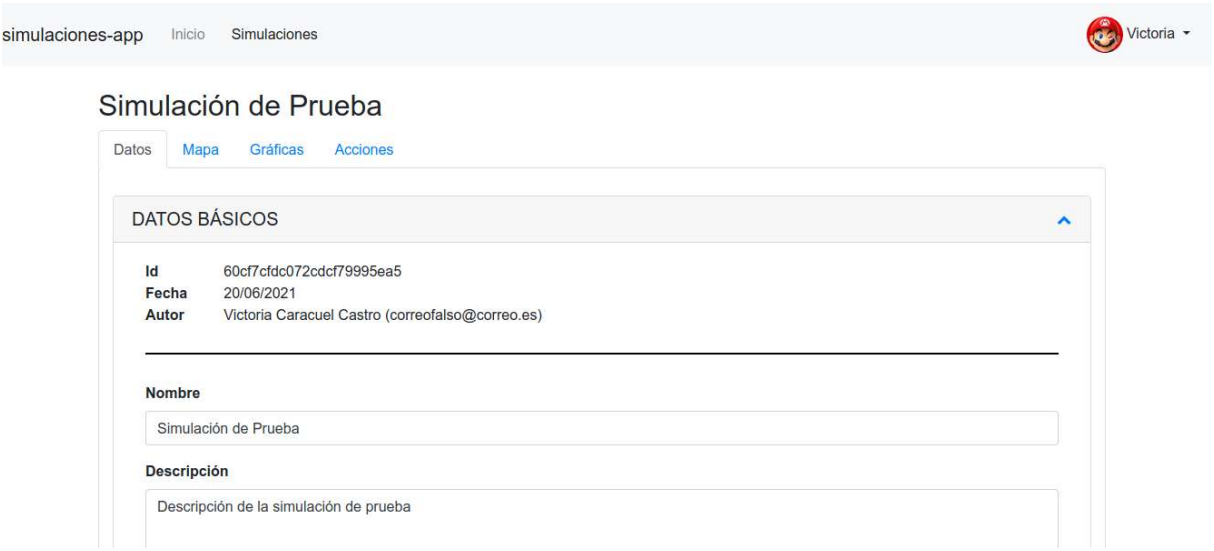


Figura A.16: Datos básicos del detalle de una simulación

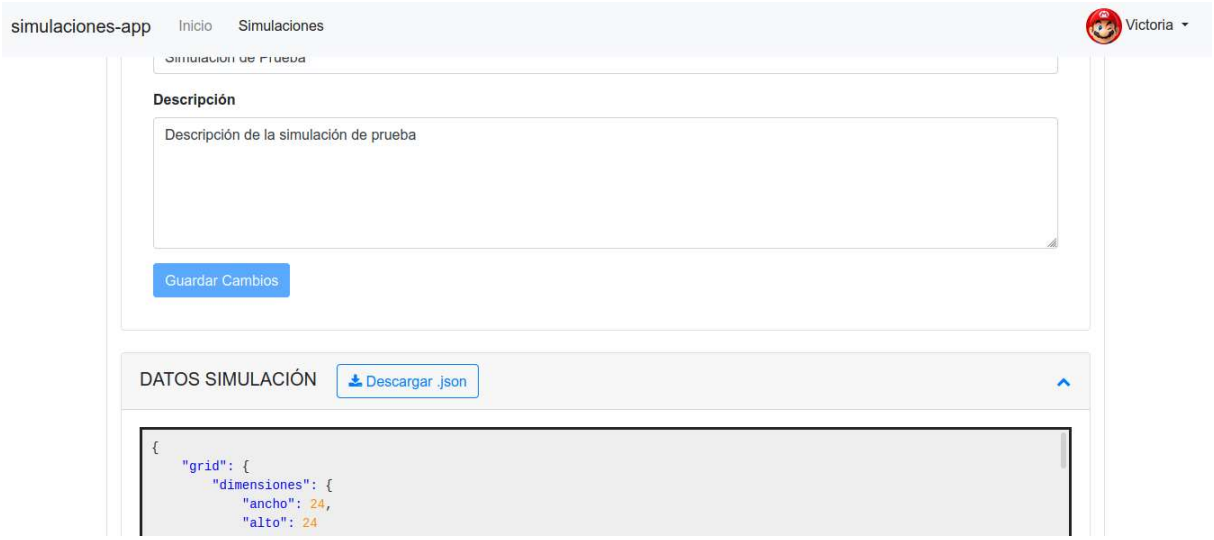


Figura A.17: Datos de simulación del detalle de una simulación.

Si en la figura A.17 pulsamos el botón "Descargar .json", obtendremos un archivo JSON con los parámetros de entrada de la simulación.

Mapa Simulación

Para tener acceso al mapa de la simulación, debemos pulsar sobre la pestaña "Mapa".

En esta pestaña tenemos acceso a varios controles, como podemos ver en las figuras A.19 y A.20. Los controles que aparecen en la zona de la derecha son para aumentar,

disminuir o resetear el tamaño del mapa. Los controles de la izquierda permiten avanzar en los pasos de la simulación, pudiendo ir hacia delante o hacia atrás, tanto de forma automática como manual.

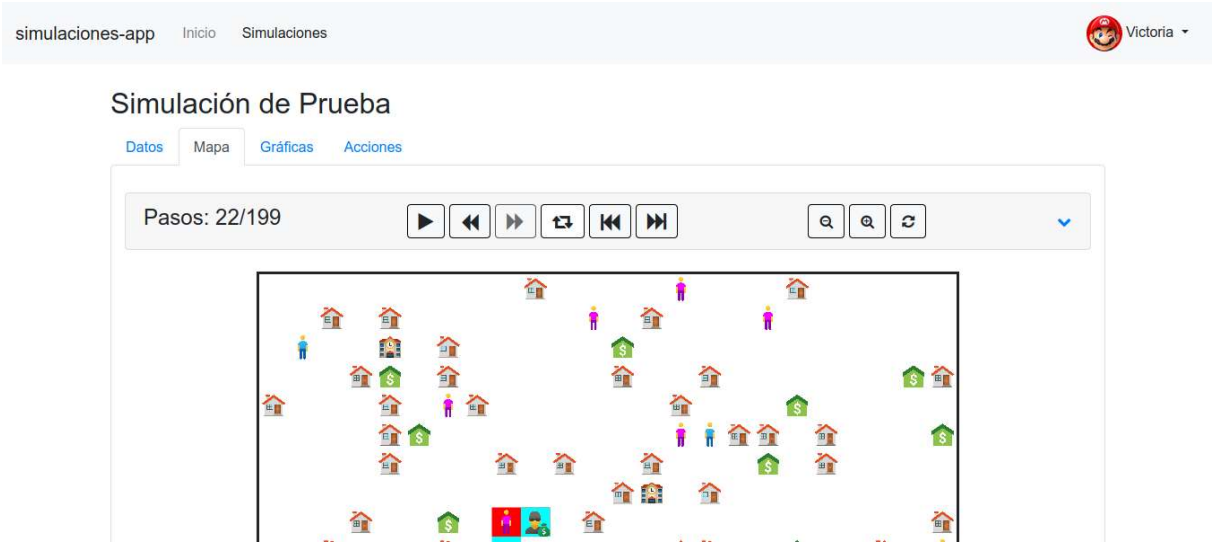


Figura A.18: Mapa de la simulación.

Figura A.19: Controles automáticos de la simulación

Figura A.20: Controles manuales de la simulación.

Ver Gráficas

Pulsando sobre la pestaña "Gráficas" podremos ver las gráficas generadas a partir de los datos de salida de la simulación (figura A.22 y A.23).

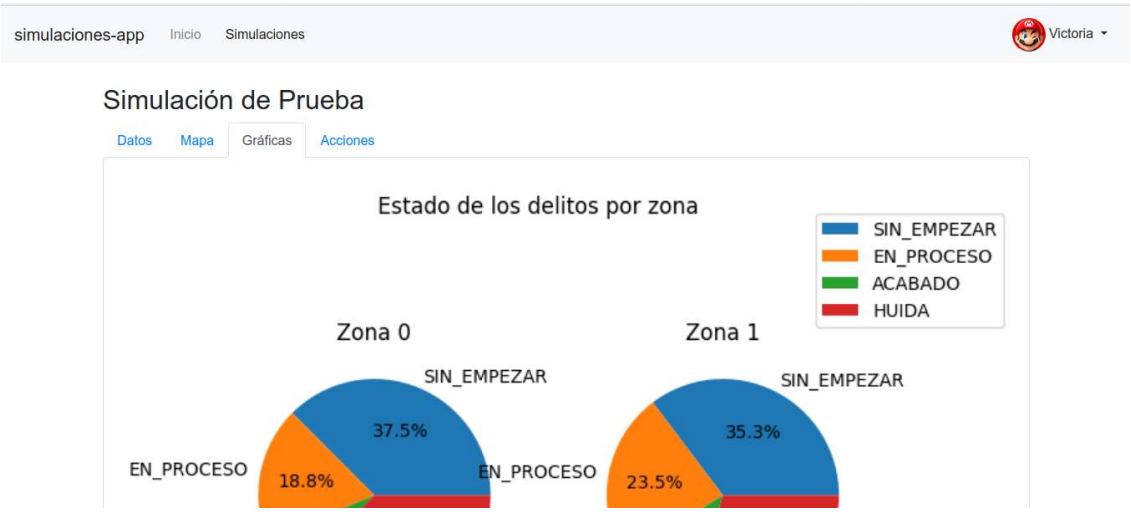


Figura A.21: Pestaña de gráficas.

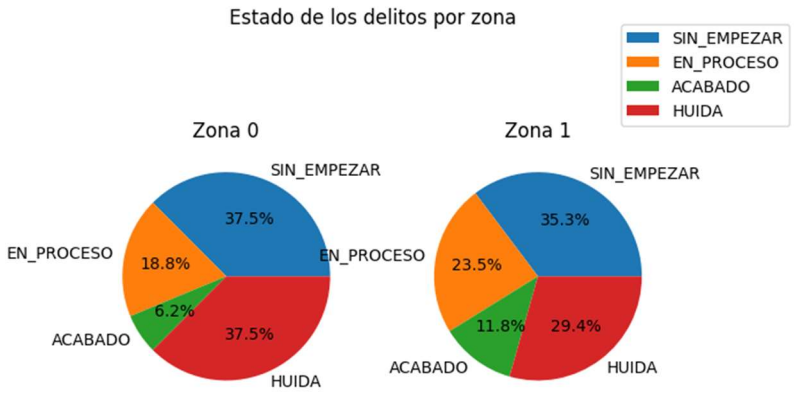


Figura A.22: Gráfica 1.

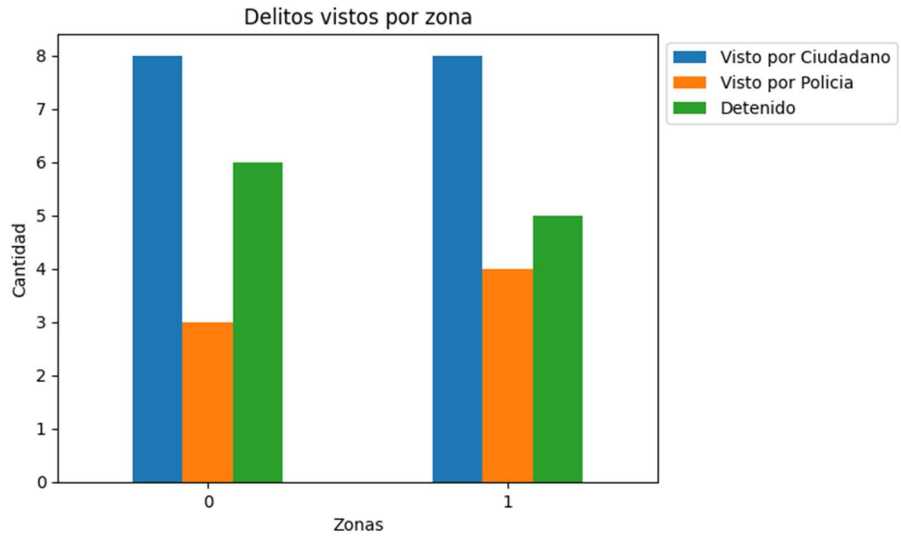
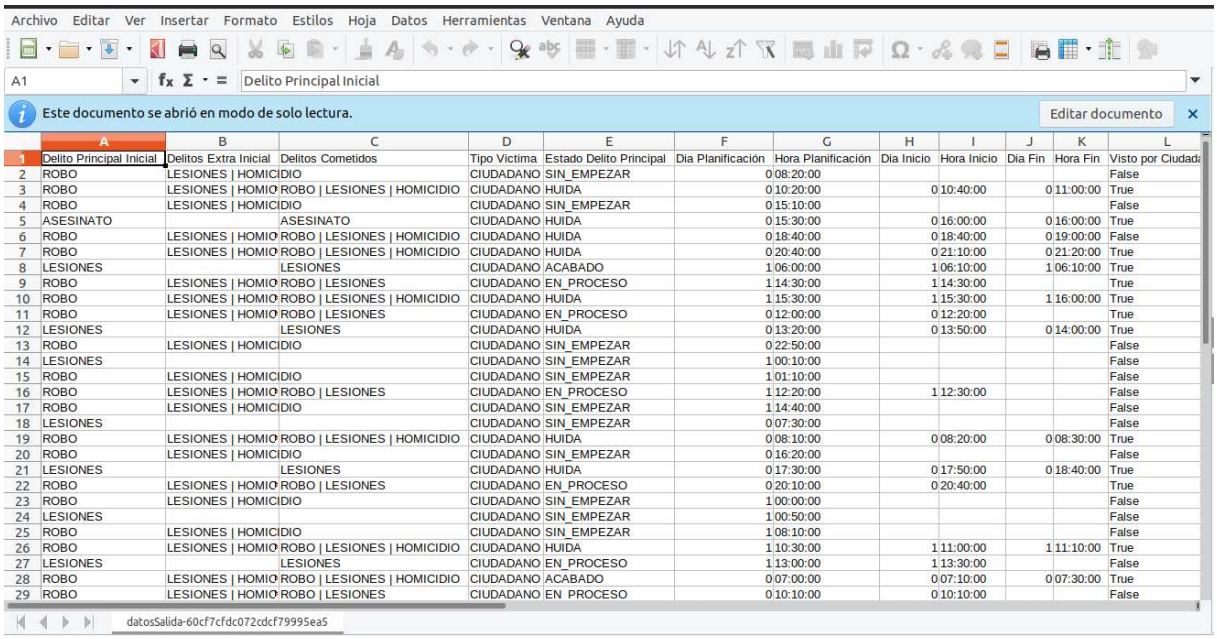


Figura A.23: Gráfica 2.

Obtener los Datos de Salida

Pulsando en la pestaña "Acciones" podremos obtener los datos de salida en formato CSV o borrar la simulación.

Figura A.24: Pestaña de acciones.



Archivo Editar Ver Insertar Formato Estilos Hoja Datos Herramientas Ventana Ayuda

A1 | Delito Principal Inicial

Este documento se abrió en modo de solo lectura. Editar documento

	A	B	C	D	E	F	G	H	I	J	K	L
1	Delito Principal Inicial	Delitos Extra Inicial	Delitos Cometidos	Tipo Victima	Estado Delito Principal	Dia Planificación	Hora Planificación	Dia Inicio	Hora Inicio	Dia Fin	Hora Fin	Visto por Ciudad
2	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	0	08:20:00					False
3	ROBO	LESIONES | HOMICI	ROBO | LESIONES | HOMICIDIO	CIUDADANO	HUIDA	0	10:20:00	0	10:40:00	0	11:00:00	True
4	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	0	15:10:00					False
5	ASESINATO		ASESINATO	CIUDADANO	HUIDA	0	15:30:00	0	16:00:00	0	16:00:00	True
6	ROBO	LESIONES | HOMICI	ROBO | LESIONES | HOMICIDIO	CIUDADANO	HUIDA	0	18:40:00	0	18:40:00	0	19:00:00	False
7	ROBO	LESIONES | HOMICI	ROBO | LESIONES | HOMICIDIO	CIUDADANO	HUIDA	0	20:40:00	0	21:10:00	0	21:20:00	True
8	LESIONES		LESIONES	CIUDADANO	ACABADO	1	06:00:00	1	06:10:00	1	06:10:00	True
9	ROBO	LESIONES | HOMICI	ROBO | LESIONES	CIUDADANO	EN_PROCESO	1	14:30:00	1	14:30:00			True
10	ROBO	LESIONES | HOMICI	ROBO | LESIONES | HOMICIDIO	CIUDADANO	HUIDA	1	15:30:00	1	15:30:00	1	16:00:00	True
11	ROBO	LESIONES | HOMICI	ROBO | LESIONES	CIUDADANO	EN_PROCESO	0	12:00:00	0	12:20:00			True
12	LESIONES		LESIONES	CIUDADANO	HUIDA	0	13:20:00	0	13:50:00	0	14:00:00	True
13	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	0	22:50:00					False
14	LESIONES			CIUDADANO	SIN_EMPEZAR	1	00:10:00					False
15	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	1	01:10:00					False
16	ROBO	LESIONES | HOMICI	ROBO | LESIONES	CIUDADANO	EN_PROCESO	1	12:20:00	1	12:30:00			False
17	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	1	14:40:00					False
18	LESIONES			CIUDADANO	SIN_EMPEZAR	0	07:30:00					False
19	ROBO	LESIONES | HOMICI	ROBO | LESIONES | HOMICIDIO	CIUDADANO	HUIDA	0	08:10:00	0	08:20:00	0	08:30:00	True
20	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	0	16:20:00					False
21	LESIONES		LESIONES	CIUDADANO	HUIDA	0	17:30:00	0	17:50:00	0	18:40:00	True
22	ROBO	LESIONES | HOMICI	ROBO | LESIONES	CIUDADANO	EN_PROCESO	0	20:10:00	0	20:40:00			True
23	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	1	00:00:00					False
24	LESIONES			CIUDADANO	SIN_EMPEZAR	1	00:50:00					False
25	ROBO	LESIONES | HOMICI	DIO	CIUDADANO	SIN_EMPEZAR	1	08:10:00					False
26	ROBO	LESIONES | HOMICI	ROBO | LESIONES | HOMICIDIO	CIUDADANO	HUIDA	1	10:30:00	1	11:00:00	1	11:10:00	True
27	LESIONES		LESIONES	CIUDADANO	EN_PROCESO	1	13:00:00	1	13:30:00			False
28	ROBO	LESIONES | HOMICI	ROBO | LESIONES | HOMICIDIO	CIUDADANO	ACABADO	0	07:00:00	0	07:10:00	0	07:30:00	True
29	ROBO	LESIONES | HOMICI	ROBO | LESIONES	CIUDADANO	EN_PROCESO	0	10:10:00	0	10:10:00			False

datosSalida-60cf7cfdc072cdcf79995ea5

Figura A.25: Ejemplo de datos de salida de la simulación.

