MEMORIA DEL PROYECTO

PLATAFORMA DE SIMULACIÓN DE DEMANDA DE VEHÍCULOS DE MOVILIDAD LIGERA

Trabajo de Fin de Grado

INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

TUTORES

Hernández González, Guillermo

Rodríguez González, Sara

González Briones, Alfonso

AUTOR

Andrei Brinza

RESUMEN

El objetivo de este trabajo ha sido la realización de una plataforma de simulación de demanda de vehículos de movilidad ligera en la cual el usuario pudiera poner a prueba casos de estudio, desplegando vehículos en zona geográfica determinada.

En la actualidad cada vez es más frecuente observar en las ciudades redes de vehículos como bicicletas o patinetes eléctricos en las cuales los usuarios, con una acción simple en sus dispositivos, pueden localizar los vehículos cercanos a su posición para poder moverse por la ciudad de forma poco contaminante.

Para poner a prueba los casos de estudio se decidió usar como mapa de referencia la ciudad de Utrecht de los Países Bajos, que cuenta con una gran red de carriles de bicicletas y que nos permitía encontrar una zona idónea para poner a prueba nuestra simulación.

El desarrollo de la plataforma se ha enfocado hacia una aplicación web en la cual cualquier usuario desde un dispositivo compatible pudiera acceder para realizar simulaciones sin tener que instalar nada. La aplicación web nos va a proporcionar acceso a la simulación con una cuenta de usuario en la que podremos gestionar nuestro perfil y podremos realizar las simulaciones que deseemos. La simulación nos permite desplegar vehículos en varios puntos de la ciudad de Utrecht, así como observar la tendencia de movimiento de nuestro vehículo hasta un destino que hayamos elegido. Además, la simulación nos permite desencadenar una serie de eventos asociados a los puntos del mapa y que nos permiten ver en tiempo real las repercusiones sobre nuestro vehículo.

La herramienta principal para el componente de simulación es el *framework* de AnyLogic*,* el cual esta implementado en el lenguaje Java. Este entorno de desarrollo es ideal para poder crear una simulación ya que cuenta con un amplio catálogo de funciones y sobre todo con la biblioteca *"Road Traffic Library"* la cual nos proporciona facilidades a la hora de crear una simulación sobre carretera.

La finalidad de este proyecto es, por tanto, la creación de un sistema que nos ofrezca una plataforma con la que podamos ver datos sobre los vehículos en tiempo real y que el usuario pueda interactuar con la simulación para explorar las variaciones que esto produce en dichos datos.

Palabras clave: Vehículos de movilidad ligera, redes de vehículos, simulación de vehículos, aplicación web, AnyLogic, Biblioteca de Tráfico de carretera.

SUMMARY

The aim of this work has been to create a platform for simulating the demand for light mobility vehicles in which the user could test case studies, deploying vehicles in each geographical area.

Nowadays, it is increasingly common to observe in cities networks of vehicles such as bicycles or electric scooters in which users, with a simple action on their devices, can locate vehicles near their position to move around the city in a low-polluting way.

To test the case studies, it was decided to use as a reference map the city of Utrecht in the Netherlands, which has a large network of bicycle lanes, and which allowed us to find a suitable area to test our simulation.

The development of the platform has been focused on a web application in which any user from a compatible device could access to perform simulations without having to install anything. The web application will provide access to the simulation with a user account in which we can manage our profile and perform the simulations we want. The simulation allows us to deploy vehicles at various points in the city of Utrecht, as well as to observe the movement trend of our vehicle to a destination of our choice. In addition, the simulation allows us to trigger a series of events associated with the points on the map, allowing us to see in real time the repercussions on our vehicle.

The main tool for the simulation component is the AnyLogic framework, which is implemented in the Java language. This development environment is ideal for creating a simulation as it has a wide range of functions and, above all, the "Road Traffic Library", which makes it easy to create a simulation on the road.

The aim of this project is, therefore, to create a system that offers us a platform with which we can see data on vehicles in real time and that the user can interact with the simulation to explore the variations that this produces in this data.

Keywords: Light mobility vehicles, networks of vehicles, simulation of vehicles, web application, AnyLogic, Road Traffic Library.

CERTIFICADO DE LOS TUTORES

D. Hernández González, Guillermo, D. ª Rodríguez González, Sara, D. González Briones, Alfonso profesores/as del Departamento de Informática y Automática de la Universidad de Salamanca

CERTIFICAN:

Que el trabajo titulado “Plataforma de simulación de demanda de vehículos de movilidad ligera” ha sido realizado por D. Andrei Brinza, con NIE X9842586N y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado de la Titulación Grado en Ingeniería Informática de esta Universidad.

Y para que así conste a todos los efectos oportunos.

En Salamanca, a 11 de Enero de 2023

Firmado por HERNANDEZ GONZALEZ GUILLERMO - ***1124** el día 11/01/2023 con un certificado emitido por AC FNMT Usuarios

RODRIGUEZ GONZALEZ SARA - 70864126E

Firmado digitalmente por RODRIGUEZ GONZALEZ SARA - 70864126E
Fecha: 2023.01.11 10:23:42 +01'00'

GONZALEZ BRIONES ALFONSO - 52415611Z

Firmado digitalmente por GONZALEZ BRIONES ALFONSO - 52415611Z
Fecha: 2023.01.11 10:18:56 +01'00'

TABLA DE CONTENIDO

TABLA DE FIGURAS

1.INTRODUCCIÓN

Hoy en día cada vez es más común observar vehículos de movilidad ligera en el ámbito de la circulación urbana. Esto incluye bicicletas, patinetes y otros tipos de vehículos de la misma índole.

Debido al incremento de uso de estos transportes, los propios gobiernos de países y ciudades han puesto en marcha planes para poder ofrecer a la población plataformas de alquiler de bicicletas o patinetes por toda la ciudad. [Pérez, A. (2020, 13 enero). Patinetes eléctricos Madrid: alquilar, normativas, precios]

Países como Alemania, España, Francia o Holanda han apostado cada vez más por reducir la contaminación en las ciudades y optar un transporte libre de emisiones de $C0_2$ y lo más limpio posible. [Estudios e Información Capital Madrid S.L, 2022]

Por tanto, surge una gran demanda de plataformas para poder solicitar este tipo de transporte de forma eficiente en una ciudad llena de vehículos y habitantes.

El desarrollo de este tipo de plataformas conlleva crear una red de usuarios para una aplicación en la que puedan solicitar vehículos en puntos concretos de una ciudad y tener conocimiento del estado de la ciudad en un entorno urbano como puede ser el tráfico o las aglomeraciones de personas en centros.

Esto nos va a permitir como usuarios de este tipo de plataformas tener una opción de transporte muy sencilla y rápida que nos permita controlar los trayectos que hagamos y monitorizar los tiempos de nuestros trayectos sobre la carretera. La aplicación desarrollada nos va a proporcionar los servicios antes mencionados ya que cuenta con un sistema de simulación que recrea los escenarios. Además de mostrarnos en tiempo real lo que está pasando.

Para poder explicar la creación y el desarrollo de una plataforma de simulación de vehículos ligeros se ha seguido la siguiente estructura en la memoria:

Objetivos del proyecto: Objetivos necesarios para el desarrollo del sistema.

Conceptos teóricos: Conceptos usados durante el desarrollo y que son relevantes para la aplicación.

Técnicas y herramientas: descripción de las tecnologías y herramientas usadas durante el desarrollo.

Aspectos relevantes del desarrollo del proyecto: Otras nociones importantes para el proyecto.

Limitaciones encontradas: Limitaciones encontradas durante el transcurso del desarrollo de la plataforma.

Conclusiones y líneas de trabajo futuras: Se comentará brevemente las conclusiones obtenidas y los posibles desarrollos futuros.

Referencias: Se enumera las citas empleadas para el desarrollo del proyecto.

Además, se recogen en la documentación los siguientes anexos técnicos :

Anexo-1 Plan de Proyecto Software: Se detalla la información respecto a la planificación de las tareas de proyecto y la estimación de esfuerzo que conlleva.

Anexo-2 Especificación de Requisitos Software: Se exponen los requisitos de software que componen el proyecto.

Anexo-3 Análisis de Requisitos Software: Se recoge la información respecto al análisis de los requisitos definidos en el Anexo-2 para hacer un refinamiento del sistema.

Anexo-4 Diseño del Sistema Software: Se recoge la fase de diseño y se definen los elementos que estarán en la fase final del proyecto.

Anexo-5 Documentación Técnica: Se explica con más detalle el código desarrollado durante el proyecto.

Anexo-6 Manual de Usuario: Guía para que un usuario pueda hacer uso del sistema y sus funcionalidades

2.OBJETIVOS DEL PROYECTO

El proyecto va a constar de una serie de objetivos en los cuales vamos a encontrar el objetivo principal del sistema, objetivos funcionales, objetivos no funcionales y finalmente los objetivos personales del propio desarrollador.

2.1OBJETIVO PRINCIPAL DEL SISTEMA

El objetivo principal del sistema es desarrollar una plataforma de simulación capaz de simular casos de estudio en un entorno controlado y además proveer una interfaz de usuario posibilitando la creación perfiles y tener acceso a la simulación sin necesidad de instalaciones a mayores.

2.2 OBJETIVOS FUNCIONALES

GESTIÓN DE USUARIOS

El sistema deberá proporcionar funciones necesarias para crear altas, bajas y modificación de usuarios. Además, para poder controlar el acceso al sistema de forma adecuada se dispondrá de varios roles de usuarios.

GESTIÓN DE SIMULACIÓN

El sistema debe ser capaz de proporcionar a los usuarios un entorno de simulación controlado en el que puedan realizar casos de estudio en tiempo real sobre el mapa. Todo ello con una interfaz amigable en la que puedan fácilmente crear vehículos en distintos puntos del mapa y observar el comportamiento hacia su destino.

GESTIÓN DE EVENTOS

El sistema debe ser capaz de permitir al usuario la activación de eventos en tiempo real dentro de la simulación para observar las consecuencias que estos tienen sobre el entorno de la simulación y los tiempos de viajes de los vehículos creados.

GESTIÓN DE ESTADÍSTICAS

El sistema debe ser capaz de recoger la información generada por los casos de estudio dentro de la simulación y mostrarla de forma ordenada en graficas que se actualizarán en tiempo real.

2.3 OBJETIVOS NO FUNCIONALES

USABILIDAD

El sistema deberá ser comprensible y fácil de usar por los usuarios de la plataforma por tanto debe tener una interfaz sencilla e intuitiva.

COMPATIBILIDAD

El sistema deberá tener una interfaz compatible con la mayoría de los navegadores actuales.

SEGURIDAD

El sistema deberá tener unos mínimos de seguridad ya que se guarda información sensible de los usuarios.

REUSABILIDAD

El sistema podrá contar con parte reutilizables que se podrán utilizar en la construcción de otros programas para economiza tiempo y recursos.

DISPONIBILIDAD

El sistema deberá estar disponible para los usuarios el mayor tiempo posible ya que se realizan tareas que necesitan continuidad.

RENDIMIENTO

El sistema debe tener un tiempo de respuesta razonable para que los usuarios puedan llevar a cabo casos de estudio dentro de la simulación de la forma más eficiente posible.

2.4 OBJETIVOS PERSONALES

El objetivo principal es conseguir desarrollar un proyecto desde cero, creando los aspectos más relevantes de una aplicación full stack. Además, es una oportunidad para poder aprender nuevos lenguajes de programación, nuevos entornos de desarrollo y otras formas de ver la informática a través de un proyecto con características muy interesantes.

Otro objetivo personal es ver la capacidad de adaptarme y gestionar un proyecto de software desde el comienzo.

Finalmente, otro objetivo era la realización de un proyecto en el que estuviera involucrado con herramienta de simulación.

3.CONCEPTOS TEÓRICOS

En este apartado de explicaran algunos de los conceptos que se tratan en el TFG y que son importantes comprender.

3.1 API REST

Una API REST o RESTfull, es una interfaz de programación de aplicaciones que se enmarca en los límites de la arquitectura REST y permite la interacción con los servicios web. Cuando el cliente envía una solicitud a través de una API RESTfull, esta hace de intermediario y hace llegar esta petición al *backend*.

La información se entrega mediante el protocolo HTTP (GET, POST, PUT, UPDATE, …) con un formato concreto que puede ser JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato. JSON es uno de los formatos populares debido a que es fácil de comprender y no depende directamente de ningún lenguaje, pero también hay otros como YAML.

Además de esto, la información está sujeta a una cabecera o *header,* que es de vital importancia, ya que especifica aspectos importantes de la información. Contienen información de identificación, metadatos, la autorización, el identificador uniforme de recursos (URI), el almacenamiento en caché, las cookies y otros elementos de la solicitud.

En la aplicación que ha sido desarrollada se decidió usar esta arquitectura ya que es una de las extendidas para la creación de APIs y nos permite cumplir con los objetivos de creación.

3.2 APLICACIÓN WEB

Las aplicaciones web es un recurso utilizado frecuentemente ya que permite la conexión de multitud de clientes a los que se les ofrece un servicio desde un servidor sin necesidad de instalación de ningún software especifico, simplemente contando con una conexión a internet y un navegador web compatible.

3.3 BACKEND

El *backend* es una parte del desarrollo web que se encarga de toda la lógica de una página web funcione correctamente como por ejemplo la comunicación con el servidor o funciones concretas que no se serán visibles para el usuario. El *backend* va a ser el que trate la información recogida desde el cliente mediante el *frontend* y la va a procesar.

3.4 FRONTEND

El *frontend* es la parte de la aplicación que interactúa directamente con el usuario y la parte del cliente del desarrollo web. Esto implica que conlleva todos los elementos visuales como pantalla, efectos, textos, formularios, etc. Las interfaces creadas para el usuario serán creadas a base de lenguajes como HTML, CSS o JavaScript y se buscara que sean lo más fáciles e intuitivas posibles.

3.5 FRAMEWORK

Un *framework* es un esquema o marco de trabajo que nos va a ofrecer una estructura previa para elaborar un proyecto con unas funcionalidades específicas. Los *frameworks* son muy utilizados en desarrollo web porque agilizan procesos de elaboración y comienzo de un proyecto y además permiten una fácil depuración de errores.

3.7 MÓDULOS

Los módulos van a ser porciones de código instalables que cambiaran en función de las tecnologías o lenguajes que estemos utilizando y que nos aportaran una colección de funciones que podremos usar mediante llamadas concretas descritas previamente en la documentación del módulo instalado.

En el trabajo realizado se han usado varios módulos en el *framework* de Django por ejemplo django.contrib,django.utils,django.core,django.urls,django.forms,…

Hay que diferenciar módulos y paquetes ya que por ejemplo en el lenguaje Python son conceptos distintos.

3.8 BASE DE DATOS

Una base de datos es una forma de recopilar datos a la vez que se podrán organizar o relacionar de formas concretas. Las bases de datos nos van a permitir guardar información de usuarios en la aplicación desarrollada y que podremos acceder de distintas formas en función del tipo de base de datos que estemos usando.

En el proyecto se va ha usado bases de datos relacionales de SQL (Structured Query Language) que recopilan información y la organiza en función de un ID y que podremos acceder a través de a través de consultas.

A diferencia de las bases de datos no relacionales, la base de datos relacional que se usa en el trabajo nos da la ventaja de que la información guardada está estructurada y es menos propensa a fallos. Por eso se escogió este tipo de bases de datos para una guardado persistente de información.

3.9 PROCESO UNIFICADO

El proceso unificado es un marco de trabajo genérico que es capaz de especializarse para el desarrollo de diversos sistemas software formando por cuatro principales fases que estarán regidas por una serie de disciplinas necesarias para definir el proceso de desarrollo. Además, será iterativo e incremental y basado en hitos.

En el trabajo se ha escogido este marco de desarrollo de software debido a que nos proporciona una buena guía. Gracias a esto y que está estructurado en fases incrementales flexibles son idóneas para el proyecto.

También se barajó alternativas como la utilización de metodologías Scrum ya que tienen muchos puntos en común siendo incremental y basado en hitos, pero al estar enfocado para pequeños proyectos, mucho más intensivos y sin fecha límite.

3.10 UML

UML (Lenguaje Unificado de Modelado) es un lenguaje de modelo visual creado para la implementación de sistemas de software ayudando a definir su estructura y su comportamiento. Está basado principalmente en diagramas que describirán los objetos que componen un sistema de software y como se relacionan de forma visual mediante una simbología característica del lenguaje de modelado.

En el proyecto realizado se utilizó para modelar gran parte de la base de la aplicación creando distintos diagramas que ayudaron a la especificación.

3.11 HERRAMIENTAS CASE

Las herramientas CASE (Computer Aided Software Engineering) son diversas aplicaciones informáticas que están destinadas a aumentar la productividad del desarrollo de software intentando reducir los costes de tiempo y de recursos.

Estas aplicaciones la definiremos con más detalle en el apartado de técnicas y herramientas donde trataremos con Visual Paradigm, Microsoft Project, EZEstimate, REM.

3.12 MOVILIDAD URABANA

La movilidad de una ciudad es un conjunto de todo tipo de desplazamientos, desde peatones pasando por bicicletas hasta coches o camiones. En las ciudades grandes se ha extendido mucho el concepto de ciudad inteligente, que lo que busca es una organización eficiente de los desplazamientos con el menor impacto medioambiental.

En el proyecto realizado se trata específicamente los vehículos de movilidad ligera, es decir, los no motorizados. Esto comprende desde las bicicletas hasta los patinetes. En el desarrollo del trabajo se lleva a cabo una simulación dentro de un sector de la ciudad de Utrecht, donde se simula la movilidad de vehículos entre cinco puntos principales y recogiendo datos significativos del tiempo que pueden tardar los vehículos en desplazarse de un destino a otro. Esto nos permite observar y elegir la mejor ruta a la hora de movernos de forma eficiente.

3.13 SIMULACIÓN

Una simulación se caracteriza por ser un entorno controlado donde se llevan a cabo una serie de experimentos o casos de estudio para sacar información valiosa de cara a tener unas conclusiones o resultados.

Nuestro trabajo integra la simulación de movilidad y mediante el software proporcionado por la herramienta AnyLogic podemos crear un entorno de simulación controlado por nosotros y desplegar distintos elementos. En nuestro proyecto usamos esto para poder representar el mapa a escala, carreteras, destinos, edificios, vehículos, peatones, etc. Además, contamos con distintas opciones para definir el comportamiento y la lógica detrás del movimiento de los vehículos.

Todo esto es importante para el proyecto ya que es donde podemos probar los casos de estudio para predecir de forma aproximada los tiempos de desplazamiento entre dos puntos concurridos de nuestro mapa y sacar conclusiones en función de los resultados mostrados.

4.TÉCNICAS Y HERRAMIENTAS

En el apartado siguiente se van a exponer las diferentes técnicas y herramientas utilizadas en el proceso de desarrollo del sistema.

4.1 HERRAMIENTAS CASE

4.1.1 VISUAL PARADIGM

Visual Paradigm es una herramienta CASE que proporciona un conjunto de ayudas para el desarrollo de programas informáticos permitiendo la planificación, pasando por el análisis y el diseño. Creando así en el proceso casos de uso, diagramas de secuencia, diagramas de paquetes, diagrama de despliegue, … Todos ellos usados en el desarrollo de la aplicación. Esta herramienta ha sido básica para poder llevar a cabo el proyecto y se ha escogido debido a que se tiene una licencia por parte de la universidad que nos da acceso académico y también a que se usó previamente en la carrera en asignaturas enfocadas a la ingeniería de software por tanto se conocía el entorno y se hacía mucho más ágil su uso.

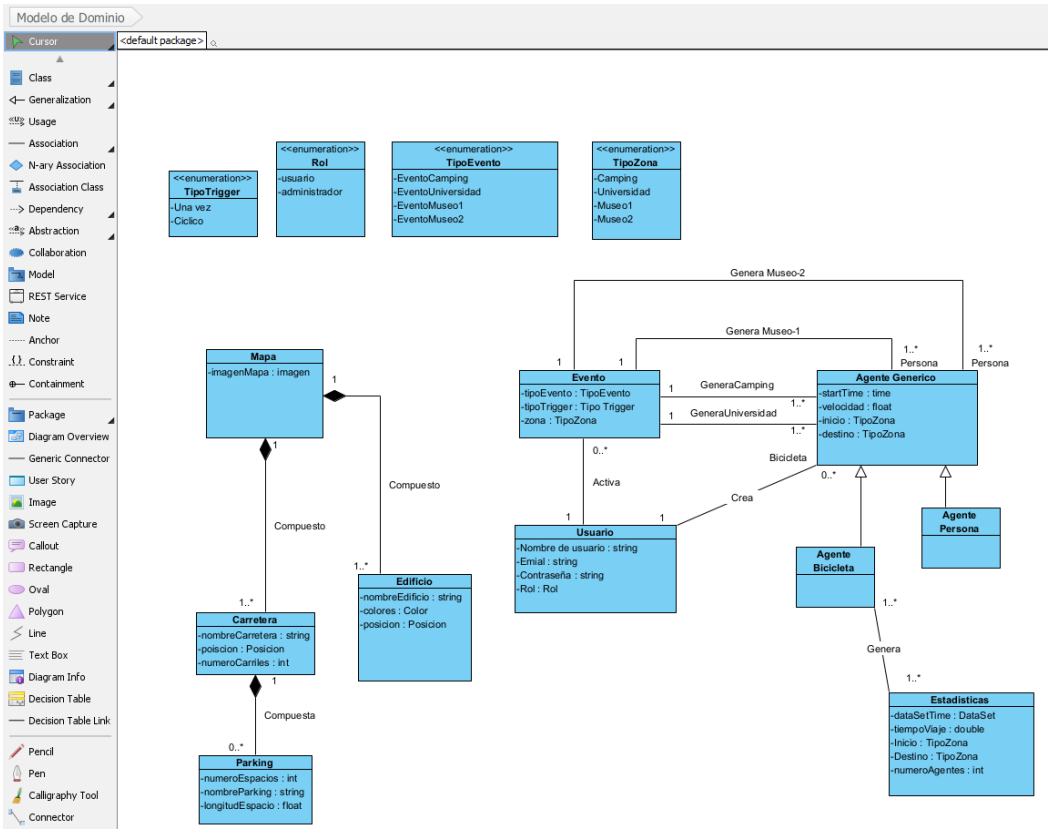


Figura 1: Visual Paradigm (Modelo de Dominio)

4.1.2 REM

REM (Requirements Management) es una herramienta experimental gratuita que nos a permitir la gestión de requisitos del sistema. Está enfocada principalmente para dar soporte a la fase de ingeniería de requisitos del desarrollo software y nos ha permitido crear y ordenar de forma sencilla los requisitos, además de proporcionar una interfaz bastante sencilla para especificar detalles sobre los actores, objetivos, requisitos funcionales, no funcionales, de información, etc. Se ha utilizado esta herramienta principalmente porque agilizó el proceso de creación de requisitos y su organización (sobre todo la matriz de rastreabilidad) y al ser gratuita no ha supuesto ninguna limitación.

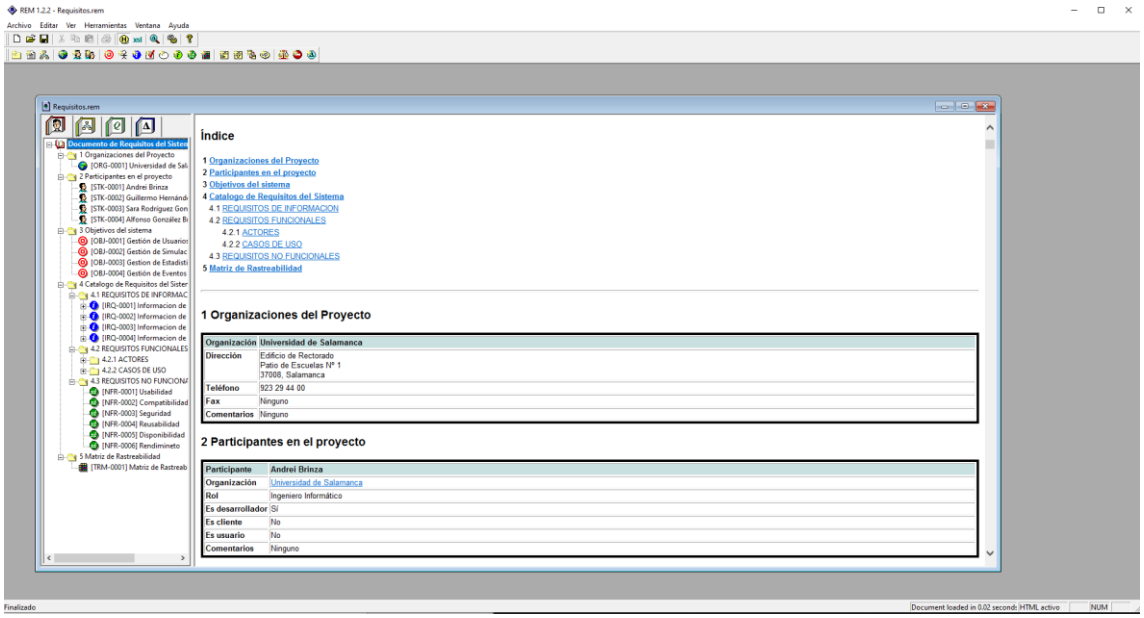


Figura 2: Captura de requisitos del proyecto en la aplicación REM

4.1.3 MICROSOFT PROJECT

Microsoft Project es un conjunto de herramientas que ofrece soporte a la gestión de proyectos mediante el uso de diagramas de Gantt que nos permitieron definir una serie de tareas o actividades que se pueden organizar y visualizar temporalmente desde la propia aplicación. Además, nos permitieron definir otros atributos como horarios laborales, calendarios y otras especificaciones relacionadas con el mundo laboral. Se ha utilizado esta herramienta debido a que nos proporcionó una base para el proyecto de software y la organización. Además, fue una herramienta utilizada previamente en la carrera.

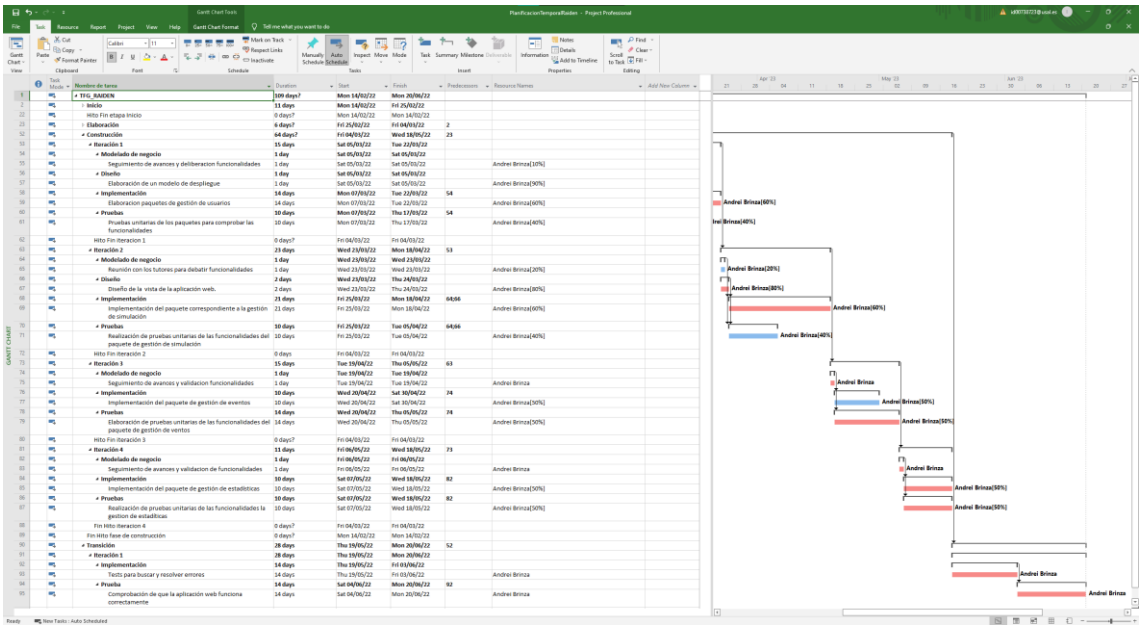


Figura 3: Microsoft Project

4.1.4 EZESTIMATE

Es una herramienta de apoyo a la planificación del proyecto y que nos va a permitir estimar de forma razonable el tiempo que se debe invertir en el sistema ajustando una serie de parámetros específicos de gestión de software.

