Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Desarrollo de un simulador de escenarios con usuarios móviles para la evaluación de algoritmos de recomendaciones

---

Autor
Slavcho Georgiev Ivanov

Director
Sergio Ilarri Artigas

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Escuela de Ingeniería y Arquitectura
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática e Ingeniería de Sistemas

—

Zaragoza, 23 de marzo de 2016

Desarrollo de un simulador de escenarios con usuarios móviles para la evaluación de algoritmos de recomendaciones

Resumen

Los sistemas de recomendación proporcionan sugerencias acerca de elementos que pueden resultar de interés para el usuario (hoteles, restaurantes, libros, películas, etc.). En este TFG se pretende desarrollar un simulador de escenarios con usuarios móviles (mapas de ciudades con objetos móviles y estáticos) que permita la evaluación de algoritmos de recomendación.

Agradecimientos

Me gustaría agradecer este Trabajo Fin de Grado a todas las personas que lo han echo posible con su apoyo y dedicación.

En su primer lugar a mi director Sergio Ilarri por la oportunidad que me ha dado para realizar este proyecto, su paciencia y ayuda, sin la cual este proyecto no hubiera sido posible. A mis compañeros y amigos de clase, con los que he compartido estos años de carrera, por hacer que los momentos de estudio y prácticas fuesen agradables y amenos. A mi familia y amigos más cercanos, por su paciencia y por motivarme para seguir adelante en los momentos más complicados.

Y por supuesto a la Universidad de Zaragoza y a todos aquellos profesores de lo que he aprendido tanto a los largo de estos años.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este capítulo se mostrará la motivación existente para la realización de este Trabajo Fin de Grado, los objetivos que han sido marcados por el proyecto, las librerías y herramientas utilizadas para su elaboración, el modelo de trabajo seleccionado y también se analizará el contexto tecnológico. Finalmente se mostrará la estructura seguida en este documento.

1.1. Motivación del proyecto

Han sido varias las razones que me llevaron a elegir desarrollar este Trabajo Fin de Grado. La primera y principal ha sido el interés personal en los sistemas de recomendaciones y su amplia aplicación en sistemas comerciales y la web. Por otro lado, realizar un proyecto complejo, partiendo desde cero y sin tener ningún conocimiento particular de este ámbito, suponía un gran reto que deseaba afrontar porque me permitiría ampliar mis conocimientos en campos diversos como Ingeniería del Software, Arquitecturas de Software etc., de las que poseía unos conocimientos limitados. Además, consideré que la experiencia y conocimientos que adquiriría en este proyecto aumentarían mis posibilidades de desarrollar mi carrera profesional en este ámbito.

1.2. Objetivos

El Trabajo Fin de Grado que se describe en este documento tiene los siguientes objetivos:

- Desarrollar un simulador de escenarios con usuarios móviles, que cuente con mapas de ciudades con objetos móviles y estáticos.

- Desarrollar lo necesario para que se permita que los mapas de ciudades sean reales de tal forma que estos sean obtenidos de un sistema que proporcione mapas.
- Desarrollar lo necesario para que los usuarios puedan crear, editar, borrar y configurar los mapas y escenarios del simulador.
- Desarrollar lo necesario para que todas las configuraciones de mapas y escenarios sean parametrizables desde la interfaz de usuario.
- Desarrollar lo necesario para que la simulación de escenas funcione en tiempo real de tal forma que los eventos de una escena se reflejen en los dispositivos de los usuarios conectados al mismo mapa y escena.
- Desarrollar una interfaz que permita la integración con un recomendador externo de tal forma que exista una comunicación bidireccional basada en eventos entre el simulador y el recomendador.

Además de los objetivos marcados por la propuesta del Trabajo Fin de Grado, también se han tenido en cuenta como objetivos lograr que el simulador utilice los recursos hardware mínimos, permitir que este sea fácilmente escalable y que pueda desplegarse en un entorno distribuido. De esta forma logramos ahorrar costes de infraestructura y futuros desarrollos.

1.3. Contexto Tecnológico

Habitualmente los sistemas de recomendación se evalúan con conjuntos de datos estáticos clásicos. El inconveniente que esto conlleva es que si ejecutamos el mismo algoritmo sobre un conjunto de datos estáticos varias veces siempre obtenemos el mismo resultado. Con el fin de evitar este inconveniente surge la necesidad de evaluar los algoritmos de recomendaciones con conjuntos de datos dinámicos.

Una de las formas de recolectar y proveer conjuntos de datos dinámicos es mediante el crowdsourcing. El crowdsourcing, también conocido como colaboración abierta distribuida, consiste en delegar tareas a un grupo de personas o comunidad a través de una convocatoria abierta.

Así que a medida que el grupo de personas o comunidad vayan realizando las tareas asignadas el sistema va recolectado y suministrando datos sobre las acciones que realizan estos para llevar a cabo sus tareas. Esto nos da la oportunidad para suministrar datos a los algoritmos de recomendaciones en tiempo real, es decir, suministrar datos a medida que el grupo de personas vayan cumpliendo con sus tareas.

1.4. Herramientas utilizadas

En esta sección se listan las tecnologías, liberarías externas y herramientas utilizadas para el desarrollo del proyecto acompañada de una breve descripción.