Crear Simulación

Si en la figura A.12 pulsamos sobre "Nueva Simulación", aparecerá un asistente que nos permitirá generar una simulación. Para poder ir a la pantalla de resumen, debemos rellenar correctamente todos los datos.

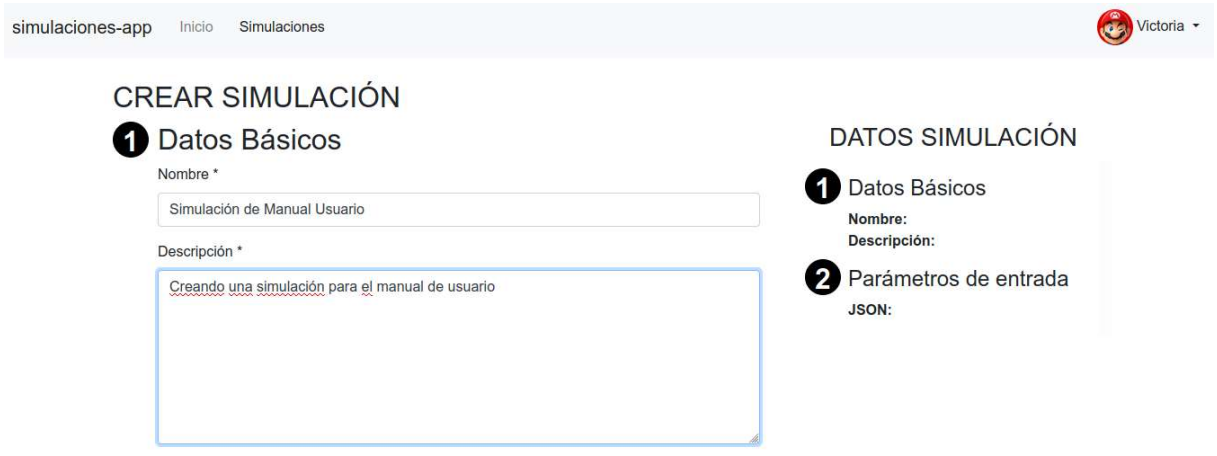


Figura A.26: Crear simulación. Datos básicos.

Como podemos ver en la figura A.27, la aplicación permite cargar los datos directamente desde un archivo JSON. Una vez cargado el archivo, podemos modificar los datos que aparecen sobre la caja.

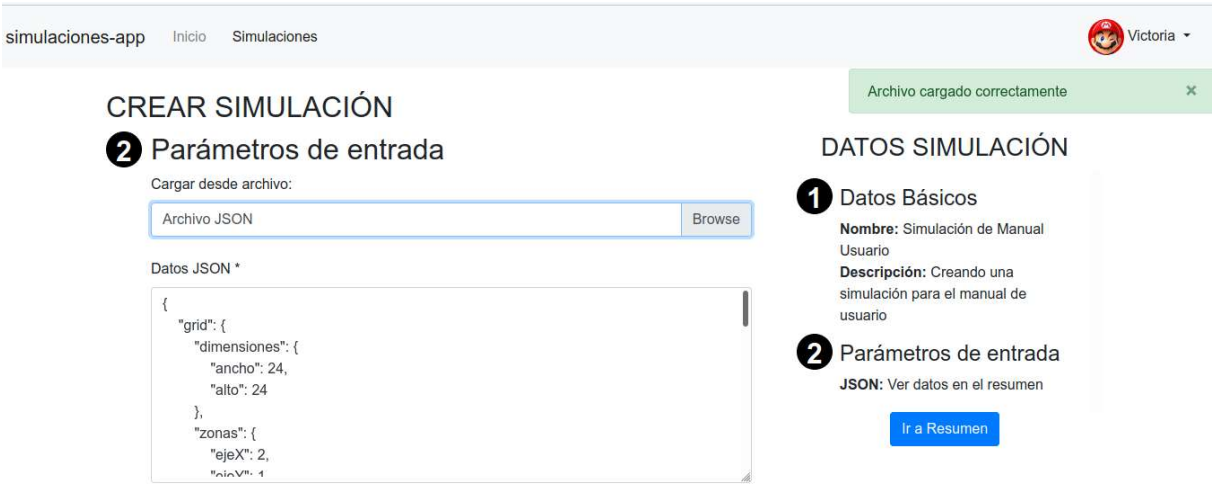


Figura A.27: Crear simulación. Parámetros de entrada.

En las figuras A.28 y A.29 mostramos el resumen de los datos. Al darle a "Crear Simulación", la aplicación procederá a generar la simulación con los datos de entrada introducidos. Si pulsamos sobre los círculos que contienen los números, volvemos a los pasos correspondientes para editarlos.

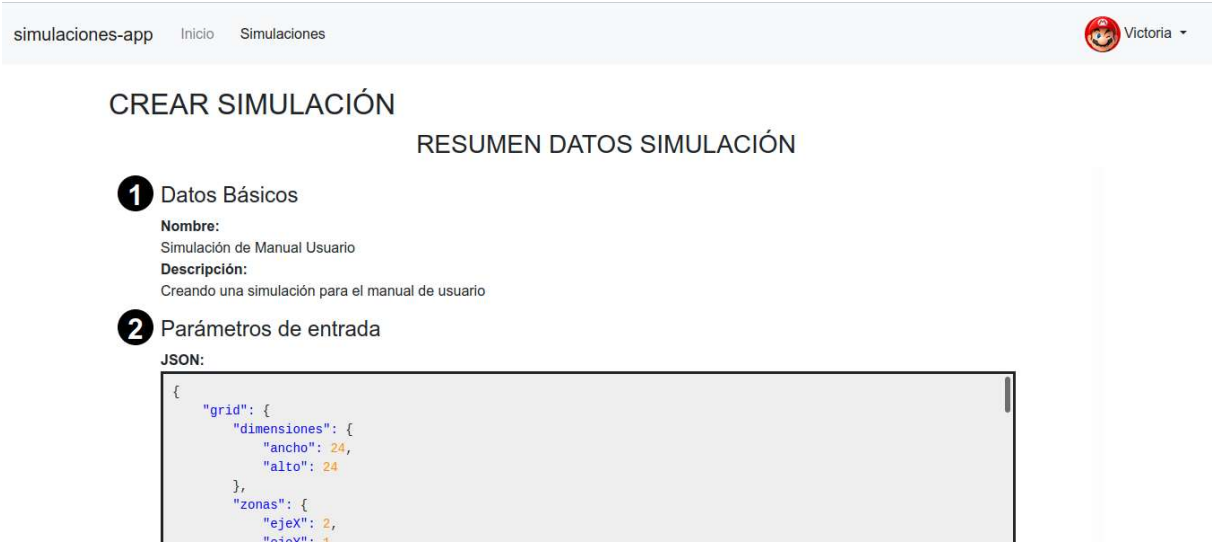


Figura A.28: Crear simulación. Resumen.

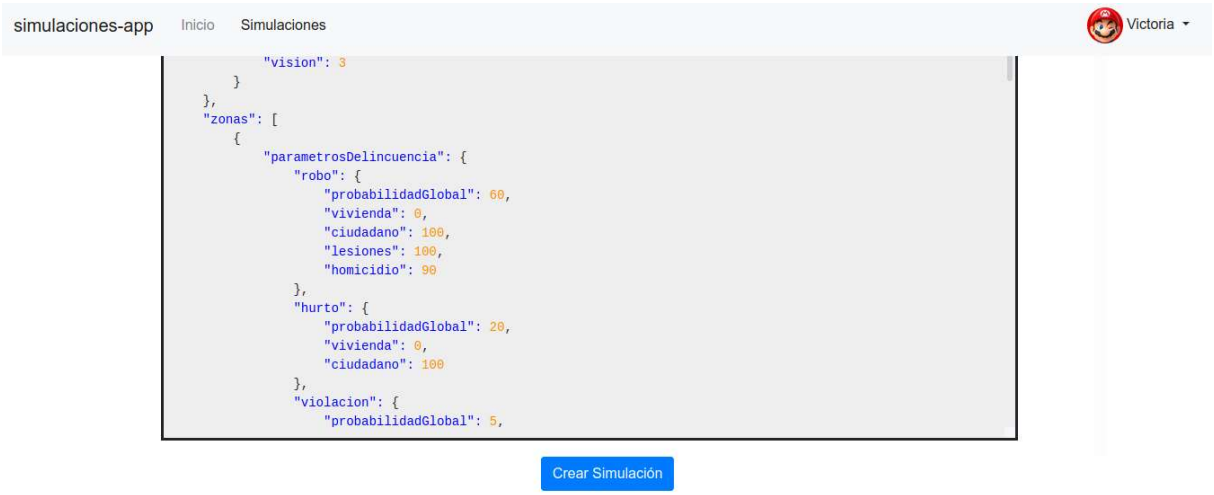


Figura A.29: Crear simulación. Botón.

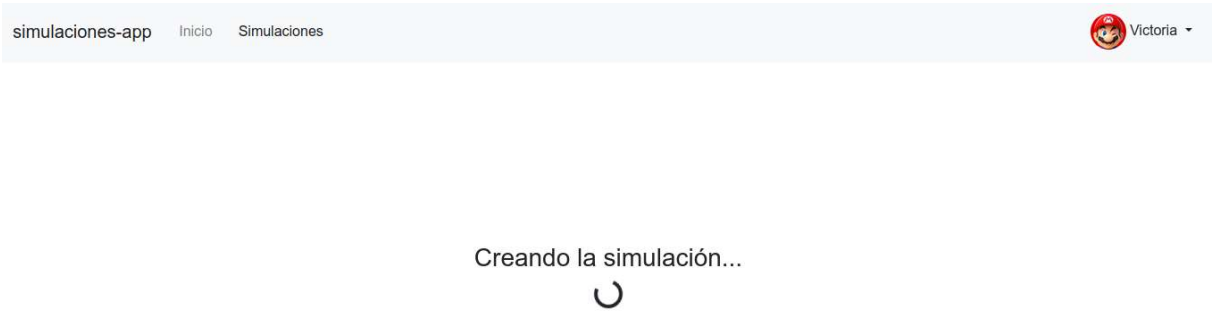


Figura A.30: Aplicación creando la simulación.

Apéndice B

Bocetos

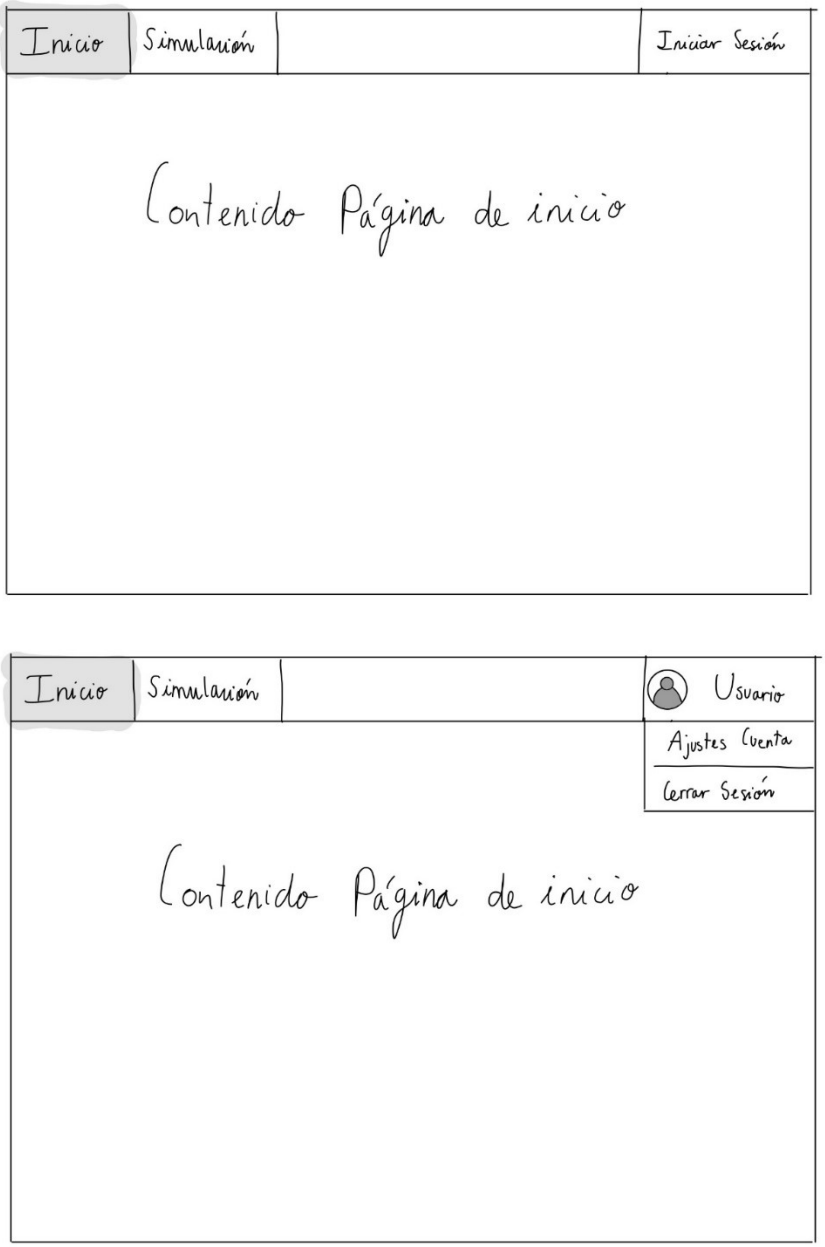


Figura B.1: Barra de Inicio.

Figura B.2: Editar Perfil.

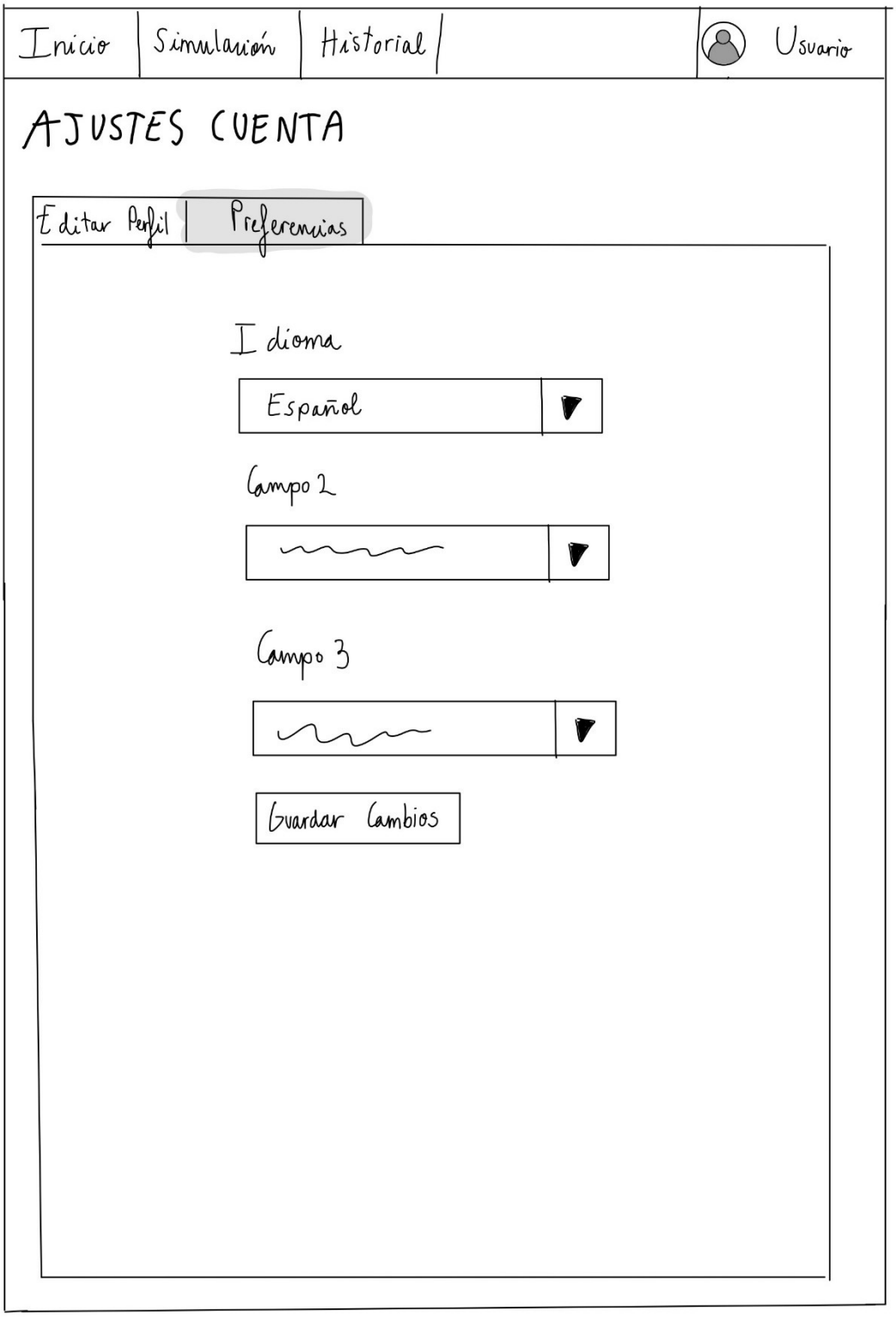


Figura B.3: Preferencias.