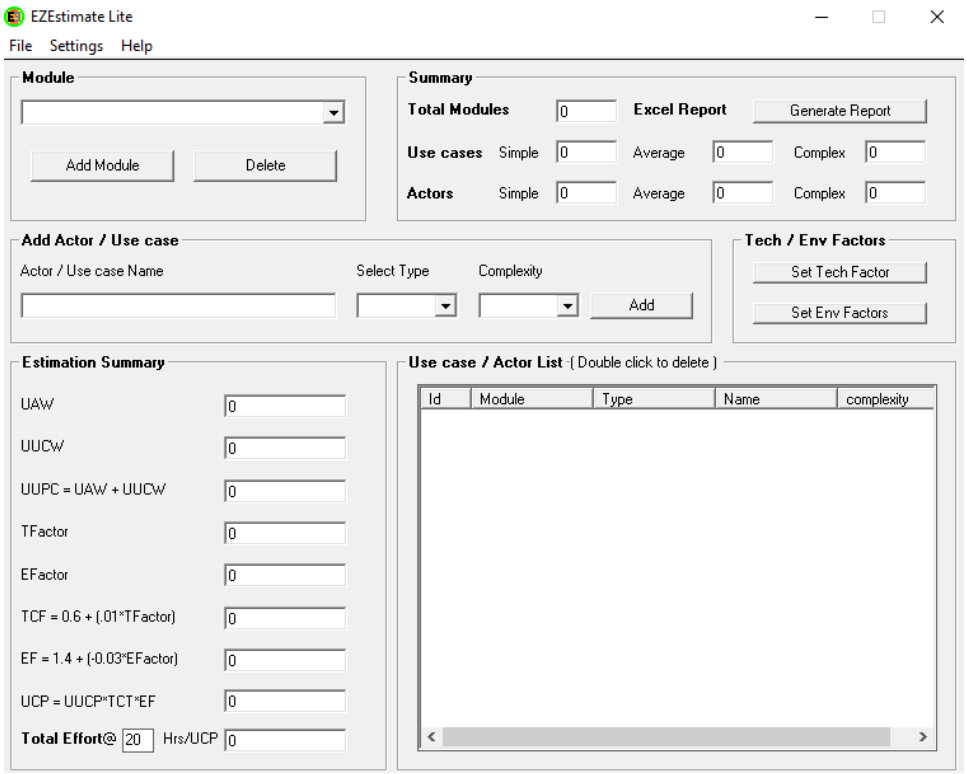


Figura 4: EZEstimate Main

4.2 FRAMEWORKS

4.2.1 DJANGO

Django es un *framework* open source de alto nivel basado en Python que nos permitió un desarrollo rápido y limpio. Está enfocado para desarrollo el web y cuenta con numerosas librerías que facilitan el desarrollo web. Se ha eligió este *framework* para el desarrollo del *backend* debido a que está basado en Python y que tiene distintos métodos para tratar datos, también con un sistema de autentificación de usuarios. Además de lo anterior Django cuenta con una buena documentación y una comunidad con constantes actualizaciones.

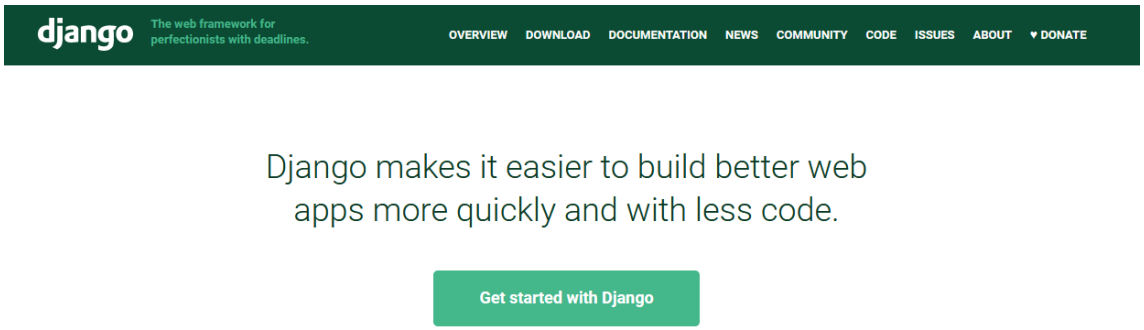


Figura 5: Django

4.2.2 VUE

Vue es un marco de trabajo de JavaScript para la construcción de interfaces de usuario. Se construye sobre HTML, CSS y JavaScript estándar, y proporciona un modelo de programación declarativo y basado en componentes que ayuda a desarrollar eficientemente las interfaces de usuario. Se ha escogido este *framework* para el desarrollo del *frontend* debido a que proporciona una gran variedad de módulos que podemos utilizar para crear nuestro proyecto de la forma más eficiente posible.

Vue es un soporte idóneo para el *backend* desarrollado con Django, pero también se barajó la posibilidad de utilizar React o Angular para desarrollar el *frontend* aunque fueron descartados ya que tenían una complejidad algo mayor.

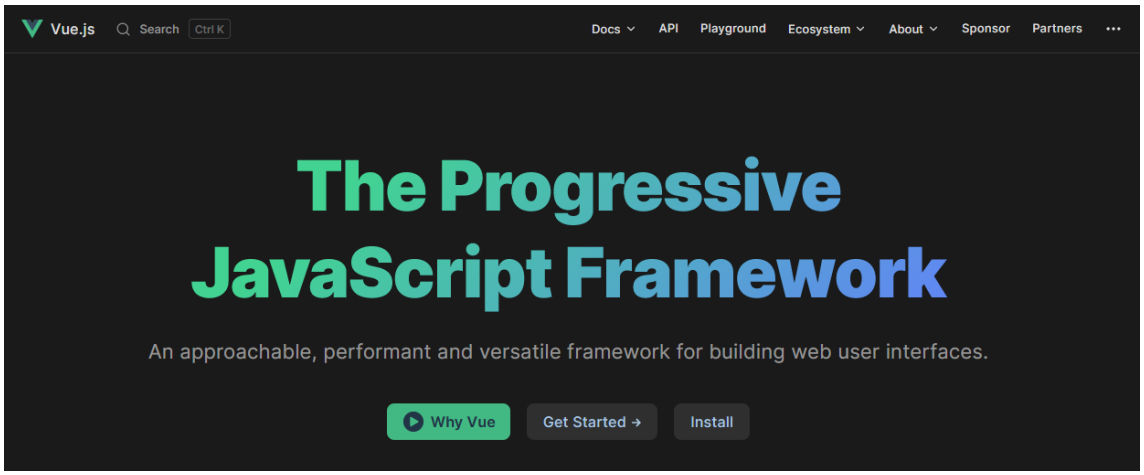


Figura 6: Vue

4.2.3 VUETIFY

Vuetify es un *framework* progresivo que se puede integrar con vue para dar soporte a la creación de interfaces de usuario pudiendo así desarrollar de forma rápida y sencilla formularios, menús o cualquier tipo de elemento comúnmente usado en una página web como botones, listas, barras de navegación, etc.

Se ha escogido este *framework* debido a que es una buena extensión de vue y fácilmente integrable que nos facilitó mucho el desarrollo de la interfaz gracias a la buena documentación que tiene.

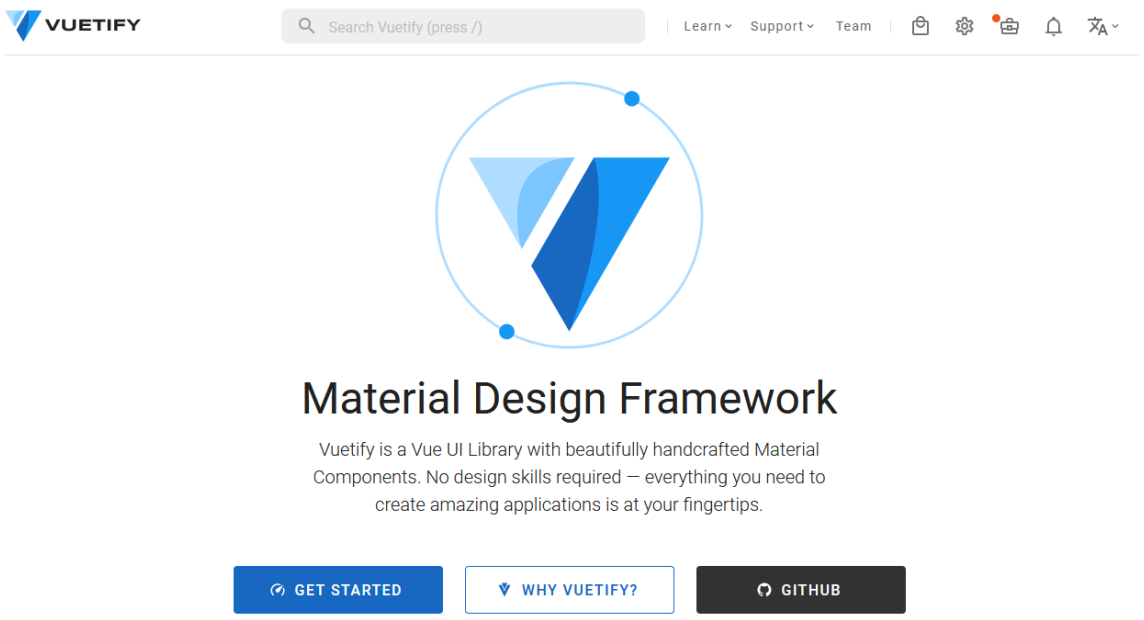


Figura 7: Vuetify

4.3 SOFTWARE DE SIMULACIÓN

4.3.1 ANYLOGIC

AnyLogic Simulation Software es el software de simulación basado en Java más utilizado en aplicaciones industriales y de negocio. AnyLogic incluye un lenguaje de modelado gráfico y también permite que los usuarios puedan ampliar los modelos de simulación con código de Java. *[Engineering. (s. f.). Software de modelado de simulación multimétodo AnyLogic. Engineering USA]*

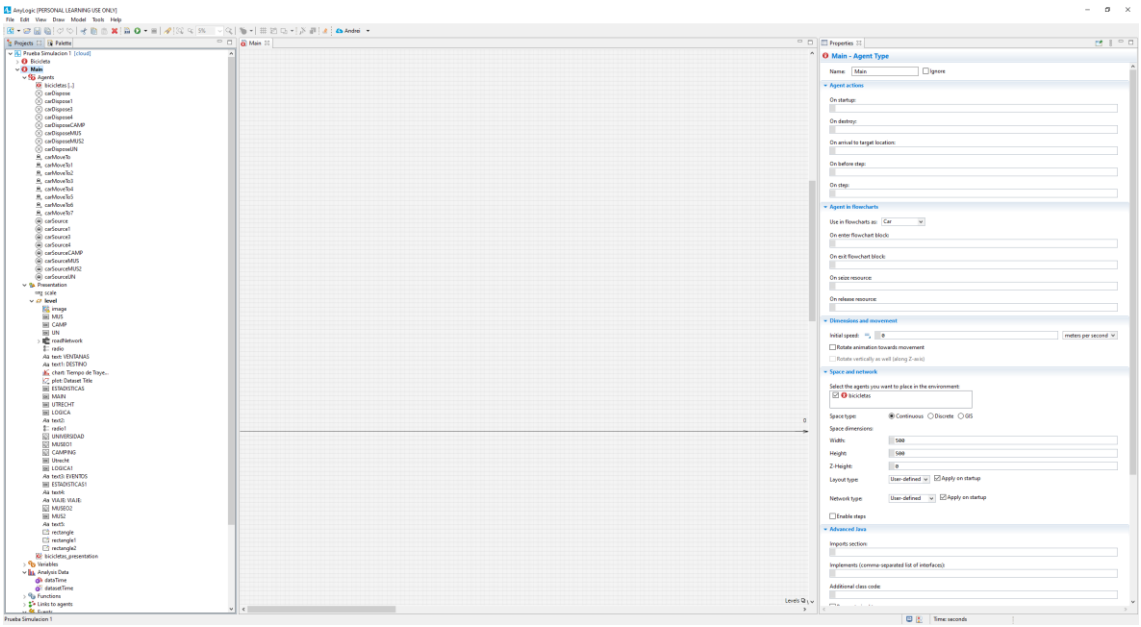


Figura 8: AnyLogic

AnyLogic soporta y permite desarrollar modelos usando cualquiera de las tres metodologías de simulación modernas que son modelado de simulación basado en agentes, modelado de simulación de eventos discretos y modelado de simulación por dinámicas de sistema.

Además, Anylogic nos proporciona una gran cantidad de librerías compuestas por numerosos elementos gráficos y lógicos para poder desarrollar todo tipo de simulaciones. El proyecto se va a centrar principalmente en el uso de la librería *Road Traffic Library* la cual nos permitió crear una red de carreteras, desplegar agentes y definir la lógica de funcionamiento.

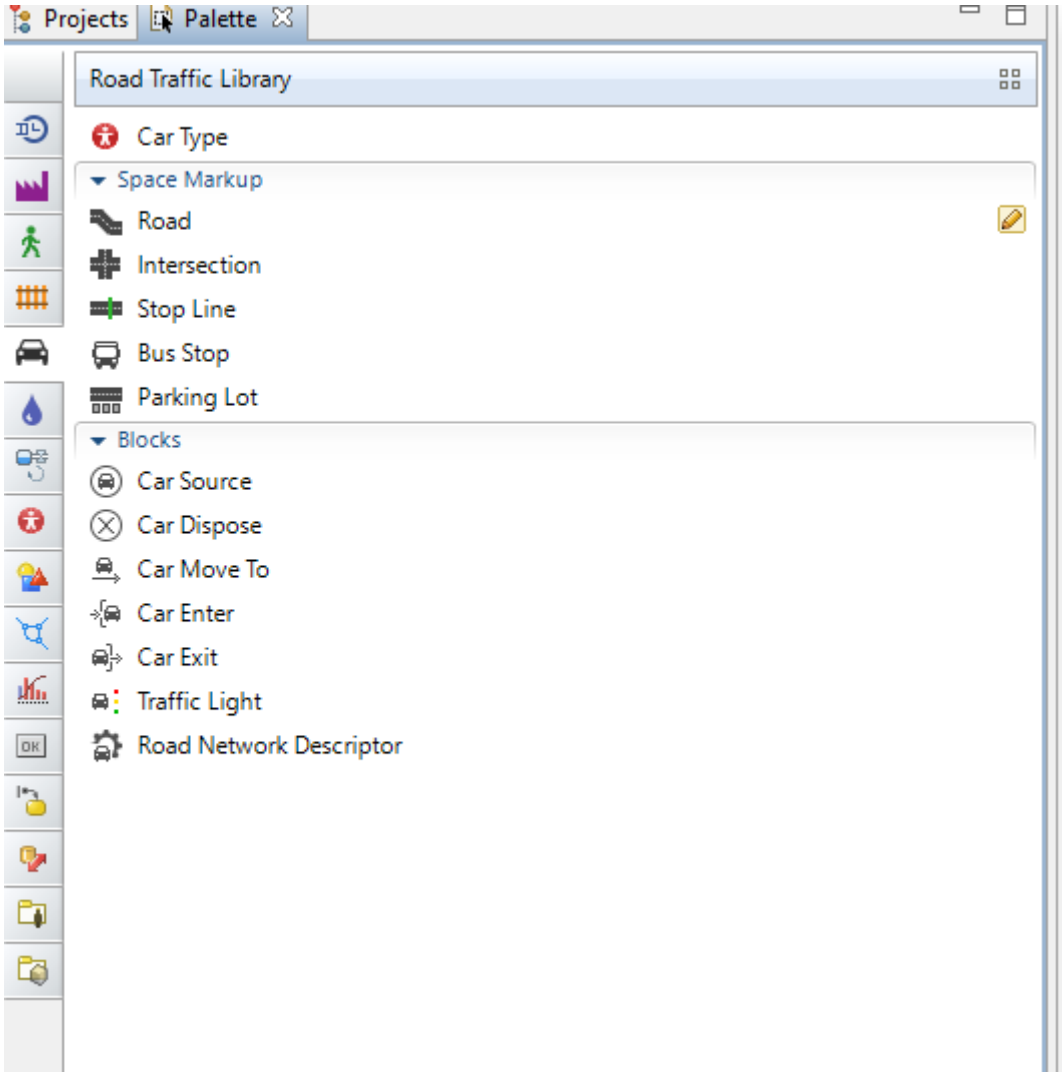


Figura 9: Road Traffic Library

Este software también nos ha permitido la inclusión de eventos en los cuales podremos definir su comportamiento y su lógica mediante el lenguaje Java. Y por último nos permitirá la creación de estadísticas y graficas a partir de los datos manejados dentro de la simulación.

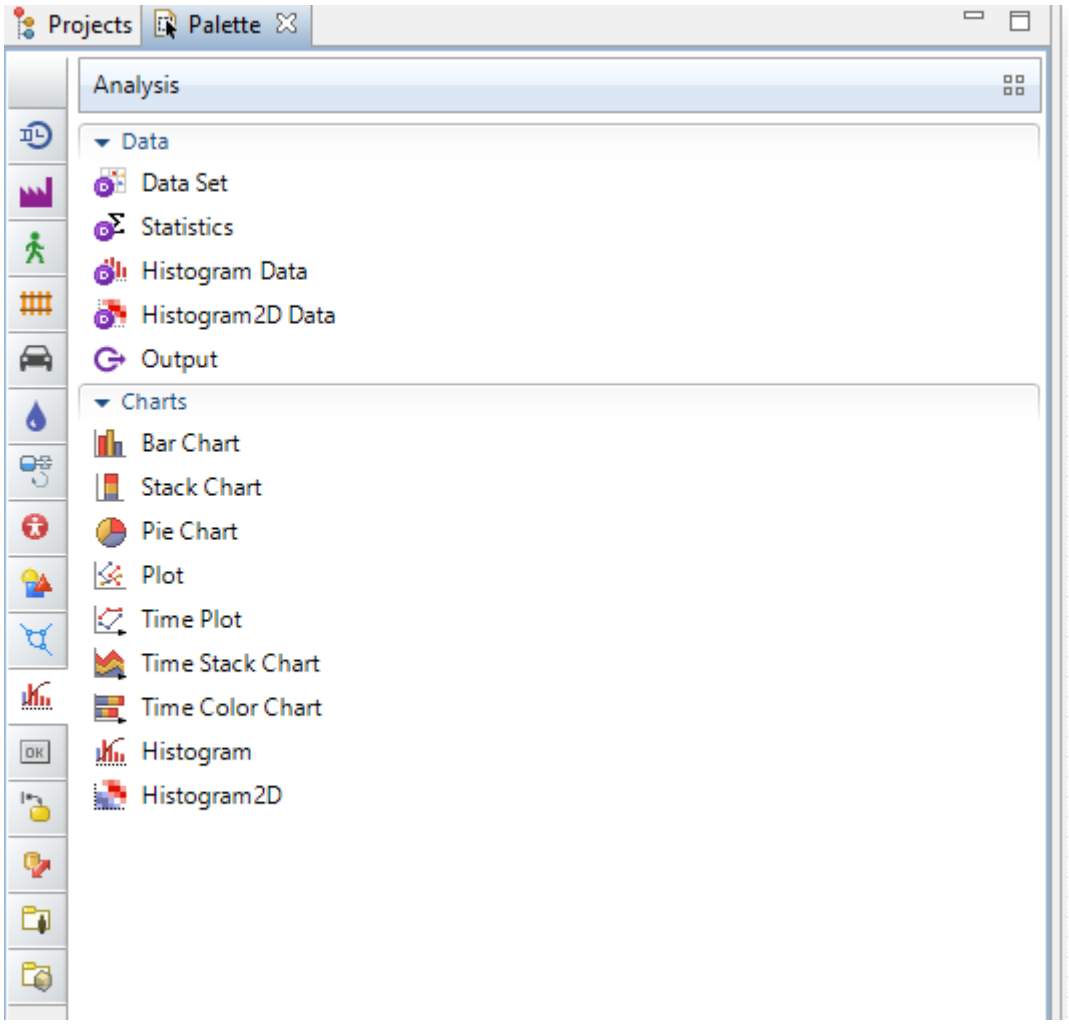


Figura 10: Analysis Library

Se disponía de varias herramientas de simulación principales a nuestra disposición como SUMO, MATSim, PTV VISSIM y AnyLogic.

SUMO fue una herramienta descartada debido a que era simple y no tenía a disposición muchas posibilidades de crear una simulación diversa y con eventos.

MATSim y PTV VISSIM también acabaron siendo descartadas ya que estaban más enfocadas a análisis de datos y tampoco se podían desplegar eventos de forma sencilla.

Finalmente se escogió AnyLogic, debido a que es la más completa del mercado y permite manejar eventos de forma sencilla y con una interfaz comprensible.

A pesar de no ser gratuita completamente nos va a permitir con una versión académica realizar simulaciones con una duración máxima de una hora. Con la adquisición de la licencia completa el software de AnyLogic tendrá ningún tipo de limitación en la simulación.

4.3.2 ANYLOGIC CLOUD

AnyLogic Cloud es uno servicio web para análisis de simulación. Permite a los usuarios almacenar, acceder, ejecutar y compartir modelos de simulación online, así como analizar los resultados de los experimentos. Además, se pueden cargar los modelos desarrollados en la nube de AnyLogic y configurar paneles web que se pueden compartir para trabajar con modelos en línea y nos facilita la ejecución de modelos usando navegadores web o dispositivos móviles.

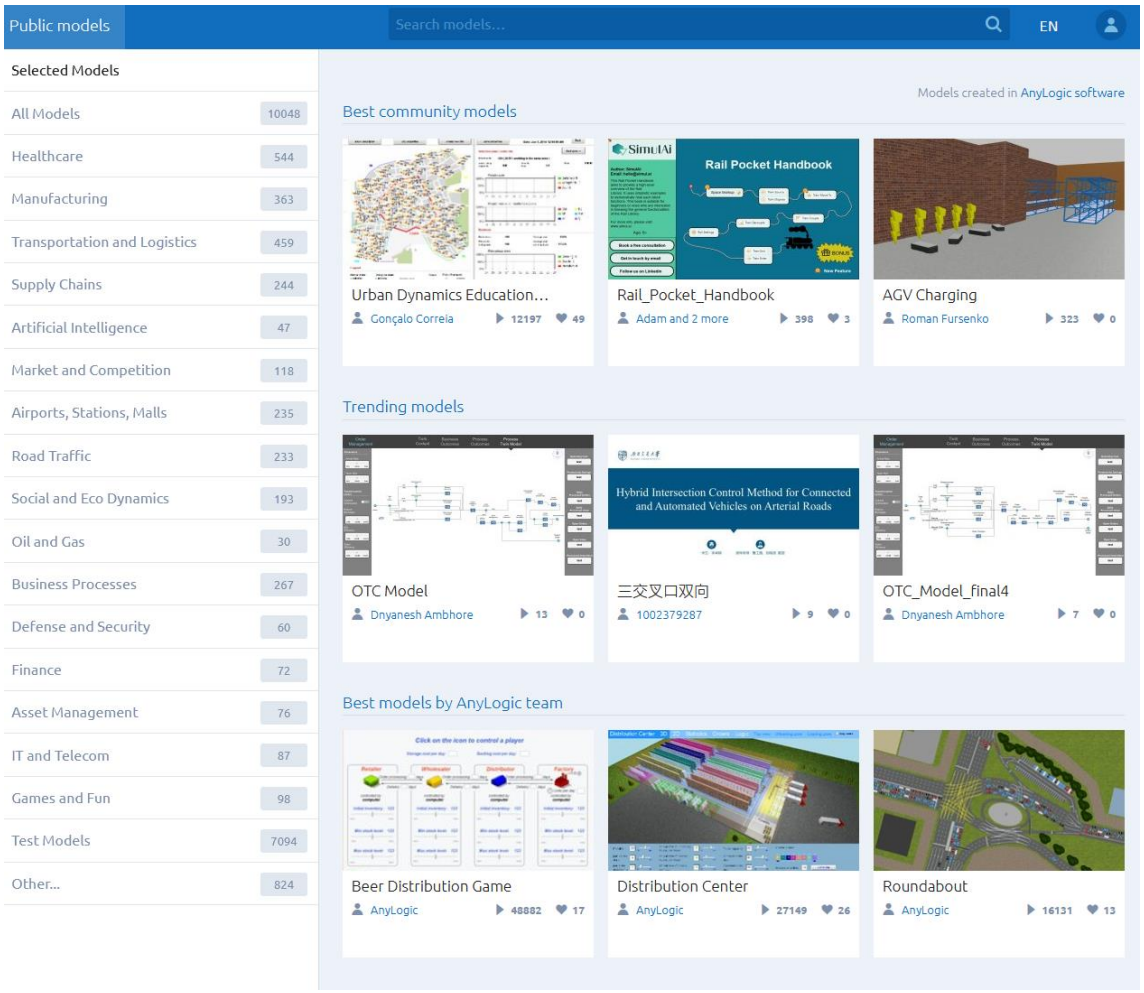


Figura 11: AnyLogic Cloud

4.4 ENTORNOS DE DESARROLLO

4.4.1 VISUAL CODE

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft. Es software libre y multiplataforma. VS Code tiene una buena integración con Git, cuenta con soporte para depuración de código, y dispone de una gran variedad de extensiones que pueden per especialmente útiles. Se ha escogido este entorno principalmente porque da soporte para la creación de código facilitando mucho la tarea de programación y también te da la posibilidad de escribir y ejecutar código en cualquier lenguaje de programación.

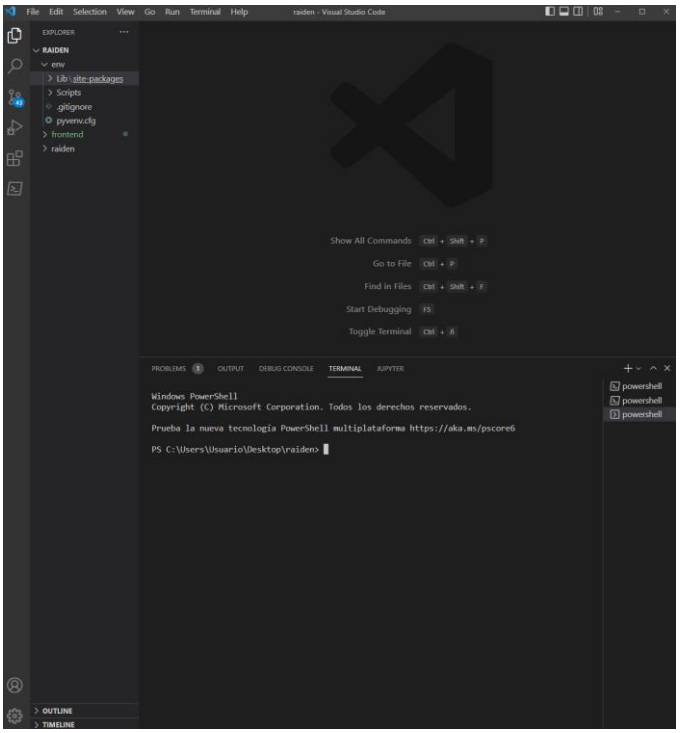


Figura 12: Visual Studio Code

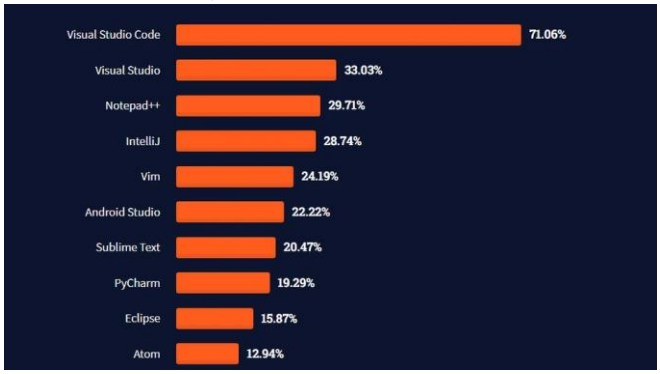


Figura 13: Visual Studio Code Grafica

4.5 BACKUPS Y CONTROL DE VERSIONES

4.5.1 GIT

Git es un software de control de versiones diseñado por el creador de Linux. Se centra en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Se ha utilizado durante el proyecto para tener un control de versiones y poder retroceder de forma sencilla en la línea temporal del código desarrollado.

Figura 14: Git

4.5.2 GITHUB

GitHub es una plataforma mundialmente conocida en la se pude alojar proyectos de software y junto a la herramienta Git nos proporciona una manera eficaz de guardar los avances que se hagan en el proyecto y llevar un control de las versiones de nuestro proyecto.

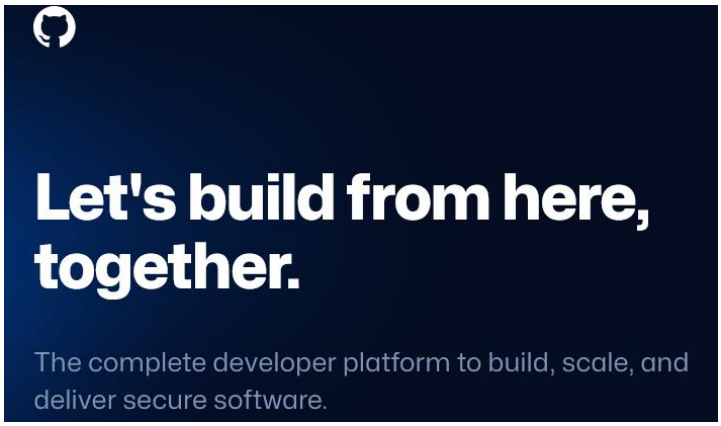


Figura 15: Github

`

4.6 COMPILACIÓN, EJECUCIÓN Y TEST

4.6.1 NODEJS

Este entorno de ejecución en tiempo real para la capa del servidor que incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript y siendo muy usado para el desarrollo de aplicaciones.

Figura 16: Nodejs

4.6.2 NPM (NODE PACK MANAGER)

NPM es el Node Package Manager que viene incluido en Node y ayuda al desarrollo de aplicaciones permitiendo la instalación de distintos módulos de forma fácil y rápida. Además, nos permite mediante otros comandos el arranque de servidores locales, compilación del proyecto e instalar varios módulos y administrar sus dependencias.