Librerías usadas

Para el desarrollo del proyecto se ha hecho uso de diversas librerías externas que han permitido la implementación en un tiempo razonable de ciertas funciones necesarias que no formaban parte de los objetivos del proyecto:

- **Openlayers**: framework de OpenStreetMap que nos permite el uso libre de mapas.
- **Node.js v0.12.4**: entorno Javascript del lado del servidor basado en el motor V8 de Google.
- **Express v4.12.4**: framework de Node.js destinado a la creación de APIs Rest
- **Angular.js**: framework javascript que facilita la creación de aplicaciones en una sola página.
- **Mongoose v4.1.2**: framework de Node.js destinado al modelado de objetos para MongoDB.
- **jwt.io v5.0.4**: framework destinado a la creación y distribución de web tokens.
- **socket.io v1.4.4**: framework de Node.js destinado a la creación de comunicaciones bidirecionales basadas en eventos
- **Apache mahout v0.11.1**: framework de Java destinado al aprendizaje automático.
- **socket.io-client v0.1.0**: cliente Java para socket.io desarrollado por Naoyuki Kanezawa

Herramienta de desarrollo

Durante el desarrollo de este Trabajo Fin de Grado se han utilizado las siguientes herramientas:

- **Eclipse Java EE IDE**: editor de código Java versión Mars 4.5.1
- **Maven v3.2.5**: gestor de paquetes para desarrollos Java
- **Brackets.io v1.6.0**: editor de código para desarrollos web
- **Git v1.9.4**: sistema de control de versiones
- **GitHub**: repositorio de código
- **Cmder**: emulador cmd de Windows
- **Sublime text 2**: editor de texto avanzado
- **CityEngine 2014**: software de modelado de ciudades
- **Unity 5**: motor gráfico para la creación de videojuegos
- **OSM2Word**: visualizador de datos con formato OSM

Herramienta de documentación

Se han usado las siguientes herramientas para la elaboración de la documentación del proyecto:

- **Latex**: lenguaje usado para la elaboración de este documento
- **GanttProject**: editor de diagramas de Gantt
- **Gliffy Diagrams**: editor de múltiples tipos de diagramas (diagramas de flujos, UML, etc.)

1.5. Modelo de proceso seleccionado

El modelo de trabajo seleccionado está basado en el modelo de espiral. Las actividades de este modelo forman una espiral de tal forma que cada iteración representa un conjunto de actividades. Se ha elegido este modelo de trabajo porque nos permitiría integrar el desarrollo con el mantenimiento y evaluar en cada iteración si dichos requisitos siguen encajando de lo que se esperaba de la aplicación para conseguir los objetivos propuestos. De esta forma se reduce el riesgo del proyecto y se incorporan objetivos de calidad.

1.6. Estructura de la memoria

El contenido de la memoria está distribuido de la siguiente forma:

- En el capítulo 2 se expone el trabajo desarrollado para la elaboración de este simulador
- En el capítulo 3 se expone la gestión del proyecto y las distintas etapas por las que ha pasado este proyecto
- En el capítulo 4 se muestran las conclusiones del proyecto y el posible trabajo futuro de cara a mejorar el simulador

Capítulo 2

Trabajo desarrollado

En este capítulo se explican las funcionalidades básicas del simulador desarrollado centradose únicamente en los aspectos más importantes. Para profundizar más sobre estos aspectos debe acudir a los anexos.

2.1. Resumen del simulador

Se trata de un sistema de colaboración abierta distribuida que permite configurar distintos escenarios con objetos móviles y estáticos sobre mapas de ciudades reales obtenidos a partir del servicio de mapas de OpenStreetMap.

El simulador de escenarios está basado en el simulador Mavsim desarrollado por el Grupo de Sistemas de Información Distribuidos de la Universidad de Zaragoza utilizado para la simulación de VANETs en el cual hay muchos vehículos distribuidos en una amplia zona geográfica.

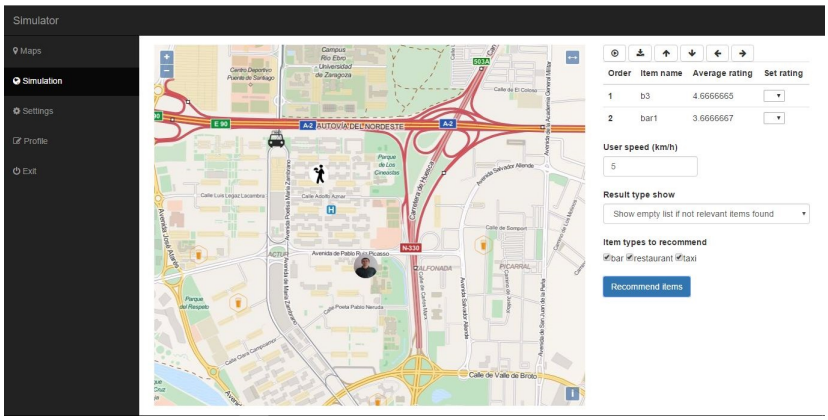


Figura 2.1: Simulación en Actur, Zaragoza con un solo usuario

Puede ser usado a través de cualquier dispositivo (PC, tablet, móvil etc.) con conexión a Internet y un navegador web. Permite a los usuarios crear sus propios mapas y escenarios. Durante la creación de una escena el usuario elige cual es la cuidad donde se realiza la simulación, el recomendador a utilizar, si el mapa es colaborativo o no, introducir los objetos estáticos y configurar cuales son los objetos móviles y sus rutas.

Para realizar una simulación el usuario tiene que buscar y seleccionar el mapa y escenario donde moverse tanto para obtener recomendaciones como para realizar votaciones sobre los distintos objetos de este entorno.