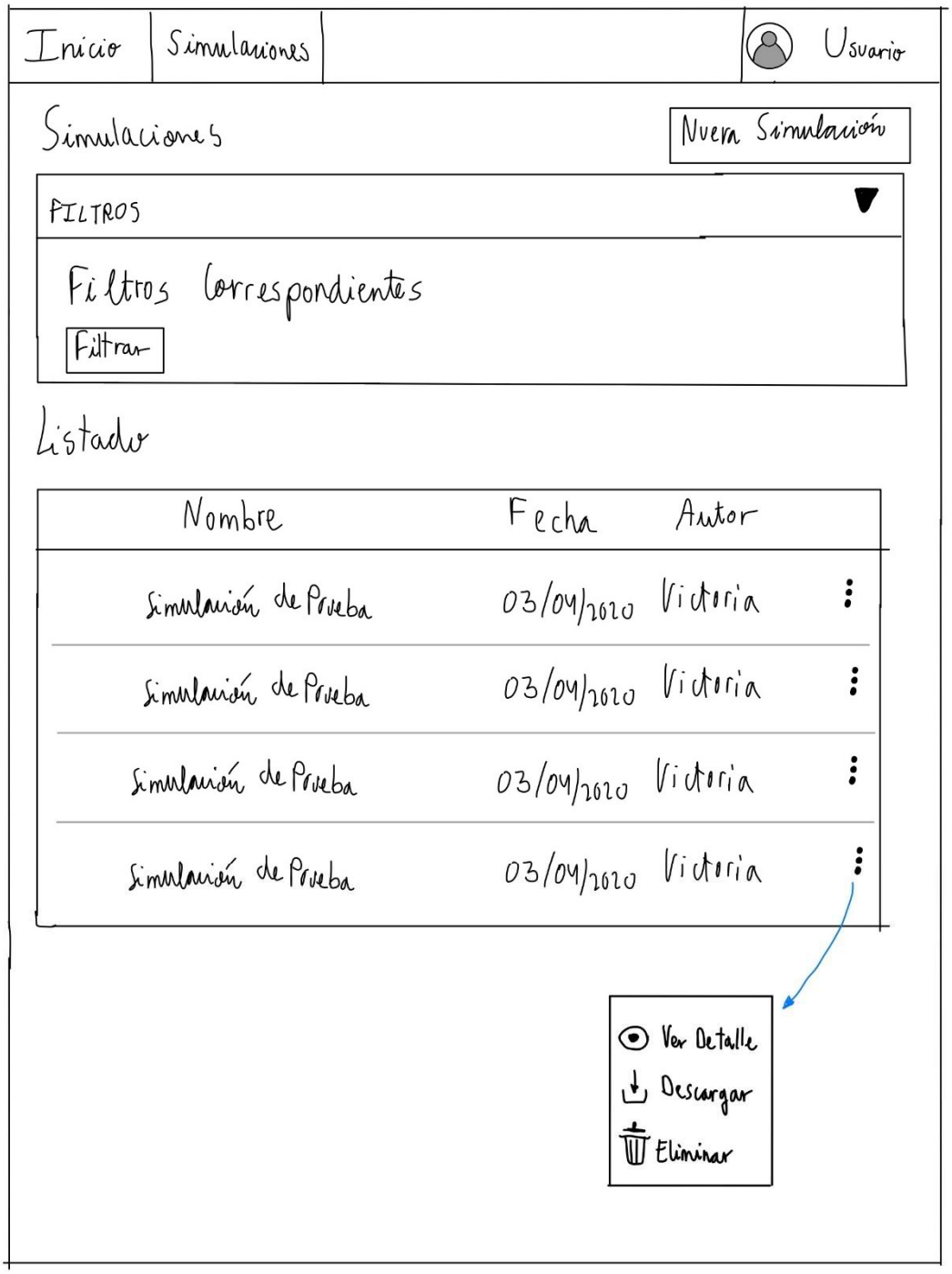


Figura B.4: Listado de Simulaciones.

Figura B.5: Detalle de Simulación.

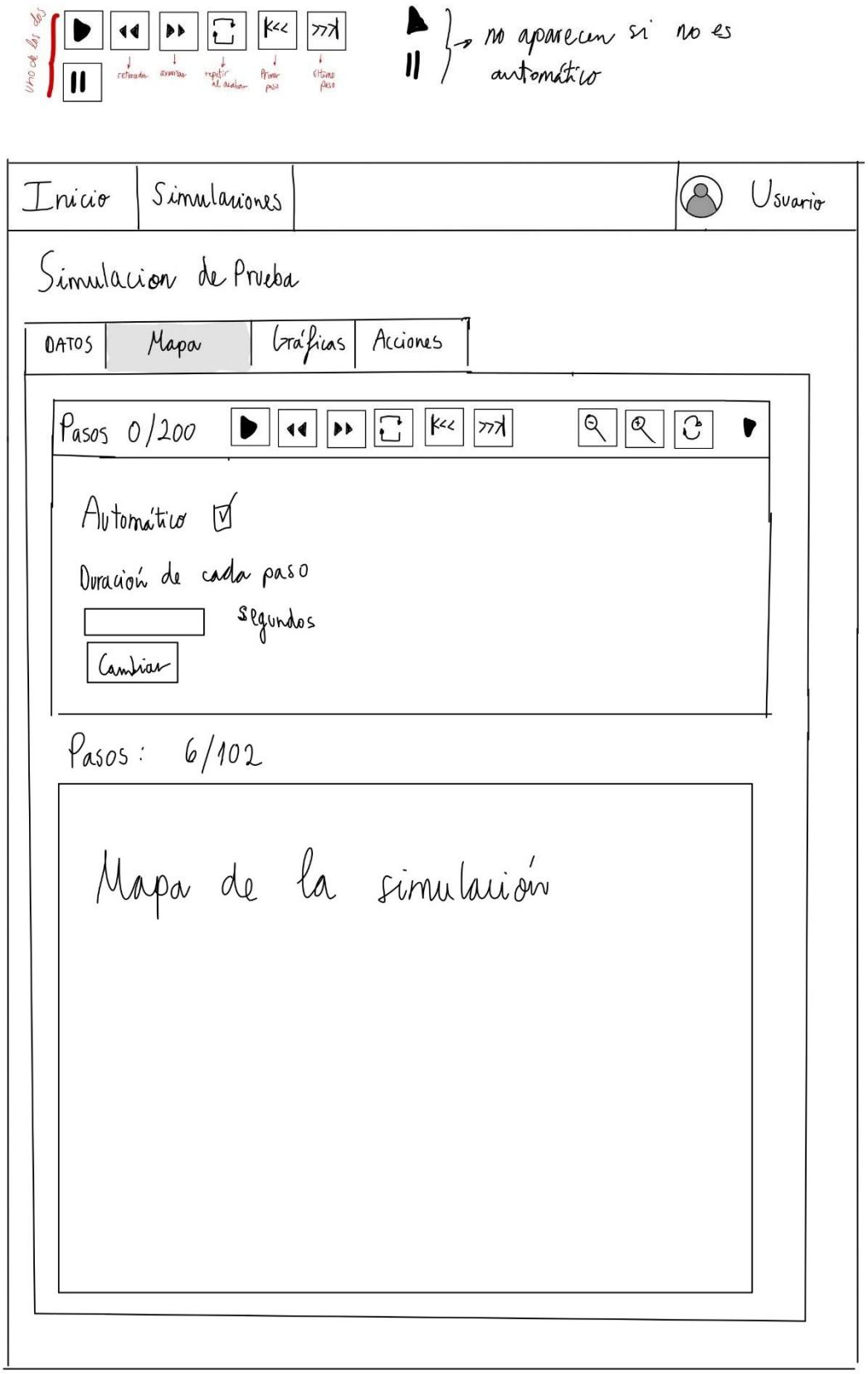


Figura B.6: Mapa de Simulación.

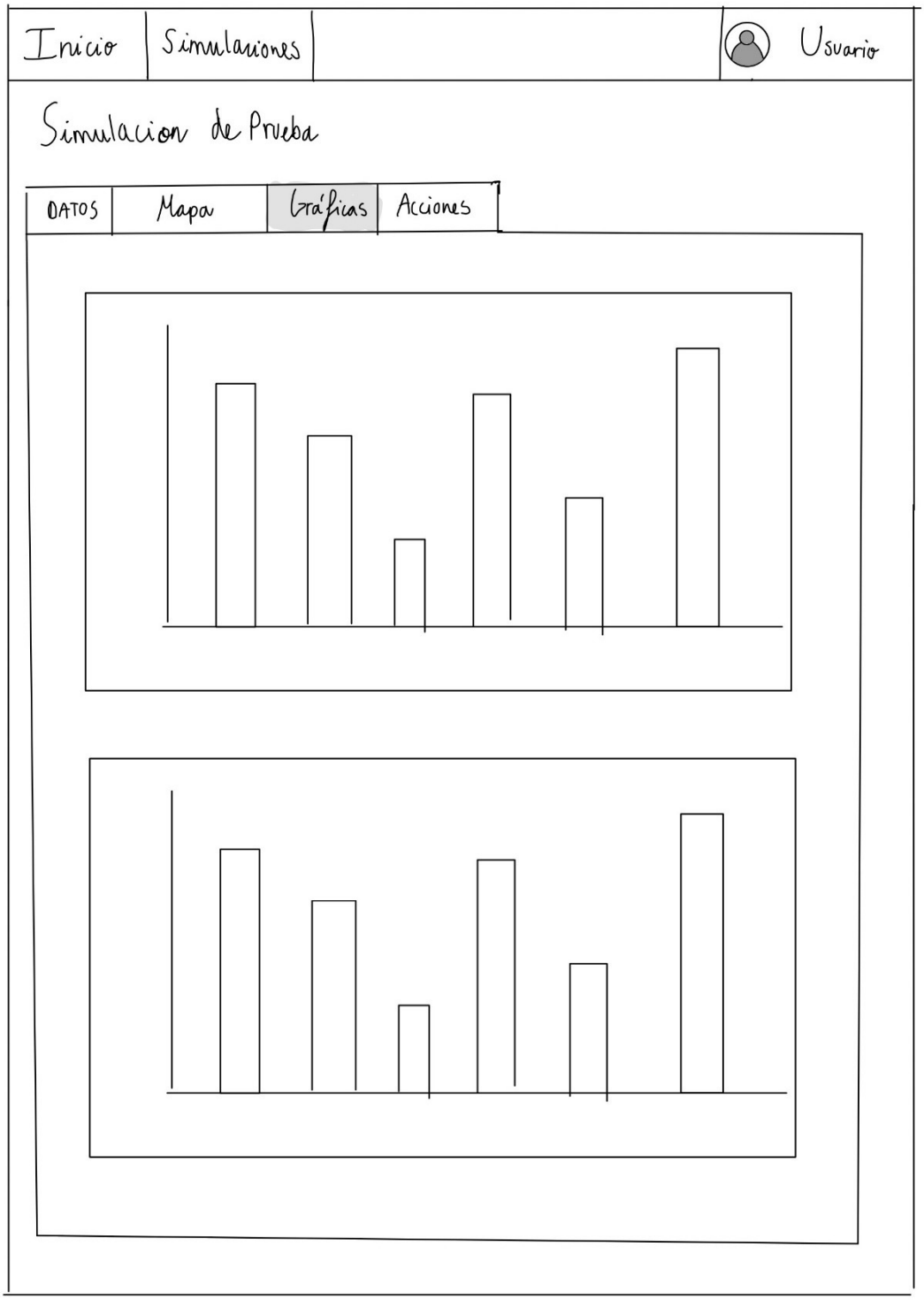


Figura B.7: Gráficas de simulación.

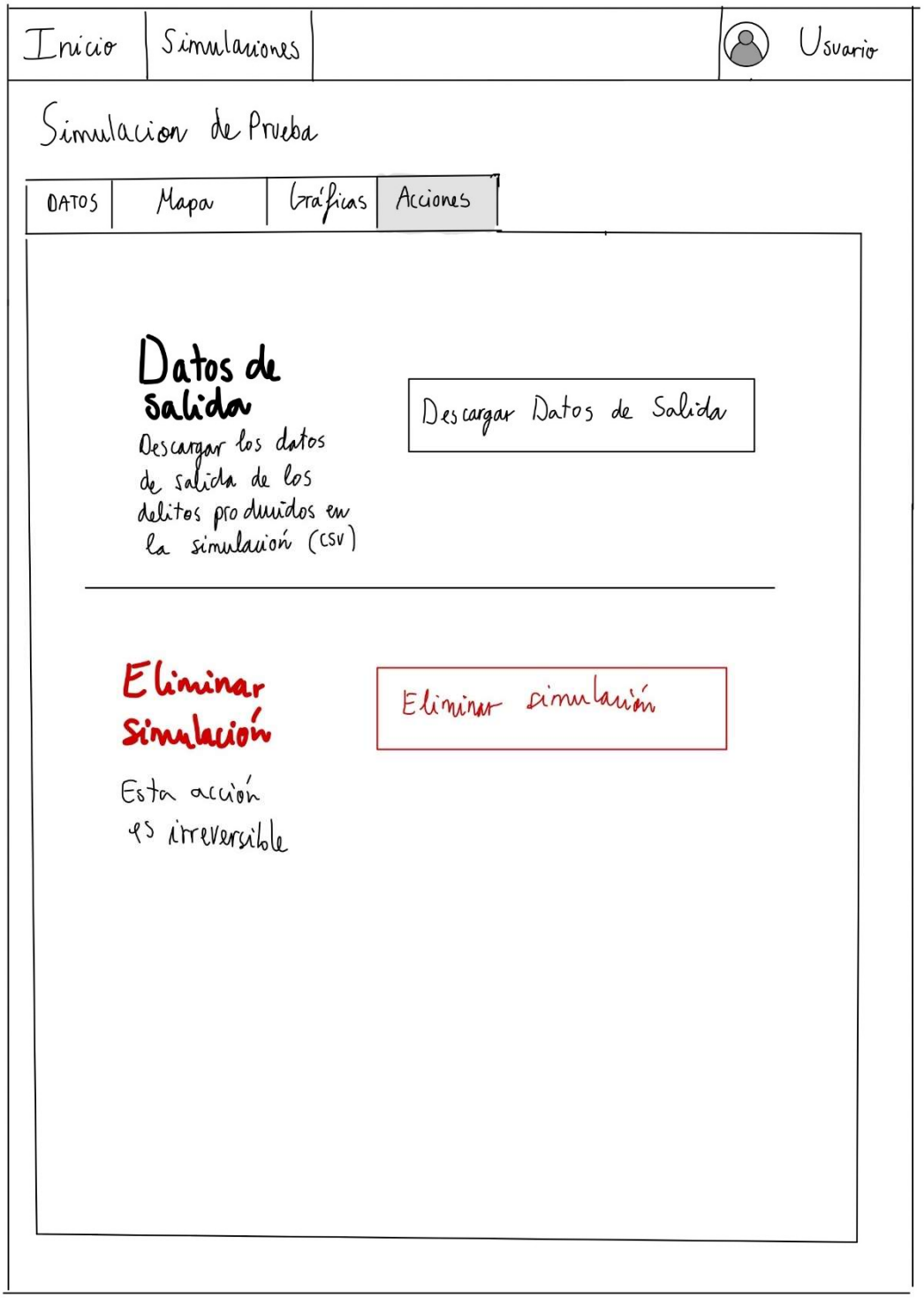


Figura B.8: Acciones de Simulación.

Figura B.9: Crear Simulación. Datos Básicos.

Figura B.10: Crear Simulación. Parámetros de entrada.

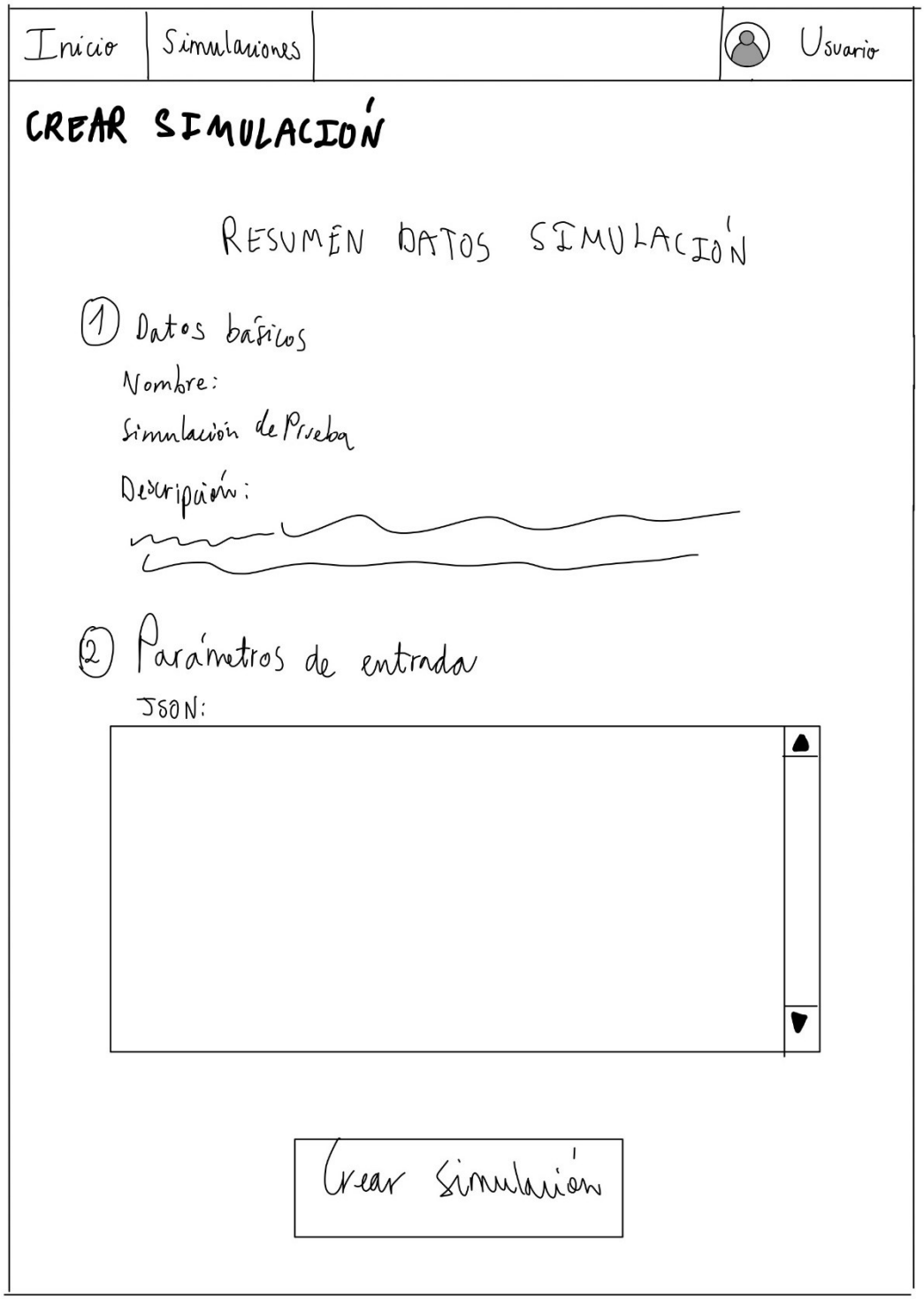


Figura B.11: Crear Simulación. Resumen.