4.6.3 WEBPACK

Webpack se define como un empaquetador de módulos o *bundler* que ha sido usado en el desarrollo de nuestro *frontend* junto a vue. Especializado en el procesamiento de unos archivos de entrada para convertirlos en otros archivos de salida, para lo cual utiliza unos componentes que se denominan *loaders*.

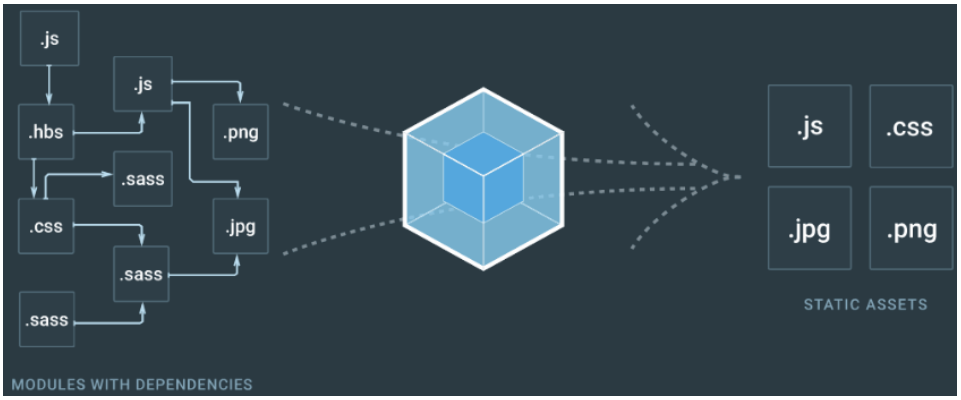


Figura 17: Webpack

4.6.4 POSTMAN

Postman nace como una extensión que podía ser utilizada en el navegador Chrome y básicamente nos permite realizar peticiones de una manera simple para testear APIs de tipo REST. Esta herramienta nos ha sido útil para testear nuestra API y comprobar su correcto funcionamiento durante del desarrollo del *backend*. Postman nos permite introducir distintos parámetros como cabeceras, seguridad, atributos, … Con esto podremos ver las respuestas de nuestra API a nuestras pruebas y saber si funciona como deseamos.

Figura 18: Postman

4.7 LENGUAJES DE PROGRAMACIÓN

4.7.1 JAVASCRIPT

Es un lenguaje de secuencias de comandos que te permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes, etc. En nuestro proyecto va a ser usado principalmente para el desarrollo de los scripts asociados a vue con los cuales podremos definir la mayoría de la interfaz de la aplicación con el soporte de HTML y CSS.

En la imagen a continuación nos encontramos con la definición de Registro.vue donde se alberga el código escrito en Javascript para poder manejar la información introducida por el usuario a la hora de registrarse en nuestro sistema.

Figura 19: Registro.vue (Javascript)

4.7.2 JAVA

Java es un lenguaje de programación clásico orientado a objetos y basado en herencia que es usado en la herramienta AnyLogic para definir ciertas funcionalidades de la simulación y realizar la creación de ciertos objetos.

Figura 20: Java

4.7.3 PYTHON

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. Se trata de un lenguaje interpretado y no es necesario compilarlo para ejecutar las aplicaciones escritas en Python. Se ha utilizado este lenguaje debido su facilidad de uso y la forma de tratar los datos para nuestro *backend*.

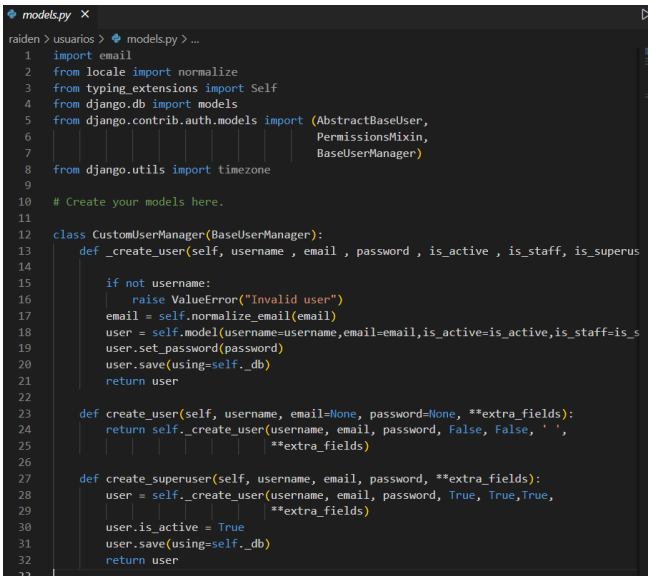


Figura 21: Python

4.7.4 HTML y CSS

HTML (HyperText Markup Language) es un lenguaje de marcas para etiquetar texto, imágenes y otro contenido para mostrarlo en un navegador Web. Define el significado y la estructura del contenido.

```
<template>
    <v-container fill-height align-center justify-center>

        <img src="@/assets/logoRaiden.png" height="500" width="600" alt="">

    </v-container>

</template>
```

Figura 22: HTML

CSS (Cascading Style Sheets) es un lenguaje que determina el estilo de los documentos HTML. Abarca opciones relativas a fuentes, colores, márgenes, altura, anchura, etc.

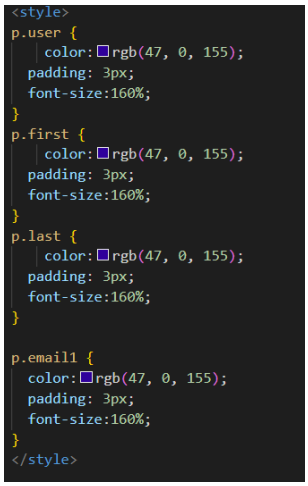


Figura 23: CSS

4.7.5 JSON

JavaScript Object Notation se trata de un formato para guardar e intercambiar información que cualquier persona pueda leer. Los archivos json contienen solo texto y usan la extensión. json. Es usado para la comunicación entre el *backend* y el *frontend* con un formato determinado.

5.ASPECTOS RELEVANTES DEL DESARROLLO DEL PROYECTO

En el apartado expuesto a continuación se va a tratar los aspectos más importantes que han tenido lugar durante el desarrollo del proyecto.

5.1 METODOLOGÍA

La metodología seguida es el proceso unificado que es un marco de trabajo genérico capaz de especializarse para sistemas software formado por cuatro fases de desarrollo.

El proceso unificado ha sido aplicado durante todo el desarrollo del proyecto y como hemos mencionado anteriormente divido en cuatro fases:

Inicio: Se definió el alcance de nuestro proyecto y se realizaron tareas de investigación para poder elegir las herramientas adecuadas.

Elaboración: Se planifico el proyecto y se plantean las directrices para especificar en detalle la mayoría de los casos de uso y también se diseñó la arquitectura del sistema.

Construcción: En la etapa de construcción se pasó a implementar por partes nuestro trabajo siguiendo el análisis y las especificaciones anteriores y creando un producto.

Transición: En la fase de transición el código desarrollado en el trabajo se depura y se convierte en una versión funcional que podremos incorporar en el sistema final.

También se definen las siguientes características principales del proceso unificado. [Jacobson et al., 1999]

Casos de uso: Un caso de uso es una metodología utilizada en el análisis de sistemas para identificar, aclarar y organizar los requisitos del sistema. El caso de uso está formado por un conjunto de posibles secuencias de interacciones entre los sistemas y los usuarios en un entorno concreto y relacionado con un objetivo determinado.

Centrado en la arquitectura: La arquitectura se representará mediante las vistas del modelo. Cada vista será una parte del modelo y estos serán la forma de visualizar, especificar, construir y documentar la arquitectura.

Iterativo e incremental: El proceso unificado estará divido iteraciones, y en cada una de estas a su vez estará dividida en disciplinas que nos van a permitir sacar de forma iterativa los casos de uso de nuestro sistema y un diseño basado en la arquitectura que se ha definido. Además, será implementado mediante componentes que forman el proceso unificado y se verificara que satisfacen los casos de uso. Si una iteración llega su fin se pasará a la siguiente hasta que la fase termine y se complete un hito. Las iteraciones irán formando el sistema de forma incremental.

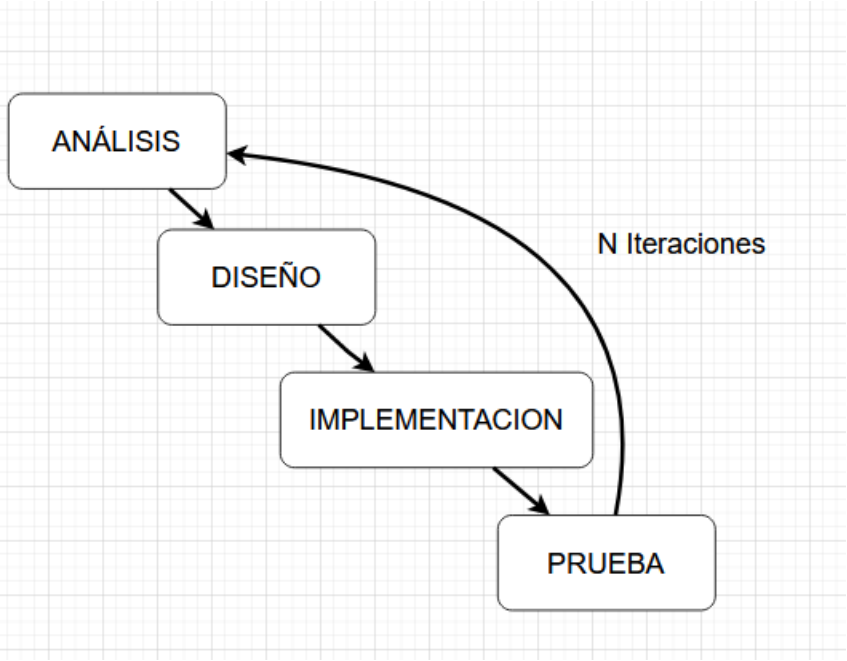


Figura 24: Proceso Unificado

5.2 ESTIMACIÓN DE ESFUERZO Y PLANIFIACIÓN TEMPORAL

5.2.1 ESTIMACIÓN DE ESFUERZO

La estimación de esfuerzo consiste en realizar predicciones de las cantidades más probables de esfuerzo que se requieren para la construcción del sistema software. Para ello se pueden llevar a cabo distintas metodologías con distintas métricas. Nuestro sistema utilizara la métrica de UCP (Use Case Points).

Para el cálculo de la estimación de esfuerzo vamos a usar la aplicación EZEstimate. Con esta aplicación podremos introducir nuestros casos de uso junto a un parámetro de complejidad. Este parámetro de complejidad lo podremos deducir a partir del número de transacciones que tenga nuestros casos de uso y que representan los pasos necesarios para que el caso de uso se ejecute. EZEstimate devolverá un valor en meses por persona que estará calculado a partir de una serie de parámetros establecidos para la estimación de esfuerzo.

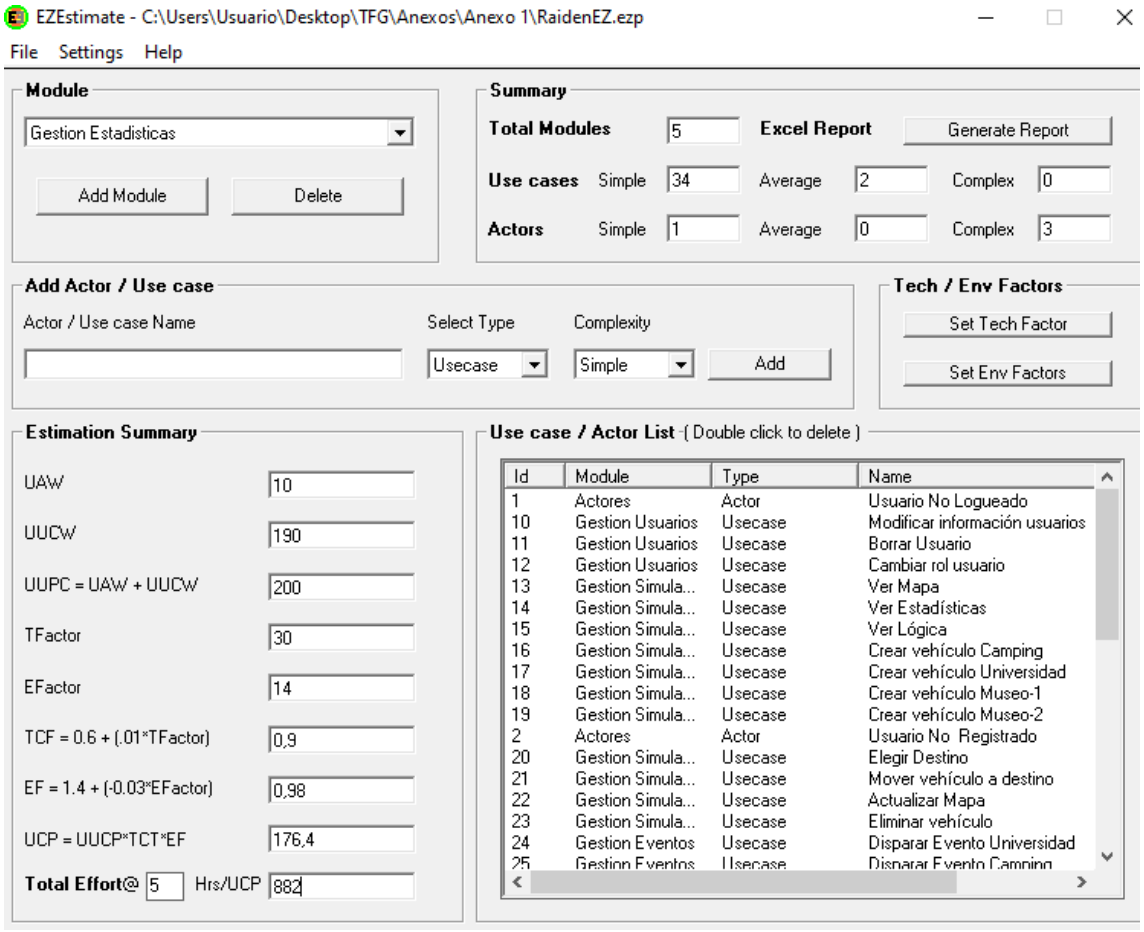


Figura 25: EZEStimate

La complejidad de los casos de uso se ha determinado mediante la cantidad de transacciones los componen y se usan para llevar a cabo su funcionalidad. Podremos definir tres tipos de complejidad en función de lo anteriormente mencionado:

Simple: Tres o menos transacciones.

Media: Entre cuatro y siete transacciones.

Complejo: Mas de siete transacciones.

Para determinar la complejidad de los actores se va a seguir las siguientes definiciones:

Simple: Si el actor es un sistema y la aplicación se comunica con él mediante una API.

Medio: Si el actor es un sistema y la aplicación se comunica con él mediante un protocolo web.

Complejo: Persona que interactúa con el sistema mediante una interfaz gráfica.

La estimación se va a realizar con la métrica UCP que va a ser usada para evaluar la funcionalidad. Para ello hace uso de trece factores de complejidad técnica (TCF) y ocho factores de complejidad del entorno (ECF).

A cada factor se le asigna un peso(W) acorde a su impacto y una complejidad percibida(F) correspondiente a la percepción de complejidad que tiene el equipo de desarrollo.

CALCULO TCF

$$TCF = C_1 + C_2 \sum_{i=1}^{13} W_i + F_i$$

CALCULO ECF

$$ECF = C_1 + C_2 \sum_{i=1}^{8} W_i + F_i$$

Figura 26: Ecuaciones factores

Factores de complejidad técnica: Va a representar el esfuerzo que supondrá para el desarrollador la implementación de dicho factor.

Factores de complejidad de entorno: Va a representar la experiencia que tiene el desarrollador en los diferentes factores.

Componentes del factor de complejidad técnica		
F_1 Copias de seguridad y recuperación fiables	F_6 Entrada interactiva de datos	F_{11} Reusabilidad
F_2 Comunicación de datos	F_7 Facilidad operativa	F_{12} Facilidad de instalación
F_3 Funciones distribuidas	F_8 Actualización interactiva	F_{13} Múltiples sitios
F_4 Rendimiento	F_9 Interfaces complejas	F_{14} Facilidad de cambios
F_5 Configuración muy usada	F_{10} Procesamiento complejo	

Factores de complejidad del entorno	
E_1 Familiaridad con UML	E_5 Experiencia en orientación a objetos
E_2 Trabajadores a tiempo parcial	E_6 Motivación
E_3 Capacidad de los analistas	E_7 Dificultad del lenguaje de programación
E_4 Experiencia en la aplicación	E_8 Estabilidad de los requisitos

Figura 27: Factores

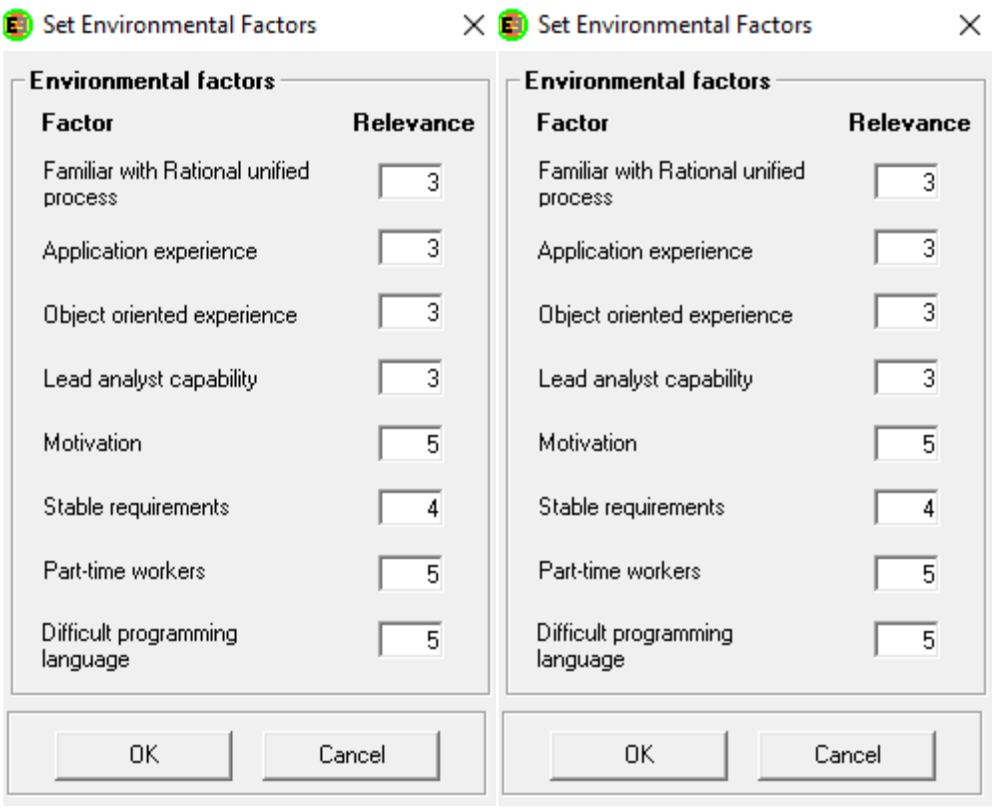


Figura 28: EZEstimate Factores

El resultado final ha sido 882 horas el cual el desarrollador en un total de días es 110 días.

Un TFG en nuestro grado tiene una asignación de 12 ECTS, por tanto, tendrá una duración aproximada de 300 horas de trabajo, además de esto se deben desarrollar las memorias y los anexos. Esto implica que las horas de trabajo puedan incrementar considerablemente y aproximarse a cifras que hemos estimado.

Para más detalles se puede consultar el Anexo I del proyecto.

5.2.2 PLANIFICACIÓN TEMPORAL

La planificación temporal es una actividad que evoluciona con el tiempo y que permite identificar, definir y programar las actividades específicas que se requieren para llevar a cabo un proyecto .

En nuestro proyecto la planificación temporal ha sido utilizada para poder generar un esquema del trabajo que se ha ido realizando y especificar cada periodo de tiempo de las tareas que hay que se han ido llevando a cabo.

Dentro de las etapas anteriormente mencionadas del proceso unificado van a estar compuestas por iteraciones y a su vez van a estar compuestas por las disciplinas. Las disciplinas o flujos de trabajo organizan las actividades fundamentales de gestión y desarrollo del proyecto.

Modelado de negocio: Se centra en reuniones con el cliente y el seguimiento de avances se extenderá a lo largo de todo el proyecto. Dentro del desarrollo de nuestro proyecto esto se traduce en reuniones con los tutores.

Requisitos: El objetivo de la disciplina de los requisitos es definir e implementar los requisitos que formaran nuestro proyecto. Deben quedar bien definidos en las primeras fases del proceso debido a que posteriormente se implementara en base a ellos.

Análisis: En esta disciplina se analizarán los diferentes aspectos referentes al sistema.

Diseño: En la disciplina de diseño se va a intentar pensar como realizar las diferentes implementaciones de los requisitos y los patrones que se van a seguir para conseguirlo. Se llevará a cabo hasta la fase de construcción debido a que es necesario prácticamente hasta el final del desarrollo del proyecto.

Implementación: En la disciplina de la implementación se va a proceder a la creación de código propiamente dicha siguiendo las indicaciones que se han estipulado anteriormente en el diseño. Se llevará a cabo durante la fase de construcción y elaboración ya que es donde se podrá implementar código.

Pruebas: En la disciplina de pruebas se va a centrar en la búsqueda de fallos, errores o bugs que puedan ser perjudiciales y se buscara una solución viable para ellos. Se llevará a cabo durante la fase de construcción ya que es principalmente donde reside la mayor creación de código y también en la fase de transición donde se buscará refinar al máximo el producto final.

Para el desarrollo de la planificación temporal en Microsoft Project se comenzó definiendo las fases anteriormente explicadas de inicio, elaboración, construcción y transición. Una vez definidas se fueron añadiendo las distintas iteraciones junto con las disciplinas correspondiente y junto a las tareas que las componen.

Tras esto se ha realizado una estimación del tiempo que se puede invertir en cada tarea de una forma razonable y buscando ser lo más coherente posible.

Cuando los tiempos de las tareas están establecidos se ha procedido a crear las relaciones que forman el diagrama de Gantt. Esto nos permite saber que tareas van antes que otras y la carga de trabajo que van a tener de forma estimada. Después para cada tarea en particular se van a asignar los recursos que tiene cada tarea. En todas ellas será el propio desarrollador del proyecto ya que no se cuenta con más miembros.

Para poder llevar a cabo la planificación temporal se ha hecho uso de la herramienta Microsoft Project la cual nos va a permitir introducir un calendario, horarios de trabajo, diagramas. Todo esto nos permite desarrollar diagramas de Gantt con las actividades del proyecto y una duración estimada. Para más detalles se puede consultar el Anexo I del proyecto.

Task Mode	Nombre de tarea	Duration	Start	Finish	Predecessors	Resource Names
	TFG_RAIDEN	**110 days?**	**Mon 14/02/22**	**Tue 21/06/22**		
	Inicio	**11 days**	**Mon 14/02/22**	**Fri 25/02/22**		
	Iteración 1	**3 days**	**Mon 14/02/22**	**Wed 16/02/22**		
	Modelado de negocio	**1 day**	**Mon 14/02/22**	**Mon 14/02/22**		
	Reunión con los tutores TFG	1 day	Mon 14/02/22	Mon 14/02/22		Andrei Brinza
	Requisistos	**1 day**	**Tue 15/02/22**	**Tue 15/02/22**		
	Busqueda de funcionalidades del sistema	1 day	Tue 15/02/22	Tue 15/02/22	5	Andrei Brinza
	Diseño	**1 day**	**Wed 16/02/22**	**Wed 16/02/22**		
	Primera propuesta arquitectónica	1 day	Wed 16/02/22	Wed 16/02/22	7	Andrei Brinza
	Hito Fin Iteración 1	0 days	Mon 14/02/22	Mon 14/02/22		
	Iteración 2	**8 days**	**Thu 17/02/22**	**Fri 25/02/22**	**3**	
	Modelado de negocio	**1 day**	**Thu 17/02/22**	**Thu 17/02/22**		
	Se definen los principales objetivos del proyecto	1 day	Thu 17/02/22	Thu 17/02/22		Andrei Brinza
	Requisitos	**2 days**	**Fri 18/02/22**	**Sat 19/02/22**	**12**	
	Establcer requisitos funcionales, no funcionales y de información	2 days	Fri 18/02/22	Sat 19/02/22		Andrei Brinza[50%]
	Definición de los actores del sistema	2 days	Fri 18/02/22	Sat 19/02/22		Andrei Brinza[50%]
	Análisis	**4 days**	**Mon 21/02/22**	**Thu 24/02/22**	**12;14**	
	Selección de lenguajes y tecnologias como frameworks para el desarollo del proyecto	4 days	Mon 21/02/22	Thu 24/02/22		Andrei Brinza[20%]
	Diseño	**5 days**	**Mon 21/02/22**	**Fri 25/02/22**	**12;14**	
	Elaboración de un prototipo de la aplicacion web con Wix	5 days	Mon 21/02/22	Fri 25/02/22		Andrei Brinza[40%]
	Hito Fin Iteración 2	0 days	Mon 14/02/22	Mon 14/02/22		
	Hito Fin etapa Inicio	0 days?	Mon 14/02/22	Mon 14/02/22		
	Elaboración	**7 days?**	**Fri 25/02/22**	**Sat 05/03/22**	**2**	

Figura 29: Diagrama Gantt-1

Nombre de tarea	Duration	Start	Finish	Predecessors	Resource Names
TFG_RAIDEN	**110 days?**	**Mon 14/02/22**	**Tue 21/06/22**		
Inicio	**11 days**	**Mon 14/02/22**	**Fri 25/02/22**		
Hito Fin etapa Inicio	0 days?	Mon 14/02/22	Mon 14/02/22		
Elaboración	**7 days?**	**Fri 25/02/22**	**Sat 05/03/22**	**2**	
Iteración 1	**4 days**	**Sat 26/02/22**	**Wed 02/03/22**		
Modelado de negocio	**1 day**	**Mon 28/02/22**	**Mon 28/02/22**		
Reunión con los tutores	1 day	Mon 28/02/22	Mon 28/02/22		Andrei Brinza[10%]
Requisitos	**1 day**	**Mon 28/02/22**	**Mon 28/02/22**		
Definición de casos de uso	1 day	Mon 28/02/22	Mon 28/02/22		Andrei Brinza[90%]
Análisis	**2 days**	**Tue 01/03/22**	**Wed 02/03/22**	**25;27**	
Elaboración diagramas de casos de uso	2 days	Tue 01/03/22	Wed 02/03/22		Andrei Brinza[70%]
Diseño	**2 days**	**Tue 01/03/22**	**Wed 02/03/22**	**25;27**	
Elaboración del un modelo de dominio	2 days	Tue 01/03/22	Wed 02/03/22		Andrei Brinza[30%]
Implementación	**1 day**	**Sat 26/02/22**	**Sat 26/02/22**		
Instalación se software y frameworks necesarios para el proyecto	1 day	Sat 26/02/22	Sat 26/02/22		Andrei Brinza[20%]
Pruebas	**1 day**	**Sat 26/02/22**	**Sat 26/02/22**		
Se comprueba el funcionamiento correcto del software instalado	1 day	Sat 26/02/22	Sat 26/02/22		Andrei Brinza[5%]
Hito Fin iteracion 1	0 days	Fri 25/02/22	Fri 25/02/22		
Iteración 2	**3 days**	**Thu 03/03/22**	**Sat 05/03/22**	**24**	
Modelado de negocio	**1 day**	**Thu 03/03/22**	**Thu 03/03/22**		
Revaloración de los costes y el tiempo	1 day	Thu 03/03/22	Thu 03/03/22		Andrei Brinza[5%]
Requisistos	**1 day**	**Thu 03/03/22**	**Thu 03/03/22**		
Refinamiento de los casos de uso	1 day	Thu 03/03/22	Thu 03/03/22		Andrei Brinza[5%]
Análisis	**3 days**	**Thu 03/03/22**	**Sat 05/03/22**		
Elaboración de los diagramas de secuencia	3 days	Thu 03/03/22	Sat 05/03/22		Andrei Brinza[25%]
Diseño	**1 day**	**Thu 03/03/22**	**Thu 03/03/22**		
Elaboración de un modelo de diseño software	1 day	Thu 03/03/22	Thu 03/03/22		Andrei Brinza[25%]
Implementación	**1 day**	**Thu 03/03/22**	**Thu 03/03/22**		
Se hacen pequeñas pruebas sobre los frameworks	1 day	Thu 03/03/22	Thu 03/03/22		Andrei Brinza[30%]
Pruebas	**1 day**	**Thu 03/03/22**	**Thu 03/03/22**		
Se testea el codigo escrito en los frameworks	1 day	Thu 03/03/22	Thu 03/03/22		Andrei Brinza[10%]
Hito Fin Iteración 2	0 days?	Fri 25/02/22	Fri 25/02/22		
Construcción	**64 days?**	**Sat 05/03/22**	**Thu 19/05/22**	**23**	

Figura 30: Diagrama Gantt-2

5.3 ESPECIFICACIÓN DE REQUISITOS

En esta fase se ha llevado a cabo la especificación de requisitos de software, donde se han definido los participantes del proyecto los objetivos primordiales, los requisitos de información del sistema, los actores que están involucrados en el sistema, creación de los requisitos funcionales y descripción de los casos de uso y finalmente la definición de requisitos no funcionales.