2.2. Análisis de requisitos

2.2.1. Requisitos no funcionales

Código	Descripción
RNF-1	La aplicación tratara de un simulador. Dicha simulación se realizara sobre mapas online
RNF-2	La navegación por los menús de la aplicación se realizara mediante una interfaz gráfica
RNF-3	Los textos por defecto de la aplicación serán en inglés
RNF-4	La autenfiticación de usuarios se realizará mediante web tokens
RNF-5	La interfaz gráfica debe ser responsive desarrollada con bootstrap
RNF-6	El back-end debe ser desarrollado con node.js, sockets.io y express
RNF-7	El front-end debe ser desarrollado con Angular.js

Cuadro 2.1: Requisitos no funcionales

2.2.2. Requisitos funcionales

Código	Descripción
RF-1	La aplicación permitirá crear un nuevo usuario
RF-2	La aplicación permitirá al usuario buscar mapas por su nombre, tipo, estado, cuidad y fecha de creación
RF-3	La aplicación permitirá al usuario crear un nuevo mapa
RF-4	La aplicación permitirá al usuario crear una nueva escena asociada a un mapa existente
RF-5	La aplicación listara todas las escenas de un mapa
RF-6	La aplicación permitirá al usuario editar un mapa existente
RF-7	La aplicación permitirá al usuario editar las escenas de un mapa existente
RF-8	La aplicación permitirá al usuario crear un nuevo tipo de objeto estático
RF-9	La aplicación permitirá al usuario crear un nuevo tipo de objeto dinámico
RF-10	La aplicación listará todos los tipos de objetos estáticos creados
RF-11	La aplicación listará todos los tipos de objetos dinámicos creados
RF-12	La aplicación permitirá al usuario editar los objetos estáticos credos
RF-13	La aplicación permitirá al usuario editar los objetos dinámicos creados
RF-14	La aplicación permitirá al usuario cambiar el nombre
RF-15	La aplicación permitirá al usuario cambiar su contraseña
RF-16	La aplicación permitirá al usuario cambiar la imagen asociada a un usuario
RF-17	La aplicación permitirá al usuario configurar un nuevo tipo de recomendador
RF-18	La aplicación permitirá al usuario editar la configuración de recomendador existente
RF-18	La aplicación permitirá al usuario asociar un recomendador existente a una escena
RF-19	La aplicación permitirá al usuario definir los limites de una escena
RF-20	La aplicación permitirá al usuario asociar un objeto estático a una escena
RF-21	La aplicación permitirá al usuario cargar todos los objetos estáticos desde un fichero JSON

Código	Descripción
RF-22	La aplicación permitirá asociar un objeto dinámico y su definir su ruta en una escena
RF-23	La aplicación permitirá al usuario cargar todos los objetos dinámicos y sus rutas desde un fichero JSON
RF-24	La aplicación listará todos objetos estáticos asociados a una escena
RF-25	La aplicación listará todos los objetos dinámicos asociados a una escena
RF-26	La aplicación permitirá borrar un objeto estático asociado a una escena
RF-27	La aplicación permitirá borrar un objeto dinámico asociado a una escena
RF-28	La aplicación permitirá al usuario elegir un si mapa es colaborativo o no
RF-29	La aplicación permitirá al usuario ejecuta una simulación sobre la escena de un mapa
RF-30	La aplicación permitirá al usuario solicitar recomendaciones mientras se está ejecutando una simulación siempre y cuando el recomendador asociado a la escena es de tipo pull
RF-31	El usuario recibirá recomendaciones sin haberlas solicitado siempre y cuando el recomendador asociado a la escena de es tipo push
RF-32	El usuario puede arrancar/pausar una simulación

Cuadro 2.2: Requisitos funcionales

2.3. Arquitectura del sistema

La arquitectura del sistema consta de cliente o navegador web, servidor web Node.js, servidor de recomendaciones y base de datos mongoDB (figura 2.2).

El la figura 2.2 observamos que el navegador web se conecta al servidor Node.js mediante dos maneras: la primera es HTTP y la segunda es un sistema bidireccional dirigido por eventos. Las funcionalidades como creación de escenas, búsqueda de mapas etc. están desarrollados sobre una REST API y el intercambio de mensajes JSON.

El sistema bidireccional dirigido por eventos es utilizado durante la simulación por una parte para reflejar los eventos generados por un usuario al resto de usuarios, y por otra para integrar el navegador, el servidor Node.js y el recomendador. De esta manera conseguimos compartir información entre los distintos componentes sin que estos los hayan solicitado evitando muchas peticiones innecesarias.

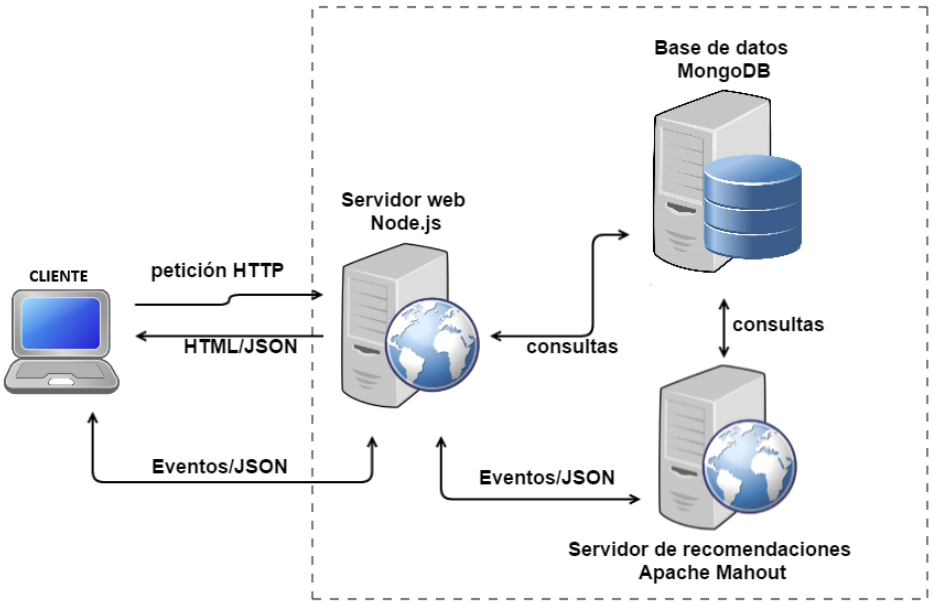


Figura 2.2: Arquitectura de componentes del sistema

2.3.1. Arquitectura del front-end

Para el desarrollo del front-end se han utilizado los frameworks Angular.js[1] y bootstrap[2]. Se ha decidido utilizar estas tecnologías porque nos ofrece varias ventajas: ahorro de recursos[3], mejora de la productividad y la posibilidad de realizar una simulación sobre dispositivos móviles[4] con el mismo código fuente.