Apéndice C

Diseño de Interfaces con Figma

Figura C.1: Barra de navegación sin iniciar sesión.

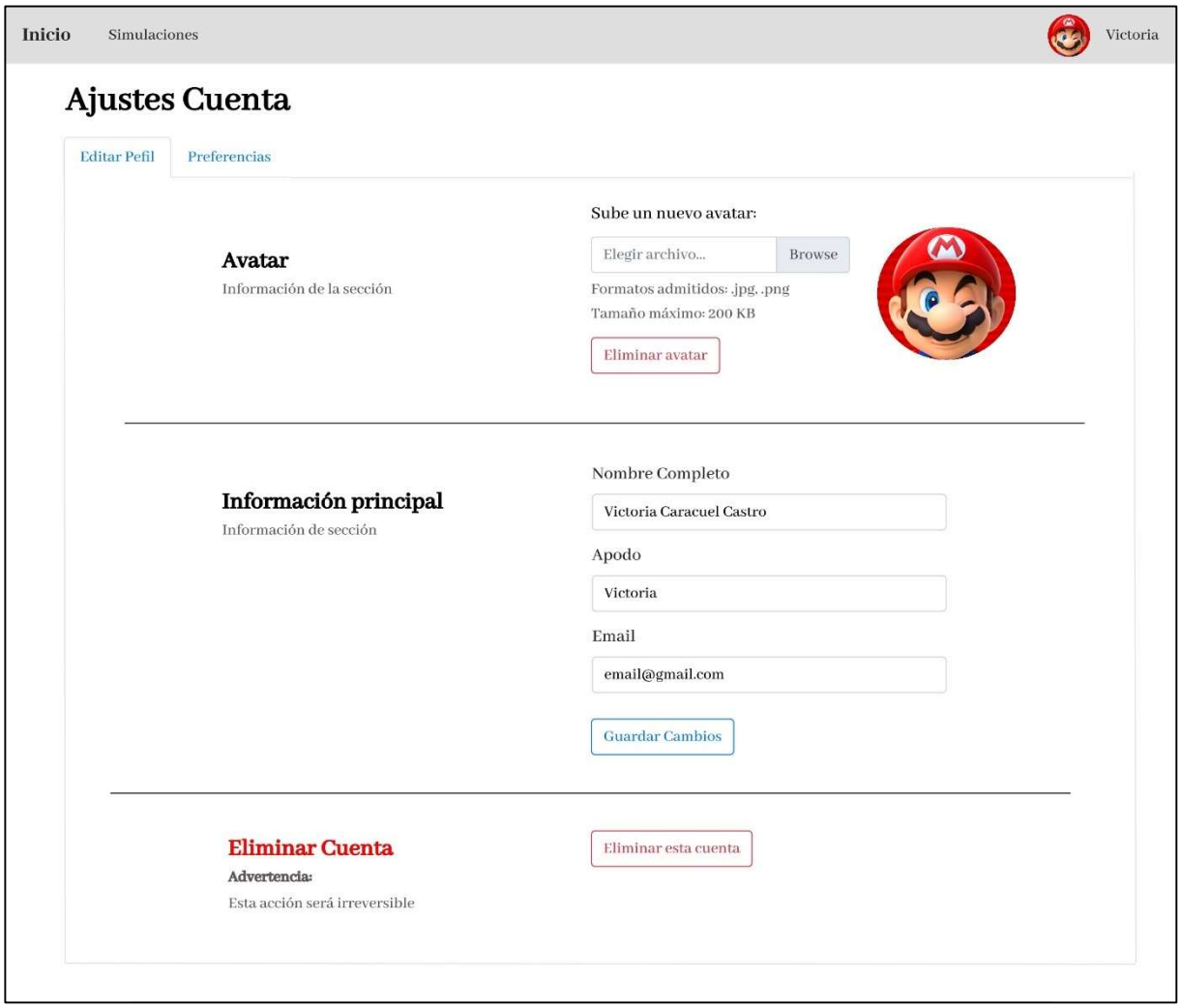


Figura C.2: Editar Perfil.

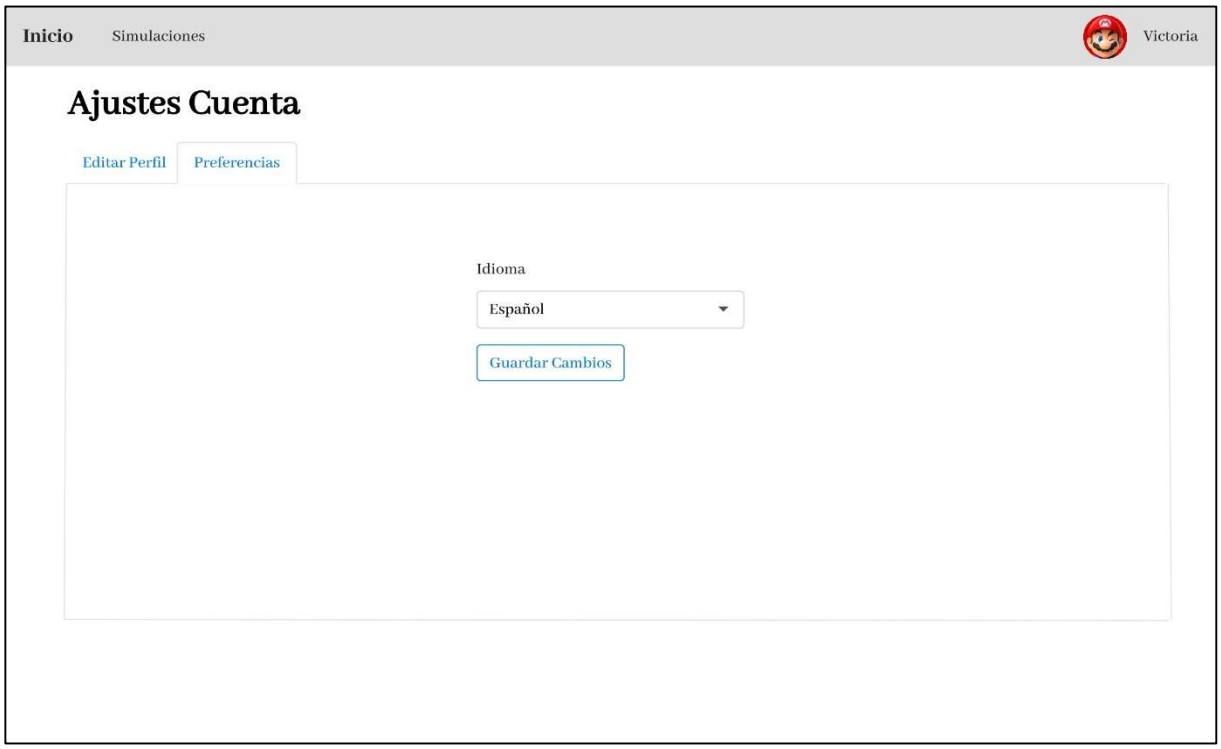


Figura C.3: Preferencias.

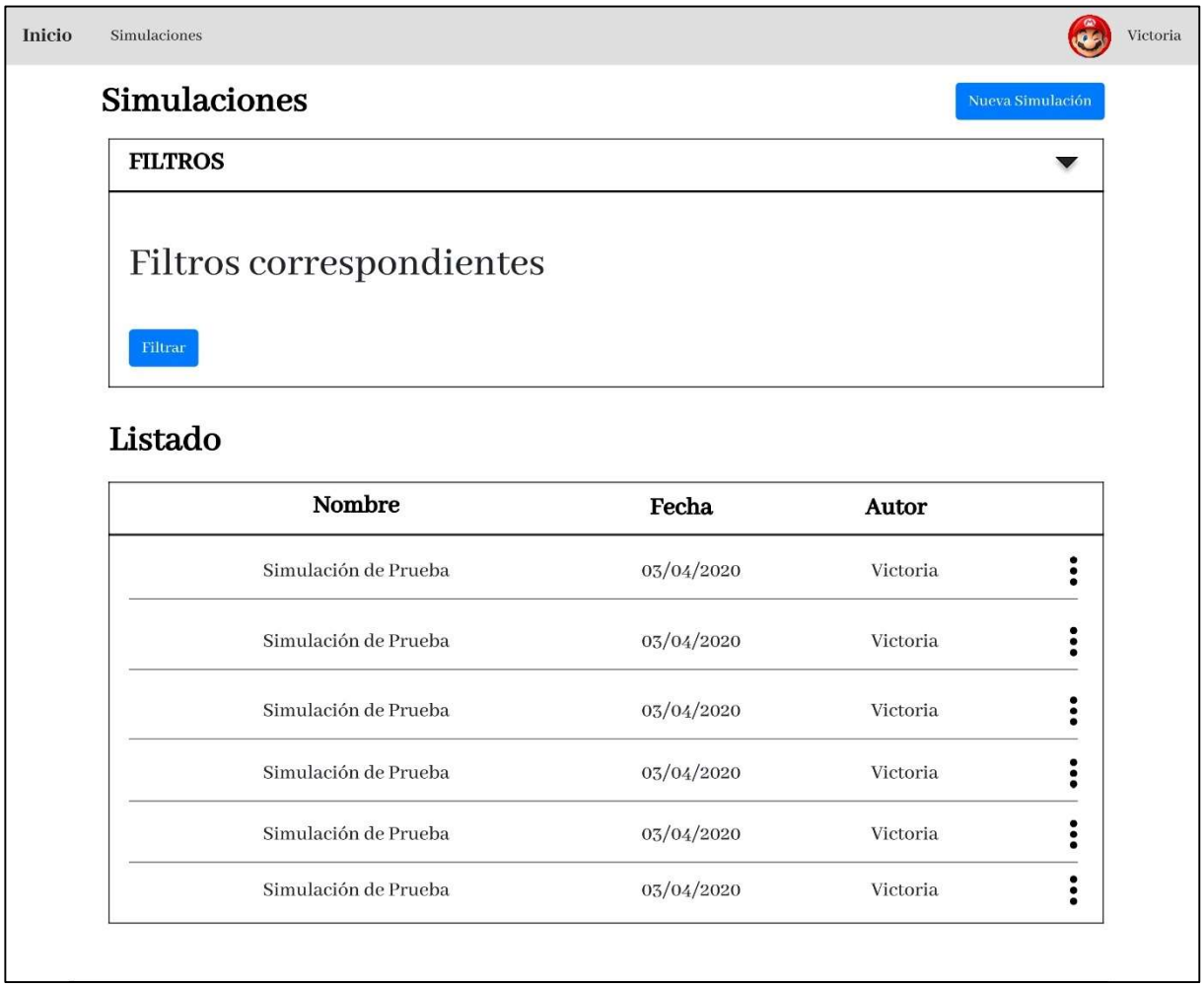


Figura C.4: Listado de simulaciones.

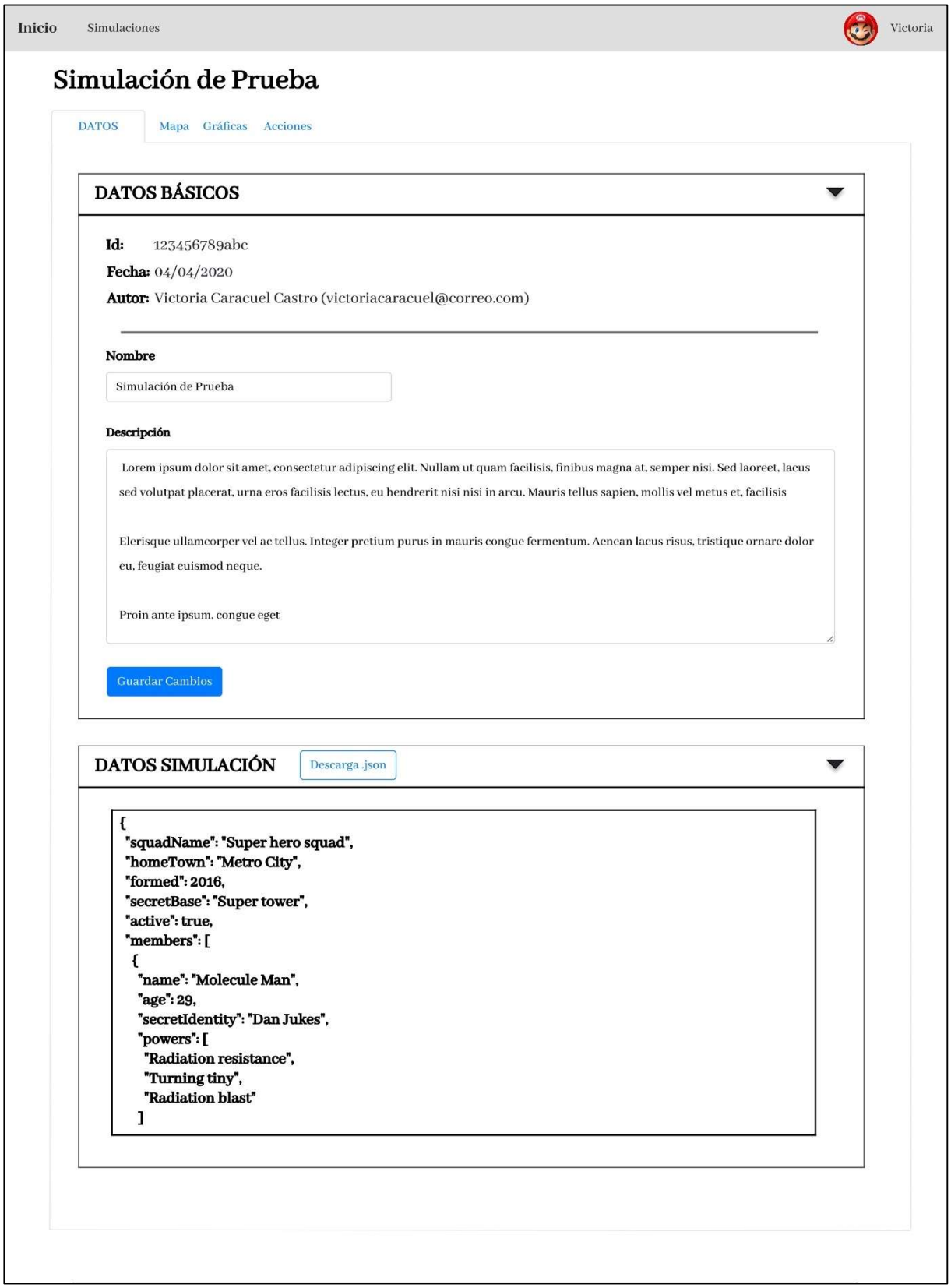


Figura C.5: Detalle de simulación.

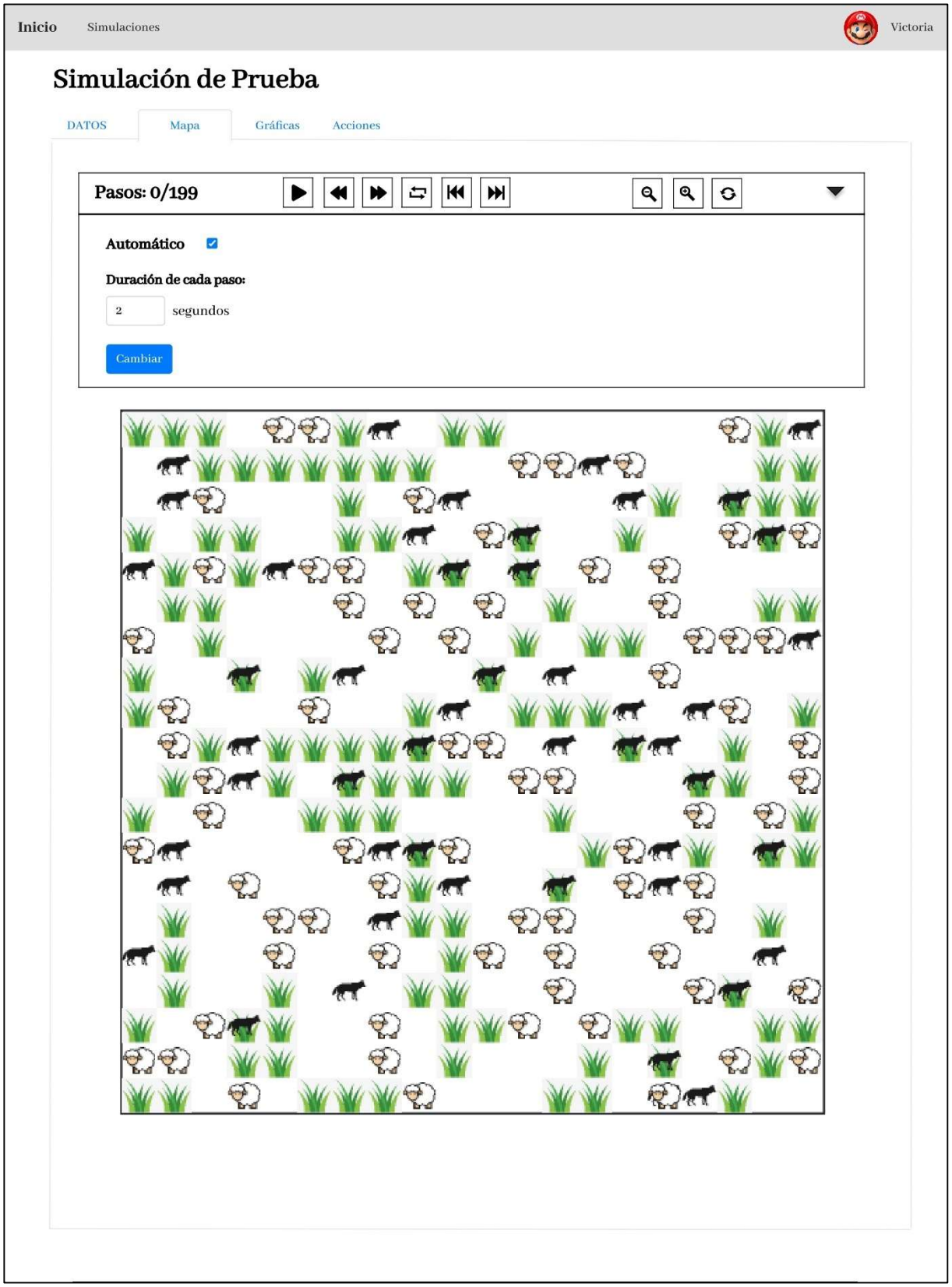


Figura C.6: Mapa de simulación.