Para una ampliación se puede consultar el Anexo II del proyecto.

5.3.1 PARTICIPANTES DEL PROYECTO

- Andrei Brinza
- Guillermo Hernández González
- Sara Rodríguez González
- Alfonso González Briones

Participante	Andrei Brinza
Organización	Universidad de Salamanca
Rol	Ingeniero Informático
Es desarrollador	Sí
Es cliente	No
Es usuario	No
Comentarios	Ninguno

5.3.2 OBJETIVOS PRIMORDIALES

Gestión de Usuarios: El sistema deberá proporcionar funciones necesarias para crear altas, bajas o modificación de usuarios. Además, para poder controlar el acceso al sistema de forma adecuada se dispondrá de varios roles de usuarios.

Gestión de Simulación y control de agentes: El sistema debe ser capaz de proporcionar a los usuarios un entorno de simulación controlado en el puedan realizar casos de estudio en tiempo real sobre el mapa. Todo ello con una interfaz amigable en la puedan fácilmente crear vehículos en distintos puntos del mapa y observar el comportamiento hacia su destino.

Gestión de Estadísticas: El sistema debe ser capaz de recoger la información generada por los casos de estudio de los usuarios dentro de la simulación y mostrarla de forma ordenada en graficas que se actualizarán en tiempo real.

Gestión de Eventos: El sistema debe ser capaz de permitir al usuario la activación de eventos en tiempo real dentro de la simulación para observar las consecuencias que estos tienen sobre el entorno de la simulación y los tiempos de viajes de los vehículos creados.

OBJ-0004	Gestión de Eventos
Versión	1.0 (31/08/2022)
Autores	• Andrei Brinza
Fuentes	• Alfonso González Briones • Guillermo Hernández González • Sara Rodríguez González
Descripción	El sistema deberá *tener una serie de eventos que se puedan controlar por el usuario y que se vean reflejados en la simulación*
Subobjetivos	Ninguno
Importancia	vital
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

5.3.3 REQUISITOS DE INFORMACIÓN

Los requisitos de información van a ser necesarios para que el sistema funcione ya que es la definición de la información con la que se trabaja en el sistema.

Información de los usuarios: El sistema deberá almacenar la información correspondiente a los usuarios.

Información de la Simulación: El sistema deberá almacenar la información correspondiente a los usuarios.

Información de las Estadísticas: El sistema deberá almacenar la información correspondiente a las estadísticas de los casos de estudio.

Información de Eventos: El sistema deberá almacenar la información correspondiente a los eventos que se pueden dar en la simulación.

IRQ-0001	Información de los usuarios	
Versión	1.0 (31/08/2022)	
Autores	• Andrei Brinza	
Fuentes	• Alfonso González Briones • Guillermo Hernández González • Sara Rodríguez González	
Dependencias	• [OBJ-0001] Gestión de Usuarios	
Descripción	El sistema deberá almacenar la información correspondiente a *los usuarios* . En concreto:	
Datos específicos	• Nombre de Usuario • Email • Nombre • Apellido • Usuario-EstaActivo • Usuario-EsStaff • Usuario-EsSuperusuario	
Tiempo de vida	**Medio**	**Máximo**
	PD	**PD**
Ocurrencias simultáneas	**Medio**	**Máximo**
	PD	**PD**
Importancia	vital	
Urgencia	hay presión	
Estado	validado	
Estabilidad	alta	
Comentarios	Ninguno	

5.3.4 REQUISITOS FUNCIONALES

En los requisitos funcionales se representan el comportamiento del sistema. Inicialmente se va a comenzar por hacer un diagrama de paquetes que nos va a permitir ver una vista general de cómo se va a organizar el sistema en paquetes. A continuación, se definirán los actores involucrados en el sistema mediante un diagrama también donde se podrá observar la herencia de ellos. Finamente se comenzará a crear los diagramas de casos de uso y su respectiva descripción con UML y REM respectivamente.

5.3.4.1 DIAGRAMA DE PAQUETES

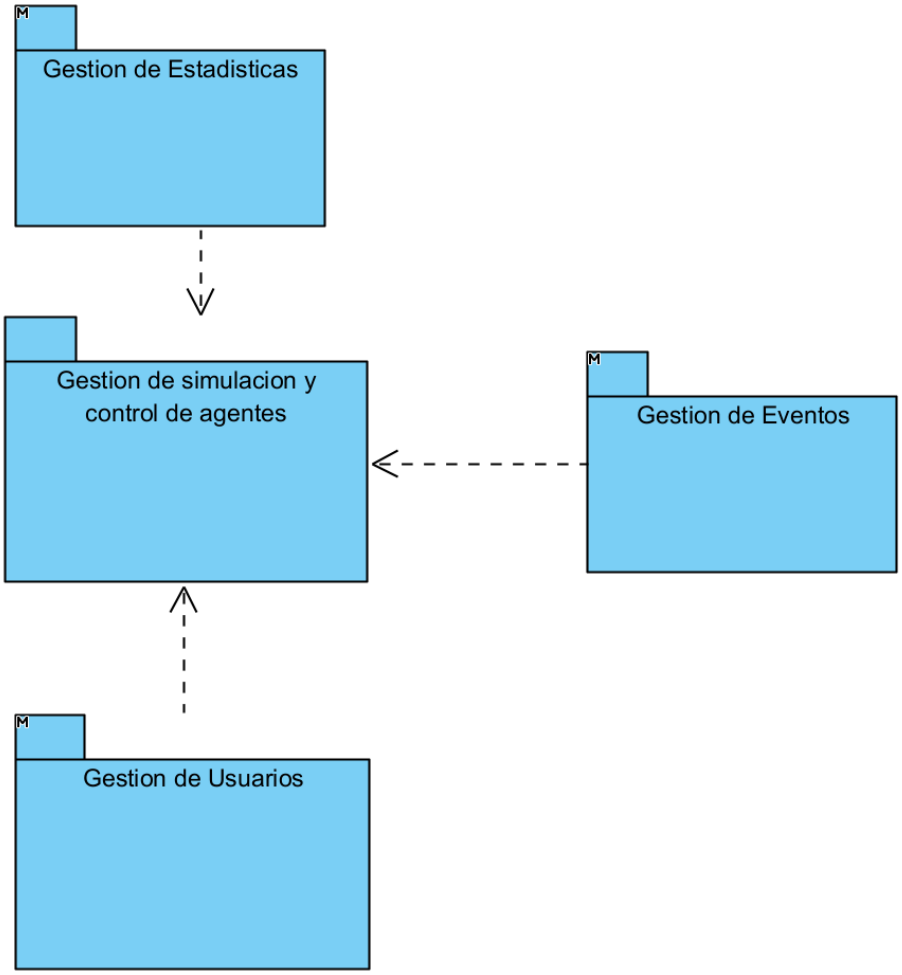


Figura 31: Diagrama de paquetes

5.3.4.2 DEFINICIÓN DE LOS ACTORES

Se definen los siguientes actores dentro del sistema:

- Usuario No Logueado
- Usuario No Registrado
- Usuario
- Administrador
- <<System>> Timer

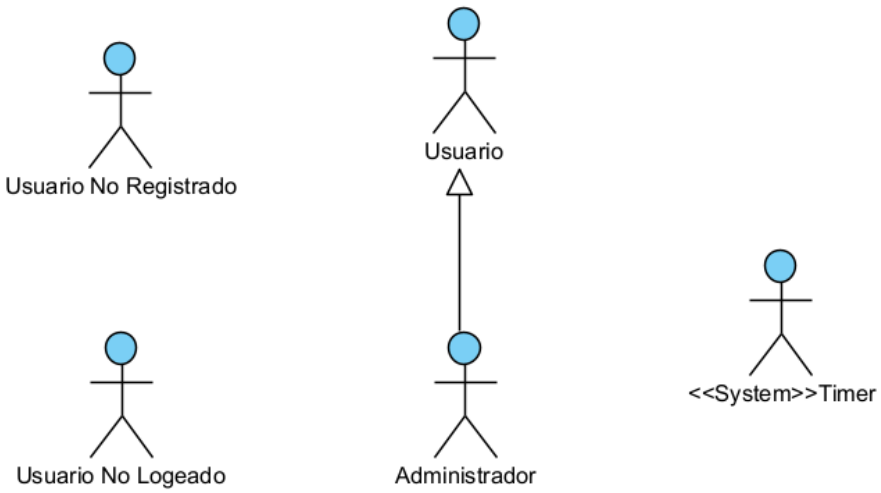


Figura 32: Definición de actores

ACT-0002	**Usuario**
Versión	1.0 (31/08/2022)
Autores	• Andrei Brinza
Fuentes	• Alfonso González Briones • Guillermo Hernández González • Sara Rodríguez González
Descripción	Este actor representa *al usuario estándar del sistema que podría usar la plataforma con sus funcionalidades básicas*
Comentarios	Ninguno

5.3.4.3 DEFINICIÓN DE LOS CASOS DE USO

En la definición de los casos de uso se comienza desarrollando el diagrama de casos de uso del propio paquete y creando una idea general del comportamiento. A continuación, se realiza la tabla de caso de uso con las dependencias apropiadas.

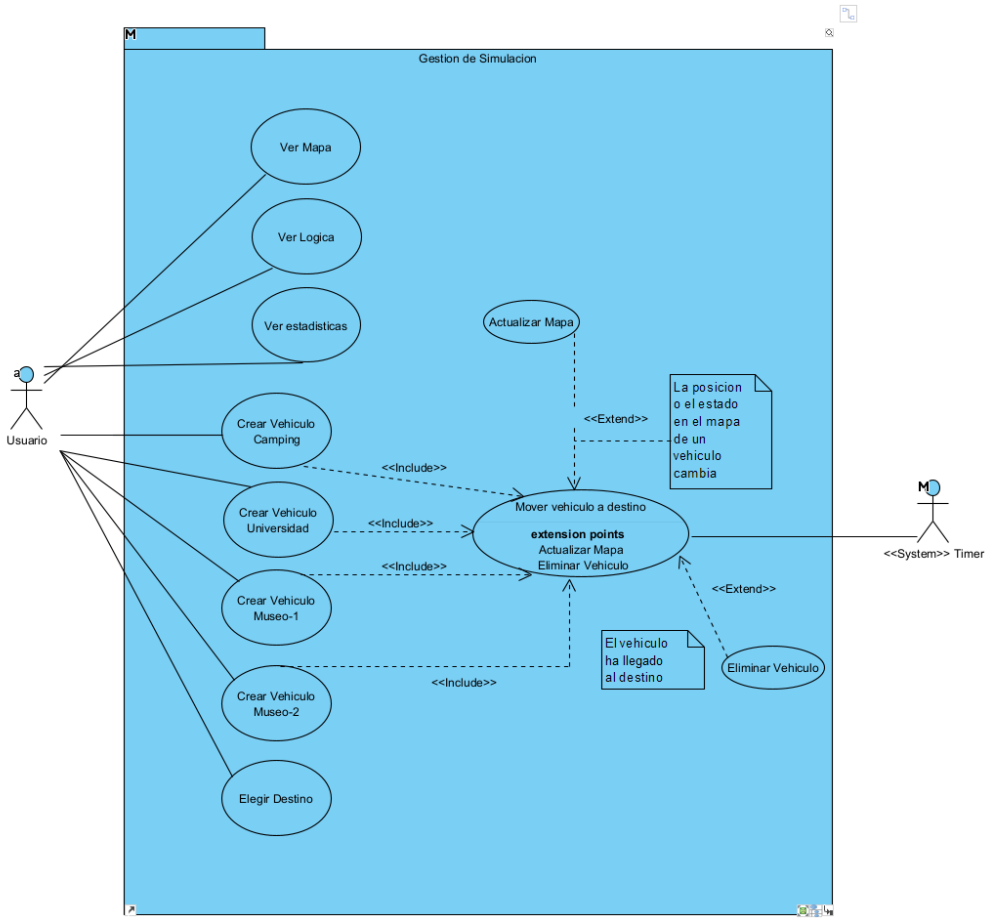


Figura 33: Gestión de Simulación

UC-0017	Mover Vehiculo a destino	
Versión	1.0 (01/09/2022)	
Autores	• Andrei Brinza	
Fuentes	• Alfonso González Briones • Guillermo Hernández González • Sara Rodríguez González	
Dependencias	• [IRQ-0003] Información de las Estadísticas • [OBJ-0002] Gestión de Simulación y control de agentes • [IRQ-0002] Información de la Simulación • [OBJ-0003] Gestión de Estadísticas	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *un vehículo es creado y tiene asociado un destino* o durante la realización de los siguientes casos de uso: [UC-0012] Crear Vehículo Camping, [UC-0013] Crear Vehículo Universidad, [UC-0014] Crear Vehículo Museo-1, [UC-0015] Crear Vehículo Museo-2	
Precondición	el vehículo debe existir previamente	
Secuencia normal	**Paso**	**Acción**
	1	El actor <<System>> Timer (ACT-0004) *acciona el bloque carMoveTo correspondiente y cambia las variables de posición del vehículo acorde a su velocidad establecida cada segundo*
	2	Se realiza el caso de uso Actualizar Mapa (UC-0018)
	3	Si *el vehículo a llegado a su destino,* se realiza el caso de uso Eliminar Vehiculo (UC-0019)
	4	Se realiza el caso de uso Actualizar Mapa (UC-0018)
Postcondición	el vehículo cambia su posición en el plano del modelo	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Tiempo máximo**
	-	-
Frecuencia esperada	1 veces por décima(s) de segundo	
Importancia	vital	
Urgencia	inmediatamente	
Estado	validado	
Estabilidad	alta	
Comentarios	Ninguno	

5.3.5 REQUISITOS NO FUNCIONALES

Los requisitos no funcionales representan características generales y restricciones del sistema:

Usabilidad: Fácil uso y comprensibilidad.

Compatibilidad: Posibilidad de usar varias plataformas.

Seguridad: Mínimos de seguridad ya que se guarda información de los usuarios.

Reusabilidad: Reutilización de código o módulos del sistema.

Disponibilidad: El sistema deberá ofrecer una buena disponibilidad.

Rendimiento: Tiempo de respuesta cortos y consumo reducido de recursos.

NFR-0001	**Usabilidad**
Versión	1.0 (01/09/2022)
Autores	• Andrei Brinza
Fuentes	• Alfonso González Briones • Guillermo Hernández González • Sara Rodríguez González
Dependencias	• [OBJ-0002] Gestión de Simulación y control de agentes • [OBJ-0003] Gestión de Estadísticas • [OBJ-0004] Gestión de Eventos
Descripción	El sistema deberá *ser comprensible y fácil de usar por los usuarios de la plataforma por tanto se debe tener una interfaz sencilla e intuitiva*
Importancia	vital
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

5.4 ANÁLISIS DE REQUISISTOS

Tras acabar con la fase de especificación de requisitos se pasa al análisis de estos mismos, para ello se va a hacer un refinamiento y organización de los requisitos del sistema con el objetivo de asegurar la ausencia de errores en el planteamiento o en la calidad del producto final.

Para una ampliación se puede consultar el Anexo III del proyecto.

La estructura será la siguiente:

- **Modelo de dominio:** Diagrama de clases que representara los principales requisitos del domino del problema.
- **Realización de casos de uso:** Diagramas de secuencia que representara las acciones de los casos de uso.
- **Clase de análisis:** Descomposición del sistema para definir las relaciones de cada paquete.
- **Descripción de la arquitectura:** Se muestra la vista de la arquitectura del modelo de análisis.

5.4.1 MODELO DE DOMINIO

El modelo de dominio describe los requisitos de datos estáticos de la aplicación Permite identificar las clases conceptuales significativas en el dominio y como se relacionan entre ellas mediante abstracciones y un vocabulario basándose en requisitos de información.

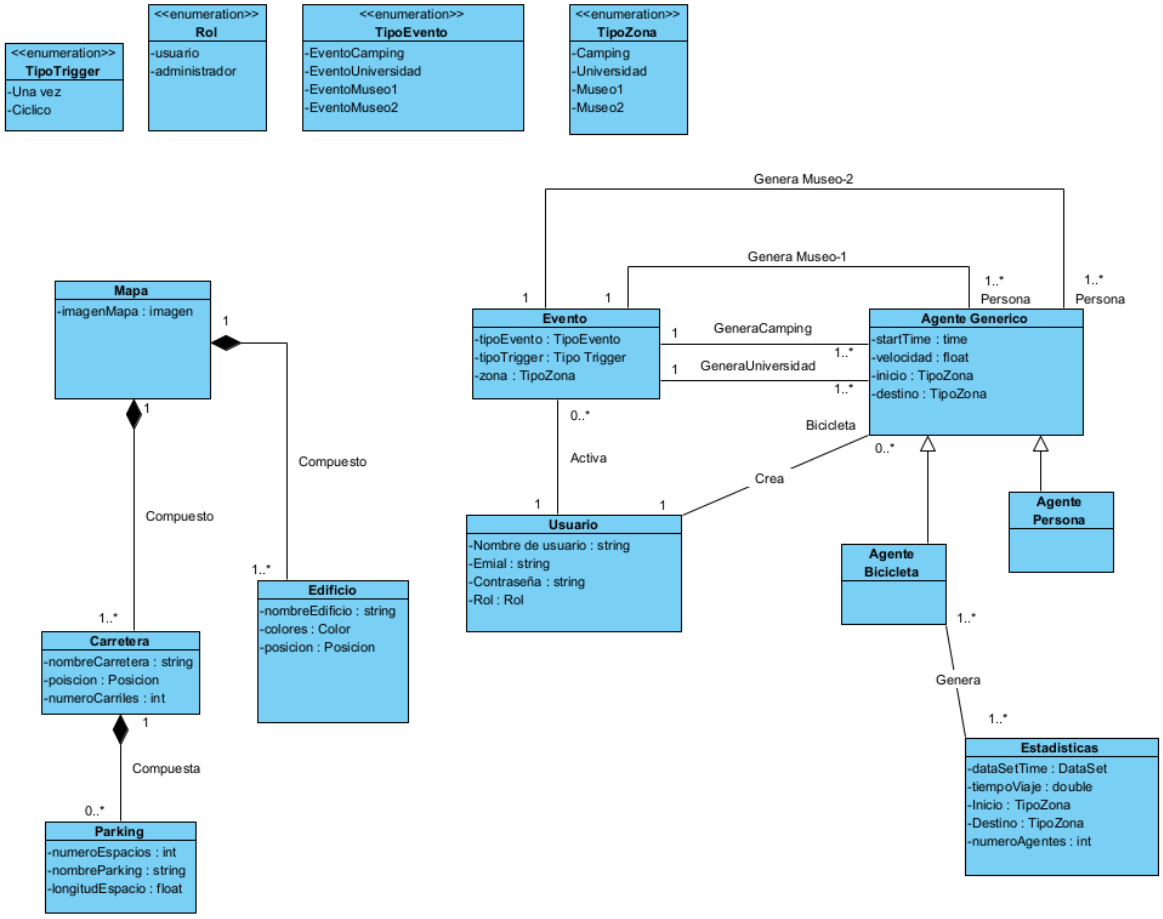


Figura 34: Modelo de Domino

5.4.2 REALIZACIÓN DE LOS CASOS DE USO

Se realizan los distintos diagramas de secuencia asociados a los casos de uso donde podremos observar los intercambios de información entre interfaces, controladores y modelos.

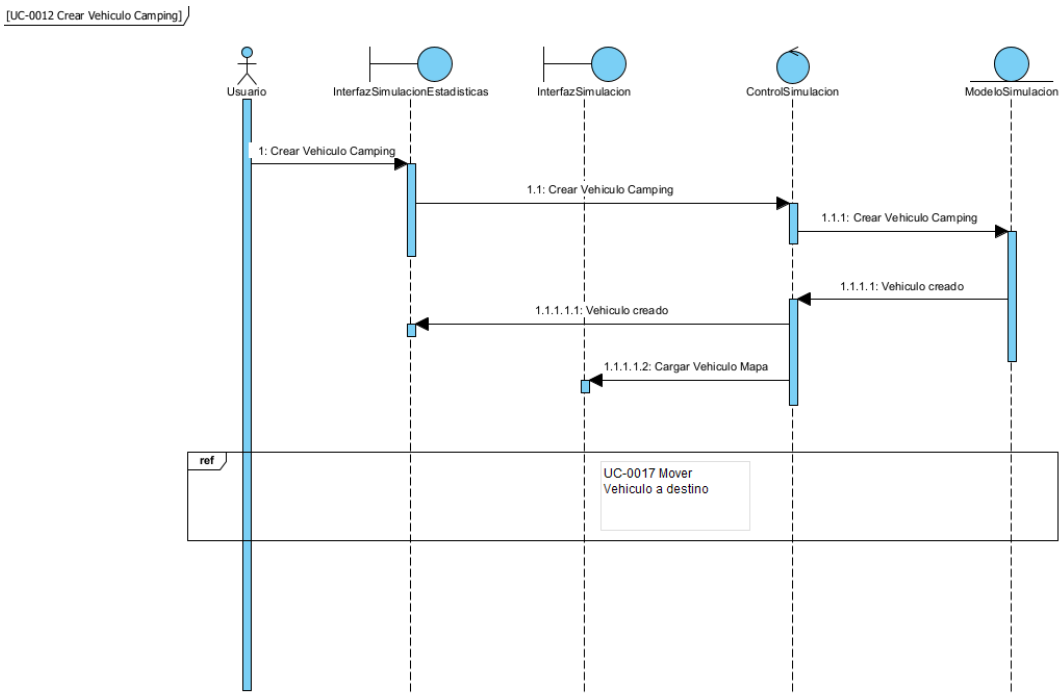


Figura 35: UC-0012 Crear Vehículo Camping

5.4.3 CLASE DE ANÁLISIS

Se van a representar las distintas clases de análisis (Interfaz, Control, Entidad) en sus respectivos paquetes las relaciones que tienen entre ellos mediante diagramas de comunicación.

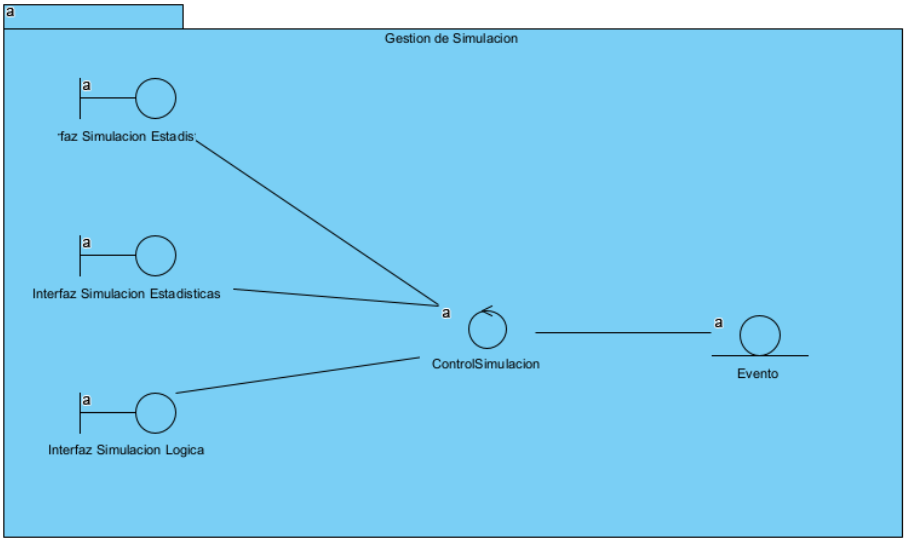


Figura 36: Gestión de Simulación clase análisis

5.4.4 VISTA DE ARQUITECTURA

En la siguiente figura se puede ver la descripción de la arquitectura del modelo de análisis junto con los paquetes y la relación entre ellos.

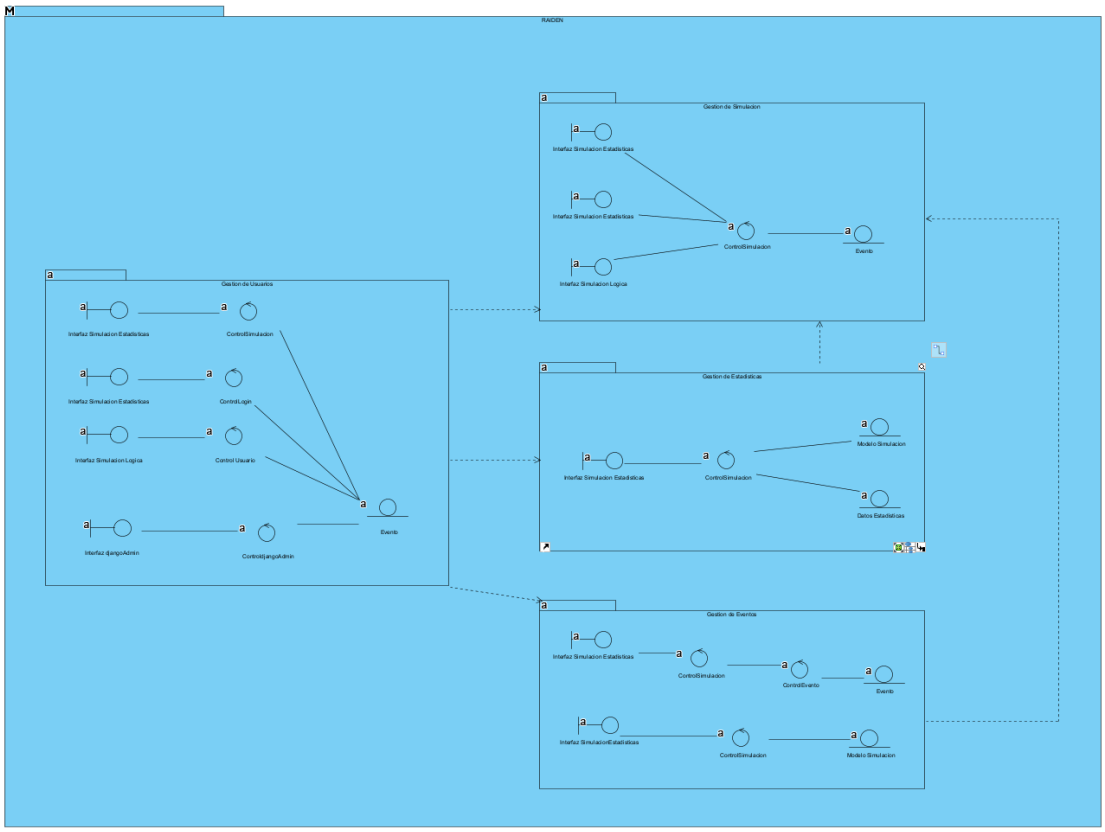


Figura 37: Vista arquitectura

5.5 DISEÑO DEL SISTEMA SOFTWARE

En fase de diseño de software, la cual después de una fase de modelación y análisis de requisitos se pasa a un plano menos abstracto donde empezaremos a definir elementos que estarán en la fase final del proyecto. El modelo de diseño será mucho más específico y cercano a la implementación ya que es un modelo físico a diferencia del anterior y tratara el alcance de requisitos funcionales y no funcionales dentro del sistema.

La estructura será la mostrada a continuación:

- **Patrones arquitectónicos:** Se especifica los patrones utilizados para diseñar y desarrollar la aplicación.
- **Subsistemas de diseño:** Descomposición del sistema en paquetes que componen la aplicación y la relación entre ellos.
- **Clases de diseño:** Se especifican las clases que forman la aplicación.
- **Descripción de la arquitectura:** Expone la vista de la arquitectura del modelo de diseño.
- **Realización de casos de uso:** Se describen mediante diagramas la interacción de las clases.
- **Modelo de despliegue:** Describe la distribución de los distintos nodos que componen el sistema.

5.5.1 PATRONES ARQUITECTÓNICOS

5.5.1.1 PATRÓN MVVM (MODEL-VIEW-VIEWMODEL)

Para el desarrollo de nuestra aplicación hemos utilizado el *framework* de Vuejs, este *framework* está basado en el patrón MVVM. Este patrón ayuda a separar fácilmente la lógica de negocio y presentación de una aplicación de su interfaz de usuario. Este tipo de diseño ayuda a las pruebas, mantenimiento o futuros cambios dentro de la aplicación. Las diferentes partes del patrón son las siguientes:

- **Modelo:** El modelo son clases no visuales que encapsulan los datos de la aplicación por tanto se considera que representa el modelo de dominio de la aplicación, por ende, también el modelo de datos.

- **Vista:** La vista es la responsable de definir la apariencia y el diseño de lo que el usuario ve en la pantalla de la aplicación.

- **Vista-Modelo:** El modelo de vista es el que implementa propiedades y lógica donde se van a enlazar datos y notificar a la vista los cambios de estado a través de notificaciones o eventos. En vuejs esto ocurre de forma reactiva y en cuanto se disponga de un dato solicitado por el modelo de vista será automáticamente enlazado con la vista del usuario.