La arquitectura del front-end está basada en el patrón Modelo-Vista-Controllador de tal forma que para cada vista existe un controlador que contiene la lógica de negocio de esta. El controlador también es el encargado de establecer comunicación con el back end. Esta comunicación se realiza mediante los llamados Servicios[5] de Angular.js y la REST API del back end. Los Servicios de Angular.js son muy necesarios y útiles ya que nos permiten crear un envoltorio sobre la REST API que nos ofrece el back end y de esta forma centralizar las llamas a la API.

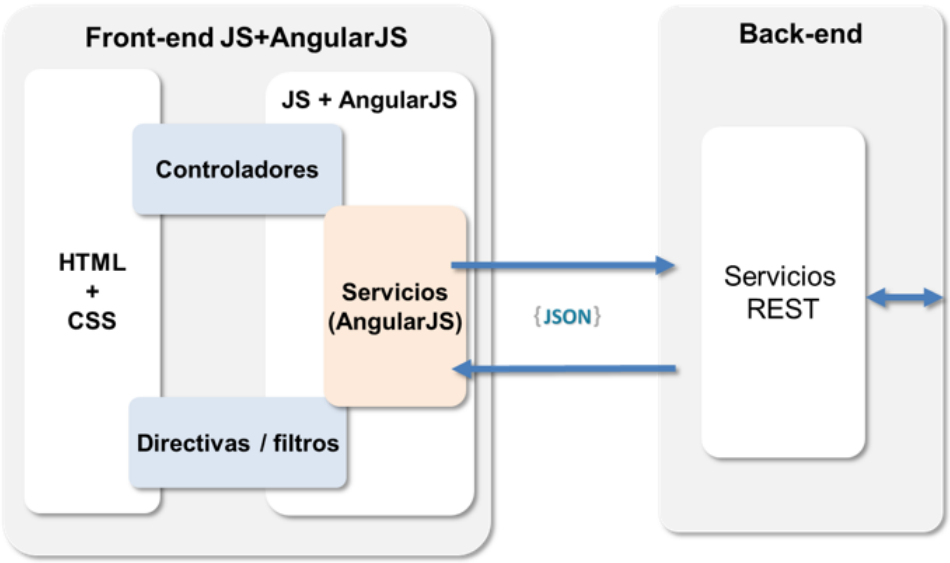


Figura 2.3: Arquitectura del front-end

[1] framework que nos permite desarrollar aplicaciones de una sola página

[2] framework que nos ofrece un sistema de componentes reutilizables y adaptables a la pantalla del dispositivo

[3] angular.js va transmitiendo las vistas de la interfaz gráfica y las cachea al lado del cliente para ser reutilizadas posteriormente. Vuelve a solicitar una vista si y solo si esta ha sufrido algún cambio en el servidor

[4] esto nos da la oportunidad de utilizar la posición geográfica del usuario para obtener recomendaciones en el entorno de una cuidad real. De está manera obtenemos datos reales y muchas más precisión a la hora de evaluar los algoritmos de recomendaciones

[5] es pequeña fabrica de funciones y objetos inyectada en los controladores

2.3.2. Arquitectura del back-end

Para el desarrollo del back-end se han utilizado Node.js[6], Express[7], socket.io[8] y mongoose [9]. Se ha decidido utilizar estas tecnología por que nos ofrecen las siguientes ventajas: mejora la productividad a la hora de desarrollar el back-end, nos permite desarrollar un back-end ligero que consuma pocos recursos y el modulo sockets.io nos ofrece la posibilidad de desarrollar un sistema bidireccional dirigido por eventos.

La arquitectura del back-end se basa en la filosofía de desarrollo de aplicaciones con Node.js y Express y consiste de la siguiente estructura de directorios:

```
Simulator
├── app.js
├── package.json
├── bin
│   └── www
├── models
├── public
│   ├── images
│   ├── javascript
│   │   ├── angular
│   │   ├── bootstrap
│   │   ├── jquery
│   │   ├── openlayers
│   │   └── socketIO
│   ├── stylesheets
│   ├── views
│   │   ├── configurations
│   │   ├── maps
│   │   ├── settings
│   │   ├── index.html
│   │   └── register.html
│   └── index.html
├── routes
│   ├── configuratios.js
│   ├── maps.js
│   ├── simulation.js
│   └── user.js
```

[6] javascript al lado del servidor
[7] modulo de Node.js que nos ofrece la posibilidad de desarrollar una REST API
[8] sistema bidireccional dirigido por eventos
[9] modelado de objetos sobre mongoDB

A continuación vamos a ver más detalladamente cual es la función de cada elemento de este directorio:

- app.js: centraliza las configuraciones de nuestra aplicación como por ejemplo en que puerto arranca el servidor, establecer conexiones con la base de datos, configuraciones del router[10] etc.
- package.json: es un gestor de paquetes y contiene los módulos que se están utilizando en nuestra aplicación.
- public: es un directorio que contiene la parte visual, es decir el front-end. Podemos ver que este contiene varios subdirectorios:
  - en images se ubican las imágenes o iconos usados en la aplicación.
  - en javascript se ubican los frameworks Javascript usados en en el front-end como Angular.js, Bootstrap, OpenLayers etc. En el subdirectorio angular podemos encontrar las configuraciones de Angular.js, los controladores de las vistas, los Servicios etc.
  - en views se ubican las vistas del front-end.
  - en stylesheets están ubicadas las hojas de estilos
- en routes se ubican las distintas rutas de Express. En nuestro caso tenemos una para cada menú de la aplicación. Por ejemplo todas las operaciones referentes al menú Maps se encuentra en el fichero maps.js etc.

2.3.3. Arquitectura de recomendador

El desarrollo del recomendador está realizado con Java 7, la librería Apache Mahout y socket.io-client. El recomendador desarrollado es un recomendador pull[11] de ejemplo basado en los usuarios (User based recommender).

Aunque sea un recomendador de ejemplo este está pensado para se expandido, tanto para otro tipo de recomendadores (como pueden ser los recomendadores de tipo push[12]) como para la implementación de nuevos tipos de estrategias para el recomendador de tipo pull.

Durante el arranque el servidor del recomendador lanza un hilo por cada tipo de recomendador. En nuestro caso solo lanza un hilo que se corresponde al servidor de tipo pull. Este hilo es el que contiene los eventos que invoca

[10] el router de Node.js es el encargado del direccionamiento de las peticiones y hace referencia a la definición de puntos finales de aplicación (URI) y cómo responden a las solicitudes de cliente

[11] tipo de recomendador en el cual los usuarios solicitan recomendaciones

[12] tipo de recomendador que realiza recomendaciones sin que el usuario los haya solicitado

el simulador de escenarios. En la figura 2.4 observamos que tenemos solo dos eventos: uno para recuperar los tipos de implementaciones[13] y otro para realizar las recomendaciones.

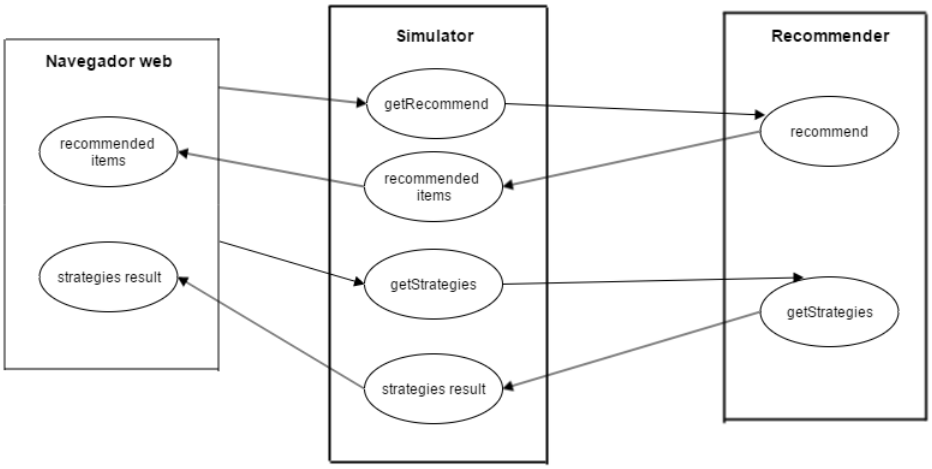


Figura 2.4: Diagrama de eventos

El evento recommend es el que se dispara cuando uno de los usuarios solicita una recomendación. Para que se puedan utilizar distintos tipos de implementaciones del recomendador de tipo pull se ha implementado un patrón de diseño de tipo Strategy. Este patrón de diseños nos permite cambiar de estrategia de recomendación en tiempo de ejecución:

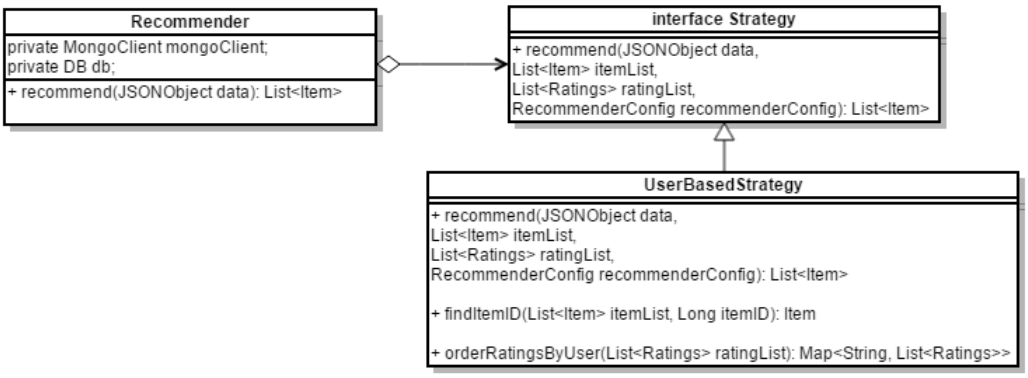


Figura 2.5: Diagrama uml del patrón de diseño de tipo Strategy

[13] en nuestro caso es User based recommender pero existen otros tipos como Item based recommender

2.4. Menús del simulador de escenarios

Como en todas las aplicaciones, el simulador de escenarios cuenta con un sistema de menús que dan acceso a las distintas opciones del simulador. En este caso se ha seguido la paradigma WIMP para la organización de los menús y distintas opciones (figura 2.6). Existen 4 tipos de caminos organizados en forma de árbol:

- busqueda y gestión (creación, edición y borrados) de mapas y escenas
- simulación de un mapa y escena
- gestión de tipos de recomendadores, objetos estáticos[14] y dinámicos[15]
- configuraciones del perfil del usuario

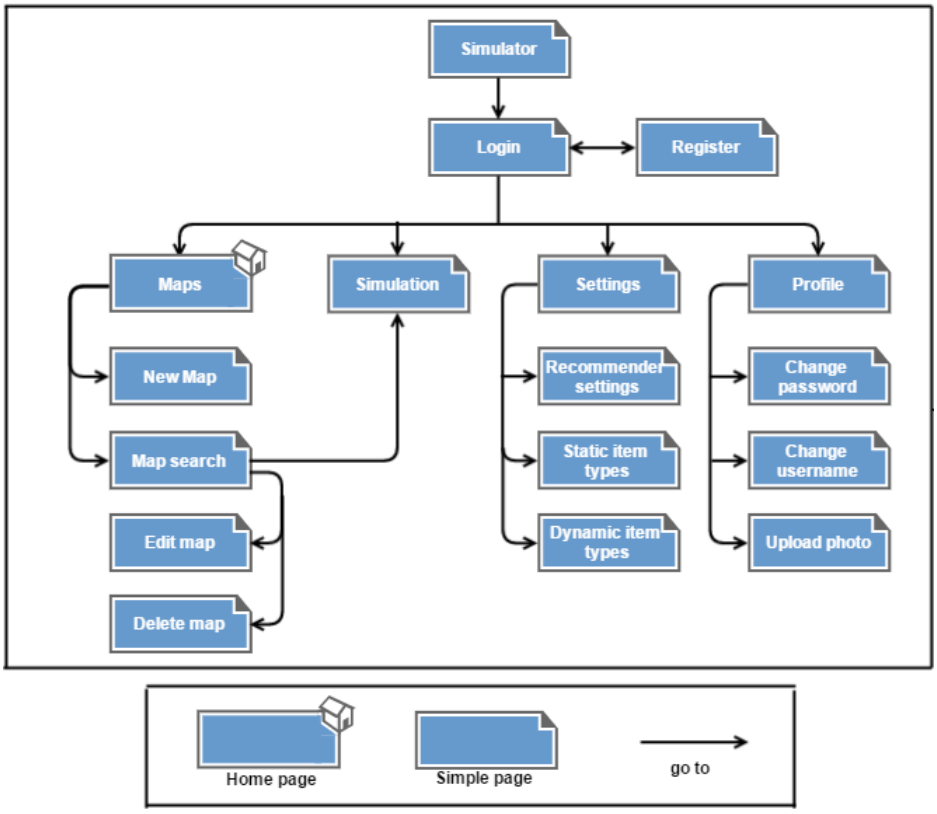


Figura 2.6: Mapa de navegación

[14] objetos que no cambian de posición a medida que pasa el tiempo

[15] objetos que cambian de posición a medida que pasa el tiempo. Tienen una ruta definida durante la creación de la escena

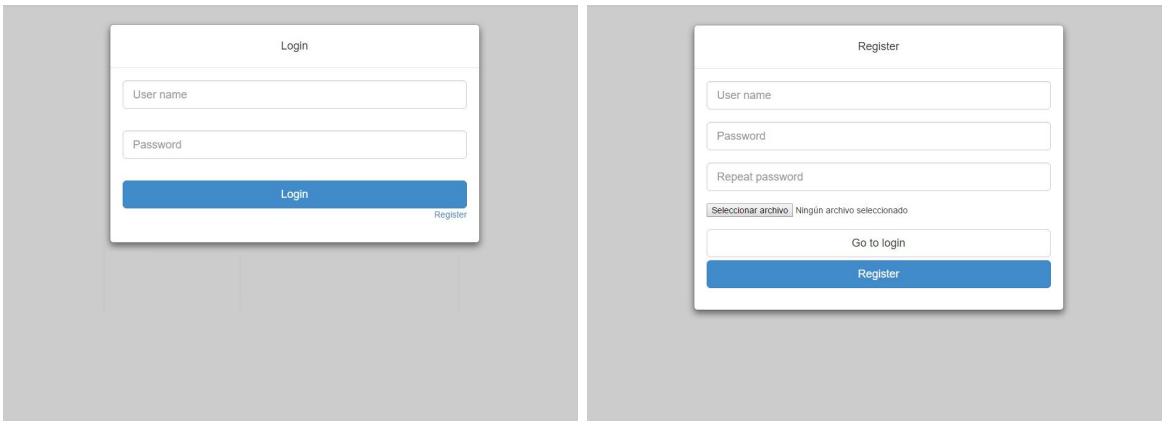
2.5. Navegación por estima

La navegación por estima es una técnica que se aplica en el cliente. Consiste en procesar en cada ciclo el estado de los objetos móviles. Se trata de una técnica analítica utilizada en la náutica para la navegación y situación de los barcos y se tienen en cuenta los siguientes elementos: la situación actual, rumbo y velocidad. Es decir, sabiendo la velocidad, el rumbo de la nave y el tiempo transcurrido se puede estimar la posición de la misma al cabo del tiempo.

Con este método conseguimos calcular cual es la siguiente posición geográfica donde tenemos que colocar un objeto móvil al acabo de un tiempo (el tiempo de refresco de la pantalla). Como ventaja conseguimos disminuir el error en el cálculo de las posiciones de los objetos móviles. Así obtenemos movimientos muy precisos incluso en grafos de movimientos con nodos muy cercanos. Para evitar problemas en estos casos se han puesto tiempos bajos de refresco de la pantalla.

2.6. Diseño final

En esta sección se dejan solo algunas de las pantallas más relevantes de la aplicación. Para ver más pantallas debe acudir al manual de usuario en los anexos.



(a) Autentificación de un usuario (b) Crear un nuevo usuario

Figura 2.7: Registro y autentificación de un usuario

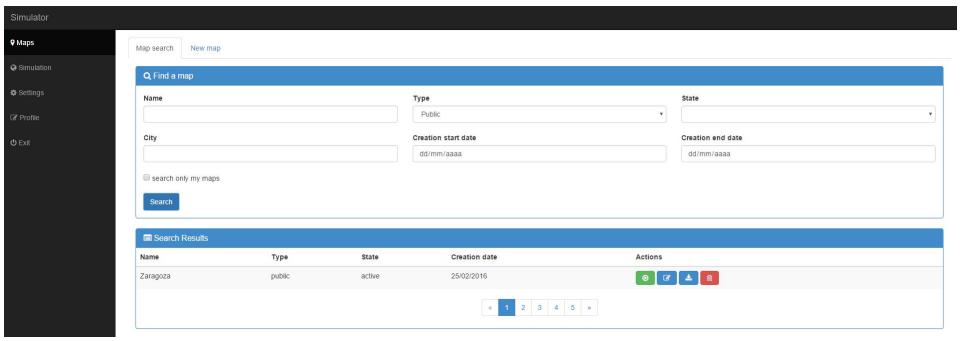


Figura 2.8: Busqueda de mapas

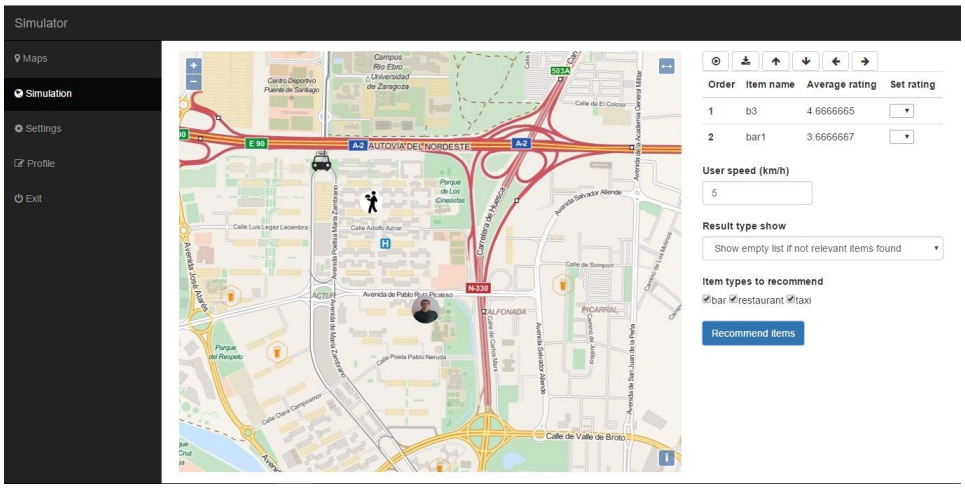


Figura 2.9: Simulación en Actur, Zaragoza

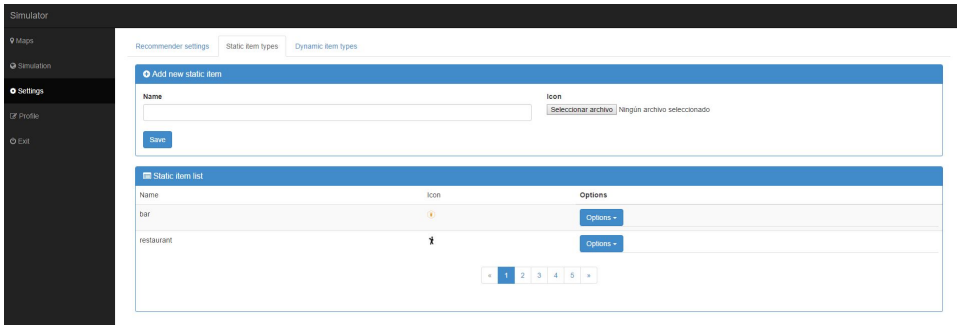


Figura 2.10: Configuración de los objetos estáticos

Capítulo 3

Gestión del proyecto

En este capítulo se explican el modelo de proceso seleccionado y las distintas etapas por las que ha pasado el este Trabajo Fin de Grado centradose únicamente en los aspectos más importantes. Para profundizar más sobre estos aspectos debe acudir a los anexos.

3.1. Modelo de proceso seleccionado

Este Trabajo Fin de Grado ha surgido una importante evolución desde el primer planteamiento hasta la obtención del sistema final. La primera idea para la evaluación de los algoritmos de recomendaciones era la utilización de un videojuego ya que este podría recolectar datos de forma transparente mientras los usuarios se divertian. El videojuego consistiría en el cumplimiento de misiones en una cuidad basandose en algún videojuego de aventuras como el videojuego Paperboy del año 1985 donde un chico reparte periódicos en una cuidad.

Por esto se ha planteado usar el motor gráfico Unity 5 para el desarrollo del videojuego ya que no tenía sentido desarrollar un videojuego usando simplemente un lenguaje de programación. Pero existía la incertidumbre si todos los requisitos podrían ser implementados. Esto era debido porque por una parte no se conocían de antemano todos los requisitos y por otra no se conocía si el motor gráfico tenía algún tipo de limite.

Para solucionar estos problemas se ha tomado la decisión de elegir el modelo en espiral como modelo de trabajo. Las actividades de este modelo forman una espiral de tal forma que cada iteración representa un conjunto de actividades. Esto nos permitiría segmentar el trabajo en tareas más pequeñas e ir definiendo los requisitos mientras se desarrollaba el videojuego. De esta manera podríamos evaluar si los requisitos propuestos podrían o no ser implementados con Unity 5. En caso de que el motor gráfico tuviese

algún límite tendríamos la capacidad de reorientar el proyecto.

3.1.1. Etapa 1

En la tabla 3.1 podemos encontrar las tareas que han sido definidas para esta etapa:

Código	Descripción
RQ-0	Formación básica con Unity 5
RQ-1	Establecer puntos en el mapa para que el jugador pueda ir a buscarlos.
RQ-2	Los puntos establecidos en el requisitos RQ-1 deben de ser extraibles desde un servidor externo
RQ-3	Establecer una conexión HTTP/JSON entre Unity 5 y un servidor externo.
RQ-4	Sacar el modelado geométrico de un servidor externo
RQ-5	El videojugo debe de tener un menú principal
RQ-6	Investigar si se puede usar Longitud y Latitud con Unity 5
RQ-7	Investigar si los edificio generados con CityEngine 2014 son reales o no
RQ-8	Permitir que el juego se realice sobre mapas de ciudades reales
RQ-9	Investigar si es posible el desarrollo de videojuego en 3D

Cuadro 3.1: Tareas etapa 1

Como conclusión de esta etapa obtenemos el siguiente resultado:

- puede establecerse una conexión con un servidor externo mediante HTTP/JSON (RQ-3).
- pueden establecerse puntos en el mapa para que el jugador pueda ir a buscarlos(RQ-1).
- pueden extraerse puntos desde un servidor externo para que el usuarios pueda ir a buscarlos (RQ-1 y RQ-3).
- la realizacion del menú principal del videojuego ha sido posible.
- Unity 5 utiliza eje de abscisas. Por lo tanto si queremos usar coordenadas geográficas tenemos que calcular la Longitud y Latitud a partir del eje de coordenadas (x, y, z).
- los datos sobre los edificios de OpenStreetMap no están completos.

- se ha decido de dejar atrás el modelado 3D ya que se han detectado los siguientes problemas: coste en cuanto a tiempo demasiado grande para el diseño de una ciudad, CityEngine 2014 no interpreta correctamente los datos exportados además no existe ninguna otra herramienta que nos permita la correcta interpretación de los datos (por lo menos yo no podido encontrarla).

3.1.2. Etapa 2

En la tabla 3.2 podemos encontrar las tareas que han sido definidas para esta etapa:

Código	Descripción
RQ-10	Investigar como se desarrollan grafos en Unity para la posterior implementación de la IA sobre estos para el movimiento de los vehículos etc.
RQ-11	Diseñar y desarrollar un mapa del juego

Cuadro 3.2: Tareas etapa 2

Como conclusión de esta etapa obtenemos el siguiente resultado:

- Unity 5 dispone de su propio modulo de IA y por lo tanto no hace falta implementar algoritmos de IA para el comportamiento de los objetos dinámicos.
- en esta etapa se ha definido el requisito que el juego tiene que funcionar sobre cualquier mapa del mundo. Por lo tanto en la siguiente etapa hay que investigar si es posible la integración de Unity 5 con OpenStreetMap.

3.1.3. Etapa 3

En la tabla 3.3 podemos encontrar las tareas que han sido definidas para esta etapa:

Código	Descripción
RQ-11	Investigar en que consiste el formato OSM
RQ-12	Investigar si es posible la integración de OpenStreetMap con Unity

Cuadro 3.3: Tareas etapa 3

Como conclusión de esta etapa obtenemos el siguiente resultado:

- Unity 5 no puede ser integrado con OpenStreetMap y no es capaz de renderizar mapas a partir sus datos (ficheros OSM).
- en esta etapa se ha decido reorientar el proyecto por las siguientes razones: Unity 5 no puede ser integrado con OpenStreetMap por lo tanto la única opción para desarrollar un videojuego sobre mapas reales es desarrollando el videojuego desde cero con algún lenguaje como Java y el inconveniente que esto representa es que es demasiado costoso en cuanto a tiempo y dificultad.

3.1.4. Etapa 4

En la tabla 3.4 podemos encontrar las tareas que han sido definidas para esta etapa:

Código	Descripción
RQ-12	instalar mongoDB
RQ-13	instalar Node.js y NPM
RQ-14	instalar git
RQ-14	instalar Java y Apache maven
RQ-16	Desarrollar y configurar la base de la aplicación con Node.js
RQ-17	Instalar y configurar Angular.js
RQ-18	Instalar y configurar OpenLayers.js
RQ-19	Instalar y configurar Bootstrap

Cuadro 3.4: Tareas etapa 4

Como conclusión de esta etapa obtenemos el siguiente resultado:

- se han instalado y configurado todas las herramientas y frameworks necesarios para el desarrollo de la aplicación.

3.1.5. Etapa 5

En la tabla 3.6 podemos encontrar las tareas que han sido definidas para esta etapa:

Código	Descripción
RQ-20	Definir lo requisitos funcionales y no funcionales de la aplicación
RQ-21	Desarrollar el front-end
RQ-22	Deseñar la base de datos
RQ-23	Diseñar y desarrollar el back-end
RQ-24	Diseñar y desarrollar el recomendador
RQ-25	Realizar pruebas funcionales de la aplicación

Cuadro 3.5: Tareas etapa 5

Como conclusión de esta etapa obtenemos el siguiente resultado:

- la parte mas costosa de esta etapa ha sido la definición de los requisitos de la aplicación.
- una vez definidos los requisitos el resto del trabajo ha sido puramente técnico.

3.1.6. Etapa 6

La última etapa consiste en documentar el trabajo realizado en este Trabajo Fin de Grado.

3.2. Tiempo dedicado

Como ya se ha comentado anteriormente, el desarrollo del proyecto se ha realizado siguiendo el modelo en espiral. En esta sección se mostrará el tiempo dedicado y también un cronograma de las diferentes etapas.

Fase	**Horas**
Fase 1	54
Fase 2	6.5
Fase 3	38
Fase 4	3.5
Fase 5	247
Fase 6	42.5
Reuniones	7
Total	398.5

Cuadro 3.6: Separación por horas de las distintas fases

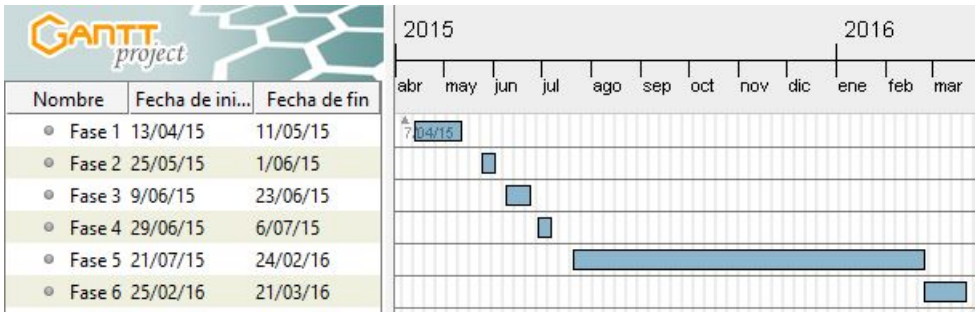


Figura 3.1: Diagrama de Gantt de las distintas fases del proyecto

Capítulo 4

Conclusiones