Figura C.7: Gráficas de simulación.

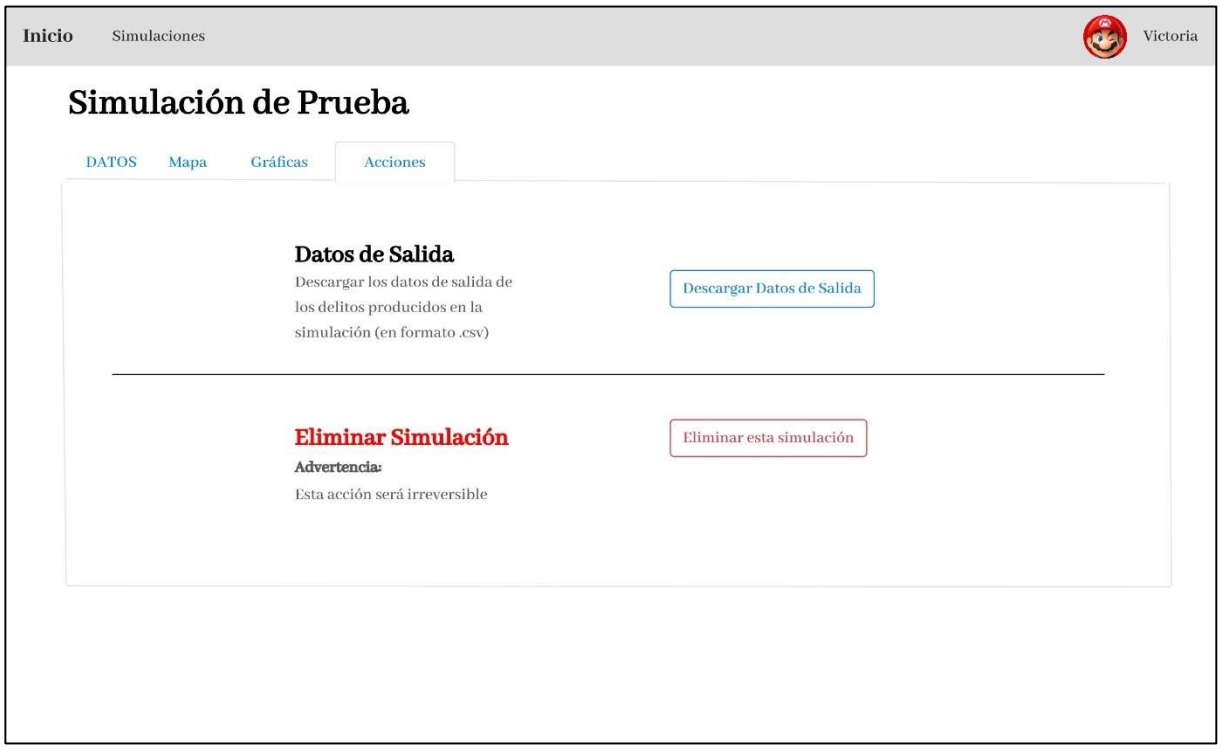


Figura C.8: Acciones de simulación.

Figura C.9: Crear simulación. Resumen.

Figura C.10: Crear simulación. Datos básicos.

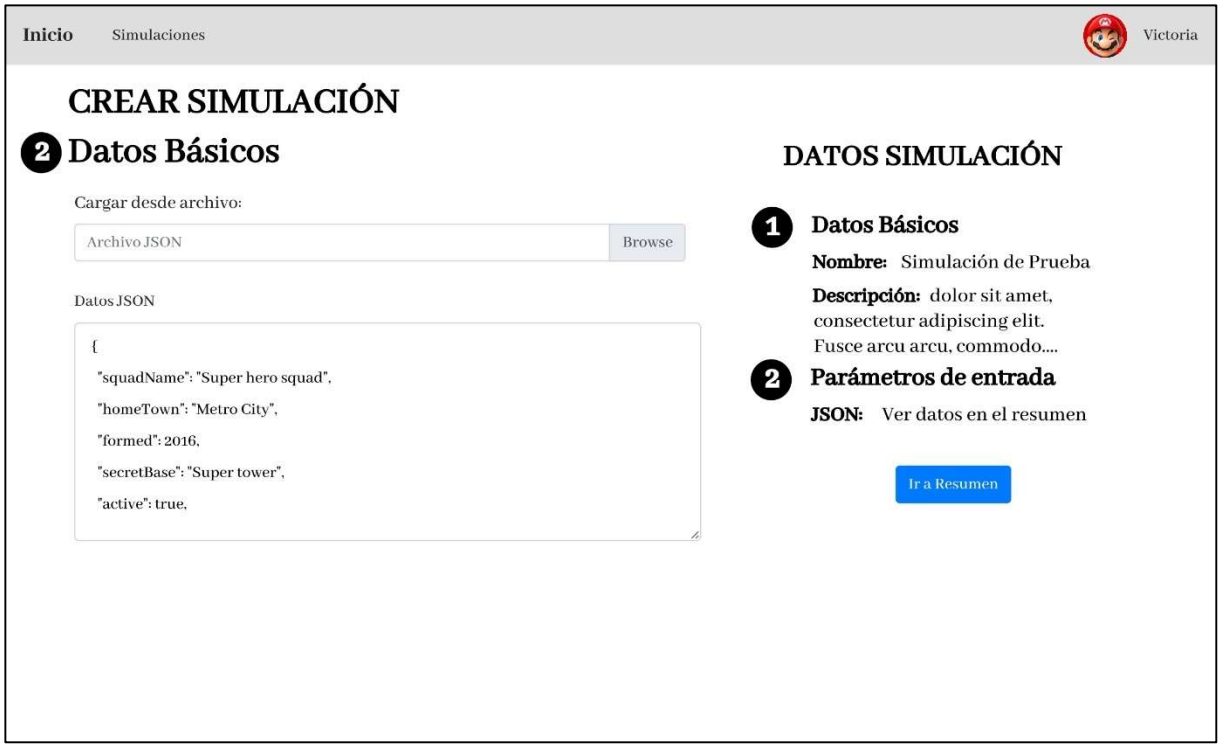


Figura C.11: Crear simulación. Parámetros de entrada.

Apéndice D

Casos de uso y Casos de Prueba

RF01: Los usuarios podrán registrarse usando su cuenta de Google.

Caso de uso

CASO DE USO	**RF01:** Los usuarios podrán registrarse usando su cuenta de Google.
ÁMBITO	Registro y acceso
ACTOR PRINCIPAL	Usuario sin cuenta creada
PARTICIPANTES	• Usuario sin cuenta creada
OBJETIVOS E INTERESES	
Los usuarios que no estén registrados en la plataforma querrán crear una cuenta para poder interactuar con la aplicación.	
PRECONDICIÓN	
El usuario no tiene una cuenta registrada.	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha podido crear la cuenta correctamente.	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá creado la cuenta e iniciado sesión.	
ESCENARIO PRINCIPAL DE ÉXITO	

1. El usuario pulsa el botón "Iniciar Sesión con Google"
2. El sistema muestra una pantalla para poder seleccionar una cuenta de Google
3. El usuario selecciona una cuenta
4. El sistema recoge los datos enviados por Google
5. El sistema guarda los datos del nuevo usuario
6. El sistema muestra los datos del usuario en la barra de navegación
7. El sistema muestra un mensaje de éxito

ESCENARIOS ALTERNATIVOS

0.a. El usuario selecciona una opción de la barra de navegación que requiere iniciar sesión.
0.a.1. El sistema muestra una ventana modal que indica la situación y un botón que pone "Iniciar Sesión con Google".
0.a2. Volvemos al paso 1.

2.a. El usuario no tiene cuentas cargadas en la sesión
2.a.1 El sistema solicita realizar el inicio de sesión usando la cuenta de Google
2.a.2 El usuario introduce sus datos
2.a.3 Volvemos al paso 4

2.b.2 El usuario cierra la ventana emergente
2.b.3 Fin del caso de uso

3.a. El usuario cierra la ventana emergente

3.a.1 El sistema informa al usuario de la situación
3.a.2 Fin del caso de uso

4.a. Se ha producido un error al obtener los datos de Google

4.a.1 El sistema informa al usuario de la situación
4.a.2 Fin del caso de uso

5.a. Se ha producido un error al guardar los datos del usuario

5.a.1 El sistema informa al usuario de la situación
5.a.2 Fin del caso de uso

Tabla D.1: Caso de Uso RF01.

Caso de prueba

```
Feature: (RF01) Registrar usuario

Background:
  Given un usuario que no está registrado

Scenario: (01) Usuario registrado con éxito
  When pulsa el botón "Iniciar sesión con Google"
  And selecciona una cuenta
  Then el sistema muestra los datos del usuario en la barra de navegación
  And el sistema muestra un mensaje de éxito

Scenario: (02) Usuario accede a ruta protegida
When accede a una ruta protegida
And el sistema muestra un modal con la opción de iniciar sesión
And pulsa el botón "Iniciar sesión con Google"
  And selecciona una cuenta
  Then el sistema muestra los datos del usuario en la barra de navegación
  And el sistema muestra un mensaje de éxito

Scenario: (03) Usuario sin cuenta cargadas en la sesión
  When pulsa el botón "Iniciar sesión con Google"
  And introduce sus datos
  Then el sistema muestra los datos del usuario en la barra de navegación
  And el sistema muestra un mensaje de éxito

Scenario: (04) El usuario cierra la ventana emergente
  When pulsa el botón "Iniciar sesión con Google"
  And cierra la ventana emergente
  Then el sistema muestra un mensaje de error

Scenario: (05) Error al registrar el usuario
  When pulsa el botón "Iniciar sesión con Google"
  And se produce un error al guardar los datos
```

```
  Then el sistema muestra un mensaje de error

Scenario: (06) Error al registrar el usuario desde ruta protegida
When accede a una ruta protegida
And el sistema muestra un modal con la opción de iniciar sesión
And pulsa el botón "Iniciar sesión con Google"
  And se produce un error al guardar los datos
Then el sistema muestra un mensaje de error
But mantiene el modal abierto
```

Tabla D.2: Caso de Prueba RF01.

RF02: Los usuarios podrán iniciar sesión usando su cuenta de Google.

Caso de uso

CASO DE USO	**RF02:** Los usuarios podrán iniciar sesión usando su cuenta de Google
ÁMBITO	Registro y acceso
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán iniciar sesión para poder interactuar con la aplicación.	
PRECONDICIÓN	
El usuario no ha iniciado sesión.	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha podido iniciar sesión correctamente.	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá iniciado sesión.	
ESCENARIO PRINCIPAL DE ÉXITO	

1. El usuario pulsa el botón "Iniciar Sesión con Google"
2. El sistema muestra una pantalla para poder seleccionar una cuenta de Google
3. El usuario selecciona una cuenta
4. El sistema recoge los datos enviados por Google
5. El sistema obtiene los datos guardados del usuario
6. El sistema muestra los datos del usuario en la barra de navegación
7. El sistema muestra un mensaje de éxito

ESCENARIOS ALTERNATIVOS

0.a. El usuario selecciona una opción de la barra de navegación que requiere iniciar sesión.
0.a.1. El sistema muestra una ventana modal que indica la situación y un botón que pone "Iniciar Sesión con Google".
0.a2. Volvemos al paso 1.

2.a. El usuario no tiene cuentas cargadas en la sesión
2.a.1 El sistema solicita realizar el inicio de sesión usando la cuenta de Google
2.a.2 El usuario introduce sus datos
2.a.3 Volvemos al paso 4

2.b.2 El usuario cierra la ventana emergente
2.b.3 Fin del caso de uso

3.a. El usuario cierra la ventana emergente

3.a.1 El sistema informa al usuario de la situación
3.a.2 Fin del caso de uso

4.a. Se ha producido un error al obtener los datos de Google

4.a.1 El sistema informa al usuario de la situación
4.a.2 Fin del caso de uso

5.a. Se ha producido un error al obtener los datos del usuario

5.a.1 El sistema informa al usuario de la situación
5.a.2 Fin del caso de uso

Tabla D.3: Caso de Uso RF02.

Caso de prueba

```
Feature: (RF02) Inciar sesión

Background:
  Given un usuario que está registrado

Scenario: (01) Sesión iniciada con éxito
  When pulsa el botón "Iniciar sesión con Google"
  And selecciona una cuenta
  Then el sistema muestra los datos del usuario en la barra de navegación
  And el sistema muestra un mensaje de éxito "{{nombre}}, has iniciado sesión
correctamente"

Scenario: (02) Usuario accede a ruta protegida
When accede a una ruta protegida
And el sistema muestra un modal con la opción de iniciar sesión
And pulsa el botón "Iniciar sesión con Google"
  And selecciona una cuenta
  Then el sistema muestra los datos del usuario en la barra de navegación
  And el sistema muestra un mensaje de éxito "{{nombre}}, has iniciado sesión
correctamente"

Scenario: (03) Usuario sin cuenta cargadas en la sesión
  When pulsa el botón "Iniciar sesión con Google"
  And introduce sus datos
  Then el sistema muestra los datos del usuario en la barra de navegación
  And el sistema muestra un mensaje de éxito "{{nombre}}, has iniciado sesión
correctamente"

Scenario: (04) El usuario cierra la ventana emergente
  When pulsa el botón "Inciar sesión con Google"
  And cierra la ventana emergente
  Then el sistema muestra un mensaje de error "No se ha podido iniciar sesión"
```

```
Scenario: (05) Error al iniciar sesión
  When pulsa el botón "Inciar sesión con Google"
  And se produce un error al obtener los datos
  Then el sistema muestra un mensaje de error "No se ha podido iniciar sesión"

  Scenario: (06) Error al iniciar sesión desde ruta protegida
When accede a una ruta protegida
And el sistema muestra un modal con la opción de iniciar sesión
And pulsa el botón "Iniciar sesión con Google"
  And se produce un error
Then el sistema muestra un mensaje de error
But mantiene el modal abierto
```

Tabla D.4: Caso de Prueba RF02.

RF03: Los usuarios podrán cerrar sesión.

Caso de uso

CASO DE USO	**RF03:** Los usuarios podrán cerrar sesión.
ÁMBITO	Registro y acceso
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios, cuando hayan iniciado sesión, querrán en algún momento cerrar la sesión.	
PRECONDICIÓN	
El usuario ha iniciado sesión.	
POST CONDICIÓN MÍNIMA	
El usuario habrá cerrado sesión y será informado.	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá cerrado sesión y será informado.	
ESCENARIO PRINCIPAL DE ÉXITO	

1. El usuario pulsa en su apodo o imagen de perfil en la barra de navegación. **2.** El sistema muestra un submenú donde una de las opciones será cerrar sesión. **3.** El usuario selecciona "Cerrar Sesión" **4.** El sistema cierra la sesión del usuario (desapareciendo los datos de éste y volviendo a la página principal)
ESCENARIOS ALTERNATIVOS

Tabla D.5: Caso de Uso RF03.

Caso de prueba

```
Feature: (RF03) Cerrar Sesión

Background:
  Given un usuario que ha iniciado sesión

Scenario: (01)Sesión cerrada correctamente
  When pulsa en su apodo o imagen de perfil en la barra de navegación
  And selecciona "Cerrar Sesión"
  Then el sistema deja de mostrar los datos relativos al usuario
  And el sistema envía al usuario a la página principal
```

Tabla D.6: Caso de Prueba RF03.

RF04: Los usuarios podrán gestionar sus datos en el sistema.

RF04.1: Los usuarios podrán asociar un avatar a su cuenta.

Caso de uso

CASO DE USO	**RF04.1:** Los usuarios podrán asociar un avatar a su cuenta.
ÁMBITO	Perfil de los usuarios
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	

Los usuarios que estén registrados en la aplicación querrán cambiar su foto de perfil.
PRECONDICIÓN
El usuario ha iniciado sesión y se encuentra en "Ajustes Cuenta", en la pestaña "Perfil".
POST CONDICIÓN MÍNIMA
El usuario será informado sobre si ha podido cambiar la foto correctamente.
POSTCONDICIÓN DE ÉXITO
El usuario habrá cambiado la foto del perfil.
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario pulsa el botón para cargar la nueva imagen. **2.** El usuario selecciona una imagen entre los archivos de su dispositivo. **3.** El sistema sube la imagen a Imgur y obtiene el enlace. **4.** El sistema almacena el enlace de la imagen del usuario. **5.** El sistema muestra la imagen actualizada en la barra de navegación y en el perfil. **6.** El sistema muestra un mensaje de éxito
ESCENARIOS ALTERNATIVOS
2.a. El usuario selecciona un formato de imagen no válido. **2.a.1.** El sistema informa al usuario de la situación **2.a.2.** Fin del caso de uso **3.a.** Se produce un error al subir la imagen a Imgur **3.a.1.** El sistema informa al usuario de la situación **3.a.2.** Fin del caso de uso **4.a.** Se produce un error al guardar el enlace de la imagen. **4.a.1.** El sistema informa al usuario de la situación **4.a.2.** Fin del caso de uso

Tabla D.7: Caso de Uso RF04.1.