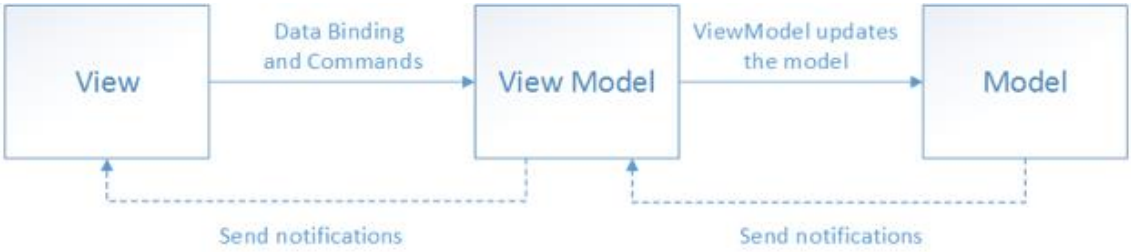


Figura 38: MVVM

Este patrón será utilizado tanto en la aplicación web como en la simulación propiamente dicha ya que al usar el entorno de simulación de AnyLogic, seguirá el mismo patrón arquitectónico a la hora de interactuar con los elementos de la simulación entre las distintas vistas y el modelo de la simulación.

Por tanto, al haber aplicado este patrón en distintas partes del trabajo como AnyLogic y en la parte desarrollo web se vio reducida la complejidad del proyecto considerablemente ya que los elementos que lo conforman estaban organizados en capas.

5.5.2 SUBSISTEMA DE DISEÑO

A continuación, se van a representar mediante un diagrama los distintos paquetes realiza la descomposición del sistema de diseño en subsistemas siguiendo la división del patrón MVVM y en los cuales vamos a poder ver las relaciones entre ellos.

El sistema que tenemos se va a descomponer en cuatro principales subsistemas:

• **Aplicación Web:** Es donde el usuario va a poder acceso al sistema y donde va a poder llevar a cabo distintas funcionalidades.

• **Simulación:** Es donde se recogen todas las funcionalidades que nos van a permitir realizar casos de estudio. La vista de la simulación está directamente relacionada con la vista de la aplicación web y se le mostrara al usuario de forma enlazada.

• **Backend:** Es el encargado de recoger las peticiones de la aplicación web guardar la información y lógica de los usuarios.

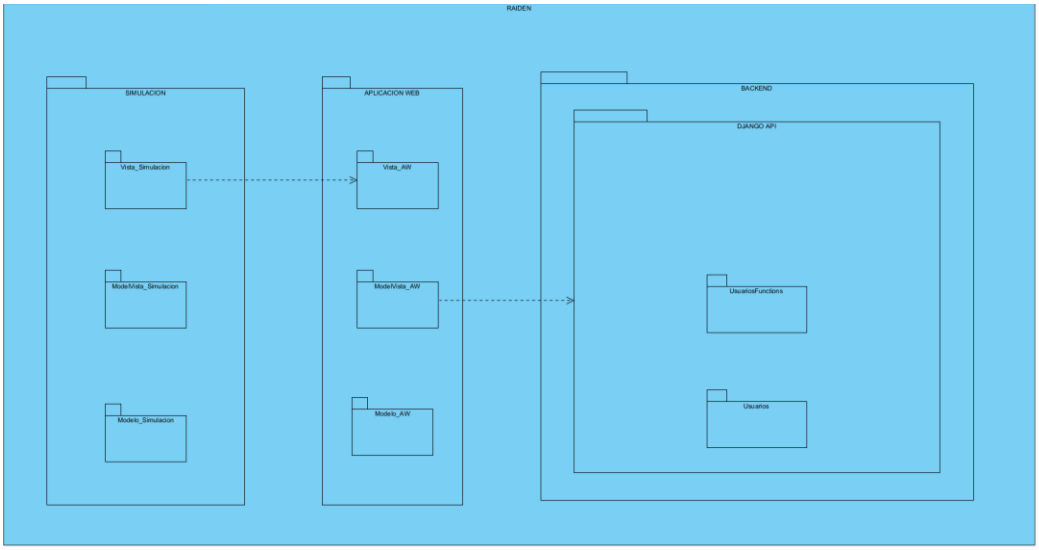


Figura 39: Subsistema

5.5.3 CLASES DE DISEÑO

En este apartado se van a describir los subsistemas de diseño mediante clases de diseño. Para ampliar la información se puede acceder el Anexo IV para ver el resto de las clases de diseño de los subsistemas.

5.5.3.1 VISTA SIMULACIÓN

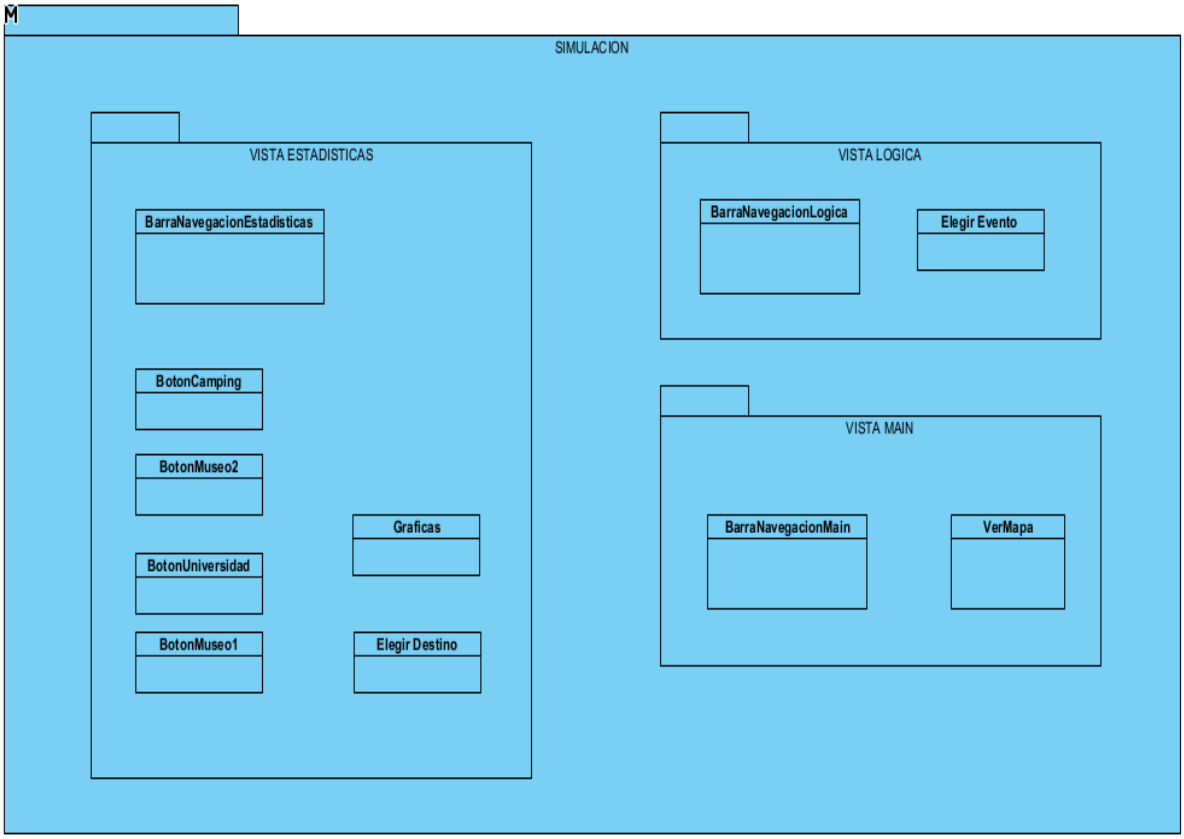


Figura 40: Vista Simulación

5.5.3.2 MODELO-VISTA SIMULACIÓN

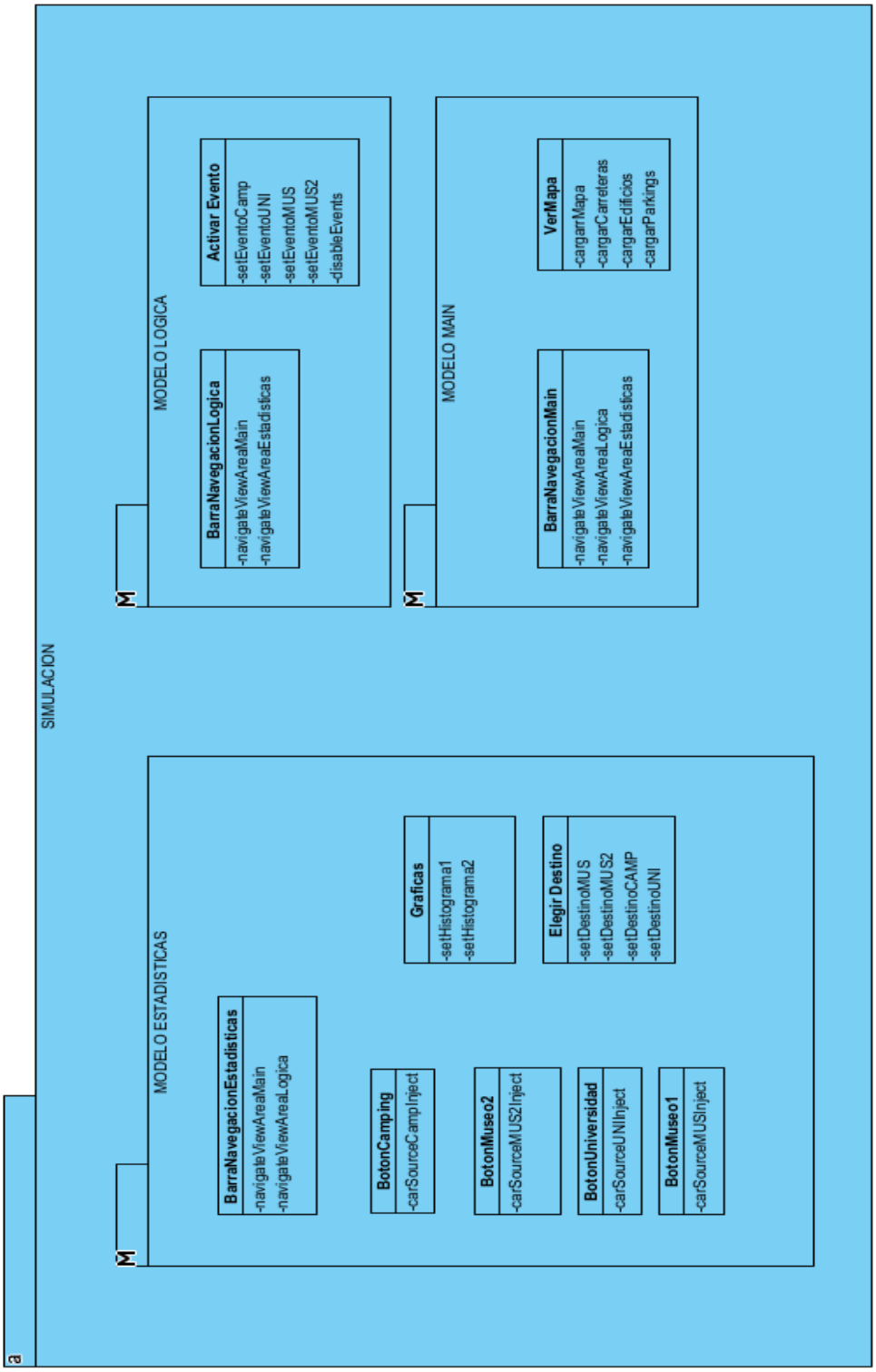


Figura 41: MODELO-VISTA SIMULACIÓN

5.5.3.3 MODELO SIMULACIÓN

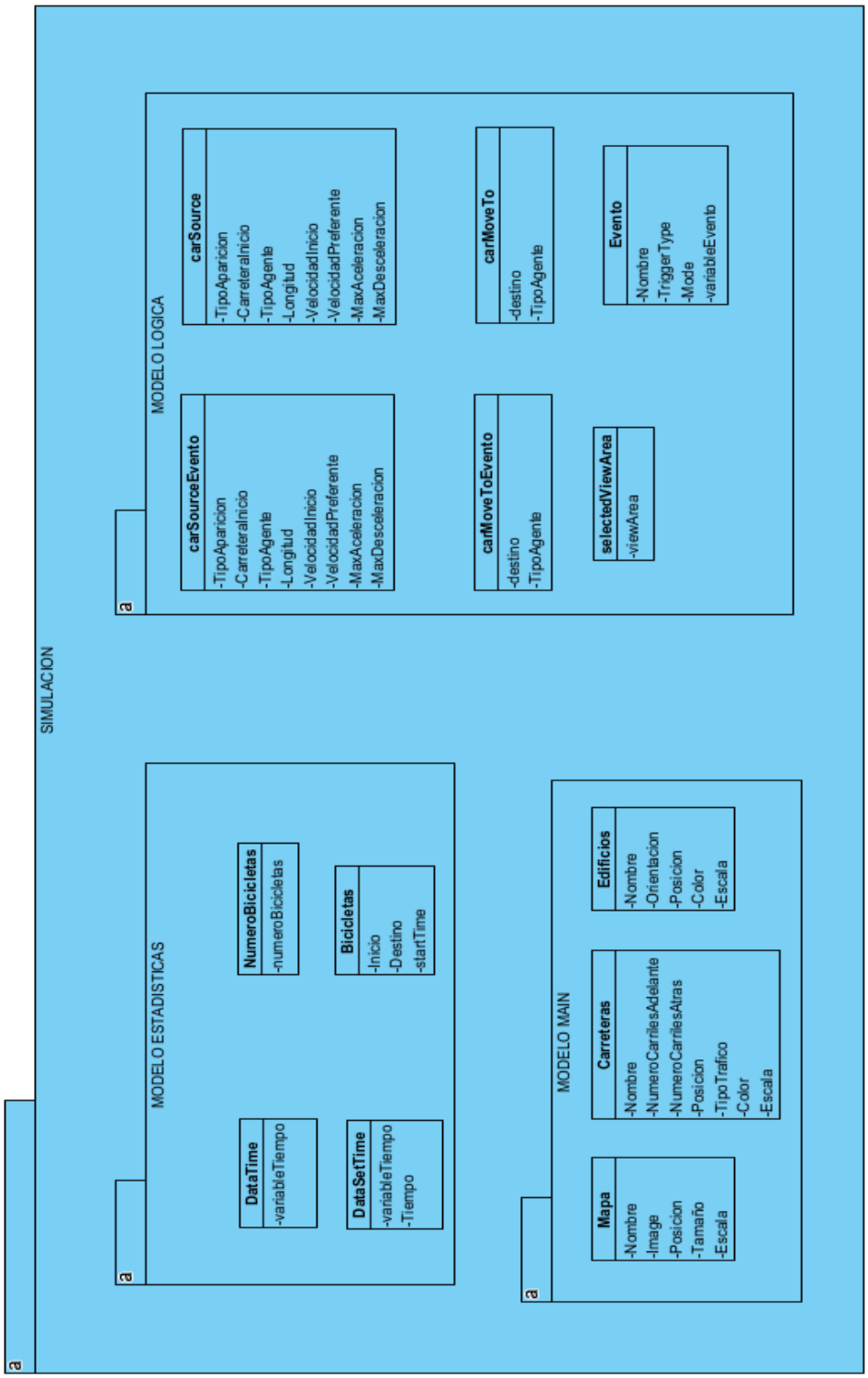


Figura 42: MODELO SIMULACIÓN

5.5.4 DESCRIPCIÓN DE LA ARQUITECTURA

Como hemos visto con anterioridad al seguir un patrón MVVM nuestra aplicación tendrá la siguiente disposición arquitectónica.

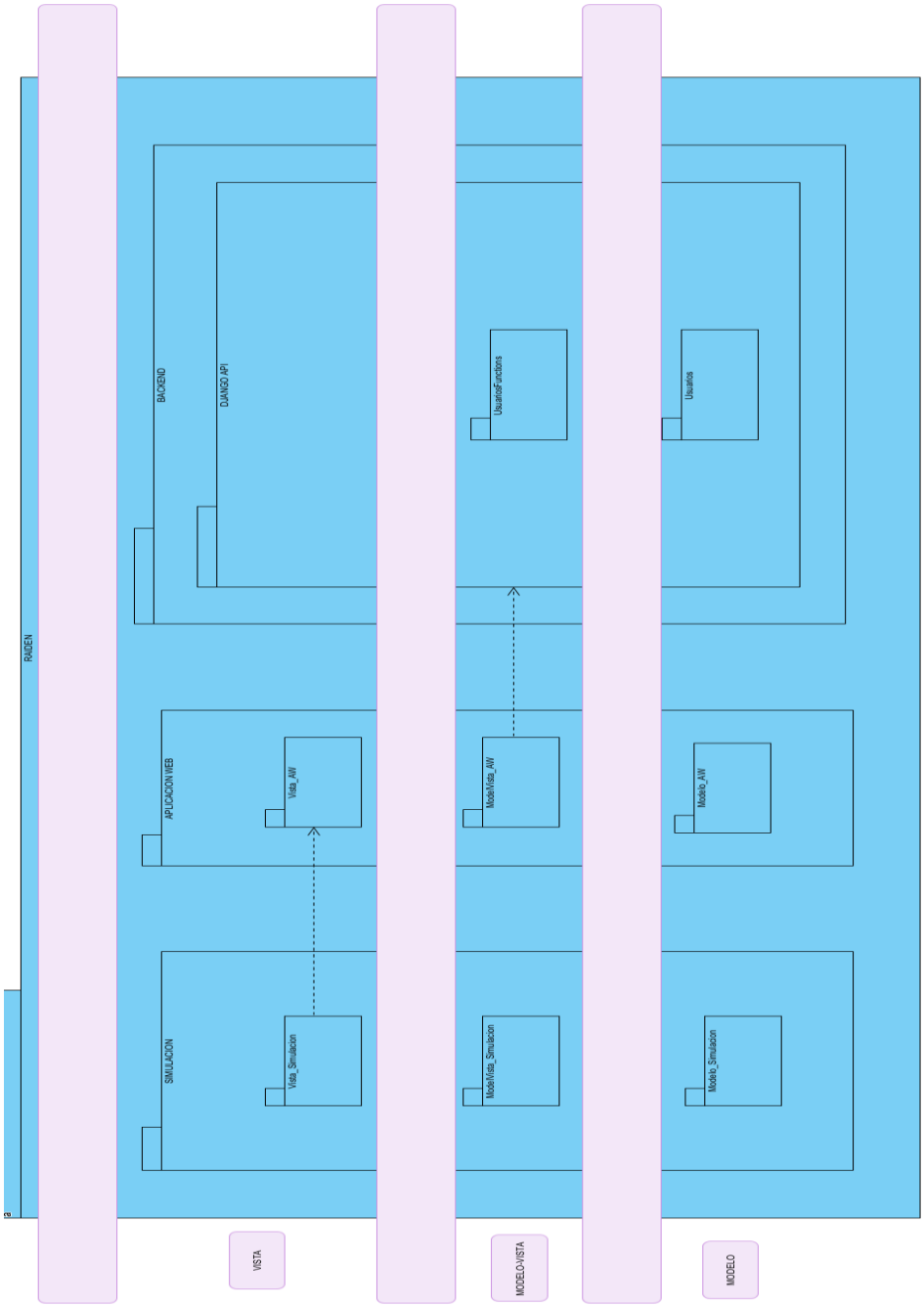


Figura 43: DESCRIPCIÓN DE LA ARQUITECTURA

5.5.5 REALIZACION DE LOS CASOS DE USO

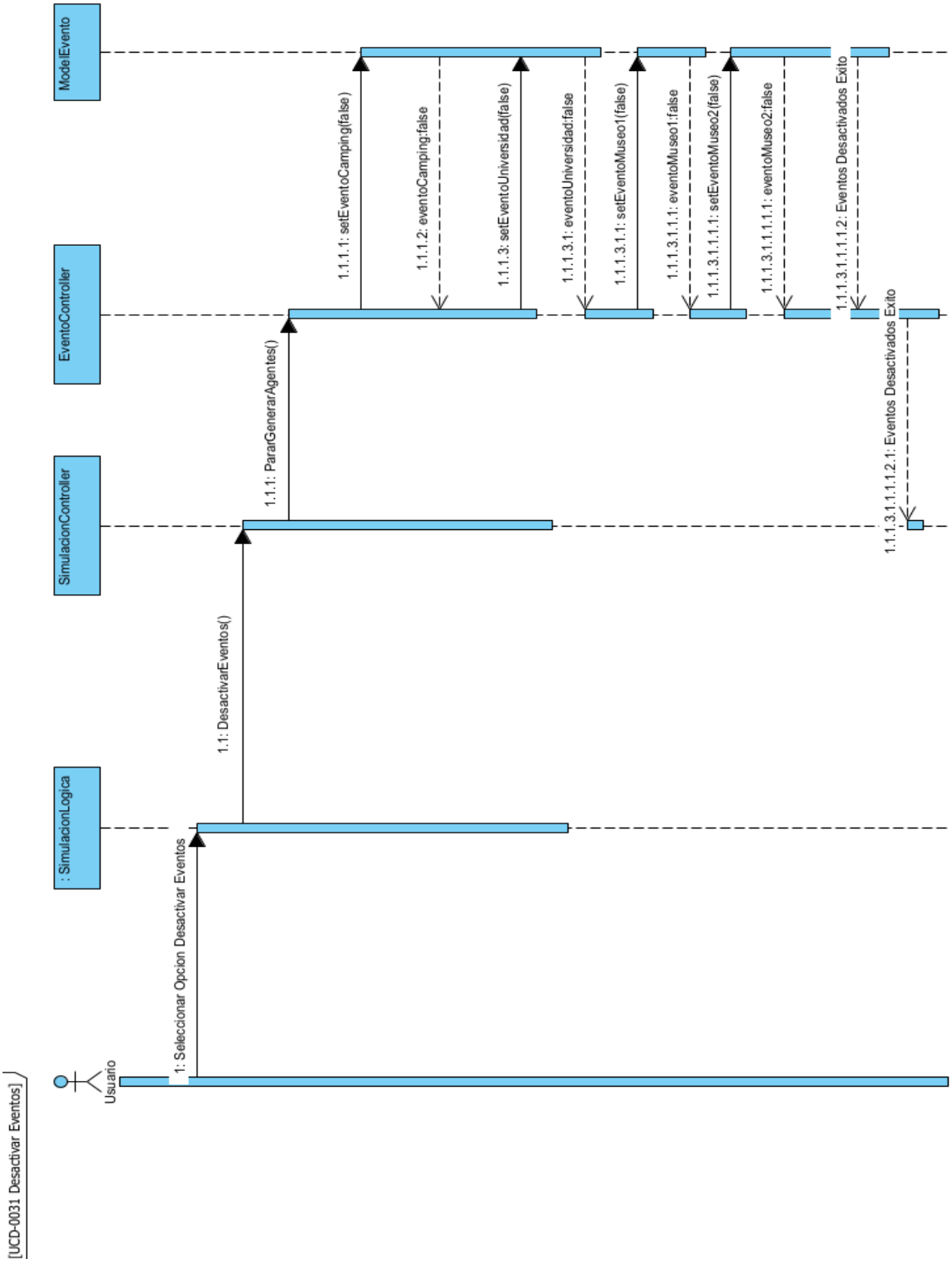
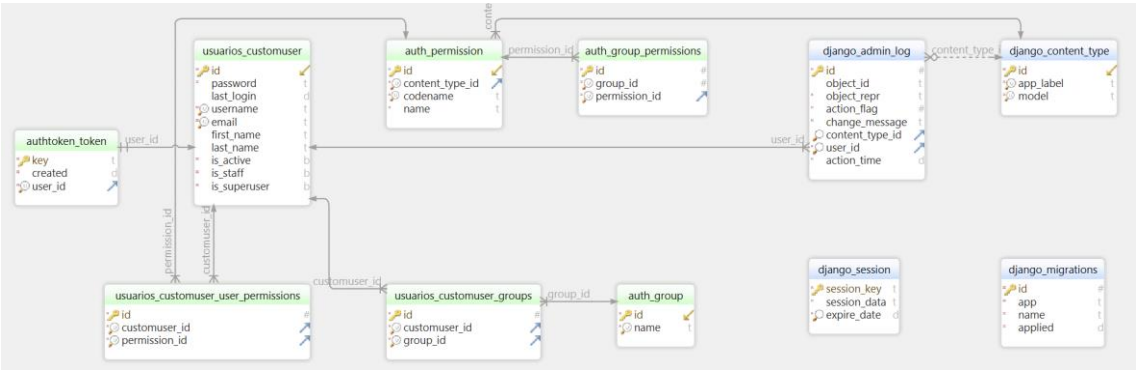


Figura 44: UCD-0031 Desactivar Eventos

5.5.6 DISEÑO DE DATOS

Para el proyecto se ha usado bases de datos relacionales de SQL (Structured Query Language) que recopilan información y la organiza en función de un ID y que podremos acceder a través de a través de consultas.

La base de datos relacional que se usa en el trabajo nos permite que la información guardada está estructurada y es menos propensa a fallos. Por eso se escogió este tipo de bases de datos para una guardado persistente de la información.



5.5.7 MODELO DE DESPLIEGUE

En el modelo de despliegue vamos a representar la distribución física de componentes o grupos de componentes mediante nodos. Cada nodo representa un recurso de cómputo, los cuales están relacionados mediante medios de comunicación. El sistema consta de los siguientes nodos:

Cliente: Representa básicamente cualquier dispositivo con acceso a Internet y pueda cargar la aplicación desde un navegador web.

ServidorWeb: Es el que va a proporcionar la aplicación web a los usuarios y hará uso de CloudAnylogic para poder desplegar la simulación.

ServidorAPI: Proporciona la API REST de los usuarios al sistema y que podremos consumir mediante axios y comunicarnos mediante protocolo HTTP.

ServidorSQLDataBase: Es la base de datos relacional de SQLite de la aplicación. Aquí se guardará la información asociada a los usuarios. SQLite no es la mejor opción si se quiere poner en producción una aplicación de grandes dimensiones, pero al ser un prototipo será suficiente. En caso de aumento será sencillo realizar una migración a una base datos mejor.

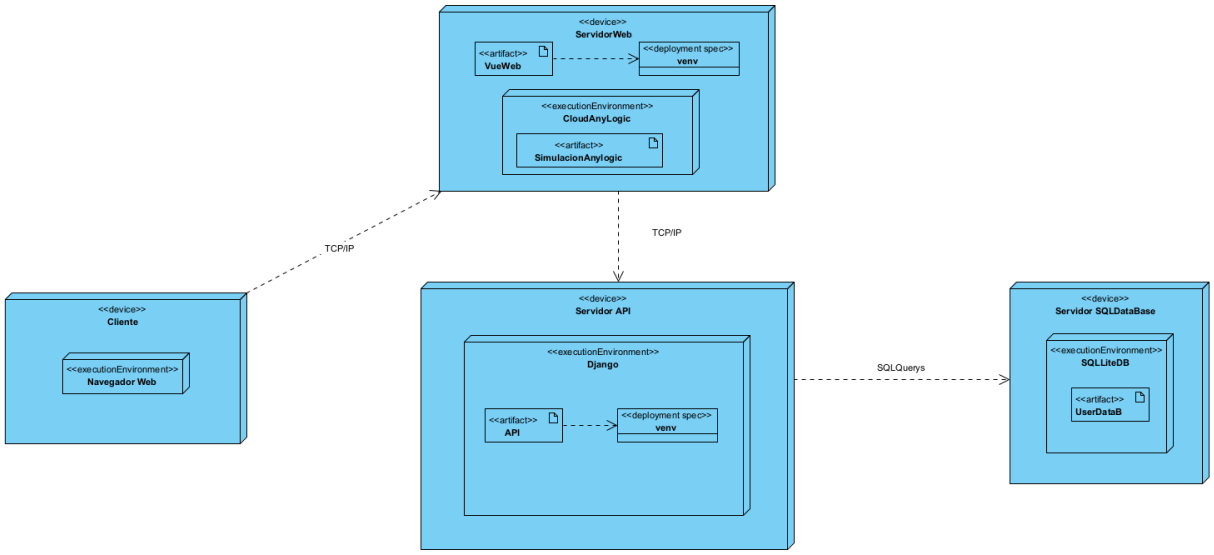


Figura 45: Modelo de Despliegue

5.6 IMPLEMENTACIÓN

En esta fase se va a exponer como se ha ido realizado la codificación propiamente del proyecto separando en dos grandes bloques. La parte de desarrollo web en la que se realiza la aplicación web y la parte del desarrollo de la simulación.

5.6.1 APLICACIÓN WEB

Durante el desarrollo de la aplicación web se generó diferente tipo código y se han usado distintos tipos de módulos y herramientas para poder crear las funcionalidades necesarias que se van a exponer en esta parte.

Figura 46: Logo Raiden

El nombre que se le asigno al principio al proyecto fue Raiden. Este nombre se ha utilizado en base a que es una plataforma de movilidad de vehículos por tanto es un nombre que encaja muy bien en este ámbito.

5.6.1.1 GESTIÓN DE USUARIOS

Nada más comenzar el proyecto se sugirió desplegar la simulación en una aplicación web por tanto se debía crear primero una lógica de acceso de usuarios a una página web. Para ello comencé a informarme primero que clase de *frameworks* me podían proporcionar una buena base para mi aplicación web y enfocarlo a la gestión de usuarios. Después de un tiempo de búsqueda y con el visto bueno de mis tutores procedí a iniciar el desarrollo del *backend* de la aplicación en Django.