Caso de prueba

```
Feature: (RF04.1) Cambiar imagen del perfil

Background:
  Given un usuario que está registrado
  And está en "Ajustes Cuenta"
  And está en la pestaña "Perfil"

Scenario: (01) Imagen cambiada con éxito
  When selecciona una imagen con formato válido
  Then el sistema muestra la imagen actualizada en la barra de navegación y en el
perfil
  And el sistema muestra un mensaje de éxito "Imagen Cambiada Correctamente"

Scenario: (02) Formato de imagen no válida
  When selecciona una imagen con formato no válido
  Then el sistema muestra un mensaje de error "Formato de imagen no válido"

Scenario: (03) Error con el servicio Imgur
  When selecciona una imagen con formato válido
  And hay un error al subir la imagen a Imgur
  Then el sistema muestra un mensaje de error "No se ha podido modificar la imagen
de perfil"

Scenario: (04) Error al guardar la imagen
  When selecciona una imagen con formato válido
  And hay un error al guardar la imagen en el sistema
  Then el sistema muestra un mensaje de error "No se ha podido modificar la imagen
de perfil"
```

Tabla D.8: Caso de Prueba RF04.1.

RF04.2: Los usuarios podrán eliminar el avatar asociado a su cuenta.

Caso de uso

CASO DE USO	**RF04.2:** Los usuarios podrán eliminar el avatar asociado a su cuenta.
ÁMBITO	Perfil de los usuarios

ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán eliminar su foto de perfil.	
PRECONDICIÓN	
El usuario ha iniciado sesión y se encuentra en "Ajustes Cuenta", en la pestaña "Perfil", y tiene una foto.	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha podido eliminar la foto correctamente.	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá eliminado la foto del perfil.	
ESCENARIO PRINCIPAL DE ÉXITO	
1. El usuario pulsa el botón "Eliminar Avatar" **2.** El sistema muestra un modal de confirmación para realizar la acción. **3.** El usuario pulsa el botón de confirmación. **4.** El sistema elimina la imagen de los datos guardados del usuario. **5.** El sistema muestra la imagen de perfil por defecto en la barra de navegación y en el perfil del usuario. **6.** El sistema muestra un mensaje de éxito	
ESCENARIOS ALTERNATIVOS	
3.a. El usuario cancela la acción **3.a.1.** Fin del caso de uso **4.a.** Se produce un error al eliminar la imagen **4.a.1.** El sistema informa al usuario de la situación **4.a.2.** Fin del caso de uso	

Tabla D.9: Caso de Uso RF04.2.

Caso de prueba

```
Feature: (RF04.2) Eliminar imagen del perfil

Background:
  Given un usuario que está registrado
  And está en "Ajustes Cuenta"
  And está en la pestaña "Perfil"
  And tiene imagen de perfil
  And pulsa el botón "Eliminar avatar" en el perfil
  And pulsa el botón "Eliminar avatar" en el modal de confirmación

Scenario: (01) Imagen eliminada con éxito
  When el sistema elimina la imagen correctamente
  Then el sistema muestra la imagen por defecto en la barra de navegación y en el
perfil
  And el sistema muestra un mensaje de éxito "Imagen Cambiada Correctamente"

Scenario: (02) Error al eliminar la imagen
  When hay un error al eliminar la imagen en el sistema
  Then el sistema muestra un mensaje de error "No se ha podido modificar la imagen
de perfil"
```

Tabla D.10: Caso de Prueba RF04.2.

RF04.3: Los usuarios podrán editar su información principal

Caso de uso

CASO DE USO	**RF04.3:** Los usuarios podrán editar su información principal.
ÁMBITO	Perfil de los usuarios
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán modificar sus datos principales (nombre completo, apodo y email).	

PRECONDICIÓN
El usuario ha iniciado sesión y se encuentra en "Ajustes Cuenta", en la pestaña "Perfil".
POST CONDICIÓN MÍNIMA
El usuario será informado sobre si ha podido modificar los datos.
POSTCONDICIÓN DE ÉXITO
El usuario habrá modificado los datos asociados a la cuenta.
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario rellena los campos pertenecientes a la sección "Información Principal" (nombre completo, apodo y email) **2.** El usuario pulsa el botón "Guardar Cambios" **3.** El sistema actualiza los datos guardados del usuario. **4.** El sistema muestra un mensaje de éxito.
ESCENARIOS ALTERNATIVOS
1.a. El usuario no rellena todos los campos **1.a.1.** El sistema muestra un mensaje de error sobre el campo que falte por rellenar **1.a.2.** Volvemos al paso 1. **1.b.** El usuario introduce en el campo de entrada del email una cadena de caracteres que no tiene el formato de un email. **1.b.1.** El sistema muestra un mensaje de error explicando la situación **1.b.2.** Volvemos al paso 1. **3.a.** Se produce un error al guardar los datos **3.a.1.** El sistema informa al usuario de la situación **3.a.2.** El sistema vuelve a mostrar los datos que tiene guardados del usuario **3.a.3.** Fin del caso de uso

Tabla D.11: Caso de Uso RF04.3.

Caso de prueba

```
Feature: (RF04.3) Editar datos principales

Background:
```

```
  Given un usuario que está registrado
  And está en "Ajustes Cuenta"
  And está en la pestaña "Perfil"

Scenario: (01) Datos editados correctamente
  When el usuario rellena correctamente los campos de entrada de la sección
"Información Principal" (nombre completo, apodo y email)
  And pulsa el botón "Guardar Cambios"
  Then el sistema guarda los datos
  And el sistema muestra un mensaje de éxito "Datos Cambiados Correctamente"

Scenario: (02) Falta un campo
    When el usuario no rellena todos los campos de entrada de la sección "Información
Principal" (nombre completo, apodo y email)
  And pulsa el botón "Guardar Cambios"
  Then el sistema informa al usuario de la situación "El apodo es obligatorio"

Scenario: (03) Formato email incorrecto
  When el usuario introduce en el campo de entrada del email una cadena de
caracteres que no tiene el formato de un email.
  And pulsa el botón "Guardar Cambios"
  Then el sistema informa al usuario de la situación "Tiene que ser una dirección de
email válida "

Scenario: (04) Error al guardar los datos
  When el usuario rellena correctamente los campos de entrada de la sección
"Información Principal" (nombre completo, apodo y email)
  And pulsa el botón "Guardar Cambios"
  And se produce un error al guardar los datos
  Then el sistema muestra un mensaje de error "No se han podido modificar los
datos"
  And el sistema vuelve a mostrar los datos guardados del usuario
```

Tabla D.12: Caso de Prueba RF04.3.

RF04.4: Los usuarios podrán modificar sus preferencias

Caso de uso

CASO DE USO	**RF04.4:** Los usuarios podrán modificar sus preferencias.

ÁMBITO	Perfil de los usuarios
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán elegir la forma en la que se muestra la información (idioma).	
PRECONDICIÓN	
El usuario ha iniciado sesión y se encuentra en "Ajustes Cuenta", en la pestaña "Preferencias".	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha podido modificar las preferencias.	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá modificado las preferencias asociadas a la cuenta.	
ESCENARIO PRINCIPAL DE ÉXITO	
1. El usuario selecciona las preferencias deseadas. **2.** El usuario pulsa el botón "Guardar Cambios" **3.** El sistema actualiza las preferencias guardadas del usuario. **4.** El sistema muestra un mensaje de éxito.	
ESCENARIOS ALTERNATIVOS	
3.a. Se produce un error al guardar los datos **3.a.1.** El sistema informa al usuario de la situación **3.a.2.** El sistema vuelve a mostrar los datos que tiene guardados del usuario **3.a.3.** Fin del caso de uso	

Tabla D.13: Caso de Uso RF04.4.

Caso de prueba

```
Feature: (RF04.4) Editar preferencias

Background:
  Given un usuario que está registrado
```

```
  And está en "Ajustes Cuenta"
  And está en la pestaña "Preferencias"

Scenario: (01) Datos guardados correctamente
  When el usuario selecciona las preferencias deseadas
  And pulsa el botón "Guardar Cambios"
  Then el sistema actualiza las preferencias guardadas del usuario
  And el sistema muestra un mensaje de éxito "Datos Cambiados Correctamente"

  Scenario: (02) Error al guardar los datos
When el usuario selecciona las preferencias deseadas
  And pulsa el botón "Guardar Cambios"
  And se produce un error al guardar los datos
  Then el sistema muestra un mensaje de error "No se han podido modificar los
datos"
  And el sistema vuelve a mostrar los datos guardados del usuario
```

Tabla D.14: Caso de Prueba RF04.4.

RF05: Los usuarios podrán eliminar su cuenta.

Caso de uso

CASO DE USO	**RF05:** Los usuarios podrán eliminar su cuenta.
ÁMBITO	Perfil de los usuarios
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán eliminar su cuenta.	
PRECONDICIÓN	
El usuario ha iniciado sesión y se encuentra en "Ajustes Cuenta", en la pestaña "Perfil".	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha podido eliminar la cuenta.	

POSTCONDICIÓN DE ÉXITO
El usuario habrá eliminado la cuenta y dejará de estar identificado.
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario pulsa el botón "Eliminar esta Cuenta" **2.** El sistema muestra un modal de confirmación para realizar la acción. **3.** El usuario pulsa el botón de confirmación. **4.** El sistema elimina los datos de la cuenta **5.** El sistema cierra la sesión del usuario. **6.** El sistema muestra al usuario a la pantalla de inicio. **7.** El sistema muestra un mensaje de éxito
ESCENARIOS ALTERNATIVOS
3.a. El usuario cancela la acción **3.a.1.** Fin del caso de uso **4.a.** Se produce un error al eliminar la cuenta **4.a.1.** El sistema informa al usuario de la situación **4.a.2.** Fin del caso de uso

Tabla D.15: Caso de Uso RF05.

Caso de prueba

```
Feature: (RF05) Eliminar cuenta

Background:
  Given un usuario que está registrado
  And está en "Ajustes Cuenta"
  And está en la pestaña "Perfil"
  And pulsa el botón "Eliminar esta cuenta" en el perfil
  And pulsa el botón "Eliminar Cuenta" en el modal de confirmación

Scenario: (01) Cuenta eliminada correctamente
When el sistema elimina los datos de la cuenta correctamente
  Then el sistema cierra la sesión del usuario
  And el sistema muestra al usuario la pantalla de inicio
  And el sistema muestra un mensaje de éxito "Cuenta eliminada correctamente"

Scenario: (02) Error al eliminar la cuenta
```

When hay un error al eliminar la cuenta en el sistema
Then el sistema cierra el modal de confirmación
And el sistema muestra un mensaje de error "No se ha podido eliminar la cuenta"

Tabla D.16: Caso de Prueba RF05.

RF06: Los usuarios podrán crear simulaciones

Caso de uso

CASO DE USO	**RF06:** Los usuarios podrán crear simulaciones.
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán crear simulaciones a partir de unos datos de entrada.	
PRECONDICIÓN	
El usuario ha iniciado sesión y se encuentra en "Simulaciones"	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha podido crear la simulación	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá creado la simulación a partir de los datos de entrada introducidos	
ESCENARIO PRINCIPAL DE ÉXITO	
1. El usuario pulsa el botón "Nueva Simulación" **2.** El sistema muestra un asistente (wizard) con los pasos y el primer paso activo (introducir el nombre y descripción de la simulación) **3.** El usuario rellena los datos (nombre y descripción) **4.** El usuario selecciona el paso 2 **5.** El sistema muestra la pantalla del paso 2 (parámetros de la simulación) **6.** El usuario pulsa el botón para cargar los datos desde un archivo json	

7. El usuario selecciona un archivo json entre los archivos de su dispositivo
8. El sistema muestra los datos del archivo en una caja de texto
9. El sistema comprueba que los datos son correctos
10. El sistema muestra el botón "Ir a Resumen"
11. El usuario pulsa el botón "Ir a Resumen"
12. El sistema muestra un resumen de los datos introducidos en el proceso de creación de la simulación
13. El usuario pulsa el botón "Crear Simulación"
14. El sistema genera la simulación
15. El sistema almacena la simulación en base de datos
16. El sistema muestra un mensaje de éxito
17. El sistema muestra la pantalla de detalle de la simulación creada

ESCENARIOS ALTERNATIVOS

6.a. El usuario introduce los datos en la caja establecida para ello
6.a.1. Vamos al paso 9

7.a. El usuario selecciona un archivo de extensión no válida
7.a.1. El sistema informa al usuario de la situación
7.a.2. Volvemos al paso 6.
7.b. El usuario selecciona un archivo json con estructura no válida
7.b.1. El sistema informa al usuario de la situación
7.b.2. Volvemos al paso 6.

9.a. No están todos los datos introducidos correctamente
9.a.1 El usuario rellena los datos como se indica en los pasos [3-8]
9.a.2 Volvemos al paso 9

13.a. El usuario pulsa en la etiqueta del paso 1 del resumen
13.a.1. Volvemos al paso 2
13.b. El usuario pulsa en la etiqueta del paso 2 del resumen
13.b.1. Volvemos al paso 5

14.a. Parámetros de entrada mal introducidos
14.a.1. El sistema informa al usuario de la situación
14.a.2. Volvemos al paso 12

14.a. Se produce un error en la simulación
14.a.1. El sistema informa al usuario de la situación
14.a.2. Volvemos al paso 12

15.a. Se produce un error al guardar en la base de datos
15.a.1. El sistema informa al usuario de la situación
15.a.2. Volvemos al paso 12

Tabla D.17: Caso de Uso RF06.

Caso de prueba

```
Feature: (RF06) Crear Simulación

Background:
  Given un usuario que ha iniciado sesión
  And se encuentra en la ventana "Simulaciones"
  And el usuario pulsa "Nueva Simulación"

Scenario: (01) Creada Correctamente
  When el usuario rellena los datos correctamente
  And el sistema muestra el botón "Ir a Resumen"
  And el usuario pulsa el botón "Ir a Resumen"
  And el usuario pulsa el botón "Crear Simulación"
  And el sistema genera la simulación correctamente
  And el sistema almacena la simulación correctamente
  Then el sistema muestra un mensaje de éxito
  And el sistema muestra la pantalla de detalle de la nueva simulación

Scenario: (02) Archivo formato no válida
  When el usuario selecciona un archivo con extensión no válida
  Then el sistema muestra un mensaje de error "Formato de archivo no válido (debe
ser .json)"

Scenario: (03) Archivo json mal formado
  When el usuario selecciona un archivo json mal formado
  Then el sistema muestra un mensaje de error "JSON mal formado en el archivo"

Scenario: (04) Datos no correctos
  When el usuario no rellena los datos correctamente
  Then el sistema no muestra el botón "Ir a Resumen"

Scenario: (05) Parámetros de entrada incorrectos
  When el usuario rellena los datos sin seguir las pautas indicadas
```

```
  And el sistema muestra el botón "Ir a Resumen"
  And el usuario pulsa el botón "Crear Simulación"
  Then el sistema muestra un mensaje de error "Parámetros de entrada introducidos de manera incorrecta"

Scenario: (06) Error al simular
  When el usuario rellena los datos correctamente
  And el sistema muestra el botón "Ir a Resumen"
  And el usuario pulsa el botón "Crear Simulación"
  And el sistema falla al generar la simulación
  Then el sistema muestra un mensaje de error "No se ha podido crear la simulación"

Scenario: (07) Error al guardar en bd
  When el usuario rellena los datos correctamente
  And el sistema muestra el botón "Ir a Resumen"
  And el usuario pulsa el botón "Crear Simulación"
  And el sistema generar la simulación correctamente
  And el sistema no almacena la simulación correctamente
  Then el sistema muestra un mensaje de error "No se ha podido crear la simulación"
```

Tabla D.18: Caso de Prueba RF06.

RF07.1: Los usuarios podrán visualizar el listado de simulaciones generadas

Caso de uso

CASO DE USO	**RF07.1:** Los usuarios podrán visualizar el listado de simulaciones generadas.
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán visualizar el listado de simulaciones creadas por ellos mismos.	
PRECONDICIÓN	
El usuario ha iniciado sesión.	

POST CONDICIÓN MÍNIMA
El usuario visualizará el listado de simulaciones (si no tiene, también se indicará)
POSTCONDICIÓN DE ÉXITO
El usuario visualizará el listado de simulaciones (si no tiene, también se indicará)
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario pulsa la sección "Simulaciones" **2.** El sistema obtiene las simulaciones creadas por el usuario **3.** El sistema muestra el listado de simulaciones
ESCENARIOS ALTERNATIVOS
2.a. Se ha producido un error al obtener las simulaciones **2.a.1.** El sistema informa al usuario de la situación **2.a.2.** El sistema muestra al usuario el botón "Cargar de nuevo" **2.a.3.** El usuario pulsa el botón "Cargar de nuevo" **2.a.4.** Volvemos al paso 2 **3.a.** No hay simulaciones creadas **3.a.1.** El sistema informa al usuario de la situación **3.a.2.** Fin del caso de uso

Tabla D.19: Caso de Uso RF07.1.

Caso de prueba

```
Feature: (RF07.1) Visualizar Listado Simulaciones

Background:
  Given un usuario que ha iniciado sesión

Scenario: (01) Listado Mostrado Correctamente
  When el usuario pulsa la sección "Simulaciones"
  And el sistema obtiene las simulaciones
  And hay simulaciones
  Then el sistema muestra el listado de simulaciones

Scenario: (02) Error al obtener las simulaciones
  When el usuario pulsa la sección "Simulaciones"
  And el sistema falla al obtener las simulaciones
```

```
  Then el sistema muestra el botón "Cargar de Nuevo"

Scenario: (03) No hay simulaciones
  When el usuario pulsa la sección "Simulaciones"
  And el sistema obtiene las simulaciones
  And no hay simulaciones
  Then el sistema muestra un mensaje "No hay simulaciones"
```

Tabla D.20: Caso de Prueba RF07.1.