Una vez que el *framework* del *backend* estaba decidido comencé a investigar cual era la forma adecuada de enfocar el desarrollo a partir de la plantilla de que nos ofrece Django

y cree mi primer proyecto en Django. Para ello primero instale un entorno virtual donde iba tener instaladas la mayoría de mis paquetes para tener problemas de versiones.

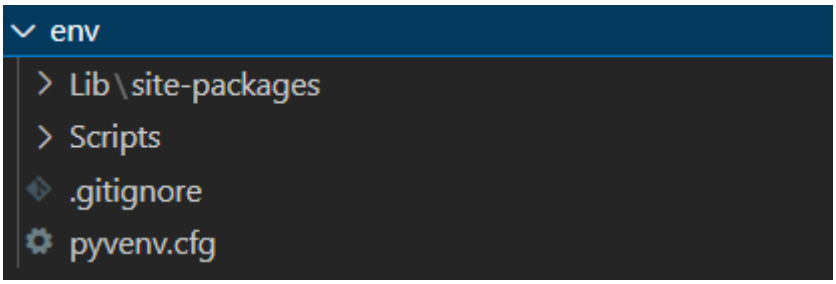


Figura 47: Virtual Environment

Una vez creado el proyecto en Django el cual llamé Raiden accedí a la documentación oficial de Django para informarme cual era el siguiente paso. Tras la creación del proyecto cree la aplicación de los usuarios que es donde se iba a alojar el código de la API de los usuarios que íbamos a desarrollar. La disposición de cada uno de los componentes es la siguiente y su uso oficial se encuentra con más detalle en el Anexo V del proyecto.

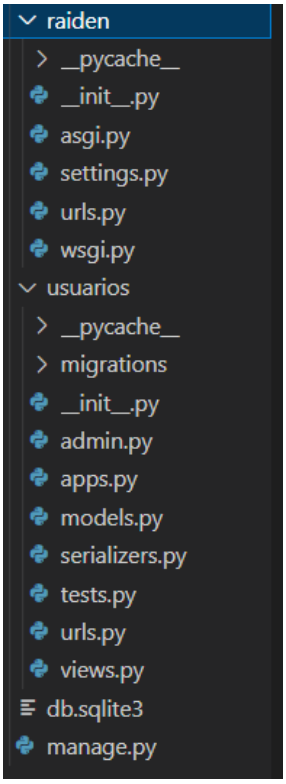


Figura 48: Proyecto Django

models.py

raiden > usuarios > models.py > ...

```python
import email
from locale import normalize
from typing_extensions import Self
from django.db import models
from django.contrib.auth.models import (AbstractBaseUser,
                                        PermissionsMixin,
                                        BaseUserManager)
from django.utils import timezone

class CustomUserManager(BaseUserManager):
    def _create_user(self, username , email , password , is_active , is_staff, is_superuser, **extra_fields):

        if not username:
            raise ValueError("Invalid user")
        email = self.normalize_email(email)
        user = self.model(username=username,email=email,is_active=is_active,is_staff=is_staff,is_superuser=is_superuser,**extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, username, email=None, password=None, **extra_fields):
        return self._create_user(username, email, password, False, False, ' ',
                                 **extra_fields)

    def create_superuser(self, username, email, password, **extra_fields):
        user = self._create_user(username, email, password, True, True,True,
                                 **extra_fields)
        user.is_active = True
        user.save(using=self._db)
        return user


class CustomUser(AbstractBaseUser,PermissionsMixin):
    username = models.CharField(max_length=30, unique=True)
    email = models.EmailField(max_length=250, unique=True)
    first_name = models.CharField(max_length=30, blank=True, null=True)
    last_name = models.CharField(max_length=30, blank=True, null=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)

    objects = CustomUserManager()

    USERNAME_FIELD = 'username'
    REQUIRED_FIELDS = ['email', ]

    def __str__(self):
        return self.email

```

Figura 49: Código models.py

La parte más importan de la aplicación de los usuarios será el archivo *models.py,* el cual va a estar escrito en Python y es donde se comenzó la definición de los campos que iban a formar a los usuarios. A pesar de que Django ofrecía un módulo por defecto para la creación de usuarios se optó por la creación de una clase personalizada a parte que usara el módulo de Django como referencia. Esto nos daba la ventaja de poder elegir los campos que fueran necesarios para la aplicación y añadir otros a nuestro gusto y no tener que seguir el modelo por defecto.

serializers.py

raiden > usuarios > serializers.py > UserCreateSerializer > Meta

```python
from django.contrib.auth.password_validation import validate_password
from django.core.exceptions import ValidationError

from rest_framework import serializers

from .models import CustomUser

class UserLoginSerializer(serializers.Serializer):

    username = serializers.CharField(required=True)
    password = serializers.CharField(required=True)


class UserSerializer(serializers.ModelSerializer):



    class Meta:
        model = CustomUser
        fields = ('pk',
                  'username',
                  'email',
                  'first_name',
                  'last_name')

class UserCreateSerializer(serializers.ModelSerializer):

    password = serializers.CharField(write_only=True)
    def create(self, validated_data):
        user = super(UserCreateSerializer, self).create(validated_data)
        user.set_password(validated_data['password'])
        user.is_active = True
        user.save()
        return user
    def validate_password(self, value):
        try:
            validate_password(value)
        except ValidationError as exc:
            raise serializers.ValidationError(str(exc))
        return value
    class Meta:
        model = CustomUser
        fields = ('pk',
                  'username',
                  'password',
                  'email',
                  'first_name',
                  'last_name')
```

Figura 50: Código serializers.py

El siguiente archivo importante del *backend* iba a ser el *serializers.py*. Al usar Django REST *framework* los serializadores iban a ser un parte importante del proyecto para tratar la información que llegaba desde el *frontend*. El uso de los serializadores y Django REST *framework* se explica con más detalle en el Anexo V.

views.py

raiden > usuarios > views.py > login

```python
from django.contrib.auth import authenticate, get_user_model

from rest_framework.decorators import api_view, permission_classes
from rest_framework.views import APIView
from rest_framework.permissions import AllowAny, IsAuthenticated
from rest_framework.authtoken.models import Token
from rest_framework.response import Response
from rest_framework import status, generics
from rest_framework.authtoken.models import Token

from .serializers import *

CustomUser = get_user_model()

@api_view(["POST"])
@permission_classes((AllowAny,))
def login(request):


    login_serializer = UserLoginSerializer(data=request.data)

    if not login_serializer.is_valid():
        return Response(login_serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    user = authenticate(
        username=login_serializer.data['username'],
        password=login_serializer.data['password']
    )

    if user and user.is_active is False:
        return Response({'detail': 'User is not active'},
                        status=status.HTTP_403_FORBIDDEN)

    if not user:
        return Response({'detail': 'Incorrect account login information'},
                        status=status.HTTP_404_NOT_FOUND)

    token, _ = Token.objects.get_or_create(user=user)
    #Se devuelven dos cosas la instancia como tal (token) y una instancia de si ha sido creada o no
    user_serialized = UserSerializer(user)


    return Response({
        'user': user_serialized.data,
        'token': token.key
    }, status=status.HTTP_200_OK)

    #El usuario no lo podemos devolver tal cual hay que serializarlo
class Registration(generics.CreateAPIView):

    serializer_class = UserCreateSerializer
    queryset = CustomUser.objects.all()

class Logout(APIView):

```

Figura 51: Código views.py

Finalmente, la última parte más importante del desarrollo del *backend* era la parte de *view.py.* En esta parte del código es donde se van a recoger las peticiones o *request* HTTP que se envíen a nuestras URIs desde el *frontend* y se va a devolver una respuesta o response.

Tras haber terminado el desarrollo de API REST de los usuarios era el momento que una fase de testeos en la que uso Postman para comprobar que todo funcionaba correctamente y los usuarios eran creado correctamente.

Cuando el *backend* de la aplicación funcionaba correctamente se procedió a elegir un *framework* para el *frontend* donde pudiera desarrollar con facilidad una interfaz y consumir la API de forma sencilla que creamos anteriormente. Para ello se decidió usar el *framework* de Vue integrado con Vuetify que nos iban a permitir crear una interfaz bonita de forma rápida.

Una vez el *framework* fue elegido se procedió acceder a la documentación oficial de Vue para comprender la creación de proyectos en vue y su posterior desarrollo en ellos. Ampliación en Anexo V del proyecto

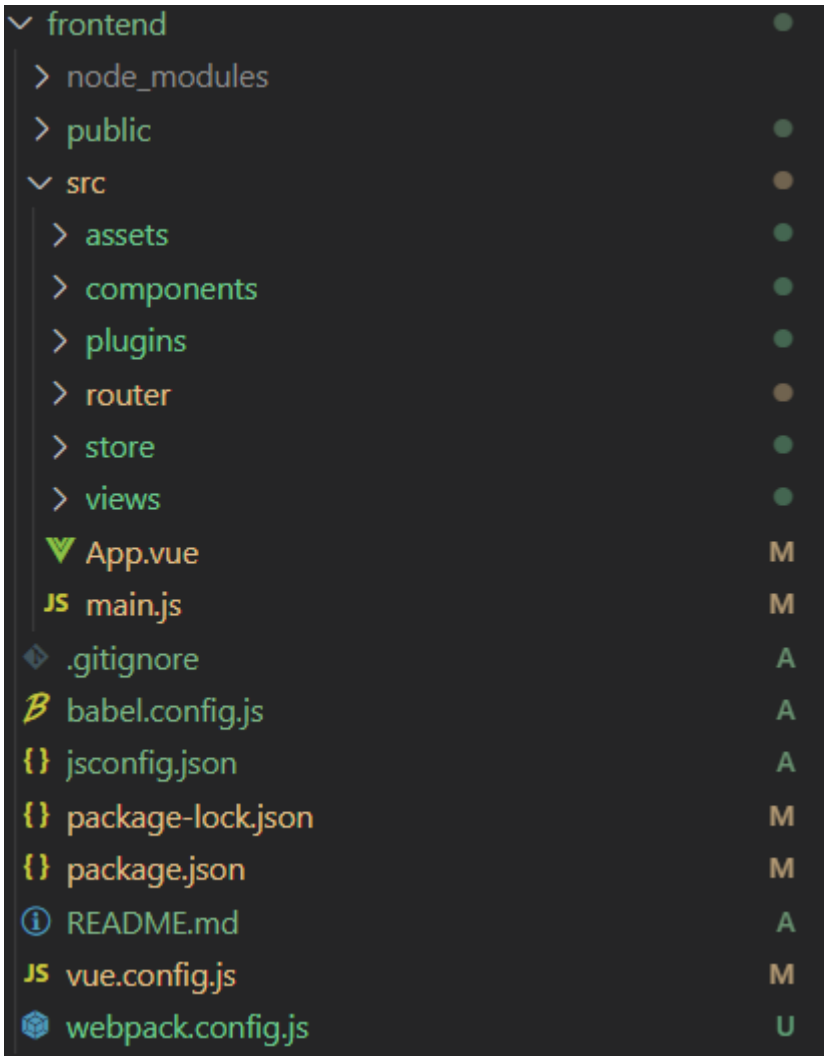
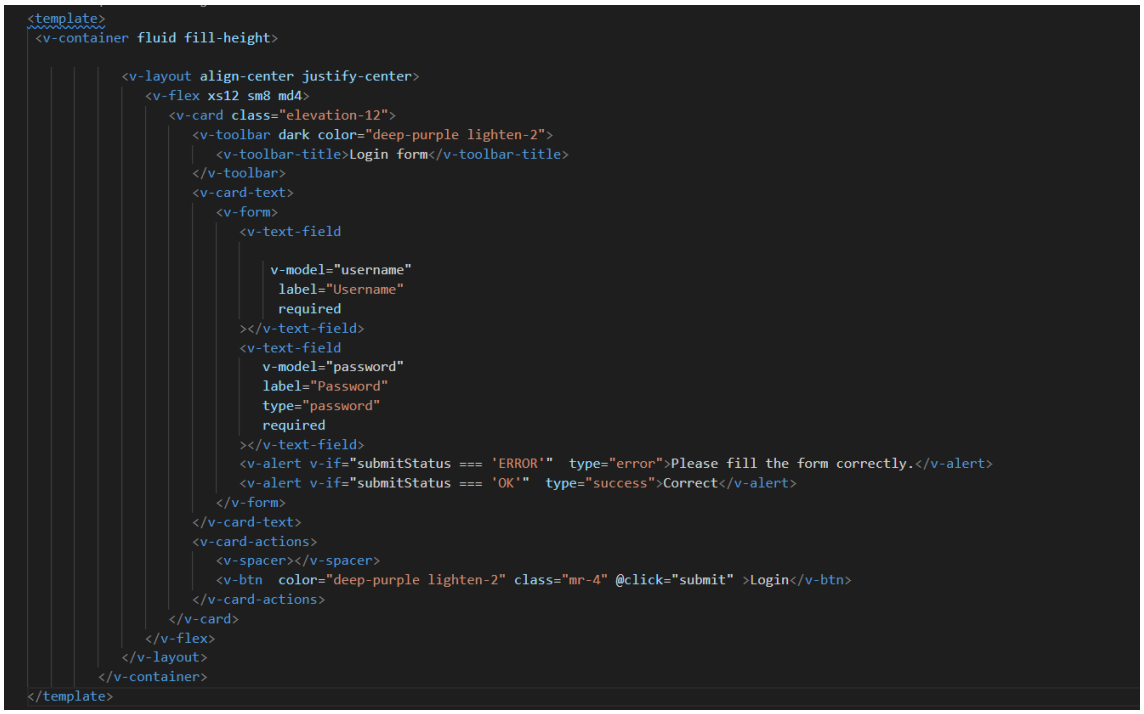


Figura 52: Proyecto Vue

Nuestro proyecto iba a necesitar distintos paquetes por tanto primero se tuvo que instalar vue-router y vuex que iban a soportar la navegación entre páginas y el almacenamiento de información necesaria para las sesiones respectivamente en cada uno.

Figura 53: Código Login.vue



```
<template>
  <v-container fluid fill-height>

          <v-layout align-center justify-center>
            <v-flex xs12 sm8 md4>
              <v-card class="elevation-12">
                <v-toolbar dark color="deep-purple lighten-2">
                  <v-toolbar-title>Login form</v-toolbar-title>
                </v-toolbar>
                <v-card-text>
                  <v-form>
                    <v-text-field

                      v-model="username"
                      label="Username"
                      required
                    ></v-text-field>
                    <v-text-field
                      v-model="password"
                      label="Password"
                      type="password"
                      required
                    ></v-text-field>
                    <v-alert v-if="submitStatus === 'ERROR'"  type="error">Please fill the form correctly.</v-alert>
                    <v-alert v-if="submitStatus === 'OK'"  type="success">Correct</v-alert>
                  </v-form>
                </v-card-text>
                <v-card-actions>
                  <v-spacer></v-spacer>
                  <v-btn  color="deep-purple lighten-2" class="mr-4" @click="submit" >Login</v-btn>
                </v-card-actions>
              </v-card>
            </v-flex>
          </v-layout>
        </v-container>
</template>
```

Figura 54: Código Login.vue

A continuación, cuando todo el *framework* estaba configurado correctamente se procedió a la implementación de las funcionalidades básicas para los usuarios como el Login, Registro,Logout,... Un ejemplo de la implementación *Login* donde el usuario dispondrá de un formulario creado con vuetify y que recogerá la información necesaria para iniciar sesión. Esta información iba a ser procesada en la parte de <script> y mediante Axios se iba comunicar con nuestra API de los usuarios (Anexo V del proyecto).

Las herramientas y módulos utilizados para la gestión de usuarios han sido utilizados son:

• Django(django-admin,django-user,django-rest,…)

• JavaScript

• Nodejs

• Npm

• Python

•Vue(vue-router,vuex,vue-loader,…)

•Vuetify

•Postman

5.6.2 SIMULACIÓN

La parte de simulación del proyecto ha sido realizada con la aplicación de AnyLogic. Esta elección se ibo después de una búsqueda bastante exhaustiva ya que la mayoría de plataformas no tenían las funcionalidades que yo buscaba.

Se encontraron cuatro principales entornos de simulación

- SUMO (Simulation of Urban MObility)
- AnyLogic
- MATSim
- PTV VISSIM

De todos ellos se acabó eligiendo AnyLogic ya que era el más completo de cara a utilización de eventos y agentes, aunque sí que supuso unas dificultades que se comentaran posteriormente en la parte de limitaciones.

5.6.2.1 GESTIÓN DE SIMULACIÓN

Para comenzar a construir la simulación primero tuve que acceder a la documentación de AnyLogic e informarme de forma contundente sobre la creación de simulaciones y la utilización de la principal librería que se iba a usar que era *Road Traffic Library.*

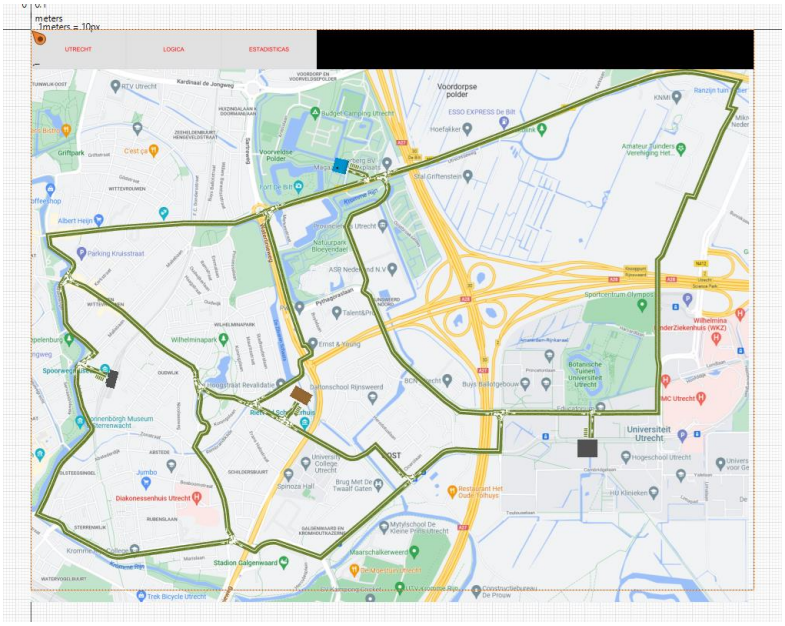


Figura 55: Vista Mapa simulación

Con los conocimientos suficientes se comenzó el desarrollo y la construcción de la simulación. Lo primero que se decidió fue elegir la ciudad en la que iba a tener lugar la simulación. Para ello se barajaron varias localizaciones de varios países, pero se terminó decidiendo por Países Bajos debido a su gran red de bicicletas. Esto supuso elegir una ciudad como Utrecht debido a que tenía una red de bicicletas muy bien definida y no excesivamente sobrecargada como Ámsterdam, por ejemplo.

Cuando el lugar estaba decidido se procedió a crear un mapa con la mayor calidad posible para ello se usó capturas desde Google Maps ya que era la única opción viable para conseguir una buena resolución (Se comentará posteriormente en limitaciones).

Finalmente, cuando el mapa ya estaba establecido se comenzó a crear la red de carretera mediante la librería que nos proporciona AnyLogic (Se amplía información en Anexo V) y la disposición de cuatro puntos distribuidos en el mapa de forma razonada para servir como partido para la creación de vehículos por parte del usuario al intentar realizar casos de estudio.

Figura 56: Vista Mapa simulación Museo-2

Figura 57: Vista Mapa simulación Museo-1

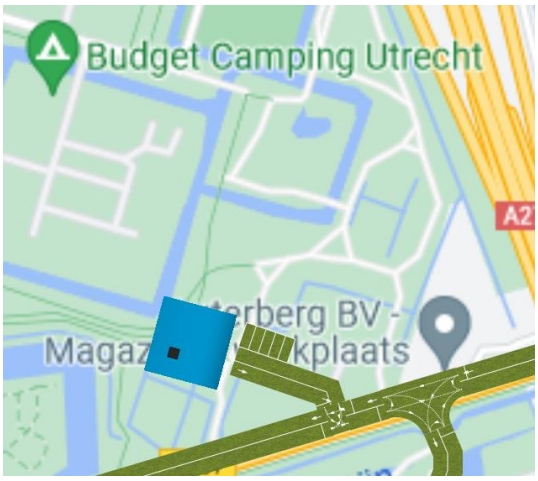


Figura 58: Vista Mapa simulación camping

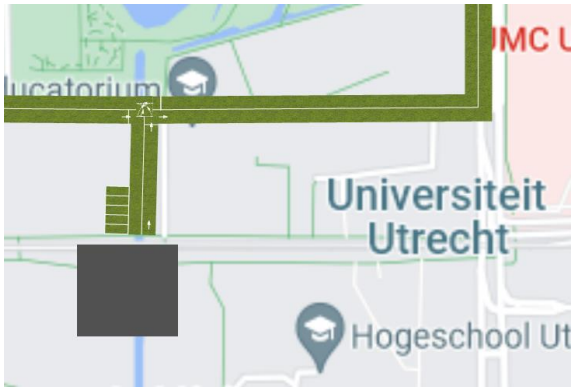


Figura 59: Vista Mapa simulación Universidad

Tras esto se procedió a crear la siguiente vista de la simulación que iban a ser la lógica de los vehículos, las estadísticas y los eventos.

5.6.2.2 GESTIÓN DE ESTADÍSTICAS

En la vista de las estadísticas se va a desplegar todo lo relacionado con la creación de vehículos por parte del usuario y además todo lo relacionado con los histogramas y los viajes realizados.

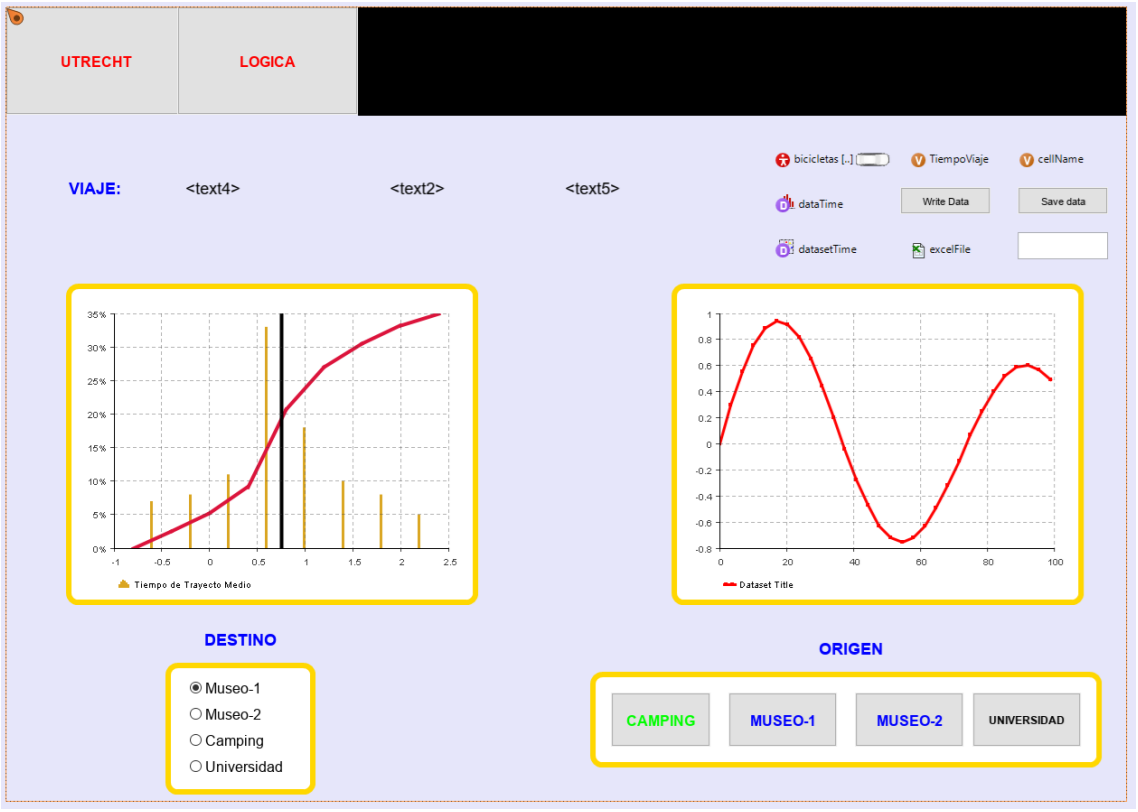


Figura 60: Vista Estadísticas

Cuando la vista del mapa estaba completa se procedió a crear unos botones que nos permitieran generar bicicletas en cualquiera de los cuatro puntos de desplegados en el mapa. Para ello se hacía una llamada al bloque *carSource* respectivo de cada punto (Ampliación Anexo V).

También se incorporó una serie de botones para poder elegir el destino de las bicicletas creadas por el usuario.

Finalmente, cuando una bicicleta llegaba a su destino actualizaba los *dataSet* y los histogramas asociados a los tiempos de viaje.

5.6.2.2 GESTIÓN DE EVENTOS

Finalmente, la última parte que se desarrolló en la simulación fue la gestión de eventos donde el usuario puede elegir de un total de cuatro eventos cada uno asociado a uno de los cuatro puntos de creación de bicicletas del mapa. (Ampliación Anexo V)

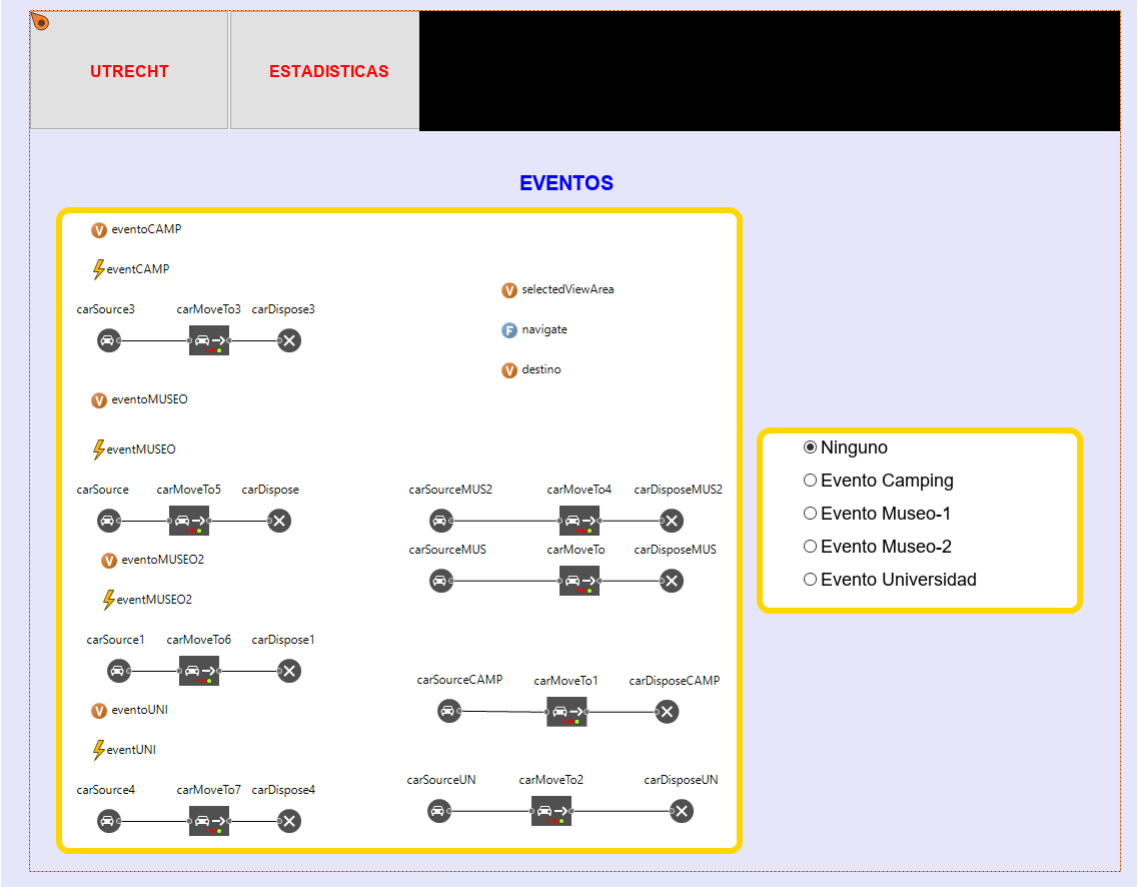


Figura 61: Vista Lógica

Cuando la parte de creación de vehículos y recogida de datos fue finalizada se pudo comenzar con la funcionalidad de los eventos para ello se usaron unos *triggers* que lo que hacen es llamar a la función *carSource* y en función del evento que este activo se comenzara a crear agentes en una zona determinada.

Esto repercutirá en la simulación ya que los agentes creados simularán aglomeraciones en distintos puntos de la ciudad por tanto el tiempo de viaje de las bicicletas creadas se verán afectados cuando su evento este activo.

Cuando la simulación estuvo completada se procedió su incorporación en la aplicación web mediante el servicio que nos ofrece AnyLogic Cloud para poder almacenar nuestros modelos y desplegarlos.

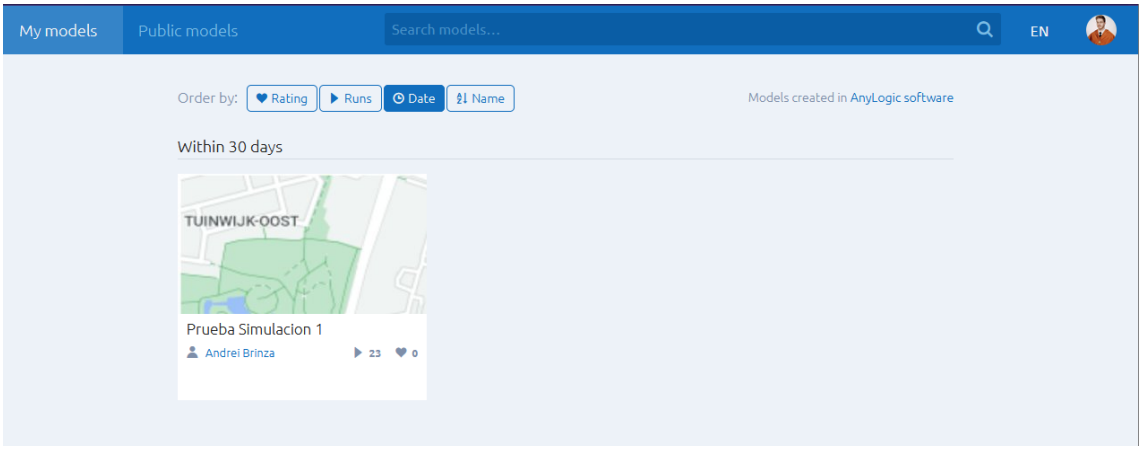


Figura 62: AnyLogic Cloud Project

Las herramientas y módulos utilizados para la gestión de usuarios han sido utilizados son:

- AnyLogic(Road Traffic Library, Statistic Analysis Library,Process Modeling Library)
- Java
- AnyLogicCloud

5.7 PRUEBAS

Una vez terminada la fase de implementación se realizaron varias pruebas. La primera fase de pruebas comenzó una terminada la API REST de Django. Una vez terminado el bloque de gestión de usuarios, mediante la herramienta Postman se comprobó que las funcionalidades de mi API tenían el comportamiento deseado y tras ello poder comenzar de forma óptima con el desarrollo del *frontend*.

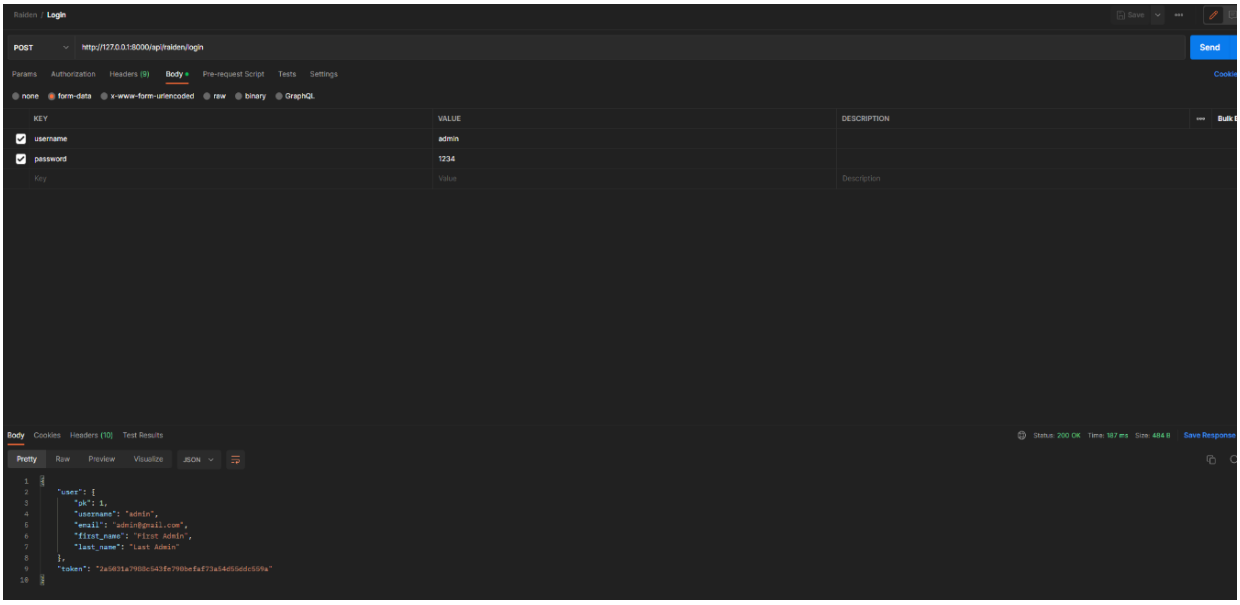


Figura 63: Login Postman Test

En esta imagen superior se realiza la prueba del correcto funcionamiento de nuestro *backend*. En este caso se prueba el *login*, donde si se han proporcionado las credenciales correctas se recibe un mensaje 200 OK con los datos esenciales del usuario y el token.

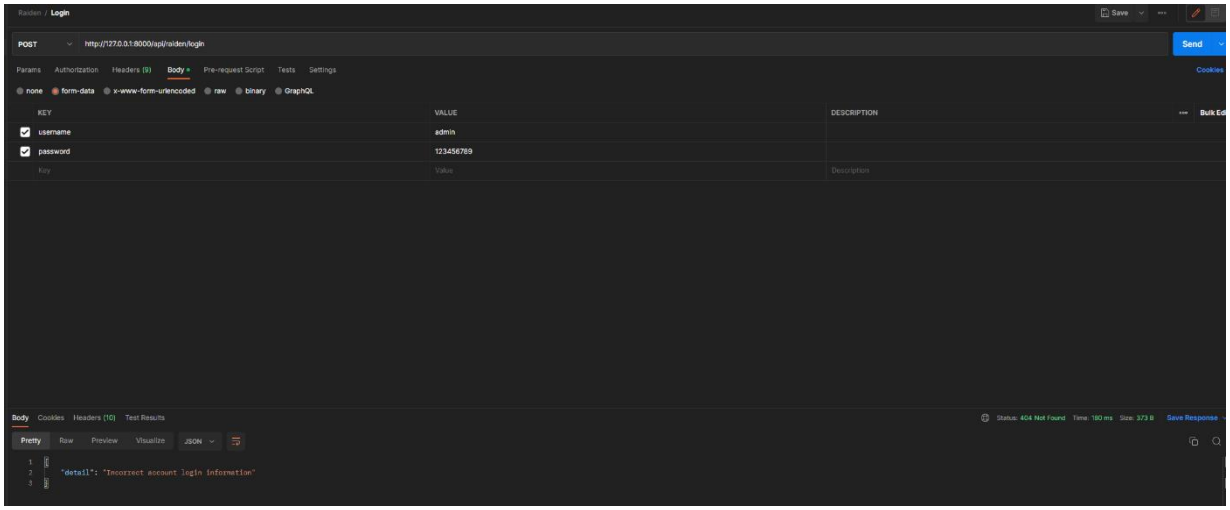


Figura 64: Login Postman Test Error 404

Si las credenciales no son correctas se recibe un mensaje de error 404 y una descripción del error.

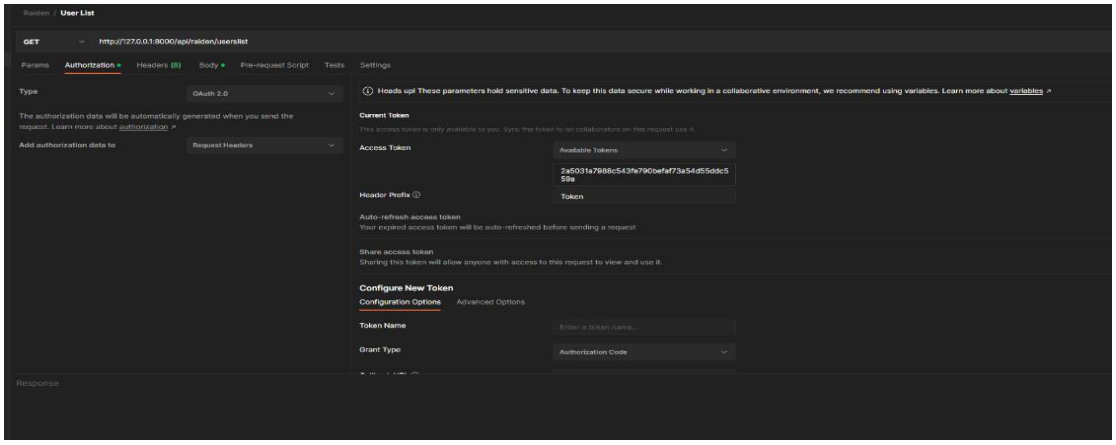


Figura 65: Configuración del Auth

Una vez obtenido el token tras el inicio de sesión, este es utilizado para el apartado de autorización y poder acceder a las otras funcionalidades.

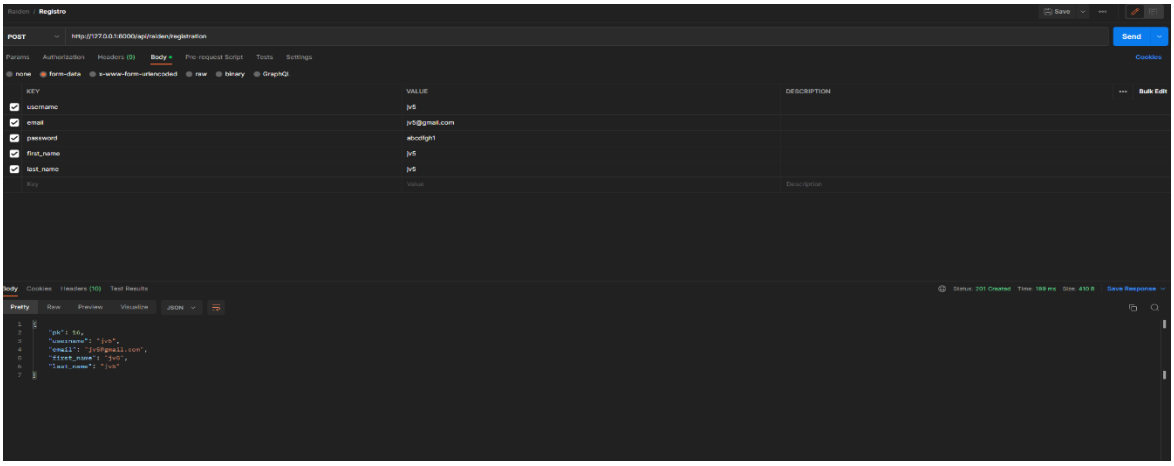


Figura 66: Registro Postman Test

En la imagen superior se realiza la prueba de la operación POST asociada al registro de usuarios. Si los datos introducidos son correctos se recibe un mensaje de confirmación 201 Created con los datos esenciales del usuario

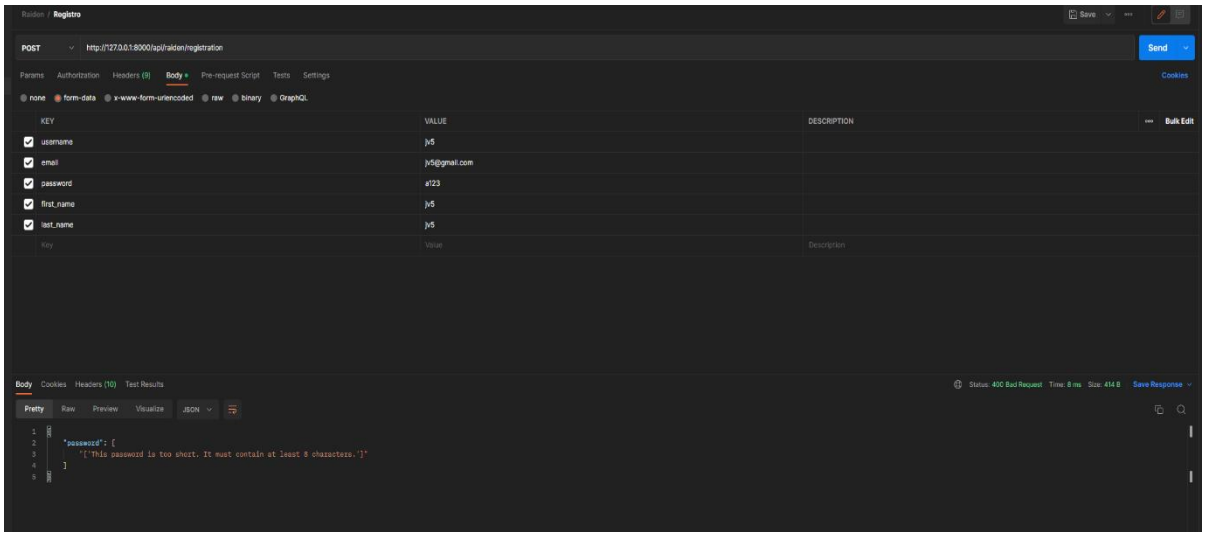


Figura 67: Registro Postman Test Error 400

Si los datos introducidos no son adecuados, se recibe un mensaje de error 400 Bad Request. En este caso la contraseña introducida no tiene el formato deseado y en la descripción se informa de ello.

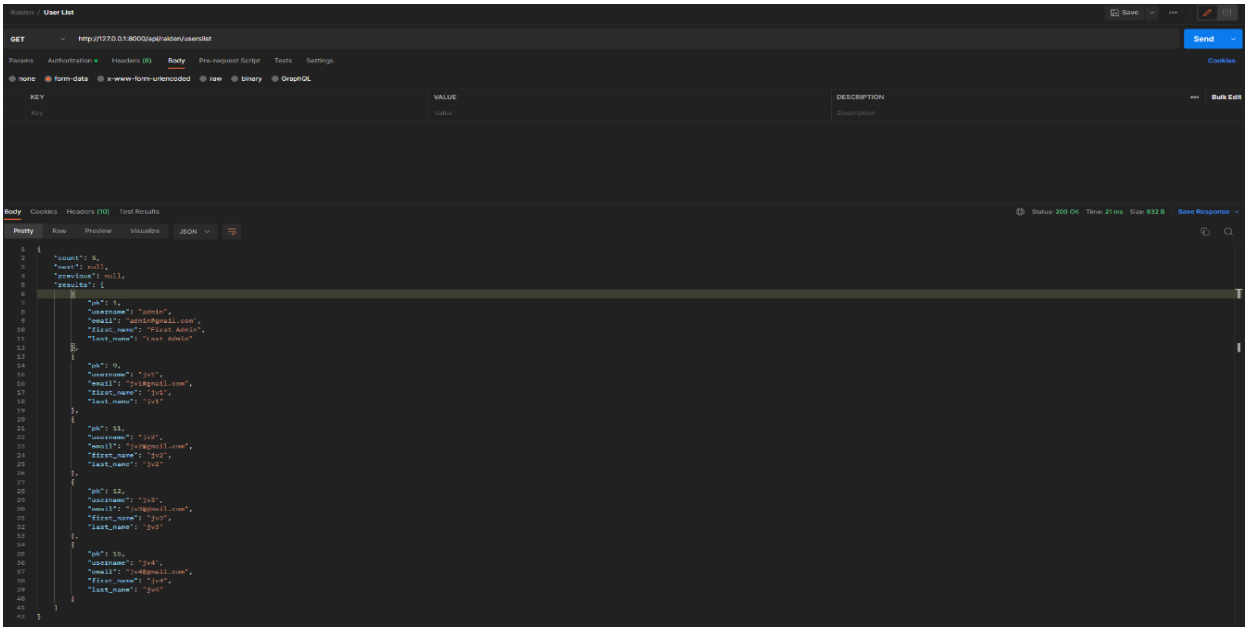


Figura 68: Listado de Usuarios Postman Test

En esta imagen se observa el resultado de la operación GET que obtiene un listado de los usuarios de la aplicación.

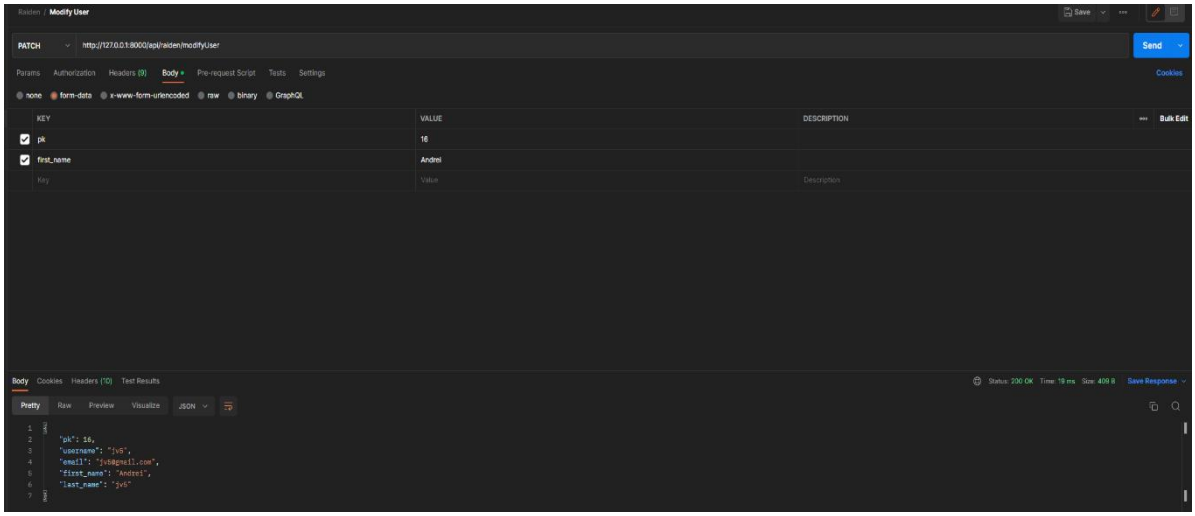


Figura 69: Modificación Postman Test

En esta imagen se realiza la prueba del método PATCH en la cual modificamos parcialmente datos del usuario. En la respuesta se recibe un mensaje con los datos esenciales del usuario.

La segunda fase pruebas comenzó con la realización del *frontend* donde se tuvieron que resolver bugs o errores que salieron al desarrollar las diferentes partes de la aplicación web y se realizaron distintas pruebas en las cuales se introducían datos de prueba para poder sacar errores que pudieran surgir de una mala codificación y usando el *debugger*. Al finalizar la parte de *frontend* se realizó la prueba de integración con el *backend* para comprobar que todo funcionaba según lo establecido.

La tercera parte de pruebas se realizó en el desarrollo de la simulación donde se depurarán los errores fueran surgiendo por consola. Cuando la simulación estuvo terminada se incorporó con la aplicación web y se realizaron varias pruebas de integración para comprobar que el conjunto de las partes nos proporcionaba el resultado deseado.

5.8 FUNCIONALIDAD DE LA APLICACIÓN

En este apartado se comentará las funcionalidades de la aplicación para mayor ampliación se puede acceder al Anexo VI Manual de Usuario del proyecto.

5.8.1 GESTIÓN DE USUARIOS

El sistema nos va a permitir las funcionalidades básicas de un usuario altas, bajas y modificaciones. Además, si el usuario es un administrador podrá realizar funciones adicionales como ver el panel de admin observar una lista de usuarios.

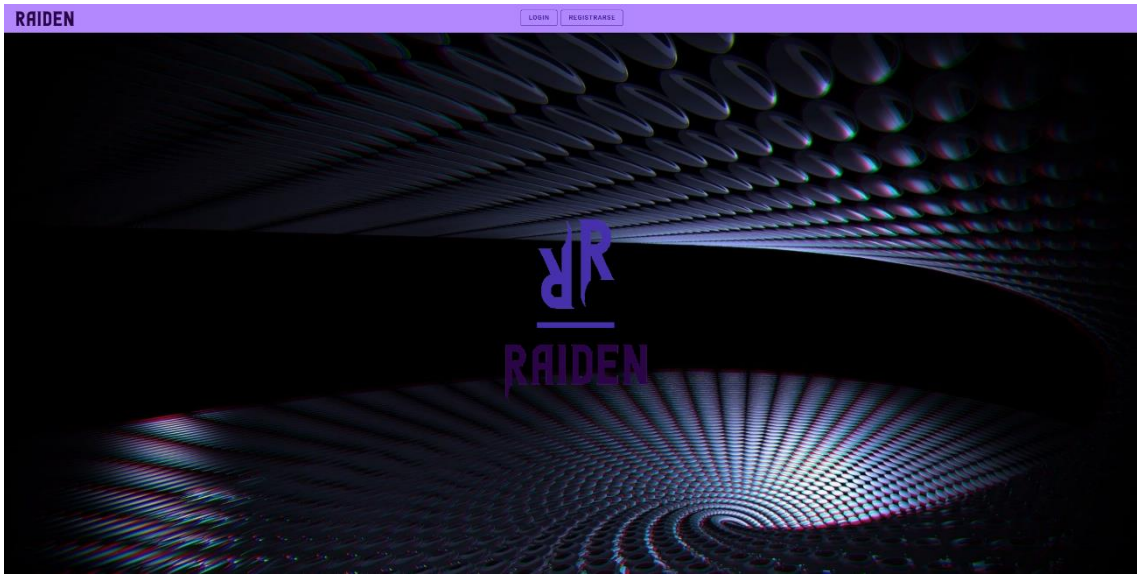


Figura 70: Inicio Raiden

Nada más entrar a nuestra aplicación nos vamos a encontrar con la página de inicio. Para poder acceder a las funcionalidades debemos tener un usuario y una contraseña en otro caso debemos registrarnos.

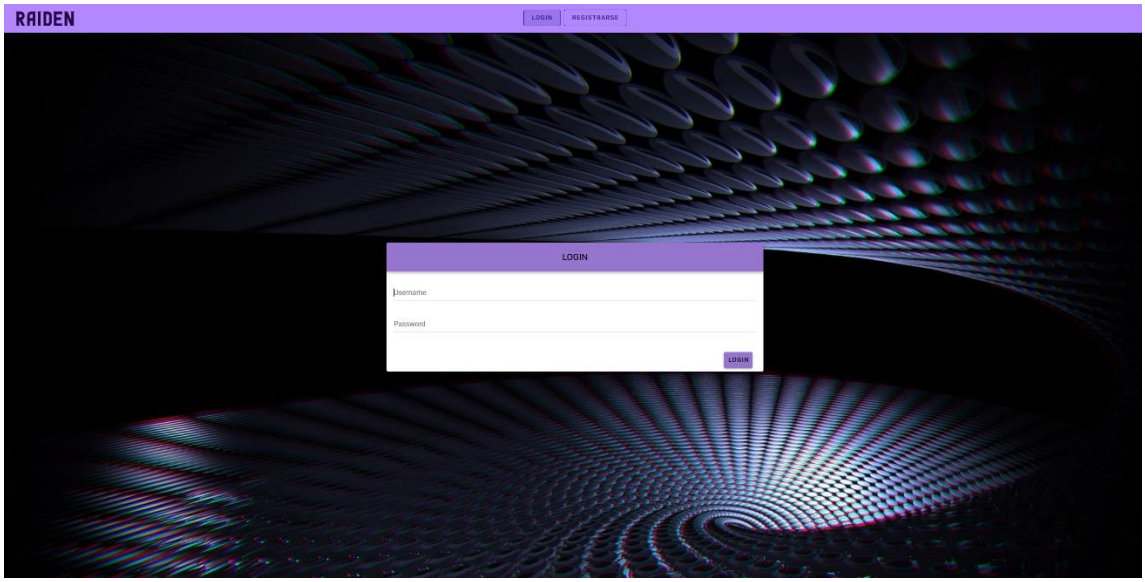


Figura 71: Inicio de Sesión

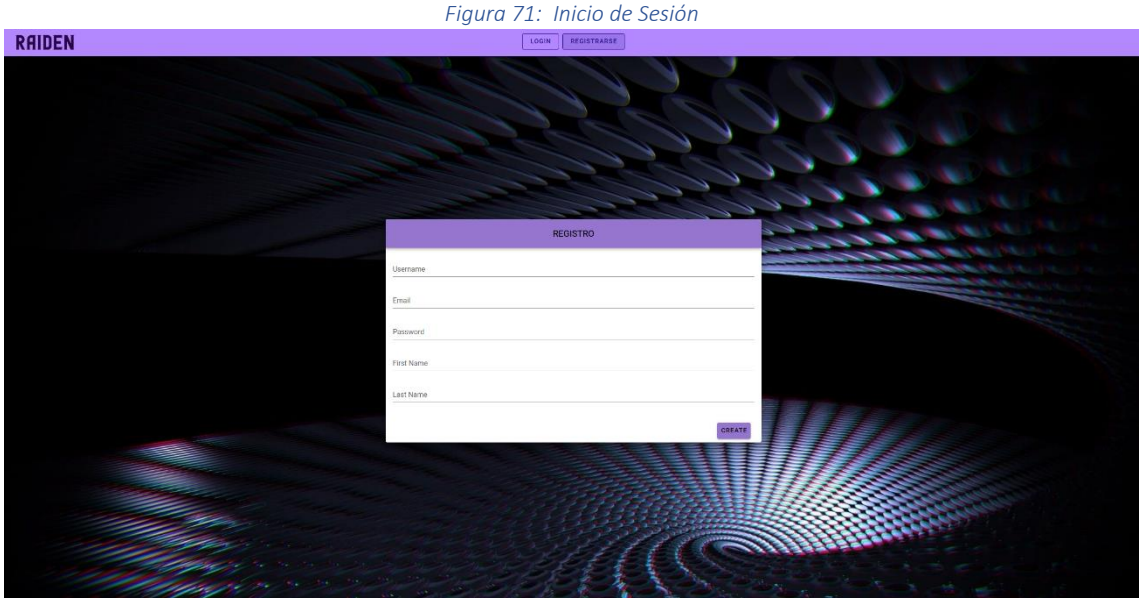


Figura 72: Registro

Si intentamos iniciar sesión con datos incorrectos nos saldrá el siguiente panel

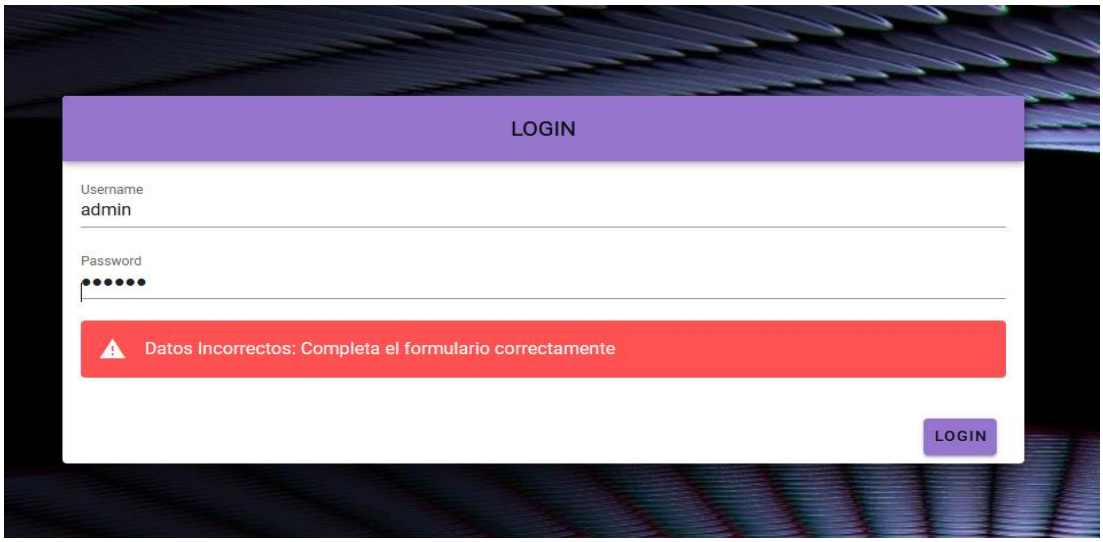


Figura 73: Inicio de Sesión Erroneo

Una vez iniciada la sesión con un usuario y contraseña correctos el usuario volverá al inicio, pero esta vez con acceso a las funcionalidades y el menú de navegación actualizado

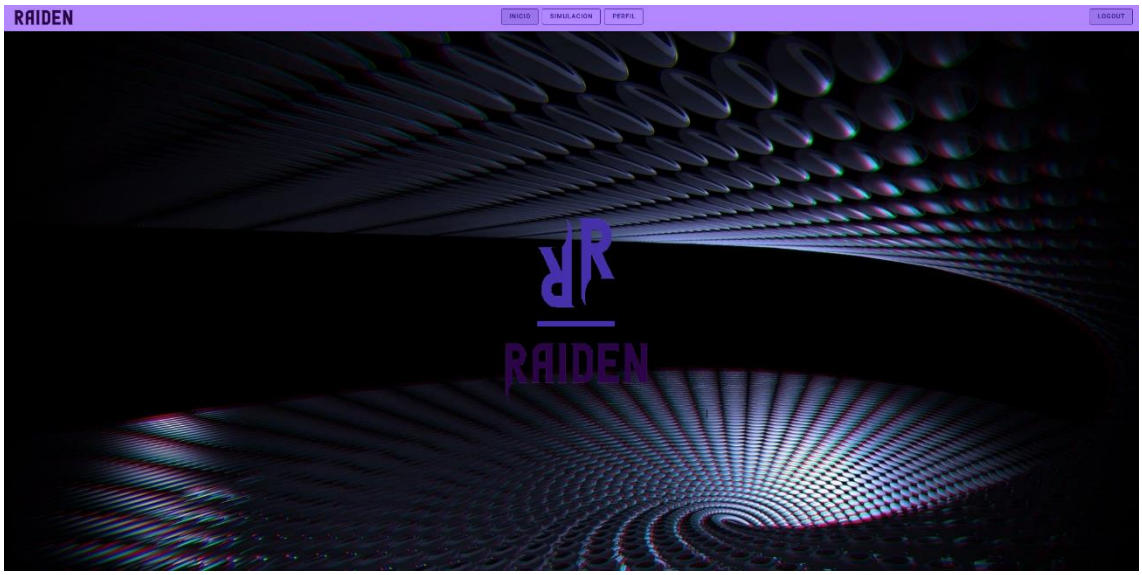


Figura 74: Inicio (El usuario ha iniciado sesión)

Podremos acceder a la simulación, perfil o cerrar sesión.

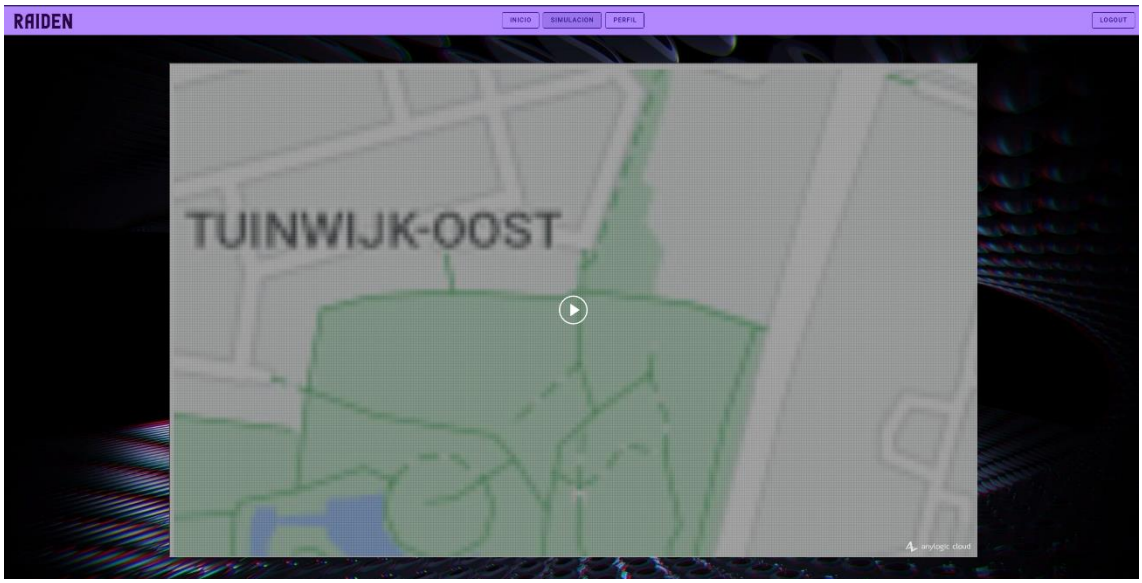


Figura 75: Simulación

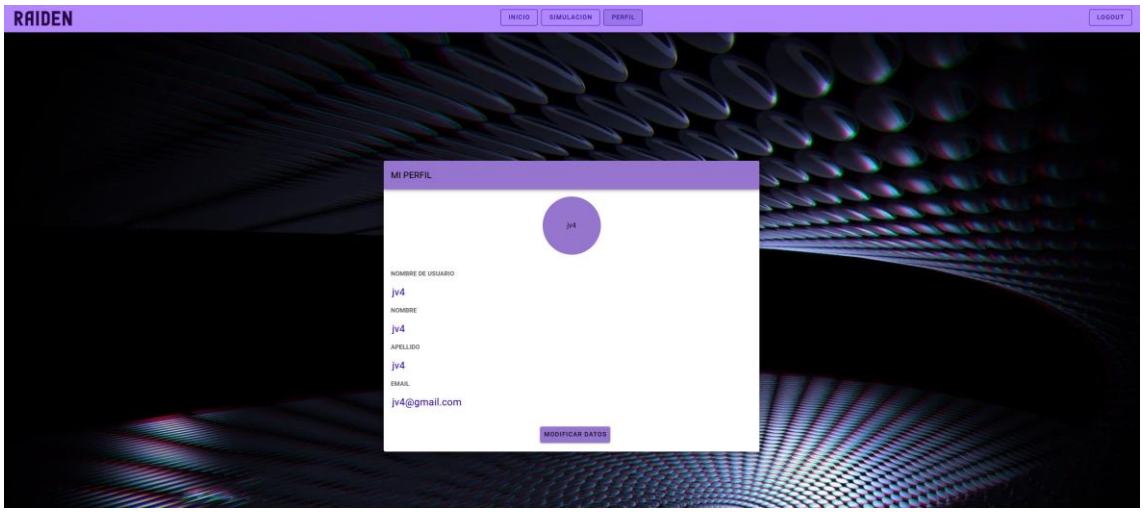


Figura 76: Perfil usuario

Si iniciamos sesión como administradores (admin). El sistema nos va a permitir ver el botón para acceder al panel de administrador, pero si iniciamos sesión en una cuenta normal se nos permite las acciones normales.

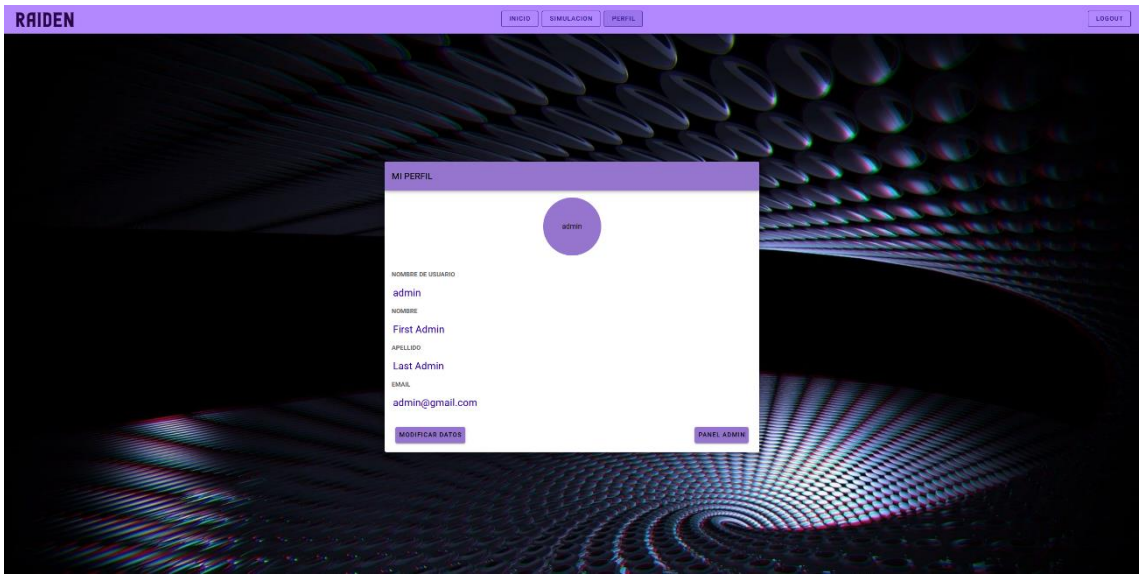


Figura 77: Perfil Admin

Si accedemos al panel de administrador se nos mostrara una lista de usuarios del sistema. Desde aquí el administrador podrá crear, eliminar o modificar información de los usuarios

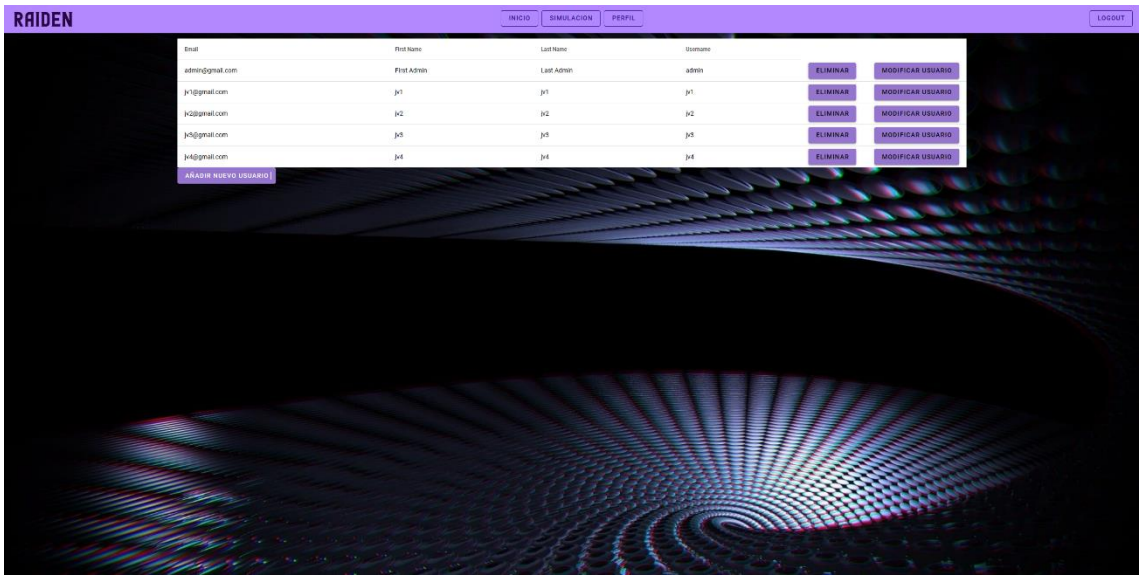


Figura 78: Lista Usuarios Admin

Finalmente, cuando queramos salir de la aplicación el usuario simplemente debe pulsar el cierre de sesión para volver a la página de inicio.

5.8.2 GESTIÓN DE SIMULACIÓN

El usuario una vez la sesión está iniciada en el sistema podrá comenzar la simulación y poner a prueba casos de uso. Se le mostrara la visión general del mapa y una barra de navegación con las opciones Estadísticas, Lógica y Utrecht.

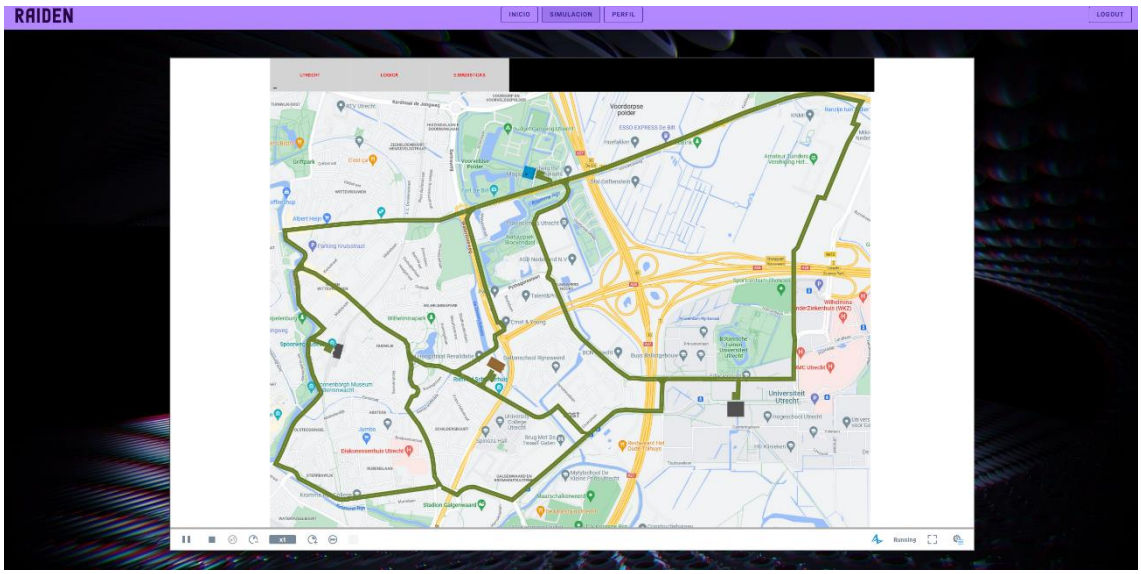


Figura 79: Vista simulación

Para comenzar a crear vehículos el usuario deberá dirigirse a la parte superior izquierda y en la barra de navegación de la simulación seleccionar la opción Estadísticas.

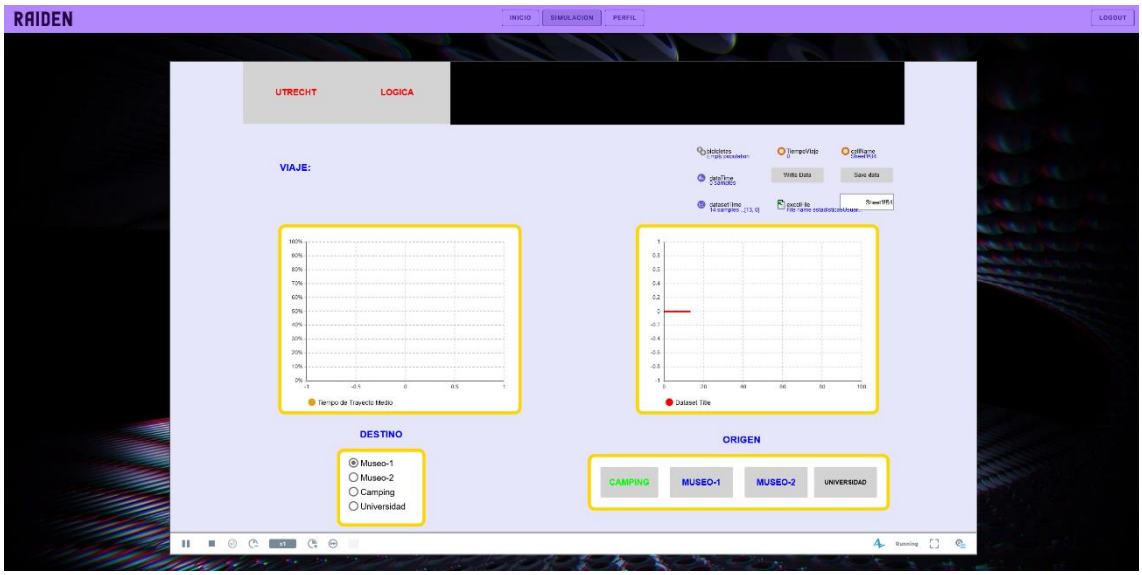


Figura 80: Vista Estadísticas

A continuación, para crear una bicicleta debemos elegir primero un destino de los cuatro posibles que hay (Museo-1, Museo-2, Camping, Universidad) y a continuación el lugar donde comenzara el viaje del vehículo (Museo-1, Museo-2, Camping, Universidad). Cuando la bicicleta se cree aparecerá en el contador de agentes bicicleta.

La bicicleta será creada con rumbo al destino elegido

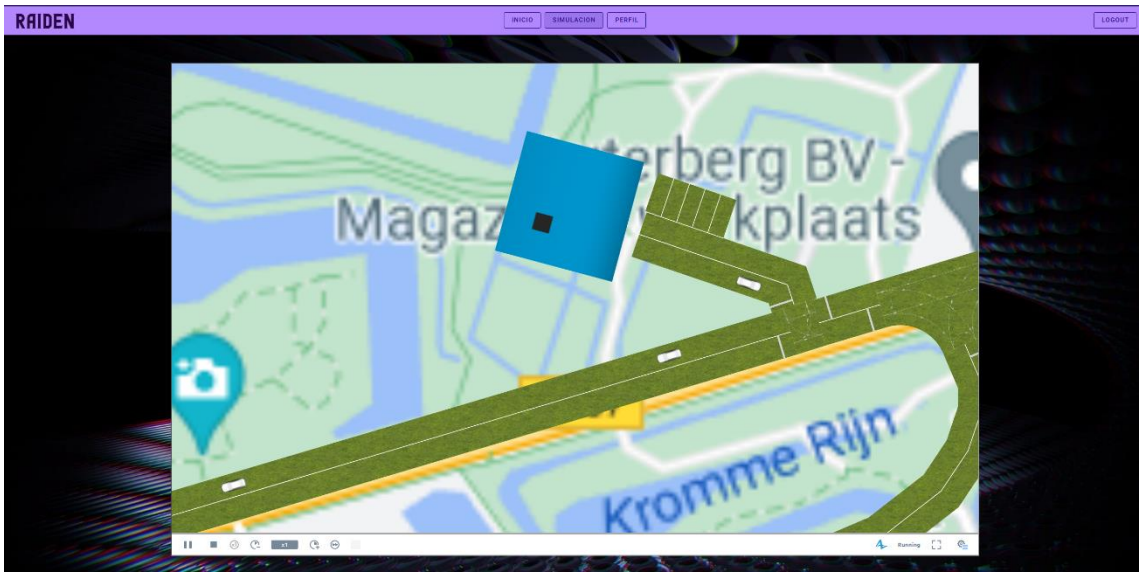


Figura 81: Bicicleta creada

5.8.3 GESTIÓN DE ESTADÍSTICAS

Dentro de la simulación tendremos la vista de las estadísticas, cuando un vehículo llegue a su destino se generarán una serie de datos como el tiempo e instante de llegada y la media de los tiempos de llegada, que el usuario podrá ver en tiempo real dentro de la simulación.

Figura 82: Estadísticas de caso de estudio

Figura 83: Estadísticas de varios casos de estudio

5.8.4 GESTIÓN DE EVENTOS

En la parte de eventos de la simulación el usuario podrá activar eventos en los puntos de la ciudad (Museo-1, Museo-2, Camping, Universidad).

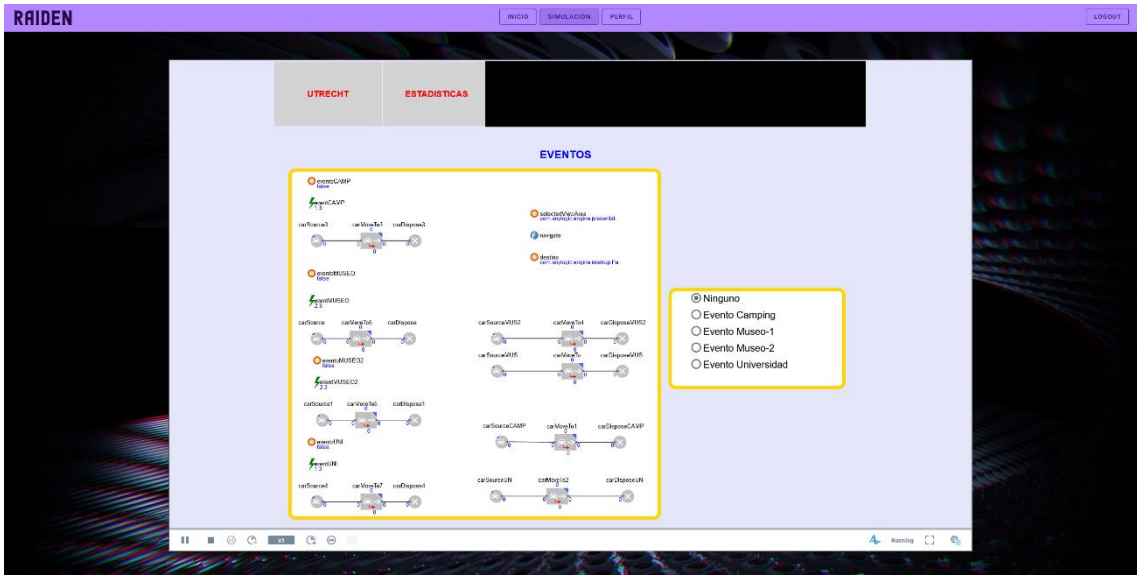


Figura 84: Eventos Simulación

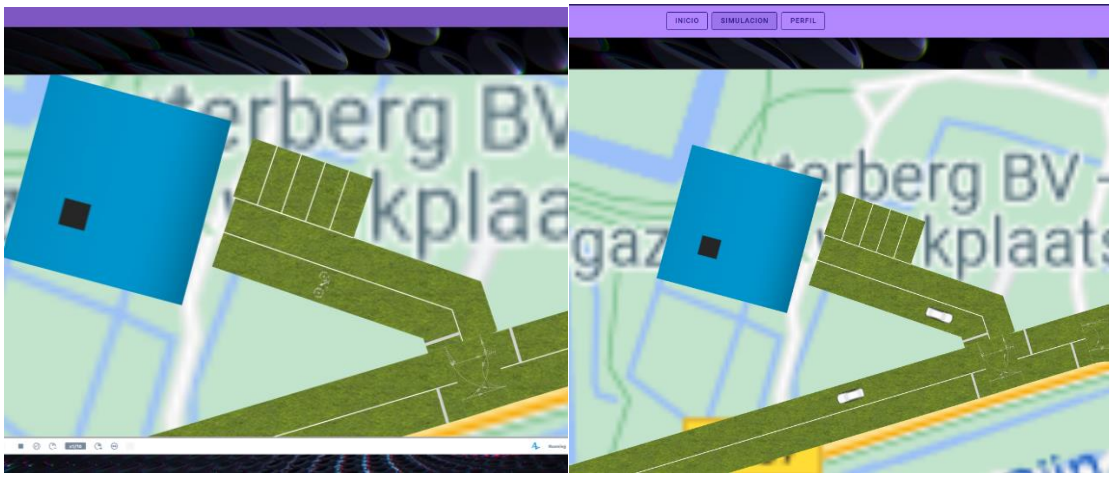
5.9 CAMBIO DE INTERFAZ

La aplicación inicialmente contaba con una interfaz sencilla y con una distribución de los elementos de navegación un poco aleatoria. Además, los colores elegidos en algunos elementos como títulos no creaban demasiado contraste.

Por ello se tomó la decisión de hacer algunos cambios en la interfaz de la aplicación. Primero se hizo un pequeño análisis sobre cómo se podría mejorar la disposición de los elementos, llegando a la conclusión de que por ejemplo los botones de la barra de navegación deberían estar más centrados para una mejor visibilidad y acceso. También se tomó la decisión de cambiar algunos colores para crear un mayor contraste.

Otro de los cambios llevado a cabo fue la creación de unos formularios más claros y sencillos con menos elementos innecesarios y con un estilo más sobrio.

Por último, a parte de la interfaz de la aplicación web se decidió hacer cambios también en la interfaz de la simulación de AnyLogic. Siguiendo la misma lógica se buscó crear más contraste para que los elementos fueran más visibles. Se cambiaron colores y se cambió también la representación de los vehículos ya que la representación original exigía acercar demasiado la visión y no se podía tener un visón general.



6.CONCLUSIONES Y LINEAS DE TRABAJO FUTURAS

6.1 CONCLUSIONES

Se puede concluir que se ha desarrollado una aplicación Raiden usando herramientas de simulación y desarrollo web para proporcionar una plataforma de simulación de movilidad para vehículos ligeros, la cual proporciona la funcionalidad de poder crear casos de estudio demandando distintos vehículos en varios puntos de la ciudad para ver como nuestro vehículo es desplegado en el mapa hacia su destino. Esto nos puede ser útil para observar tiempos, recorridos y tendencias de uso de los vehículos en un entorno urbano y que podría aplicarse en ciudades con un plan de transporte inteligente.

Además, se pueden accionar distintos tipos de eventos que nos van a permitir realizar variaciones en los casos de estudio. También se pueden recoger las gráficas y ver los resultados de nuestras simulaciones en tiempo real.

Dentro de los aspectos más relevantes del desarrollo cabe destacar la creación del backend y la gestión de usuarios en el framework de Django mediante las bibliotecas que nos proporciona, así como y el uso de Postman para realizar pruebas. Por otro lado, se ha llevado a cabo la implementación del frontend con el framework Vue y se han realizado pruebas en Visual Studio Code. Y por último, se ha realizado la creación de la simulación en AnyLogic y las pruebas realizadas en el entorno de desarrollo.

También se cumplió el objetivo personal de conseguir llevar a cabo un proyecto desde cero siguiendo un procedimiento estudiado con anterioridad en la carrera en las asignaturas dedicadas a la gestión de software y aprender a usar nuevas tecnologías como Django, AnyLogic o Vue, que han conseguido darme mucha inspiración.

6.2 LIMITACIONES ENCONTRADAS

Dentro de la realización del proyecto ha habido varias limitaciones que han entorpecido de alguna forma la realización del proyecto.

La primera limitación que hemos encontrado es que a la hora de desplegar el mapa AnyLogic no me permitía una escala con la suficiente resolución para que se pudieran ver los elementos, para eso se tuvo que buscar otra estrategia con el fin de integrar el mapa dentro de la simulación con una resolución mucho mayor y un tamaño considerable para poder dibujar todas las carreteras y puntos de la simulación.

Otra de las limitaciones encontradas ha sido que a pesar de que AnyLogic es una excelente herramienta de simulación, no permite simulaciones de más de una hora y no podrás tener la simulación activa más de un determinado periodo de tiempo. Además, los servidores de AnyLogic Cloud se caen constantemente lo que simulación a veces no está disponible hasta que se recargue la página varias veces.

6.3 LINEAS DE TRABAJO FUTURAS

En este apartado vamos a exponer algunas de las soluciones a las limitaciones encontradas en el proyecto y alguna propuesta de desarrollo o mejora en el futuro del software.

Para las limitaciones expuestas anteriormente una posible solución sería adquirir una licencia de uso mensual de Private AnyLogic Cloud, lo cual nos permitirá tener mayor capacidad de simulación con un horario mucho más extendido y sin ninguna restricción de tiempo.

En el futuro de la aplicación sería interesante incorporar nuevas formas de eventos como, por ejemplo, que tengan en cuenta la meteorología o el relieve del mapa y los agentes reaccionen de forma adecuada en tiempo real. Estas incorporaciones pueden llegar a ser algo complejas si se intentan introducir dentro de la simulación en tiempo real, ya que se tendría que comprobar en todo momento el estado de cada vehículo respecto al relieve del mapa o el tiempo meteorológico.

Además, sería útil incorporar a futuros varias ubicaciones o una forma de tener un mapa con mucha mayor escala para poder desplegar de forma óptima.

7.REFERENCIAS

Edix, R. (2022, 26 julio). Framework: qué es, para qué sirve y algunos ejemplos. Edix España. Recuperado 6 de septiembre de 2022, de https://www.edix.com/es/instituto/framework/

Guzman, H. C. (s. f.). API con Django Rest Framework | Django | Academia | Hektor Profe. Recuperado 6 de septiembre de 2022, de https://docs.hektorprofe.net/academia/django/api-rest-framework/

Qué es el lenguaje unificado de modelado (UML). (s. f.). Lucidchart. Recuperado 6 de septiembre de 2022, de https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml

EcuRed. (s. f.). Visual Paradigm - EcuRed. Recuperado 6 de septiembre de 2022, de https://www.ecured.cu/Visual_Paradigm

Departamento de Lenguajes y Sistemas Informáticos. Actividad docente. (s. f.). Recuperado 6 de septiembre de 2022, de http://www.lsi.us.es/docencia/docencia.php

Django documentation | Django documentation | Django. (s. f.). Recuperado 6 de septiembre de 2022, de https://docs.djangoproject.com/en/4.1/

Introduction | Vue.js. (s. f.). Recuperado 6 de septiembre de 2022, de https://vuejs.org/guide/introduction.html

Quick Start — Vuetify.js. (s. f.). Recuperado 6 de septiembre de 2022, de https://v15.vuetifyjs.com/es-MX/getting-started/quick-start/

The Model-View-ViewModel Pattern - Xamarin. (2021, 8 julio). Microsoft Docs. Recuperado 6 de septiembre de 2022, de https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

García Peñalvo, F. J., García Holgado, A., & Vázquez Ingelmo, A. (s.f.). Proceso unificado. Obtenido de https://repositorio.grial.eu/bitstream/grial/2470/1/7.%20PU-2021.pdf

García Peñalvo, F., & García Holgado, A. (s.f.). Ingeniería de requisitos. Obtenido de https://repositorio.grial.eu/bitstream/grial/1143/1/IS_I%20Tema%204%20-%20Ingenieria%20de%20Requisitos.pdf

García Peñalvo, F., García Holgado, A., & Vázquez Ingelmo, A. (s.f.). Flujos de trabajo del proceso unificado. Obtenido de https://repositorio.grial.eu/bitstream/grial/1945/1/IS_I%20Tema%206%20-%20Flujos%20de%20trabajo%20del%20Proceso%20Unificado.pdf

García Peñalvo, F., García Holgado, A., & Vázquez Ingelmo, A. (s.f.). Fundamentos de la vista de casos de uso. Obtenido de https://repositorio.grial.eu/bitstream/grial/1950/1/UML%20-%20Casos%20de%20uso-2020.pdf

García Peñalvo, F., Moreno García, M., García Izquierdo, A., & Vázquez Ingelmo, A. (s.f.). UML. Unified Modeling Language. Obtenido de https://repositorio.grial.eu/bitstream/grial/1949/1/IS_I%20Tema%208%20-%20UML.pdf