RF07.2: Los usuarios podrán visualizar los parámetros de entrada y datos básicos de una simulación

Caso de uso

CASO DE USO	**RF07.2:** Los usuarios podrán visualizar los parámetros de entrada y datos básicos de una simulación
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán visualizar los datos asociados a una simulación.	
PRECONDICIÓN	
El usuario ha iniciado sesión, se encuentra en la ventana "Simulaciones", ha cargado correctamente el listado y tiene alguna simulación creada.	
POST CONDICIÓN MÍNIMA	
El usuario visualizará los datos.	
POSTCONDICIÓN DE ÉXITO	
El usuario visualizará los datos.	
ESCENARIO PRINCIPAL DE ÉXITO	

1. El usuario pulsa en el botón "..." de una de las simulaciones del listado **2.** El sistema muestra un listado de acciones **3.** El usuario pulsa la opción "Ver detalle" **4.** El sistema muestra los datos de la simulación
ESCENARIOS ALTERNATIVOS

Tabla D.21: Caso de Uso RF07.2.

Caso de prueba

```
Feature: (RF07.2) Visualizar datos Simulación

Background:
  Given un usuario que ha iniciado sesión
  And se encuentra en "Simulaciones"
  And se muestran simulaciones en el listado

Scenario: (01) Datos mostrados correctamente
  When el usuario pulsa el botón "..."
  And el sistema muestra el listado de acciones
  And el usuario pulsa el botón "Ver Detalle"
  Then el sistema los datos de la simulación
```

Tabla D.22: Caso de Prueba RF07.2.

RF07.3: Los usuarios podrán visualizar el mapa del entorno de una simulación

Caso de uso

CASO DE USO	**RF07.3:** Los usuarios podrán visualizar el mapa del entorno de una simulación
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	

Los usuarios que estén registrados en la aplicación y tenga alguna simulación creada, querrán visualizar el mapa del entorno en el que se ha realizado la simulación
PRECONDICIÓN
El usuario ha iniciado sesión, tiene alguna simulación creada y se encuentra en el detalle de una de ellas.
POST CONDICIÓN MÍNIMA
El usuario verá el mapa de la simulación
POSTCONDICIÓN DE ÉXITO
El usuario verá el mapa de la simulación
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario selecciona la pestaña "Mapa" en el detalle **2.** El sistema obtiene los datos para mostrar el mapa **3.** El sistema muestra los datos en la interfaz gráfica **4.** El sistema muestra los controles para manejar la simulación
ESCENARIOS ALTERNATIVOS
2.a. Se ha producido un error al obtener los datos del mapa **2.a.1.** El sistema informa al usuario de la situación **2.a.2.** El sistema muestra el botón "Cargar de Nuevo" **2.a.3.** El usuario pulsa el botón "Cargar de Nuevo" **2.a.4.** Volvemos al paso 2

Tabla D.23: Caso de Uso RF07.3.

Caso de prueba

```
Feature: (RF07.3) Visualizar Mapa

Background:
  Given un usuario que ha iniciado sesión
  And tiene simulaciones creadas
  And está en "Detalle"

Scenario: (01) Mapa mostrado correctamente
  When el usuario pulsa la pestaña "Mapa"
```

```
  And el sistema obtiene los datos correctamente
  Then el sistema muestra los datos en el mapa
  And el sistema muestra los controles de la simulación

Scenario: (02) Error al obtener los datos
  When el usuario pulsa la pestaña "Mapa"
  And el sistema falla al obtener los datos
  Then el sistema muestra el botón "Cargar de Nuevo"
```

Tabla D.24: Caso de Prueba RF07.3.

RF07.4: Los usuarios podrán visualizar las gráficas generadas a partir de los datos de salida de una simulación

Caso de uso

CASO DE USO	**RF07.4:** Los usuarios podrán visualizar las gráficas generadas a partir de los datos de salida de una simulación
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación y tengan alguna simulación creada, querrán visualizar las gráficas generadas a partir de los datos de entrada.	
PRECONDICIÓN	
El usuario ha iniciado sesión, tiene alguna simulación creada, se encuentra en el detalle de una de ellas y se han generado las gráficas previamente	
POST CONDICIÓN MÍNIMA	
El usuario será informado sobre si ha sido posible obtener las gráficas.	
POSTCONDICIÓN DE ÉXITO	
El usuario verá las gráficas generadas.	
ESCENARIO PRINCIPAL DE ÉXITO	

1. El usuario selecciona la pestaña "Gráficas" en el detalle. **2.** El sistema obtiene las imágenes de las gráficas. **3.** El sistema muestra las imágenes.
ESCENARIOS ALTERNATIVOS
2.a. Se ha producido un error al obtener las gráficas **2.a.1.** El sistema informa al usuario de la situación **2.a.2.** Fin del caso de uso

Tabla D.25: Caso de Uso RF07.4.

Caso de prueba

```
Feature: (RF07.4) Visualizar Gráficas

Background:
  Given un usuario que ha iniciado sesión
  And tiene simulaciones creadas
  And está en "Detalle"
  And se han generado las gráficas previamente

Scenario: (01) Gráficas mostradas correctamente
  When el usuario pulsa la pestaña "Gráficas"
  And el sistema obtiene los datos correctamente
  Then el sistema muestra las imágenes de las gráficas

Scenario: (02) Error al obtener las gráficas
  When el usuario pulsa la pestaña "Gráficas"
  And el sistema falla al obtener los datos
  Then el sistema muestra un mensaje de error
```

Tabla D.26: Caso de Prueba RF07.4.

RF08: Los usuarios podrán editar el nombre y la descripción de las simulaciones

Caso de uso

CASO DE USO	**RF08:** Los usuarios podrán editar el nombre y la descripción de las simulaciones
ÁMBITO	Simulaciones

ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación y tenga alguna simulación creada, querrán editar el nombre y la simulación	
PRECONDICIÓN	
El usuario ha iniciado sesión, tiene alguna simulación creada y se encuentra en el detalle de una de ellas.	
POST CONDICIÓN MÍNIMA	
El usuario sabrá si se han editado los datos	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá editado los datos de la simulación y podrá visualizarlo.	
ESCENARIO PRINCIPAL DE ÉXITO	
1. El usuario rellena los campos Nombre y Descripción. **2.** El usuario selecciona el botón "Guardar Cambios". **3.** El sistema verifica que los campos han sido rellenados correctamente **4.** El sistema almacena los cambios indicados **5.** El sistema muestra un mensaje de éxito. **6.** El sistema recarga el detalle de la simulación.	
ESCENARIOS ALTERNATIVOS	
3.a. No se han rellenado todos los campos **3.a.1.** El sistema informa al usuario de la situación **3.a.2.** Volvemos al paso 1 **4.a.** Se produce un error al guardar los datos **4.a.1.** El sistema informa al usuario de la situación **4.a.2.** Fin del caso de uso	

Tabla D.27: Caso de Uso RF08.

Caso de prueba

```
Feature: (RF08) Editar Simulacion

Background:
  Given un usuario que ha iniciado sesión
  And tiene simulaciones creadas
  And está en "Detalle"

Scenario: (01) Datos editados correctamente
  When el usuario rellena los campos Nombre y Descripción
  And el usuario pulsa el botón "Guardar Cambios"
  And el sistema guarda los nuevos datos de la simulación
  Then el sistema muestra un mensaje de éxito
  And el sistema muestra los nuevos datos cargados

Scenario: (02) Campos vacíos
  When el usuario no rellena todos los campos
  And el usuario pulsa el botón "Guardar Cambios"
  Then el sistema muestra un mensaje de error

Scenario: (03) Error al guardar los datos
  When el usuario rellena los campos Nombre y Descripción
  And el usuario pulsa el botón "Guardar Cambios"
  And el sistema falla al guardar los datos
  Then el sistema muestra un mensaje de error "No se ha podido editar la simulación"
```

Tabla D.28: Caso de Prueba RF08.

RF09: Los usuarios podrán eliminar las simulaciones

Caso de uso

CASO DE USO	**RF09:** Los usuarios podrán eliminar las simulaciones
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	

Los usuarios que estén registrados en la aplicación y tenga alguna simulación creada, querrán poder eliminarla
PRECONDICIÓN
El usuario ha iniciado sesión, tiene alguna simulación creada y se encuentra en el detalle de una de ellas.
POST CONDICIÓN MÍNIMA
El usuario sabrá si se ha eliminado la simulación
POSTCONDICIÓN DE ÉXITO
El usuario habrá eliminado la simulación
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario selecciona la pestaña "Acciones" en el detalle **2.** El sistema muestra el contenido de la pestaña Acciones **3.** El usuario pulsa el botón "Eliminar esta simulación" **4.** El sistema muestra un modal de confirmación para realizar la acción. **5.** El usuario pulsa el botón de confirmación. **6.** El sistema elimina la simulación. **7.** El sistema muestra un mensaje de éxito **8.** El sistema muestra al usuario la pantalla "Simulaciones"
ESCENARIOS ALTERNATIVOS
5.a. El usuario cancela la acción **5.a.1.** Fin del caso de uso **5.a.** Se produce un error al eliminar la simulación **5.a.1.** El sistema informa al usuario de la situación **5.a.2.** Fin del caso de uso

Tabla D.29: Caso de Uso RF09.

Caso de prueba

```
Feature: (RF09) Eliminar simulación

Background:
  Given un usuario que está registrado
  And tiene simulaciones creadas
```

```
  And está en "Detalle"
  And está en "Acciones"
  And pulsa el botón "Eliminar esta simulación"
  And pulsa el botón "Eliminar Simulacion" en el modal de confirmación

Scenario: (01) Simulación eliminada correctamente
  When el sistema elimina la simulación correctamente
  Then el sistema muestra al usuario la pantalla de simulaciones
  And el sistema muestra un mensaje de éxito "Simulación eliminada correctamente"

Scenario: (02) Error al eliminar la simulación
  When hay un error al eliminar la simulación en el sistema
  Then el sistema cierra el modal de confirmación
  And el sistema muestra un mensaje de error "No se ha podido eliminar la
simulación"
```

Tabla D.30: Caso de Prueba RF09.

RF10: Los usuarios podrán descargar los parámetros de entrada de una simulación

Caso de uso

CASO DE USO	**RF10:** Los usuarios podrán descargar los parámetros de entrada de una simulación
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación y haya creado una simulación, querrán descargar un archivo .json con los parámetros con los que se ha generado ésta	
PRECONDICIÓN	
El usuario ha iniciado sesión, tiene alguna simulación creada y se encuentra en el detalle de una de ellas (los datos del detalle han cargado correctamente)	
POST CONDICIÓN MÍNIMA	

El usuario habrá obtenido el archivo de los parámetros de entrada de la simulación.
POSTCONDICIÓN DE ÉXITO
El usuario habrá obtenido el archivo de los parámetros de entrada de la simulación.
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario pulsa el botón "Descargar .json" **2.** El sistema envía al navegador del usuario el archivo json para que el usuario pueda visualizarlo o descargarlo.
ESCENARIOS ALTERNATIVOS

Tabla D.31: Caso de Uso RF10.

Caso de prueba

```
Feature: (RF10) Descargar JSON Datos Entrada

Background:
  Given un usuario que ha iniciado sesión
  And tiene simulaciones creadas
  And está en "Detalle"
  And los datos del detalle han cargado correctamente

Scenario: (01) Archivo descargado correctamente
  When el usuario pulsa el botón "Descargar .json"
  Then el sistema envía al navegador el archivo
```

Tabla D.32: Caso de Prueba RF10.

RF11: Los usuarios podrán descargar los datos de salida de los delitos generados en una simulación

Caso de uso

CASO DE USO	**RF11:** Los usuarios podrán descargar los datos de salida de los delitos generados en una simulación
ÁMBITO	Simulaciones

ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación y haya creado una simulación, querrán obtener un csv con los delitos generados (potenciales y realizados)	
PRECONDICIÓN	
El usuario ha iniciado sesión, tiene alguna simulación creada y se encuentra en el detalle de una de ellas.	
POST CONDICIÓN MÍNIMA	
El usuario habrá si se ha podido descargar el archivo.	
POSTCONDICIÓN DE ÉXITO	
El usuario habrá obtenido el archivo de los datos de salida de la simulación	
ESCENARIO PRINCIPAL DE ÉXITO	
1. El selecciona la pestaña "Acciones" en el detalle **2.** El sistema muestra el contenido de la pestaña Acciones **3.** El usuario pulsa el botón "Descargar Datos de Salida" **4.** El sistema obtiene los datos del archivo de salida **5.** El sistema envía al navegador del usuario el archivo csv para que el usuario pueda visualizarlo o descargarlo.	
ESCENARIOS ALTERNATIVOS	
4.a. Se produce un error al obtener los datos de salida de la simulación **4.a.1.** El sistema informa al usuario de la situación **4.a.2.** Fin del caso de uso	

Tabla D.33: Caso de Uso RF11.

Caso de prueba

```
Feature: (RF11) Descargar CSV Salida

Background:
  Given un usuario que ha iniciado sesión
```

```
  And tiene simulaciones creadas
  And está en "Detalle"

Scenario: (01) Archivo descargado correctamente
  When el usuario pulsa la pestaña "Acciones"
  And pulsa el botón "Descargar Datos de Salida"
  And el sistema obtiene el archivo de salida correctamente
  Then el sistema envía al navegador el archivo

Scenario: (02) Error al obtener los datos
  When el usuario pulsa la pestaña "Acciones"
  And pulsa el botón "Descargar Datos de Salida"
  And el sistema no obtiene el archivo de salida correctamente
  Then el sistema informa al usuario de la situación "No se ha podido descargar el
archivo"
```

Tabla D.34: Caso de Prueba RF11.

RF12: Los usuarios podrán filtrar las simulaciones del listado de simulaciones

Caso de uso

CASO DE USO	**RF12:** Los usuarios podrán filtrar las simulaciones del listado de simulaciones
ÁMBITO	Simulaciones
ACTOR PRINCIPAL	Usuario
PARTICIPANTES	• Usuario
OBJETIVOS E INTERESES	
Los usuarios que estén registrados en la aplicación querrán filtrar por nombre las simulaciones.	
PRECONDICIÓN	
El usuario ha iniciado sesión, tiene simulaciones creadas y se encuentra en la ventana "Simulaciones".	
POST CONDICIÓN MÍNIMA	
El usuario habrá filtrado el listado de simulaciones.	

POSTCONDICIÓN DE ÉXITO
El usuario habrá filtrado el listado de simulaciones.
ESCENARIO PRINCIPAL DE ÉXITO
1. El usuario pulsa la marca para desplegar la sección de los filtros. **2.** El sistema muestra la sección de filtros **3.** El usuario introduce un nombre para filtrar **4.** El usuario pulsa el botón "Filtrar" **5.** El sistema obtiene un nuevo listado filtrando por el nombre **6.** El sistema muestra el nuevo listado
ESCENARIOS ALTERNATIVOS
3.a. El usuario deja vacío el campo de Nombre **3.a.1.** El usuario pulsa el botón "Filtrar" **3.a.2.** El sistema muestra el listado completo **3.a.3.** Fin del caso de uso **5.a.** Al filtrar no hay ninguna coincidencia **5.a.1.** El sistema informa al usuario de la situación **5.a.2.** Fin del caso de uso

Tabla D.35: Caso de Uso RF12.

Caso de prueba

```
Feature: (RF12) Filtrar simulaciones

Background:
  Given un usuario que ha iniciado sesión
  And tiene simulaciones creadas
  And está en "Simulaciones"
  And el usuario pulsa en el botón de desplegar sección de filtros

Scenario: (01) Filtrado correctamente
  When el usuario introduce un nombre
  And el usuario pulsa el botón "Filtrar"
  And hay simulaciones que coinciden con el filtro
  Then el sistema muestra el listado filtrado

Scenario: (02) No hay coincidencias
```

```
  When el usuario introduce un nombre
  And el usuario pulsa el botón "Filtrar"
  And no hay simulaciones que coincidan con el filtro
  Then el sistema informa al usuario de la situación

Scenario: (03) No introduce nombre
  When el usuario no introduce un nombre
  And el usuario pulsa el botón "Filtrar"
  Then el sistema muestra el listado completo
```

Tabla D.36: Caso de Uso RF12.

Apéndice E

Pruebas realizadas

Iteración 0

Remoto -> master
Local -> release

ID	Caso Prueba	Requisito	Fecha	Realizado Por	Entorno	Pasa Test	Issue Bug	Solucionado	Notas	Resultado obtenido
it0-1	RF01-01	RF01	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-2	RF01-02	RF01	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-3	RF01-03	RF01	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-4	RF01-04	RF01	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-5	RF01-05	RF01	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-6	RF01-06	RF01	05/05/2020	Victoria	Local	NO	Angular #38	SI. Nueva prueba: it0-29	X	El mensaje se encuentra oculto por la sombra del modal
it0-7	RF02-01	RF02	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-8	RF02-02	RF02	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-9	RF02-03	RF02	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-10	RF02-04	RF02	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-11	RF02-05	RF02	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-12	RF02-06	RF02	05/05/2020	Victoria	Local	NO	Angular #38	SI. Nueva prueba: it0-30	X	El mensaje se encuentra oculto por la sombra del modal
it0-13	RF03-01	RF03	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-14	RF04.1-01	RF04.1	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-15	RF04.1-02	RF04.1	05/05/2020	Victoria	Local	NO	Angular #39	SI. Nueva prueba:	Se ha intentado subir un archivo en formato .txt	Mensaje mostrado: ALERTS.PERFIL.IMAGEN.ERROR-FORMATO Mensaje esperado: "Formato de imagen no válido"
it0-16	RF04.1-03	RF04.1	05/05/2020	Victoria	Local	X	X	X	No se puede probar	X
it0-17	RF04.1-04	RF04.1	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-18	RF04.2-01	RF04.2	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-19	RF04.2-02	RF04.2	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-20	RF04.3-01	RF04.3	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-21	RF04.3-02	RF04.3	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-22	RF04.3-03	RF04.3	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-23	RF04.3-03	RF04.3	05/05/2020	Victoria	Local	NO	Angular #40	SI. Nueva prueba: it0-31	X	Al introducir un email con este formato: "victoriaca#raquel@a" no indica ningún mensaje de error. Esperado: "Tiene que ser una direccion de email válida"
it0-24	RF04.3-04	RF04.3	05/05/2020	Victoria	Local	NO	Angular #41	SI. Nueva prueba: RF04.3-04	X	No vuelve a cargar los datos del usuario
it0-25	RF04.4-01	RF04.4	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-26	RF04.4-02	RF04.4	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-27	RF05-01	RF05	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-28	RF05-02	RF05	05/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-29	RF01-06	RF01	08/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-30	RF02-06	RF02	08/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-31	RF04.1-02	RF04.1	08/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-32	RF04.3-04	RF04.3	08/05/2020	Victoria	Local	SI	X	X	X	Esperado
it0-33	RF01-01	RF01	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-34	RF01-02	RF01	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-35	RF01-03	RF01	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-36	RF01-04	RF01	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-37	RF01-05	RF01	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-38	RF01-06	RF01	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-39	RF02-01	RF02	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-40	RF02-02	RF02	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-41	RF02-03	RF02	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-42	RF02-04	RF02	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-43	RF02-05	RF02	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-44	RF02-06	RF02	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-45	RF03-01	RF03	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-46	RF04.1-01	RF04.1	29/05/2020	Victoria	Remoto	NO	Angular #48	SI. Nueva prueba: it0-60		Aparece el mensaje "No se ha podido modificar la imagen de perfil", cuando debería haberla cambiado.
it0-47	RF04.1-02	RF04.1	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-48	RF04.1-03	RF04.1	29/05/2020	Victoria	Remoto	X	X	X	No se puede probar	X
it0-49	RF04.1-04	RF04.1	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-50	RF04.2-01	RF04.2	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-51	RF04.2-02	RF04.2	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-52	RF04.3-01	RF04.3	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-53	RF04.3-02	RF04.3	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-54	RF04.3-03	RF04.3	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-55	RF04.3-04	RF04.3	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-56	RF04.4-01	RF04.4	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-57	RF04.4-02	RF04.4	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-58	RF05-01	RF05	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-59	RF05-02	RF05	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado
it0-60	RF04.1-01	RF04.1	29/05/2020	Victoria	Remoto	SI	X	X	X	Esperado

Figura E.1: Pruebas iteración 0.

Iteración 1

Remoto -> master
Local -> release

ID	Caso Prueba	Requisito	Fecha	Realizado Por	Entorno	Pasa Test	Issue Bug	Solucionado	Notas	Resultado obtenido
it1-1	RF06-01	RF06	22/06/2020	Victoria	Local	SI	X	X	X	
it1-2	RF06-02	RF06	22/06/2020	Victoria	Local	NO	Angular #64	SI. Nueva prueba: it1-25	X	Aparece el mensaje "Formato de imagen no válido"
it1-3	RF06-03	RF06	22/06/2020	Victoria	Local	NO	Angular #64	SI. Nueva prueba: it1-26	X	Aparece el mensaje "Formato de imagen no válido"
it1-4	RF06-04	RF06	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-5	RF06-05	RF06	22/06/2020	Victoria	Local	NO	Angular #65	SI. Nueva prueba: it1-28	X	Aparece el mensaje "No se ha podido crear la simulación", debería indicar que está mal la entrada
it1-6	RF06-05	RF06	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-7	RF06-07	RF06	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-8	RF07.1-01	RF07.1	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-9	RF07.1-02	RF07.1	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-10	RF07.1-03	RF07.1	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-11	RF07.2-01	RF07.2	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-12	RF07.3-01	RF07.3	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-13	RF07.3-02	RF07.3	22/06/2020	Victoria	Local	NO	Angular #66	SI. Nueva prueba: it1-27	Se produce cuando se carga el detalle por enlace y no está disponible el servicio	Se queda el spinner dando vueltas de forma infinita
it1-14	RF08-01	RF08	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-15	RF08-02	RF08	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-16	RF08-03	RF08	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-17	RF09-01	RF09	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-18	RF09-02	RF09	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-19	RF10-01	RF10	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-20	RF11-01	RF11	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-21	RF11-02	RF11	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-22	RF12-01	RF12	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-23	RF12-02	RF12	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-24	RF12-03	RF12	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-25	RF06-02	RF06	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-26	RF06-03	RF06	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-27	RF07.3-02	RF07.3	22/06/2020	Victoria	Local	SI	X	X	X	X
it1-28	RF06-05	RF06	23/06/2020	Victoria	Local	SI	X	X	X	X

Figura E.2: Pruebas iteración 1.

Apéndice F

Horas de Tareas Realizadas

	MInutos	Horas
1-Angular	5710	95:10:00
2-API Gateway	1130	18:50:00
3-Design	4415	73:35:00
4-Microservice Usuario	530	8:50:00
5-Microservice Simulacion	6305	105:05:00

	Minutos	Horas
TOTAL	**18090**	301:30:00

Tabla F.1: Tiempos totales.

Angular

Referencia	Nombre	Horas	Minutos	Minutos Totales	Horas Totales	Tipo	Iteracion
#1	Crear proyecto con estructura inicial	1	20	**80**	1:20:00		it0
#2	Añadir barra de navegación		30	**30**	0:30:00		it0
#3	Configurar Cuenta de Google para usar OAuth		20	**20**	0:20:00		it0
#4	Implementar Inicio de Sesión	2	55	**175**	2:55:00		it0
#5	Implementar Cierre de Sesión		15	**15**	0:15:00		it0
#6	Implementar acceso al perfil del usuario		10	**10**	0:10:00		it0
#7	Curso Angular Architecture and Best Practices	4	40	**280**	4:40:00		it0
#8	Configurar Netlify	1		**60**	1:00:00		it0
#9	Configurar archivo para debugear		20	**20**	0:20:00		it0

#10	Crear submenu para el usuario		30	**30**	0:30:00		it0
#11	Investigar Internacionalizacion		20	**20**	0:20:00		it0
#12	Añadir i18n al proyecto	1		**60**	1:00:00		it0
#13	Añadir i18n a la barra de navegación		15	**15**	0:15:00		it0
#14	Bug LoggerService		50	**50**	0:50:00	bug	it0
#15	Investigación Implementación Mapa	2		**120**	2:00:00		it1
#16	Crear ModalService	2	30	**150**	2:30:00		it0
#17	Crear Alert Service		50	**50**	0:50:00		it0
#18	Añadir un spinner al cargar el usuario		10	**10**	0:10:00		it0
#19	Sistema de pestañas en el perfil		40	**40**	0:40:00		it0
#20	Formulario para editar perfil	7		**420**	7:00:00		it0
#21	Servicio Imgur		40	**40**	0:40:00		it0
#22	Funcionalidad de borrar la cuenta		50	**50**	0:50:00		it0
#23	Bug: Barra de navegación no fija		5	**5**	0:05:00	bug	it0
#24	Bug: Al cerrar el pop-up de google, sigue el spinner		5	**5**	0:05:00	bug	it0
#26	JWT Interceptor		40	**40**	0:40:00		it0
#27	Guard para controlar las rutas que requieren permisos	1	40	**100**	1:40:00		it0
#28	Separar en un componente el boton de inicio de sesion		40	**40**	0:40:00		it0
#29	Bug: Al iniciar sesion desde el modal, no cambia la barra	1	30	**90**	1:30:00	bug	it0
#30	Bug: No muestra la informacion del usuario		10	**10**	0:10:00	bug	it0
#31	Bug: No muestra bien el modal		10	**10**	0:10:00	bug	it0
#32	Formulario Preferencias -> editar datos	1	50	**110**	1:50:00		it0
#33	Traducir funcionalidades It0	2	20	**140**	2:20:00		it0
#34	Bug: Cambiar entre pestañas al editar perfil		15	**15**	0:15:00	bug	it0
#35	Comprobar el formato de la imagen de perfil		20	**20**	0:20:00		it0
#36	prototipo simulacion	7	40	**460**	7:40:00		it1
#37	Bug: al cambiar el idioma, hay que cambiar el idioma del alert	0	0	**0**	0:00:00	bug	it0
#38	Bug: alert oculto por modal	1	15	**75**	1:15:00	bug	it0
#39	Bug: Mensaje al subir foto con formato incorrecto		5	**5**	0:05:00	bug	it0
#40	Bug: Formato email		10	**10**	0:10:00	bug	it0
#41	Bug: cargar datos usuario		30	**30**	0:30:00	bug	it0
#42	Interfaz - Mostrar Listado de Simulaciones	3	5	**185**	3:05:00		it1

#43	Ver detalle simulacion - Sistema de pestañas		30	**30**	0:30:00		it1
#44	Ver detalle simulación - DATOS	2	40	**160**	2:40:00		it1
#45	Interfaz - Ver detalle simulación - Ver simulación (Mapa)	6	40	**400**	6:40:00		it1
#46	Interfaz - Ver detalle simulación - Acciones		45	**45**	0:45:00		it1
#47	Organizar modulo simulacion		30	**30**	0:30:00		it1
#48	Bug: Siempre muestra error al guardar la imagen en remoto		20	**20**	0:20:00		it0
#49	Pruebas en remoto it0		25	**25**	0:25:00		it0
#50	Bug: Eliminar avatar		15	**15**	0:15:00		it0
#51	Bug: Editar datos		0	**0**	0:00:00		it0
#52	Bug: Editar preferencias		0	**0**	0:00:00		it0
#54	Interfaz- Crear simulación y resumen	7	30	**450**	7:30:00		it1
#55	Mostrar en canvas simulacion delitos	2	25	**145**	2:25:00		it1
#56	Crear simulacion Angular - Llamada API	3	50	**230**	3:50:00		it1
#57	Detalle simulacion -Usando API	4	10	**250**	4:10:00		it1
#58	Listado simulación - Usando API	3		**180**	3:00:00		it1
#59	Eliminar simulacion	1	5	**65**	1:05:00		it1
#60	Descargar CSV salida	2	25	**145**	2:25:00		it1
#61	i18n iteracion 1	2	30	**150**	2:30:00		it1
#62	Bug: Al cerrar sesión, hay que borrar la caché		20	**20**	0:20:00	bug	it1
#63	Filtros simulacion		25	**25**	0:25:00	bug	it1
#64	Bug: formato de imagen no válido		10	**10**	0:10:00	bug	it1
#65	Bug: mensaje entrada incorrecta json		55	**55**	0:55:00	bug	it1
#66	Bug: Spinner infinito mapa		25	**25**	0:25:00	bug	it1
#67	Mostrar gráficas	1		**60**	1:00:00		it1
#68	Pagina de inicio		40	**40**	0:40:00		it1
#69	Simulaciones Desplegar app		40	**40**	0:40:00		it1

Tabla F.2: Tiempos Angular.

API Gateway

Referencia	Nombre	Horas	Minutos	Minutos Totales	Horas Totales	Tipo	Iteracion
#1	Creación Estructura Inicial del Proyecto	1		**60**	1:00:00		it0
#2	Configuración del repositorio	2	0	**120**	2:00:00		it0
#3	Creación de la ramas		5	**5**	0:05:00		it0

#4	Creación de la aplicación en heroku		5	**5**	0:05:00		it0
#5	Endpoint /login	5	20	**320**	5:20:00		it0
#6	Endpoint infoUsuario		30	**30**	0:30:00		it0
#7	Bug: Error al desplegar en heroku			**0**	0:00:00	bug	it0
#8	Problema CORS	1		**60**	1:00:00	bug	it0
#9	Bug: Error al desplegar en Heroku -> dataclass		25	**25**	0:25:00	bug	it0
#10	Endpoint editar usuario		10	**10**	0:10:00		it0
#11	Endpoint eliminar usuario		10	**10**	0:10:00		it0
#12	Middleware para obtener el token	1		**60**	1:00:00		it0
#15	Subir fotos a imgur	1	10	**70**	1:10:00		it0
#16	Endpoints Simulacion	2	25	**145**	2:25:00		it1
#17	Listado de simulaciones		55	**55**	0:55:00		it1
#18	Crear simulacion -> llamada a ms-simulacion	1	5	**65**	1:05:00		it1
#19	Editar Simulación		30	**30**	0:30:00		it1
#20	Mostrar graficas	1		**60**	1:00:00		it1

Tabla F.3: Tiempos API Gateway.

Design

Referencia	Nombre	Horas	Minutos	Minutos Totales	Horas Totales	Tipo	Iteracion
#1	Boceto Historia de usuario	1	0	**60**	1:00:00		it0
#2	Requisitos It1	0	30	**30**	0:30:00		it0
#3	Figma Historia de usuario	3	0	**180**	3:00:00		it0
#4	Modelado Base de Datos Usuario	0	15	**15**	0:15:00		it0
#5	Casos de uso it0	2	10	**130**	2:10:00		it0
#6	Casos de prueba it0	2	10	**130**	2:10:00		it0
#7	Generación esquemas de documentación	1	40	**100**	1:40:00		it0
#9	Documento pruebas ejecutadas		10	**10**	0:10:00		it0
#10	Realizar pruebas it0	1	20	**80**	1:20:00		it0
#11	Boceto simulaciones	2	35	**155**	2:35:00		it1
#12	Figma Simulaciones	5	15	**315**	5:15:00		it1
#13	Requisitos it1	1	15	**75**	1:15:00		it1
#14	Casos de uso it1	2	45	**165**	2:45:00		it1
#15	Casos de prueba it1	1	10	**70**	1:10:00		it1
#16	Realizar pruebas it1	1	40	**100**	1:40:00		it1
#17	Presentación Hackaton	3		**180**	3:00:00		it1
#18	Estructura de la memoria	2	35	**155**	2:35:00		it2
#19	Curso de microservicios	1	15	**75**	1:15:00		it2

#20	Memoria: arquitectura microservicios y api gateway	1	15	**75**	1:15:00		it2
#21	Memoria: tecnologias usadas	3	45	**225**	3:45:00		it2
#22	Memoria: punto 4 (gitlab)		50	**50**	0:50:00		it2
#23	Memoria: fases del trabajo	5	15	**315**	5:15:00		it2
#24	Memoria: rama y gestión de despliegues	1	35	**95**	1:35:00		it2
#25	Memoria: requisitos	1	30	**90**	1:30:00		it2
#26	Memoria: realización de diagramas	3	10	**190**	3:10:00		it2
#27	Memoria: modelado simulación	7	5	**425**	7:05:00		it2
#28	Memoria: modelado bd		15	**15**	0:15:00		it2
#29	Memoria: mantenimiento y pruebas		25	**25**	0:25:00		it2
#30	Memoria: conclusiones		50	**50**	0:50:00		it2
#31	Memoria: formato de word	7	40	**460**	7:40:00		it2
#32	Memoria: librerias		40	**40**	0:40:00		it2
#33	Memoria: jwt y oauth		30	**30**	0:30:00		it2
#34	Memoria: implementación angular	1		**60**	1:00:00		it2
#35	Memoria: implementacion backend		45	**45**	0:45:00		it2
#36	Memoria: Introduccion	1	30	**90**	1:30:00		it2
#37	Manual de usuario	1	50	**110**	1:50:00		it2

Tabla F.4: Tiempos Design.

Microservice Usuario

Referencia	Nombre	Horas	Minutos	Minutos Totales	Horas Totales	Tipo	Iteracion
#1	Configuración inicial del proyecto		15	**15**	0:15:00		it0
#2	Generar conexión a la base de datos	1	30	**90**	1:30:00		it0
#3	Endpoint /login		55	**55**	0:55:00		it0
#4	Endpoint infoUsuario		30	**30**	0:30:00		it0
#5	Generar modelo para base de datos	1	30	**90**	1:30:00		it0
#6	Investigar los diferentes ORMs	2		**120**	2:00:00		it0
#7	Crear cuenta en Mongo Atlas		20	**20**	0:20:00		it0
#8	Verificar el token de google		50	**50**	0:50:00		it0
#9	Endpoint editar usuario		50	**50**	0:50:00		it0
#10	endpoint eliminar usuario		10	**10**	0:10:00		it0

Tabla F.5: Tiempos Usuario.

Microservice Simulación

Referencia	Nombre	Horas	Minutos	Minutos Totales	Horas Totales	Tipo	Iteracion
#1	Investigar Simulación	1	15	**75**	1:15:00		it1
#2	Inicializar Proyecto	1	35	**95**	1:35:00		it1
#3	Aprendizaje Framework MESA	4	35	**275**	4:35:00		it1
#4	Crear Endpoint con Simulación de Ejemplo	3		**180**	3:00:00		it1
#5	Simulacion delitos	12	40	**760**	12:40:00		it1
#6	Algoritmo A*	4	55	**295**	4:55:00		it1
#7	Sistema de rutinas en simulación	4	35	**275**	4:35:00		it1
#8	Vehiculos en simulacion	2	30	**150**	2:30:00		it1
#9	Bug: A* calcula mal el camino	1	10	**70**	1:10:00	bug	it1
#10	Refactorizacion + pathfinding y camino	2	45	**165**	2:45:00		it1
#11	Cometer delito	10	30	**630**	10:30:00		it1
#12	Agregar zonas policiales	3	50	**230**	3:50:00		it1
#13	Manejar delitos por zonas	7	40	**460**	7:40:00		it1
#14	Bug: Realiza mal la huida		20	**20**	0:20:00	bug	it1
#15	Gestion de los policias	4	55	**295**	4:55:00		it1
#17	Comportamiento ciudadano al ver delito	1	40	**100**	1:40:00		it1
#18	Gestión de los delitos de vivienda		40	**40**	0:40:00		it1
#19	Creación de los agentes por inputs	5	15	**315**	5:15:00		it1
#20	Generación de datos de salida	4	5	**245**	4:05:00		it1
#23	Arreglar sistema de vehiculos	3	35	**215**	3:35:00		it1
#25	Refactorizacion	3	10	**190**	3:10:00		it1
#27	Generar datos de rutinas	3	15	**195**	3:15:00		it1
#28	Bug: Vehiculo se queda pillado		50	**50**	0:50:00	bug	it1
#29	Mostrar simulacion delitos en angular	2	35	**155**	2:35:00		it1
#30	Curso de python de matplotlib	1	30	**90**	1:30:00		it1
#31	Guardar simulacion		55	**55**	0:55:00		it1
#32	Obtener listado de simulaciones		55	**55**	0:55:00		it1
#33	Endpoint eliminar Simulacion		25	**25**	0:25:00		it1
#34	Endpoint obtener datos de una simulación	1	5	**65**	1:05:00		it1
#36	Guardar datos de entrada simulacion	2	30	**150**	2:30:00		it1
#37	Editar Simulación		25	**25**	0:25:00		it1
#38	Generación de gráficas simulacion	6		**360**	6:00:00		it1

Tabla F.6: Tiempos simulación.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga