



**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 evaluar sistemas de recomendación

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, 6 de mayo de 2016

Desarrollo de un simulador de escenarios con usuarios móviles para evaluar sistemas de recomendación

Resumen

Los sistemas de recomendación proporcionan sugerencias acerca de elementos que pueden resultar de interés para el usuario como pueden ser los hoteles, restaurantes, libros, películas, líneas de taxis y autobuses etc. Habitualmente estos sistemas se evalúan con conjuntos de datos estáticos clásicos pero el inconveniente que esto conlleva es que solo podemos realizar un análisis sobre el comportamiento de los usuarios en el pasado perdiendo la capacidad de poder calibrar correctamente los algoritmos de recomendaciones para su correcta evaluación.

El objetivo de este Trabajo Fin de Grado es desarrollar un simulador de escenarios con usuarios móviles que recolecte y proporcione conjuntos de datos dinámicos para la evaluación de algoritmos de recomendaciones.

La arquitectura del sistema consta de un navegador web, un servidor web Node.js, un servidor de recomendaciones y una base de datos mongoDB. Se trata de una aplicación web de una sola página desarrollada con Angular.js y Node.js que nos permite configurar distintos escenarios con conjuntos de datos reales obtenidos a partir del servicio de mapas de OpenStreetMap.

La integración entre los distintos componentes del sistema está realizada de dos maneras. La primera es una REST API y el intercambio de mensajería JSON para realizar operaciones de tipo CRUD para las distintas configuraciones del simulador y la segunda es un sistema bidireccional dirigido por eventos utilizado durante las simulaciones para compartir información entre los distintos componentes del sistema sin que estos la hayan solicitado evitando muchas peticiones innecesarias y consiguiendo que las simulaciones funcionen en tiempo real.

Para la gestión del proyecto se ha elegido el modelo en espiral como modelo de trabajo. Esto me ha permitido evaluar los requisitos en cada iteración y reorientar el proyecto al encontrar dificultades y limitaciones.

Agradecimientos

Me gustaría agradecer este Trabajo Fin de Grado a 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 lo largo de estos años.

Índice general

1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivos	1
1.3. Contexto Tecnológico	2
1.4. Herramientas utilizadas	3
1.5. Modelo de proceso seleccionado	4
1.6. Estructura de la memoria	5
2. Trabajo desarrollado	7
2.1. Resumen del simulador	7
2.2. Análisis de requisitos	8
2.2.1. Requisitos no funcionales	8
2.2.2. Requisitos funcionales	9
2.3. Arquitectura del sistema	11
2.3.1. Arquitectura del front-end	12
2.3.2. Arquitectura del back-end	12
2.3.3. Arquitectura de recomendador	14
2.3.4. Autenticación basada en token	16
2.4. Menús del simulador de escenarios	17
2.5. Movimiento de los objetos móviles: navegación por estima . .	19
3. Explotación	21
3.1. Motivación	21
3.2. Definir un escenario realista	22
3.2.1. Paso 1: obtener datos de restaurantes reales	22
3.2.2. Paso 2: crear un mapa nuevo o editar un mapa existente	22
3.2.3. Paso 3: crear un escenario realista	23
3.3. Simulación y evaluación de un sistema de recomendación . . .	26
3.3.1. Simulación de un escenario realista	26
3.3.2. Evaluación de un recomendador	26
3.4. Rendimiento	26
3.4.1. Rendimientos con Linux	26
3.4.2. Rendimientos con Windows	28

3.4.3. Conclusiones de las pruebas de estrés	29
4. Gestión del proyecto	31
4.1. Modelo de proceso seleccionado	31
4.1.1. Primer ciclo de la espiral: formación con las herramientas a utilizar	32
4.1.2. Segundo ciclo de la espiral: diseñar el mapa y desarrollar la IA	33
4.1.3. Tercer ciclo de la espiral: integración con OpenStreetMap	33
4.1.4. Cuarto ciclo de la espiral: instalación y configuración del entorno de trabajo para el desarrollo del simulador RecSim	34
4.1.5. Quinto ciclo de la espiral: desarrollo del simulador RecSim	34
4.1.6. Sexto ciclo de la espiral: documentar el trabajo realizado	35
4.2. Tiempo dedicado	36
5. Conclusiones	37
5.1. Resultados	37
5.2. Trabajos futuros	38
5.3. Valoración personal	38
Bibliografía	40
Anexos	41
A. Manual del usuario	43
A.1. Visión general	43
A.1.1. Introducción	43
A.1.2. ¿Qué es el simulador de escenarios con usuarios móviles?	43
A.1.3. Tipos de tecnologías utilizadas	43
A.1.4. Instalación	44
A.1.5. Primeros pasos	47
A.2. Configuración del recomendador	48
A.3. Configuración de los objetos estáticos	49
A.4. Configuración de los objetos dinámicos	49
A.5. Crear un un nuevo usuario	50
A.6. Actualizar el perfil de un usuario	51
A.7. Búsqueda de mapas	51
A.8. Crear un nuevo mapa	52
A.9. Crear una nueva escena	53
A.9.1. Paso 1: Elegir el nombre de la escena y el recomendador a utilizar	53

A.9.2. Paso 2: Límites de la escena	53
A.9.3. Paso 3: Configurar los objetos estáticos	54
A.9.4. Paso 4: Configurar los objetos dinámicos	56
A.10. Edición de mapas y escenas	59
A.11. Simulación	59
A.11.1. Controles del usuario	60
A.11.2. Recomendaciones	61
A.11.3. Votaciones	61
A.12. Evaluación de un recomendador	62
B. Análisis	63
B.1. Análisis de requisitos	63
B.1.1. Requisitos no funcionales	63
B.1.2. Requisitos funcionales	64
B.2. Objetivos de Usabilidad	65
B.3. Diagrama de casos de uso	67
C. Implementación	69
C.0.1. Implementación del front-end	69
C.0.2. Implementación del back-end	73
C.0.3. ¿Cómo crear un instalador de la aplicación?	73
D. Integración con un recomendador externo	83
D.1. Introducción	83
D.2. Recomendadores pull	84
D.2.1. ¿En que consiste el patrón de diseño de tipo Strategy?	84
D.2.2. Pasos para crear un nuevo recomendador pull	85
D.2.3. Eventos y mensajes	87
D.3. Recomendadores push	90

Índice de figuras

2.1. Simulador RecSim	8
2.2. Arquitectura de componentes del sistema	11
2.3. Arquitectura del front-end	12
2.4. Diagrama de eventos	15
2.5. Diagrama UML del patrón de diseño de tipo Strategy	15
2.6. Autenticación Basada en Token	16
2.7. Formato del Token	17
2.8. Mapa de navegación	18
3.1. Creación de un mapa nuevo en México con datos reales . . .	23
3.2. Paso 1: definir un nombre y recomendador para el escenario realista	23
3.3. Paso 2: definir los limites del escenario realista	24
3.4. Paso 3: definir los restaurantes sobre el escenario	24
3.5. Paso 3: lista de restaurantes definidos sobre el escenario . . .	25
3.6. Paso 4: lista de objetos móviles	25
3.7. Simulación realizada en San Luis Potosí, México	26
4.1. Diagrama de Gantt de las distintas fases del proyecto	36
A.1. Configuración del recomendador	48
A.2. Configuración de los objetos estáticos	49
A.3. Configuración de los objetos dinámicos	50
A.4. Crear un nuevo usuario	50
A.5. Actualizar el perfil de un usuario	51
A.6. Búsqueda de mapas	52
A.7. Crear un nuevo mapa	52
A.8. Paso 1: elegir nombre de la escena y el recomendador	53
A.9. Paso 2: Limites de la escena	54
A.10.Paso 3.1: Previsualizar los objetos estáticos a importar	55
A.11.Paso 3.1: importar los objetos estáticos desde un fichero . . .	55
A.12.Paso 3.2: definir los objetos estáticos manualmente	56
A.13.Paso 4.1: importar los objetos dinámicos	57
A.14.Paso 4.2: definir los objetos dinámicos manualmente	57

A.15.Paso 4.2: definir los objetos dinámicos manualmente	58
A.16.Paso 4.3: generar los objetos dinámicos aleatoriamente	58
A.17.Edición de mapas y escenas	59
A.18.Simulación	60
A.19.Evaluación de un recomendador	62
 B.1. Diagrama de casos de uso parte 1	67
B.2. Diagrama de casos de uso parte 2	67
 C.1. Ejemplo de un controlador	72
C.2. Two-way data binding	72
C.3. Paso 1	75
C.4. Paso 2	75
C.5. Paso 3	76
C.6. Paso 4	76
C.7. Paso 5	77
C.8. Paso 6	77
C.9. Paso 7	78
C.10.Paso 8	78
C.11.Paso 9	79
C.12.Paso 10	79
 D.1. Diagrama de eventos del recomendador de tipo pull	87

Índice de cuadros

2.1. Requisitos no funcionales	8
2.2. Requisitos funcionales	10
3.1. Pruebas de estrés con 100 usuarios concurrentes	27
3.2. Pruebas de estrés con 500 usuarios concurrentes	27
3.3. Pruebas de estrés con 1000 usuarios concurrentes	27
4.1. Tareas del primer ciclo de la espiral	32
4.2. Tareas segundo ciclo de la espiral	33
4.3. Tareas tercer ciclo de la espiral	33
4.4. Tareas cuarto ciclo de la espiral	34
4.5. Tareas quinto ciclo de la espiral	35
4.6. Separación por horas de las distintas fases	36
B.1. Requisitos no funcionales	63
B.2. Requisitos funcionales	65
B.3. Objetivos de usabilidad	66
D.1. Evento get value forecast del recomendador	88
D.2. Evento recommended ítems del simulador	88
D.3. Evento recommend del recomendador	89

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 solo podemos realizar un análisis sobre el comportamiento de los usuarios en el pasado perdiendo la capacidad de poder calibrar correctamente los algoritmos de recomendaciones para su correcta evaluación. 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 recolectando 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, librerías 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 bidireccionales 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

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.)
- **Apache Benchmark**: herramienta para hacer pruebas de carga de sitios web
- **<http://newrelic.com/>**: sitio web para motorización de servidores

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 posible explotación del simulador de escenarios. También se muestra el rendimiento obtenido por el simulador.
- En el capítulo 4 se expone la gestión del proyecto y las distintas etapas por las que ha pasado este proyecto
- En el capítulo 5 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 centrándose únicamente en los aspectos más importantes.

2.1. Resumen del simulador

El simulador RecSim permite representar escenarios adaptados a las necesidades de simulación existentes. Un escenario se compone de un mapa y de un conjunto de objetos, y puede definirse sobre cualquier mapa real existente en OpenStreetMap: el usuario puede utilizar un buscador para ver un mapa y luego seleccionar dentro de dicho mapa el área geográfica de interés. Además, se pueden definir tipos de objetos estáticos (por ejemplo, restaurantes, hoteles, cines, etc.) y tipos de objetos dinámicos o móviles (por ejemplo, taxis, buses, ambulancias, etc.), asociándoles un nombre de tipo y un icono que se utilizará para representar instancias de dicho tipo de objeto en un mapa. Luego pueden colocarse en un escenario objetos estáticos y dinámicos de los tipos deseados: los objetos estáticos se colocan en la posición deseada sobre el mapa y a los objetos dinámicos se les asigna una determinada trayectoria o un comportamiento de movimientos aleatorios. Alternativamente, pueden importarse datos acerca de objetos estáticos o dinámicos definidos en ficheros externos. Sobre el mapa se representa también un usuario móvil, que puede desplazarse utilizando el teclado (sin necesidad de que el usuario real cambie físicamente de lugar) o bien desplazarse físicamente (en cuyo caso las coordenadas se obtienen de un receptor GPS ligado al equipo).

Un aspecto clave es que podemos simular esos escenarios móviles y al mismo tiempo poner en funcionamiento sistemas de recomendación que proporcionan recomendaciones al usuario. El usuario puede valorar ítems, de forma que el sistema de recomendación puede ir aprendiendo los gustos del usuario y aumentando la información del conjunto de datos de valoraciones

disponible. Es posible exportar resultados referentes a los errores cometidos por el sistema de recomendación al valorar ítems y generar gráficas sencillas desde el mismo simulador

Como ejemplo, en la Figura 2.1a se muestra una captura de pantalla donde se puede ver un fragmento del mapa de Zaragoza con un usuario móvil y varios tipos de ítems de potencial interés: hoteles, restaurantes, hospitales, y coches. En la Figura 2.1b se muestra un ejemplo de gráfica que ilustra los errores cometidos por el sistema de recomendación al estimar la valoración de varios ítems.

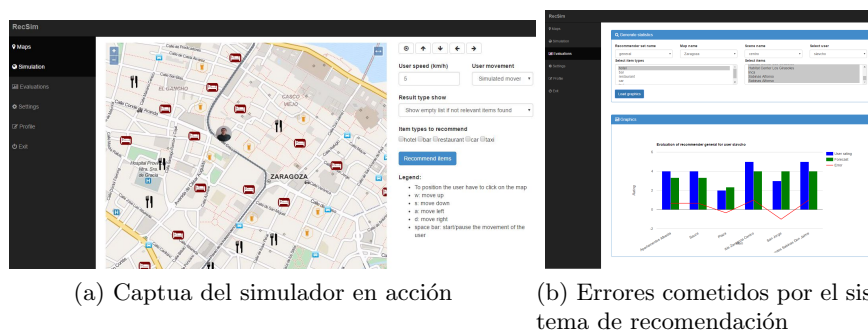


Figura 2.1: Simulador RecSim

2.2. Análisis de requisitos

2.2.1. Requisitos no funcionales

Código	Descripción
RNF-1	La aplicación tratará 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, ciudad 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 creados
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 límites 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
RF-33	La aplicación permitirá al usuario generar de forma aleatoria los grafos de movimiento de los vehículos

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).

En 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.

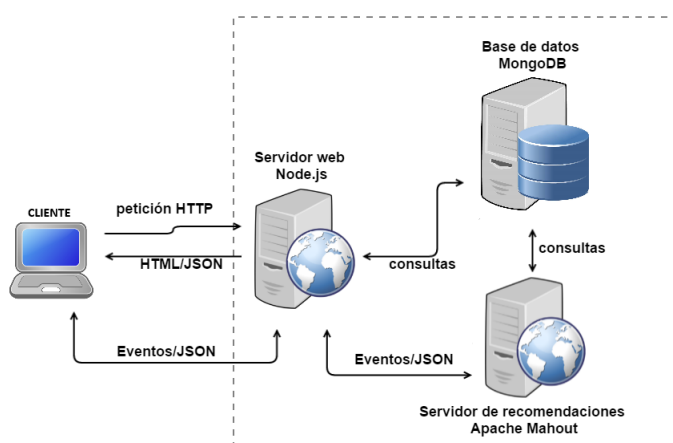


Figura 2.2: Arquitectura de componentes del sistema

2.3.1. Arquitectura del front-end

Para el desarrollo del front-end se han utilizado los frameworks Angular.js y bootstrap. Se ha decidido utilizar estas tecnologías porque nos ofrece varias ventajas: ahorro de recursos, mejora de la productividad y la posibilidad de realizar una simulación sobre dispositivos móviles 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 entre los llamados Servicios¹ 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 llamadas a la API.

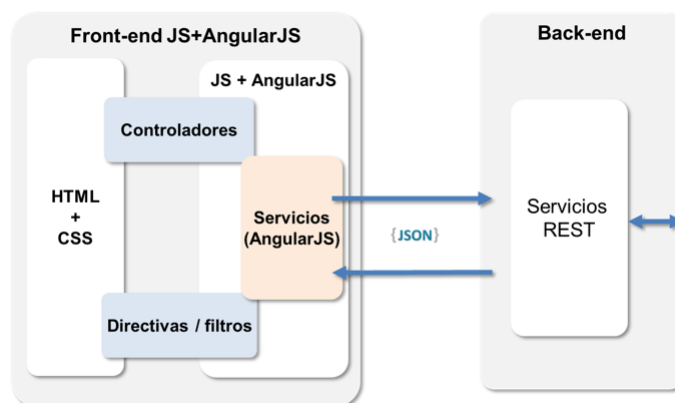


Figura 2.3: Arquitectura del front-end

2.3.2. Arquitectura del back-end

Para el desarrollo del back-end se han utilizado Node.js², Express³, socket.io⁴ y mongoose⁵. Se ha decidido utilizar estas tecnología porque 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.

¹es pequeña fabrica de funciones y objetos inyectada en los controladores

²javascript al lado del servidor

³modulo de Node.js que nos ofrece la posibilidad de desarrollar una REST API

⁴sistema bidireccional dirigido por eventos

⁵modelado de objetos sobre mongoDB

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
```

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 router⁶ 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 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 pull⁷ de ejemplo basado en los usuarios (User based recommender).

Aunque sea un recomendador de ejemplo este está pensado para ser expandido, tanto para otro tipo de recomendadores (como pueden ser los recomendadores de tipo push⁸) 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

⁶el 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

⁷tipo de recomendador en el cual los usuarios solicitan recomendaciones

⁸tipo 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 implementaciones⁹ y otro para realizar las recomendaciones.

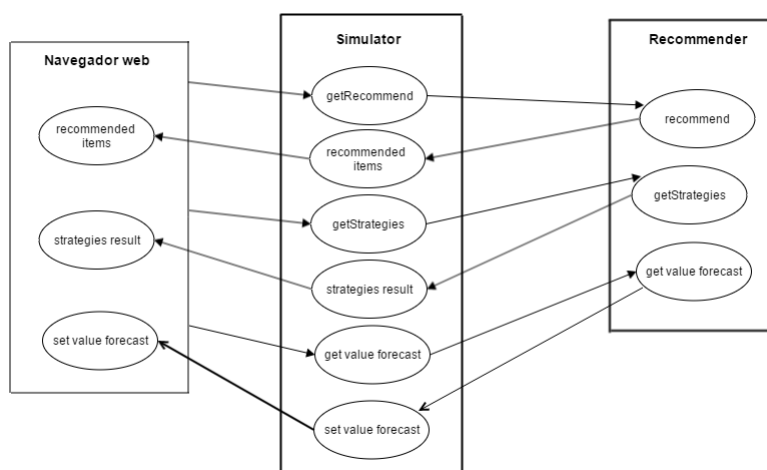


Figura 2.4: Diagrama de eventos

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 diseño nos permite cambiar de estrategia de recomendación en tiempo de ejecución:

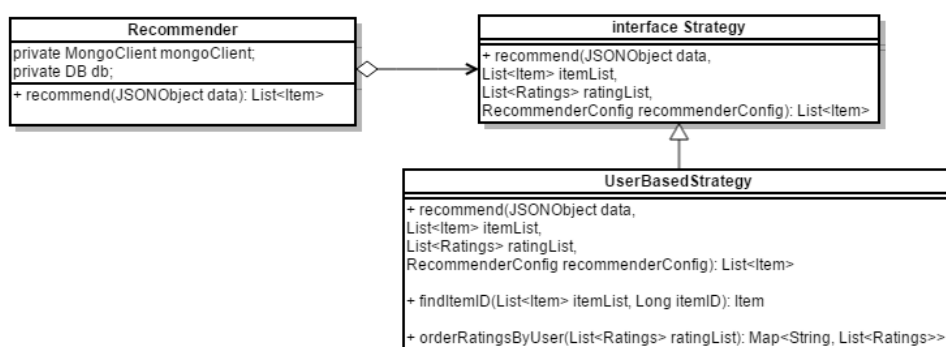


Figura 2.5: Diagrama UML del patrón de diseño de tipo Strategy

⁹en nuestro caso es User based recommender pero existen otros tipos como Item based recommender

2.3.4. Autenticación basada en token

La autenticación es una de las partes más importantes de un sistema. En este desarrollo se ha optado por una autenticación basada en token. En la autenticación basada en token, las cookies y las sesiones no se utilizan. Se emplea un token para autenticar al usuario en cada petición que se hace al servidor. El flujo de control es el siguiente:

1. el usuario provee un nombre de usuario y contraseña en el formulario de login.
2. una vez hecha la petición, se valida el usuario en el back-end mediante una consulta a la base de datos. Si la petición es válida, se crea un token utilizando la información de usuario brindada por la base de datos, y luego se retorna esa información en el encabezado de la respuesta, para así guardar el token en almacenamiento local.
3. se provee la información del token en el encabezado de cada petición para acceder a endpoints restringidos de la aplicación.
4. si el token tomado del encabezado de la petición es válido, se permite al usuario acceder al endpoint especificado, y se responde con JSON o HTML.

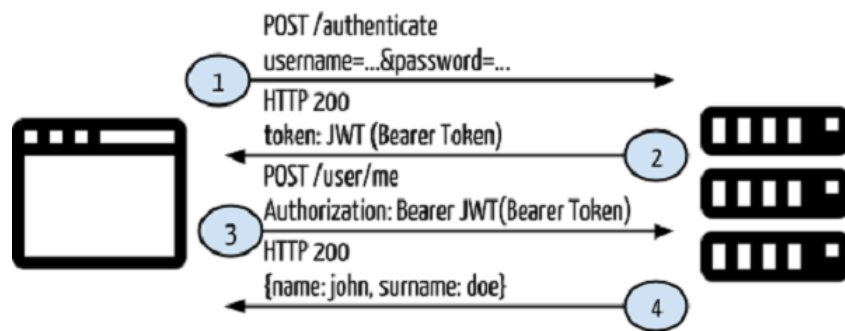


Figura 2.6: Autenticación Basada en Token

Estructura de un token

Un token consiste de tres partes:

- encabezado: es la parte del token que almacena el tipo de token y el método de encriptación, que a su vez está encriptado en base-64.
- carga útil: incluye la información que queremos encriptar.
- firma: consiste de combinaciones del encabezado, carga útil, y clave secreta. La clave secreta debe ser almacenada del lado servidor.

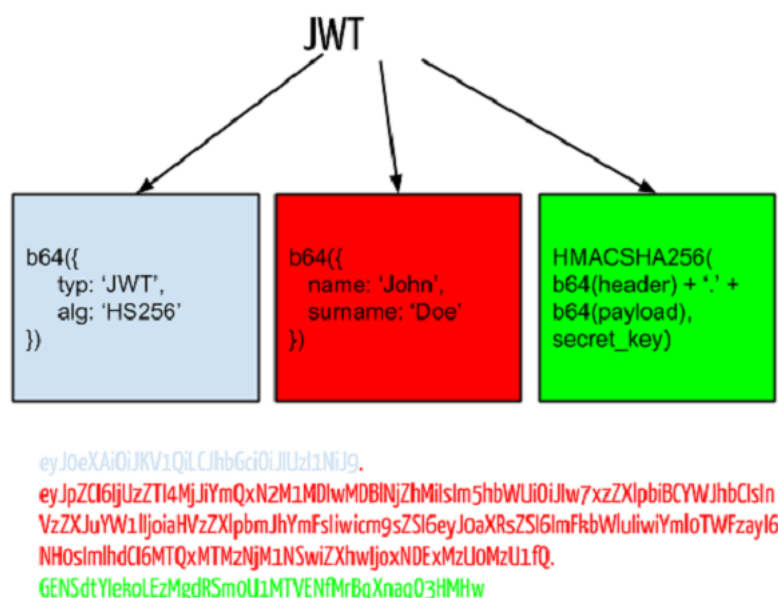


Figura 2.7: Formato del Token

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 2.8). Existen 4 tipos de caminos organizados en forma de árbol:

- búsqueda y gestión (creación, edición y borrados) de mapas y escenas

- simulación de un mapa y escena
- evaluación de un sistema de recomendación
- gestión de tipos de recomendadores, objetos estáticos¹⁰ y dinámicos¹¹
- configuraciones del perfil del usuario

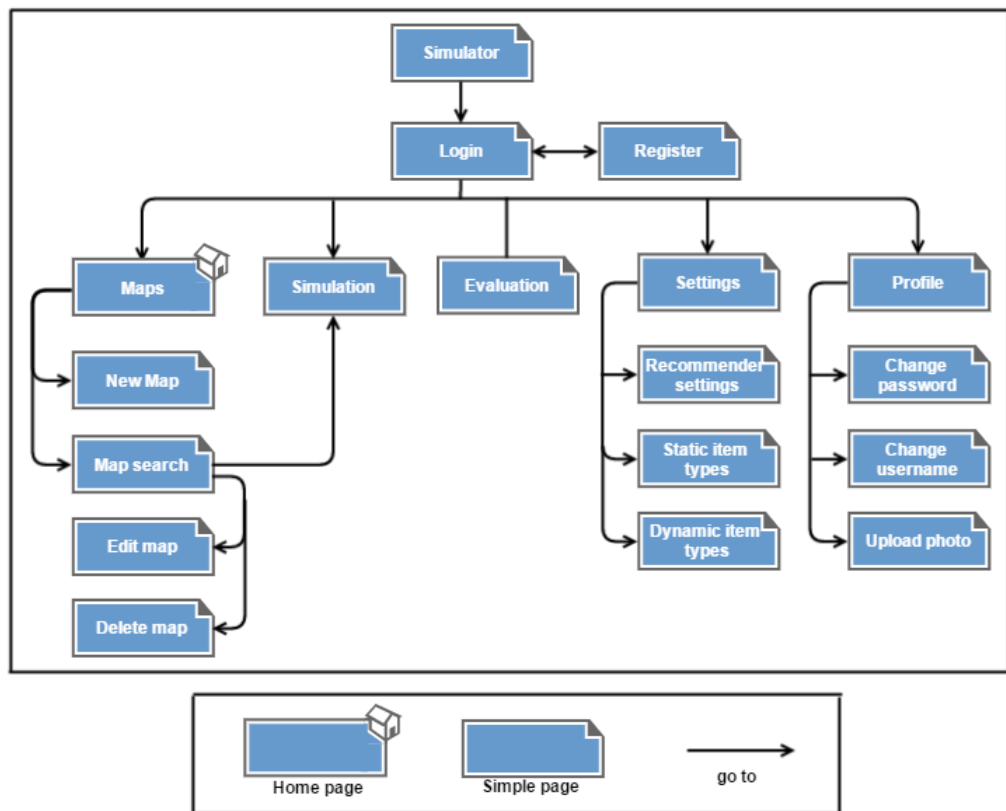


Figura 2.8: Mapa de navegación

¹⁰objetos que no cambian de posición a medida que pasa el tiempo

¹¹objetos que cambian de posición a medida que pasa el tiempo. Tienen una ruta definida durante la creación de la escena

2.5. Movimiento de los objetos móviles: 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). De esta manera conseguimos disminuir el error en el cálculo de las posiciones de los objetos móviles y obtenemos movimientos muy precisos incluso en grafos de movimientos con nodos muy cercanos.

Se podría haber utilizado una técnica mucho más simple como la interpolación pero el inconveniente que esto conlleva es que tenemos que guardar en un buffer todos los estados por que tiene que pasar cada objeto móvil en cada ciclo. Así que a medida que aumenta el número de objetos móviles también aumenta la cantidad de memoria que estamos consumiendo. Esto puede llegar a ser un problema para dispositivos que tienen poca memoria.

Capítulo 3

Explotación

En este capítulo se expone la explotación del simulador de escenarios como método para la evaluación de sistemas de recomendaciones, el trabajo realizado para permitirlo y el rendimiento obtenido del simulador. Para más información consulte el manual del usuarios disponible en los anexos.

3.1. Motivación

Debido a la gran cantidad de información resulta difícil para los usuarios elegir entre todas las alternativas existentes en los resultados mostrados por un sistema. Por ello, para facilitar la elección se utilizan los llamados sistemas de recomendación. Estos sistemas resultan de gran interés para las empresas desde punto de vista económico, ya que hacen llegar a sus cliente recomendaciones de productos relevantes para ellos. Además, también resultan de interés para los usuarios ya que actúan como filtro y les ofrecen información personalizada que les resulta de interés.

No obstante, el diseño de sistemas de recomendación se enfrenta a problema como el arranque en frío (cold start problem) o el problema de opiniones artificiales o manipuladas (spam). Esto conlleva que el sistema de recomendación muestra información no relevante para el usuario y este dejará de confiar en él y abandonarlo.

Por esto surge la necesidad de que los sistemas de recomendaciones puedan evaluarse y calibrarse correctamente. Para que esto pueda llevarse a cabo necesitamos recolectar datos reales de escenarios reales para su posterior análisis. Este análisis consiste en calcular el error cometido por el sistema de recomendación. Una forma de cuantificar el error es con la MAE (Medium Absolute Error) que consiste en restar el valor asignado por el usuario del valor estimado por el sistema de recomendación.

3.2. Definir un escenario realista

En este apartado se expone como definir un escenario realista con datos de restaurantes reales de la ciudad San Luis Potosí de México.

3.2.1. Paso 1: obtener datos de restaurantes reales

Lo primero que vamos a hacer es conseguir datos reales de restaurantes reales. Por esto accedemos al repositorio de Center for Machine Learning and Intelligent Systems y descargamos el siguiente archivo comprimido: <https://archive.ics.uci.edu/ml/machine-learning-databases/00232/RCdata.zip>

Una vez que hayamos descargado y descomprimido el fichero vemos que este contiene varios ficheros csv. A nosotros nos interesan los ficheros `geoplaces2.csv` y `rating_final.csv`. El primero contiene los datos de los restaurantes y es el que vamos a importar al crea un escenario y el segundo contiene datos con valoraciones de diferentes usuarios y el que vamos a usar como datos de entrenamiento para el recomendador de ejemplo desarrollado.

3.2.2. Paso 2: crear un mapa nuevo o editar un mapa existente

Antes de todos tenemos que crear un mapa nuevo o editar uno existente para poder asociarle el escenario que vamos a crear a continuación. Por esto tenemos que usar alguna de las dos opciones:

- crear un mapa nuevo: opción Maps → new map. Para más información por favor consulte el anexo A.8.
- editar un mapa existente: opción Maps → maps search y en la lista de resultados pulsamos sobre el icono de editar imagen. Para más información por favor consulte el anexo A.10.

En este caso se ha optado por crear un nuevo mapa con el nombre México ya que los datos que hemos descargado son de la ciudad San Luis Potosí de México. En la figura 3.1 podemos ver una captura del resultado de creación del mapa:

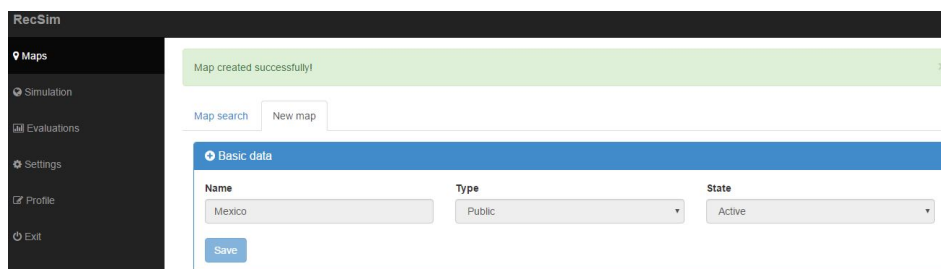


Figura 3.1: Creación de un mapa nuevo en México con datos reales

3.2.3. Paso 3: crear un escenario realista

La creación de escenarios funciona como un asistente de configuración. A continuación se muestra los pasos que hay que realizar para configurar un escenario realista:

Paso 1: definir el nombre del escenario y elegir el recomendador

En el primer paso de la creación de un escenario consiste en poner un nombre del escenario y elegir que recomendador vamos a usar en dicho escenario. En la figura 3.2 podemos una captura de este paso:

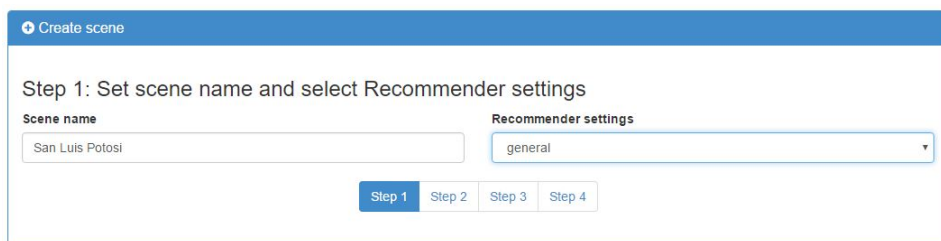


Figura 3.2: Paso 1: definir un nombre y recomendador para el escenario realista

Paso 2: definir los limites del escenario

En segundo paso de la creación del escenario realista consiste en definir los limites geográficos del escenario. Por esto lo primero que tenemos que hacer es usar el buscador para buscar la ciudad donde se va a realizar la simulación. Una vez que hayamos encontrado y seleccionado la ciudad definiremos la esquina superior izquierda ya la esquina inferior derecha. En la figura 3.3 vemos el resultado:

Step 2: Limitations of the scene

Upper left corner: longitude: -101.03500518062935
 Upper left corner: latitude: 22.198777736681336

Lower right corner: longitude: -100.91896208980903
 Lower right corner: latitude: 22.118650356099295

☐ select upper left corner ☒ select lower right corner

City: San Luis Potosí

Places: San Luis Potosí, Estados Unidos Mexico

Step 1 Step 2 Step 3 Step 4

Figura 3.3: Paso 2: definir los limites del escenario realista

Paso 3: importar los datos de los restaurantes

El siguiente paso consiste definir los restaurantes. El simulador dispone de varias opciones mediante las cuales es posible hacer esto. Nosotros vamos a importar los restaurantes del fichero `geoplaces2.csv` que hemos descargado previamente. Antes de cargar los datos el contenido del fichero se pre visualiza en la pantalla (figura 3.4)

Data Preview

placeID	latitude	longitude	the_geom_meter	name	address	city	state	country	fax	zip	alcohol	smoking_area	dress
134999	18.915421	-99.184871	0101000020957F000088568DE356715AC138C0A529FC464441	Kiku Cuernavaca	Revolucion	Cuernavaca	Morales	Mexico	?	?	No_Alcohol_Served	none	info
132825	22.1473922	-100.983092	0101000020957F00001AD016588C4858C1243261274BA54B41	puesto de tacos	esquina santos degollado y leon guzman	s.l.p.	s.l.p.	mexico	?	78280	No_Alcohol_Served	none	info
135106	22.1497088	-100.9760928	0101000020957F000064901F21634858C119AEBF528A34B41	El Rincón de San Francisco	Universidad 169	San Luis Potosí	San Luis Potosí	Mexico	?	78000	Wine-Beer	only at bar	info
132867	23.7528973	-99.1633594	0101000020957F00005D67BCDDE8157C1222A2DC8D84D4941	little pizza Emilio Portes Gil	calle emilio portes gil	victoria	tamaulipas	?	?	?	No_Alcohol_Served	none	info
132813	23.7529035	-99.165076	0101000020957F00008EBA2D86DC8157C194E03B7B504E4941	camila's_mata	ic. Emilio portes gil	victoria	Tamaulipas	Mexico	?	?	No_Alcohol_Served	permitted	info
135040	22.135617	-100.969709	0101000020957F00001B5521898B4A58C15A2AAEFD2CA24B41	Restaurant los Compadres	Camino a Simon Diaz 155 Centro	San Luis Potosí	SLP	Mexico	?	74000	Wine-Beer	none	info

Figura 3.4: Paso 3: definir los restaurantes sobre el escenario

Una vez importados los datos del fichero vemos que se muestra una lista con los restaurantes importados. Podemos ver esta lista con la figura 3.5:

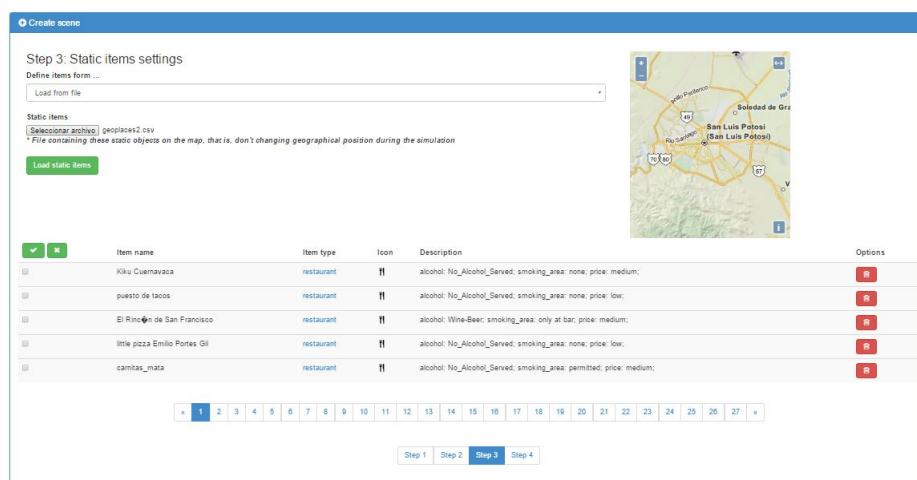


Figura 3.5: Paso 3: lista de restaurantes definidos sobre el escenario

Paso 4: definir los objetos móviles

El ultimo paso consiste en definir los objetos móviles. El simulador dispone de varias opciones mediante cuales se pueden definir objetos móviles. En este caso se han generado automáticamente (en la figura 3.6 podemos ver el resultado).

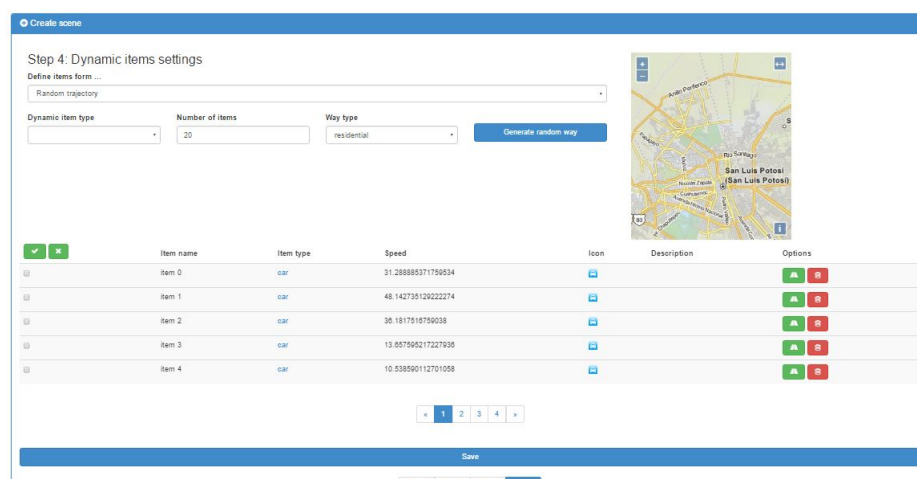


Figura 3.6: Paso 4: lista de objetos móviles

3.3. Simulación y evaluación de un sistema de recomendación

3.3.1. Simulación de un escenario realista

Una vez que hayamos realizado la configuración del escenarios realista vamos a realizar simulaciones con dos usuarios y recoger las valoraciones que han realizado para su posterior evaluación. En la figura 3.7 podemos ver una captura de una simulación realizada:

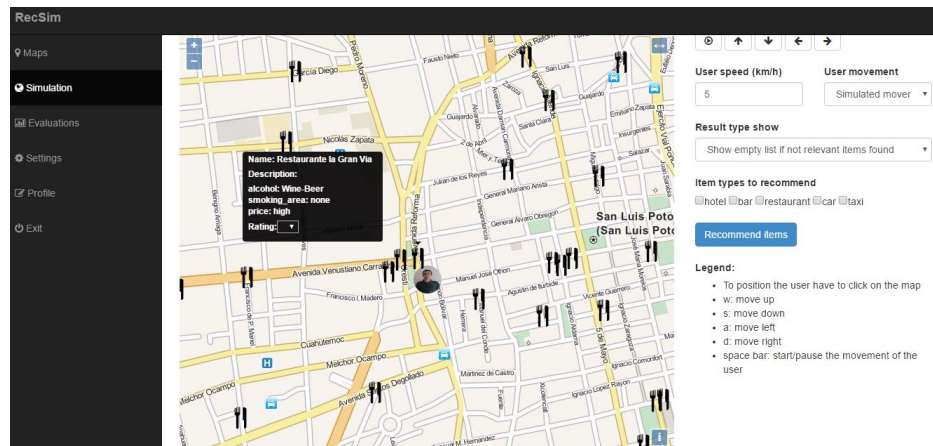


Figura 3.7: Simulación realizada en San Luis Potosí, México

3.3.2. Evaluación de un recomendador

3.4. Rendimiento

El límite teórico máximo de peticiones HTTP de las aplicaciones desarrolladas con Node.js es igual al número máximo de sockets que se puedan crear en el servidor, es decir, un servidor con un Sistema Operativo Linux puede crear 65535 sockets. Pero hay que tener en cuenta que una parte está reservada para el Sistema Operativo. Esto nos deja entre 30.000 - 45.000 sockets que podemos usar. Tenemos que comprobar cómo se comporta el simulador y por lo tanto hemos realizado una prueba de estrés tanto con Linux como con Windows.

3.4.1. Rendimientos con Linux

En este apartado se describe los resultados de las pruebas de estrés realizadas con un Sistema Operativo Linux con las siguientes características:

- Sistema Operativo: Linux v15.10 64 bits
- RAM: 3.3 GB
- CPU: Intel i3 1.8 GHz

Peticiones HTTP	CPU	Memoria	Kbytes/sec	Peticiones/seg	Segundos por petición
50.000	24.8	134MB	1593,86	508,76	0,196555
100.000	30.2	133 MB	1716,46	547,89	0,182517
500.000	45.4	138MB	1199,38	382,84	0,261204

Cuadro 3.1: Pruebas de estrés con 100 usuarios concurrentes

Peticiones HTTP	CPU	Memoria	Kbytes/sec	Peticiones/seg	Segundos por petición
50.000	25.9	147MB	1196,85	382,04	1,308774
100.000	40.6	160MB	1286,86	410,78	1,2172
500.000	44.6	162 MB	n/d	n/d	n/d

Cuadro 3.2: Pruebas de estrés con 500 usuarios concurrentes

Peticiones HTTP	CPU	Memoria	Kbytes/sec	Peticiones/seg	Segundos por petición
50.000	24.3	160 MB	1408,94	449,74	2,223517
100.000	26.3	167MB	n/d	n/d	n/d
500.000	n/d	n/d	n/d	n/d	n/d

Cuadro 3.3: Pruebas de estrés con 1000 usuarios concurrentes

En el caso de la pruebas con 500 usuarios concurrentes nos da dado el error `apr socket recv Connection timed out (110)` y se han completado 105.195 peticiones HTTP en la prueba con 500.000 peticiones HTTP. En el caso de la prueba con 1000 usuarios concurrentes nos ha dado el mismo fallo que en el caso de los 500 usuarios concurrentes y se han completado 54.599 peticiones HTTP en la prueba con 100.000 peticiones HTTP.

En el resto de casos las pruebas de carga han sido satisfactorias. Observamos que el peor de los casos se ha obtenido una media de 2,2 segundos por petición HTTP en el caso que tenemos 1000 usuarios concurrentes que han generado 50.000 peticiones HTTP.

3.4.2. Rendimientos con Windows

En este apartado se describe los resultados de las pruebas de estrés realizadas con un Sistema Operativo Windows con las siguientes características:

- Sistema Operativo: Windows 10 Home 64 bits
- RAM: 8 GB
- CPU: Intel i3 1.8 GHz

```
1 slavcho@ubuntu:~/testing$ ab -g resultados4.csv -n 50000 -c 100 http
  ://192.168.1.102:81/
2 This is ApacheBench, Version 2.3 <$Revision: 1638069 $>
3 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.
  net/
4 Licensed to The Apache Software Foundation, http://www.apache.org/
5
6 Benchmarking 192.168.1.102 (be patient)
7 Completed 5000 requests
8 Completed 10000 requests
9 Completed 15000 requests
10 apr_socket_recv: Connection timed out (110)
11 Total of 19219 requests completed
12 slavcho@ubuntu:~/testing$
```

```
1 slavcho@ubuntu:~/testing$ ab -g resultados4.csv -n 50000 -c 500 http
  ://192.168.1.102:81/
2 This is ApacheBench, Version 2.3 <$Revision: 1638069 $>
3 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.
  net/
4 Licensed to The Apache Software Foundation, http://www.apache.org/
5
6 Benchmarking 192.168.1.102 (be patient)
7 apr_socket_recv: Connection refused (111)
8 Total of 558 requests completed
```

```
1 slavcho@ubuntu:~/testing$ ab -g resultados4.csv -n 100000 -c 500 http
  ://192.168.1.102:81/
2 This is ApacheBench, Version 2.3 <$Revision: 1638069 $>
3 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.
  net/
4 Licensed to The Apache Software Foundation, http://www.apache.org/
5
6 Benchmarking 192.168.1.102 (be patient)
7 apr_socket_recv: Connection refused (111)
8 Total of 524 requests completed
```

En las pruebas realizadas observamos que nos ha dado fallo en todas las pruebas. En el caso de la prueba con 100 usuarios concurrentes 50.000 peticiones HTTP se ha completado 19.219 de las peticiones HTTP. En el

caso de la prueba con 500 usuarios concurrentes y 50.000 peticiones HTTP se han completado 558 peticiones HTTP. En el caso de la prueba con 500 usuarios concurrentes y 100.000 peticiones HTTP se han completado 524 peticiones HTTP.

3.4.3. Conclusiones de las pruebas de estrés

Como conclusión podemos determinar que el Sistema Operativo Linux se comporta bastante mejor que el Windows para esta aplicación. Por lo tanto recomiendo la utilización de un servidor Linux para el simulador de escenarios para aprovechar mejor tanto la aplicación como el hardware.

Capítulo 4

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 centrados únicamente en los aspectos más importantes.

4.1. 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 divertían. El videojuego consistiría en el cumplimiento de misiones en una ciudad basándose en algún videojuego de aventuras como el videojuego Paperboy del año 1985 donde un chico reparte periódicos en una ciudad.

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 límite.

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.

4.1.1. Primer ciclo de la espiral: formación con las herramientas a utilizar

En la tabla 4.1 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 extraíbles 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 videojuego 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 4.1: Tareas del primer ciclo de la espiral

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).

4.1.2. Segundo ciclo de la espiral: diseñar el mapa y desarrollar la IA

En la tabla 4.2 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 4.2: Tareas segundo ciclo de la espiral

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.

4.1.3. Tercer ciclo de la espiral: integración con OpenStreet-Map

En la tabla 4.3 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 4.3: Tareas tercer ciclo de la espiral

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 decidido 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.

4.1.4. Cuarto ciclo de la espiral: instalación y configuración del entorno de trabajo para el desarrollo del simulador RecSim

En la tabla 4.4 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 4.4: Tareas cuarto ciclo de la espiral

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.

4.1.5. Quinto ciclo de la espiral: desarrollo del simulador RecSim

En la tabla 4.6 podemos encontrar las tareas que han sido definidas para esta etapa:

Código	Descripción
RQ-20	Definir los requisitos funcionales y no funcionales de la aplicación
RQ-21	Desarrollar el front-end
RQ-22	Diseñar 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 4.5: Tareas quinto ciclo de la espiral

Como conclusión de esta etapa obtenemos el siguiente resultado:

- la parte más 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.

4.1.6. Sexto ciclo de la espiral: documentar el trabajo realizado

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

4.2. 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
Pruebas de carga	11.5
Total	410

Cuadro 4.6: Separación por horas de las distintas fases

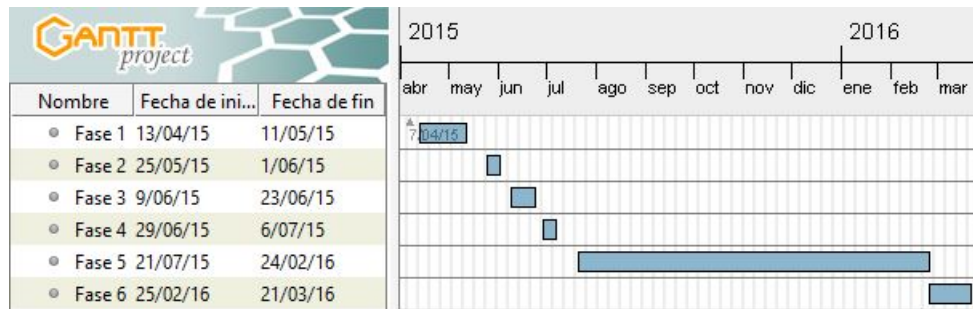


Figura 4.1: Diagrama de Gantt de las distintas fases del proyecto

Capítulo 5

Conclusiones

En este capítulo se explican los resultados que se han obtenido de este Trabajo Fin de Grado, los rendimientos del simulador, el trabajo futuro y una valoración personal.

5.1. Resultados

A lo largo de este Trabajo Fin de Grado se ha desarrollado un simulador de escenarios con usuarios móviles para la evaluación de algoritmos de recomendaciones. El simulador puede ser usado de forma cooperativa por varias personas a través de cualquier dispositivo con conexión a Internet y un navegador web. Cuenta con escenarios basados en datos reales obtenidos a través del servicio de mapas de OpenStreetMap y permite integrar de un recomendador externo.

Como se puede comprobar a continuación, se han cumplido todos los objetivos marcados inicialmente en la propuesta del Trabajo Fin de Grado:

- Se ha desarrollado un simulador de escenarios con usuarios móviles que cuenta con mapas de ciudades con objetos móviles y estáticos.
- Los mapas de las ciudades utilizados en las simulaciones están creados a partir de datos reales obtenidos a través del servicio de mapas de OpenStreetMap.
- Los usuarios puedan crear, editar, borrar y configurar los mapas y escenarios del simulador.
- Se ha desarrollado un sistema bidireccional basada en eventos que permite integrar un recomendador externo y realizar simulaciones en tiempo real de tal forma que los eventos de los usuarios se reflejen en los dispositivos conectados al mismo mapa y escena.

5.2. Trabajos futuros

A continuación se proponen algunas posibles mejoras futuras:

- implementar lo necesario para simular eventos sobre el mapa como pueden ser las calles cortadas, ofertas de descuentos en algún local cierto día o días de la semana etc.
- implementar lo necesario para exportar los datos del mapa: tanto el grafo de carreteras como la configuración de los objetos estáticos y dinámicos.
- implementar lo necesario para importar datos exportados.
- implementar lo necesario para que el servidor web Node.js pueda ejecutarse sobre multiples hilos aprovechando mejor las CPUs actuales. Actualmente solo se ejecuta sobre un proceso de un solo hilo y para escalar el sistema existe la necesidad de clusterizar el servidor web.
- utilizar algún framework de Javascript que nos permita desarrollar pruebas automaticas tanto al lado del cliente como al lado del servidor.

5.3. Valoración personal

El trabajo realizado ha sido muy satisfactorio ya que me ha aportado muchos conocimientos relacionados con la rama de Sistemas de Información en la que he estado especializandome y además me ha permitido profundizar en muchas tecnologías como Node.js y Angular.js en las que he estado interesado por el auge que están sufriendo en la actualidad. Además creo los conocimientos adquiridos me dan valor añadido como futuro profesional del sector de las tecnologías de la información.

Aún así durante el tiempo de desarrollo de todo el proyecto existían preguntas que he estado haciendome continuamente. ¿Si las tecnologías y decisiones que he tomado han sido las adecuadas? ¿Es fácil de implementar y sin perder tiempo? ¿Sería fácil de mantener el sistema en el futuro? ¿Las tecnologías elegidas tendrían algún limite que no permita o dificulte la implementación de ciertas funcionalidades en el futuro?

Como conclusión personal he obtenido que no existe la perfección y de lo complicado que es desarrollar una buena aplicación. Lo que realmente importa es ser constante y comprometido con el trabajo que se está haciendo y aprender a ser autodidacta.

Bibliografía

- [1] GitHub. Github - repositorio de código del proyecto. <https://github.com/slavcho87/Simulator>.
- [2] Unity Technologies. Unity - game engine. <https://unity3d.com/es>.
- [3] Unity Technologies. Unity - video tutoriales. <https://unity3d.com/es/learn/tutorials>.
- [4] Esri. Cityengine - 3d modeling software for urban environments. <http://www.esri.com/software/cityengine>.
- [5] Google. Angular.js - javascript mvw framework. <https://angularjs.org/>.
- [6] Node.js Foundation. Node.js - javascript server-side framework. <https://nodejs.org/en/>.
- [7] Jesús Conde. Angular.js - video tutoriales. <https://www.youtube.com/playlist?list=PLEtcGQaT56cgHfdvGguisToK90z321pRl>.
- [8] Jesús Conde. Node.js - video tutoriales. <https://www.youtube.com/playlist?list=PL38CA7BD8CB5F3FF9>.
- [9] Hüseyin Babal. Autenticación basada en token con angularjs y nodejs. <http://code.tutsplus.com/es/tutorials/token-based-authentication-with-angularjs-nodejs-cms-22543>.
- [10] Movable Type Ltd. Calcular distancia entre dos coordenadas geográficas. <http://www.movable-type.co.uk/scripts/latlong.html>.
- [11] Blog Doble Vías Transporte e ingeniería. Dirección de una línea (rumbo y azimuth). <https://doblevia.wordpress.com/2007/07/25/direccion-de-una-linea-rumbo-y-azimut/>.
- [12] Blog Doble Vías Transporte e ingeniería. Cálculos de navegación por estima. <http://mercatorlab.com/downloads/loxodromia.java>.
- [13] MIT. Socket.io - framework de node.js. <http://socket.io/>.

- [14] MIT. Socket.io - chat demo. *<http://socket.io/demos/chat/>*.
- [15] Software Freedom Conservancy. Git - sistema de control de versiones.
<https://git-scm.com/>.
- [16] Adobe Systems Incorporated. Brackets.io - editor de código.
<http://brackets.io/>.
- [17] The Eclipse Foundation. Eclipse - editor de código java.
<https://eclipse.org/>.

Anexos

Anexos A

Manual del usuario

A.1. Visión general

A.1.1. Introducción

En el siguiente documento se recoge una descripción del funcionamiento del simulador de escenarios con usuarios móviles para la evaluación de algoritmos de recomendaciones.

A.1.2. ¿Qué es el simulador de escenarios con usuarios móviles?

El simulador de escenarios con usuarios móviles es una herramienta que trata de simular distintos tipos de algoritmos de recomendaciones en el entorno de una ciudad real con el fin de evaluar su correcto funcionamiento.

Se trata de un sistema multiusuario donde distintos tipos de usuarios se conectan y se mueven en una ciudad real (el entorno es configurado de antelación). El recomendador tiene en cuenta distintos tipos de parámetros: desde sus posiciones geográficas hasta sus perfiles y preferencias.

A.1.3. Tipos de tecnologías utilizadas

Esta herramienta consta de dos partes: la primera es el simulador y la segunda es el recomendador.

El Simulador está desarrollado con nodejs, sockets-io, angularjs, bootstrap y como base de datos utiliza mongodb. El recomendador está desarrollado con java. La integración entre el navegador (cliente), simulador y recomendador está realizada mediante un sistema de eventos bidireccional.

les (sockets-io). De esta forma conseguimos comunicar todas la partes del sistema en tiempo real.

A.1.4. Instalación

Paso 1: Instalar mongoDB

Lo primero que tenemos que hacer es instalar mongodb. Los pasos para la instalación de mongodb depeden del tipo de sistema operativo que disponemos. Por esto no vamos a entrar en detalle de como se instala y vamos a seguir el tutorial disponible en la web oficial:

- Linux: <https://docs.mongodb.org/manual/administration/install-on-linux/>
- Windows: <https://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/>
- Mac OS: <https://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/>

Paso 2: Instalar Node.js y NPM

Vamos en la web oficial de nodejs (<https://nodejs.org>) y descargamos e instalamos la version v0.12.4. En el caso de Windows la instalación es igual que la de cualquier otro programa. Para la instalación en otros SO visitar <https://nodejs.org/en/download/>.

A continuación tenemos que instalar el gestor de paquetes NPM v2.10.1. En el caso de Windows vamos en la web oficial (<https://nodejs.org/en/download/>) y nos descargamos e instalamos el ejecutable. En el caso de Linux ejecutamos el siguiente comando en la consola:

```
1 sudo apt-get install npm
```

Paso 3: Instalar git

Git es un sistema distribuido de control de versiones. Para la instalación de este nos descargamos el ejecutable de <https://git-scm.com/downloads> y seguimos los pasos que nos indica este.

Paso 4: Clonar el proyecto de github e instalarlo

A continuacion tenemos que clonar el proyecto del Simulador de github. Para esto abrimos una consola y nos situamos en el directorio donde queremos clonar el proyecto. A continuacion ejecutamos el siguiente comando:

```
1 git clone https://github.com/slavcho87/Simulator
```

Vemos que se ha creado un directorio llamado Simulator. Lo primero que tenemos que hacer es bajarnos todas las dependencias del proyecto. Por esto ejecutamos el siguiente comando:

```
1 npm install
```

Una vez que nos hemos clonado el proyecto y descargado las dependencias de este podemos arrancar el servidor mediante el siguiente comando:

```
1 npm start
```

Paso 4.1: Configuraciones básicas del simulador En el fichero baseConfig.json disponemos de las siguientes basicas para el simulador como el puerto donde se ejecuta el servidor y la localizacion de la base de datos. Si editamos el fichero baseConfig.json veremos que tiene el siguiente contenido:

```
1 {  
2   "port": 81,  
3   "locationDB": "localhost",  
4   "nameDB": "simulator"  
5 }
```

Las variables del fichero baseConfig.json tienen el siguiente significado:

- port: es el puerto donde se va a ejecutar el servidor
- locationDB: es la localizacion donde se va a ejecutar mongodb.
- nameDB: es el nombre del esquema de la base de datos donde nos conectamos.

Dichas configuraciones son importantes ya que de esta forma tenemos la opción de llevarnos la base de datos en un servidor diferente para darle más potencia.

Paso 4.2: Posibles problemas durante la ejecución del simulador

Se puede dar el caso que al intentar arrancar el simulador nos de el siguiente error:

```
1 Error: listen EACCES 0.0.0.0:81
2 at Object.exports._errnoException (util.js:870:11)
3 at exports._exceptionWithHostPort (util.js:893:20)
4 at Server._listen2 (net.js:1218:19)
5 at listen (net.js:1267:10)
6 at Server.listen (net.js:1363:5)
```

El error EACCES ocurre cuando no tenemos suficientes privilegios sobre el puerto donde estamos lanzando el servidor Node.js. La solución depende del Sistema Operativo que estamos usando. En el caso de Windows tenemos que cambiar el puerto donde lanzamos el servidor siempre y cuando estemos usando un usuario que tenga suficientes privilegios. En el caso de Linux tenemos que lanzar el servidor con el comando sudo delante de la siguiente manera:

```
1 slavcho@ubuntu:~/Simulator/scripts$ sudo ./ejecutarSimulador.sh
```

Paso 5: Instalación del recomendador

Paso 5.1: Instalar Apache maven Antes de todo tenemos que instalar Apache maven que es un gestor de paquetes. Por lo tanto vamos en la web oficial (<https://maven.apache.org/>) y descargamos y descomprimos el fichero comprimido. A continuación tenemos que añadir en la variable PATH la dirección de la carpeta donde hemos descomprimido maven. Para ver que maven se haya instalado correctamente ejecutamos el siguiente comando:

```
1 mvn -v
```

De esta forma comprobamos la versión de maven que tenemos instalado. Tenemos que ver una salida como la siguiente:

```
1 Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1;
   2014-12-14T18:29:23+01:00)
2 Maven home: C:\maven
3 Java version: 1.7.0_79, vendor: Oracle Corporation
4 Java home: C:\Program Files (x86)\Java\jdk1.7.0_79\jre
5 Default locale: es_ES, platform encoding: Cp1252
6 OS name: "windows 8.1", version: "6.3", arch: "x86", family: "windows"
```

Paso 5.2: Compilar el recomendador Una vez que hayamos instalado maven correctamente tenemos que compilar el recomendador. Por esto ejecutamos el fichero `compileRecommender` que se encuentra en la carpeta `scripts`. Existen dos versiones: uno para Windows y otro para Linux.

Paso 5.3: Configuraciones básicas del recomendador Si editamos el fichero `configs/baseConfig.txt` podemos ver que tiene el siguiente formato:

```
1 <config>
2 <host>http://localhost</host>
3 <port>81</port>
4 <hostMongo>localhost</hostMongo>
5 <portMongo>27017</portMongo>
6 <nameDB>simulator</nameDB>
7 </config>
```

- `host`: es la dirección donde se ejecuta el simulador. En el ejemplo este se está ejecutando en local
- `port`: es el puerto donde se ejecuta el simulador. En el ejemplo este se ejecuta en el puerto 81
- `hostMongo`: es la dirección de la base de datos. En el ejemplo esta se está ejecutando en local
- `portMongo`: es el puerto donde se ejecuta la base de datos.
- `nameDB`: es el nombre del esquema de la base de datos

Paso 5.4: Ejecutar el recomendador Para ejecutar el recomendador tenemos que ejecutar el fichero `executeRecommender` que se encuentra en la carpeta `scripts`. Existe dos versiones de este fichero: uno para Windows y otro para Linux.

A.1.5. Primeros pasos

El primero paso al instalar el simulador es crear nuestro usuario (apartado A.5). A continuación vamos en `Settings` y realizamos las configuraciones realizadas en los capítulos A.2, A.3 y A.4.

Se trata de crear nuevas configuraciones para el recomendador y crear los tipos de objetos dinámicos y estáticos que usaremos posteriormente en la creación de mapa (apartado A.8) y escenas (apartado A.9).

A.2. Configuración del recomendador

Para configurar los parámetros del recomendador primero tenemos que estar autenticados con nuestro usuario. Una vez autenticados vamos en Settings → Recommender settings y vemos la siguiente pantalla:

Recommender set name	Recommender type	Strategy type	Maximum distance to go (meters)	Visibility radius (meters)	Number of items to recommend	Minimum score for recommending an item	Options
general	pull	User based recommender	1200	1200	10	0	Options +

Figura A.1: Configuración del recomendador

En este formulario tenemos que introducir los siguientes datos:

- Pool name: es nombre que vamos a dar al conjunto de parámetros
- Recommender type: es el tipo de recomendador. Podemos elegir entre pull (el usuario solicita una recomendación) o push (el recomendador realiza recomendaciones sin que el usuario lo haya solicitado)
- Strategy type: indicar el tipo de implementación que queremos para el recomendador
- Maximum distance to go (meters): es la distancia máxima (en metros) que está dispuesto a recorrer el usuario
- Visibility radius (meters): radio (en metros) de visibilidad del usuario. Los ítems que están fuera de este radio son invisibles para el usuario
- Number of items to recommend: número máximo de ítems que va a recomendar el recomendador cada vez
- Minimum score for recommending an item: puntuación mínima para que un ítem sea recomendado

A.3. Configuración de los objetos estáticos

Para crear un nuevo tipo de objeto estático primero tenemos que estar autenticados con nuestro usuario. Una vez autenticados vamos en Settings → Static item type y vemos la siguiente pantalla:

Name	Icon	Options
hotel		Options -
bar		Options -
restaurant		Options -

Figura A.2: Configuración de los objetos estáticos

En este formulario tenemos que introducir los siguientes datos:

- Name: es el nombre del tipo del objeto estático
- Icon: icono del tipo del objeto estático

Por debajo del formulario de creación nuevos tipos de objetos estáticos aparecerá la lista con todos los tipos de objetos estáticos que han sido creados. En el apartado opciones podemos gestionar cada uno de estos objetos.

A.4. Configuración de los objetos dinámicos

Para crear un nuevo tipo de objeto dinámicos primero tenemos que estar autenticados con nuestro usuario. Una vez autenticados vamos en Settings → Dynamic item type y vemos la siguiente pantalla:

Figura A.3: Configuración de los objetos dinámicos

En este formulario tenemos que introducir los siguientes datos:

- Name: es el nombre del tipo del objeto dinámicos
- Icon: icono del tipo del objeto dinámicos

Por debajo del formulario de creación nuevos tipos de objetos dinámicos aparecerá la lista con todos los tipos de objetos dinámicos que han sido creados. En el apartado opciones podemos gestionar cada uno de estos objeto.

A.5. Crear un un nuevo usuario

Para registrar un nuevo usuario lo que tenemos que hacer es pinchar en el enlace Register en la página lo login. Se nos muestra la siguiente pantalla:

Figura A.4: Crear un nuevo usuario

En este formulario tenemos que añadir el nombre del usuario, su contraseña y subir su imagen. Esta imagen aparecerá en el mapa del simulador.

Una vez que hayamos creado el usuario tenemos que pulsar el botón "Go to login" para volver en la pantalla de login para introducir el nombre de nuestro usuario su contraseña.

A.6. Actualizar el perfil de un usuario

Para realizar cambios en el nombre del usuario, cambiar la imagen del perfil o cambiar la contraseña tenemos que ir en el menú Perfil:

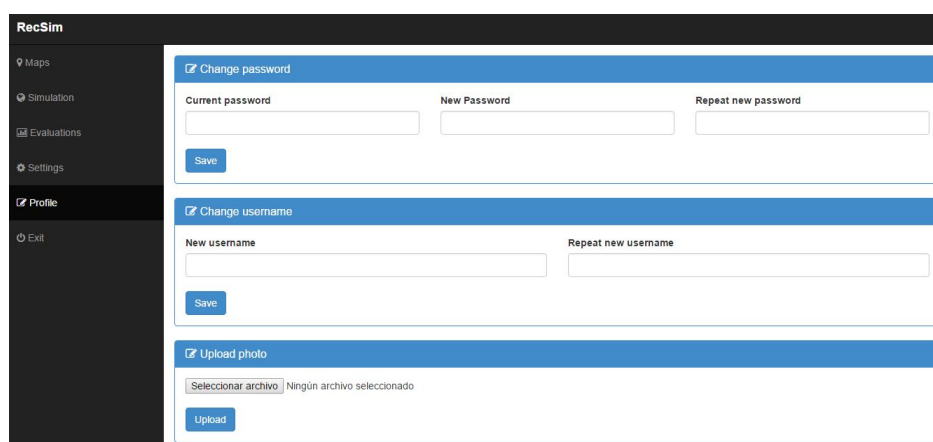


Figura A.5: Actualizar el perfil de un usuario

Vemos que hay 3 formularios:

- uno para cambiar el nombre del usuario
- otro para cambiar la contraseña del usuario
- otro para cambiar la imagen del perfil del usuarios

Al realizar cualquier cambio el el sistema nos informara si la acción ha salido bien o no.

A.7. Búsqueda de mapas

La opción Maps → map search nos permite buscar mapas a los cuales queremos conectarnos para realizar alguna simulación:

The screenshot shows the 'RecSim' application interface. On the left is a dark sidebar with navigation links: Maps, Simulation, Evaluations, Settings, Profile, and Exit. The main content area has two tabs: 'Map search' (selected) and 'New map'. Below the tabs is a 'Find a map' form with the following fields: 'Name' (text input), 'Type' (dropdown menu), 'State' (dropdown menu), 'City' (text input), 'Creation start date' (text input with format 'dd/mm/yyyy'), and 'Creation end date' (text input with format 'dd/mm/yyyy'). There is a checkbox labeled 'search only my maps' and a 'Search' button. Below the form is a 'Search Results' section with a table. The table has columns: Name, Type, State, Creation date, and Actions. One result is shown: 'Zaragoza', 'public', 'active', '11/04/2016'. The Actions column contains three icons: a green plus, a blue share, and a red delete. At the bottom of the results is a pagination control showing '1'.

Figura A.6: Búsqueda de mapas

En la imagen vemos que disponemos de distintos tipos de filtros para la búsqueda de mapas. Estos filtros son: por nombre, por tipo, por estado, por ciudad, por fechas de creación o solo buscar solo mis mapas. Si no introducimos ningún filtro devolverá todos los mapas que están creados en el sistema.

A.8. Crear un nuevo mapa

Para crear un nuevo mapa vamos en Maps → new map y vemos el siguiente formulario:

The screenshot shows the 'RecSim' application interface. On the left is the same dark sidebar as in Figure A.6. The main content area has two tabs: 'Map search' and 'New map' (selected). Above the 'New map' tab is a green notification bar that says 'Map created successfully!'. Below the tabs is a 'Basic data' form with the following fields: 'Name' (text input with 'Barcelona'), 'Type' (dropdown menu with 'Public'), and 'State' (dropdown menu with 'Active'). There is a 'Save' button at the bottom left of the form.

Figura A.7: Crear un nuevo mapa

Tenemos que rellenar los siguientes datos:

- Name: es el nombre que queremos dar al mapa.
- Type: tipo que mapa. Podemos elegir entre publico (puede conectarse cualquier usuarios) y privado (solo puede conectarse el que la ha creado).

- State: estado del mapa. Podemos elegir entre activa (que el mapa está disponible para realizar simulaciones) y borrador (el mapa todavía no está disponible para realizar simulaciones)

Una vez que hayamos rellenado el formulario pulsamos en el botón Save y el sistema nos informará si el guardado ha salido con éxito o no. Si se ha guardado con éxito por debajo de este formulario aparece el formulario de creaciones de escenas y una lista de las escenas actuales. Lo normal es que la lista de escenas creadas aparezca vacía ya que todavía no hemos creado ninguna escena. Para ver los detalles de como crear una escena consultar el capítulo A.9.

A.9. Crear una nueva escena

Una vez que hayamos creado el mapa podemos empezar a crear las escenas. La creación de escenas se realiza en 4 pasos que vamos a ver a continuación.

A.9.1. Paso 1: Elegir el nombre de la escena y el recomendador a utilizar

En el primer paso tenemos que escoger un nombre para la escena que estamos configurando y elegir el tipo de recomendador que queremos usar en esta escena.

Figura A.8: Paso 1: elegir nombre de la escena y el recomendador

A.9.2. Paso 2: Límites de la escena

En este paso tenemos que buscar la ciudad donde se va a realizar la simulación y cuales son los límites de la escena.

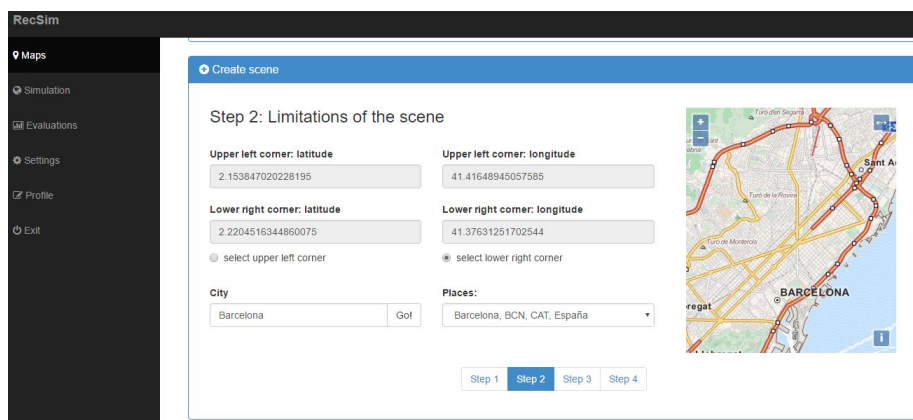


Figura A.9: Paso 2: Limites de la escena

Para buscar la ciudad donde se va a realizar la simulación tenemos poner cual es su nombre en la casilla City y pulsar el botón Go!. Vemos que al lado nos aparece un desplegable con una lista de resultados. Elegimos uno de ellos y vemos que el mapa que aparece al lado se actualiza.

Para definir los límites de la escena tenemos que establecer cual es la esquina superior izquierda y la esquina inferior derecha. Para definir la esquina superior izquierda seleccionamos el select button select upper left corner y hacemos click en el mapa para definir donde estará dicha esquina. Veremos que las cajas de texto Upper left corner: latitude y Upper left corner: longitude aparecerán las coordenadas geográficas de dicha esquina (longitud y latitud).

Para definir la esquina inferior derecha seleccionamos la opción select lower right corner y hacemos click en el mapa para definir donde estará dicha esquina. Cuando hayamos seleccionado la esquina inferior derecha vemos que en las cajas de texto Lower right corner: latitude y Lower right corner: longitude aparecerán las coordenadas geográficas de dicha escena.

A.9.3. Paso 3: Configurar los objetos estáticos

Este paso consiste en definir cuales son los objetos estáticos y sus posiciones. Para definirlos disponemos de dos opciones: importarlos desde un fichero con formato JSON y definirlos manualmente.

Paso 3.1: importar los objetos estáticos

Para importar los objetos estáticos seleccionamos la opción load from file y nos aparecerá un formulario donde podemos elegir cual es el fichero que

queremos importar. Antes de que se carguen los datos se nos previsualiza el contenido del fichero que queremos cargar:

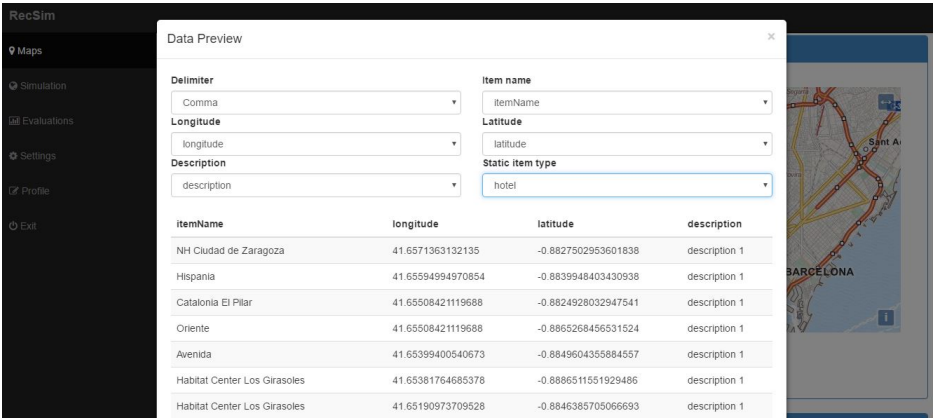


Figura A.10: Paso 3.1: Previsualizar los objetos estáticos a importar

En esta ventana tenemos que vincular los atributos del fichero a los atributos de nuestro sistema. Para cada atributo de nuestro sistema tenemos un select y en este select salen los atributos del fichero. Tenemos que seleccionar que atributo del fichero a que atributo de nuestro sistema queremos vincular. Por ultimo tenemos que elegir que tipo de objeto estático queremos asignar a los datos que estamos importando. Una vez importados los objetos nos aparecerá un table con todos los objetos importados.

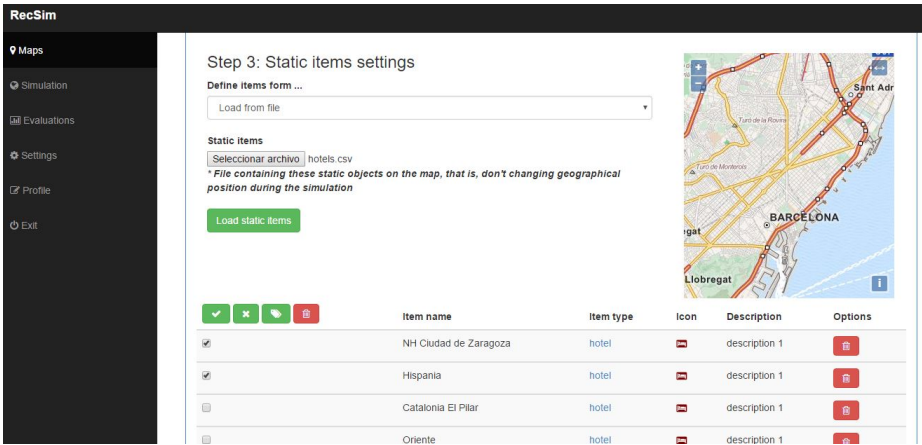


Figura A.11: Paso 3.1: importar los objetos estáticos desde un fichero

Una vez importados los objetos vemos que disponemos una serie de opciones que nos permiten gestionar los objetos importados. Podemos seleccionar un subconjunto y todos los objetos y cambiarles el tipo o borrarles. Esto se

hace con las opciones que tenemos en la cabecera de la tabla.

Paso 3.2: definir los objetos estáticos manualmente

Para definirlos de forma manual tenemos que seleccionar la opción set manually y nos aparecerá un formulario con el tipo de ítem que queremos definir, el nombre que queremos asignarle. Para definir cuales son sus coordenadas geográficas tenemos que seleccionar la posiciones directamente sobre el mapa. A continuación pulsamos sobre el botón Save Item y nos aparecerá la misma tabla que el en paso 3.1:

Figura A.12: Paso 3.2: definir los objetos estáticos manualmente

A.9.4. Paso 4: Configurar los objetos dinámicos

El últimos paso consiste en definir los objetos dinámicos y sus rutas. Disponemos de 3 posibles maneras para definirlos: importándolos desde un fichero con formato JSON, definirlos manualmente, generarlos aleatoriamente.

Paso 4.1: importar los objetos dinámicos

Para importar los objetos dinámicos desde un fichero tenemos que seleccionar la opción load from file y nos aparecerá un formulario que nos permite seleccionar el fichero a importar. Antes de cargar los objetos dinámicos podemos previsualizar los datos y vincular los atributos del fichero a los atributos del sistema. Esto se realiza mediante los selects que observamos en la siguiente pantalla:

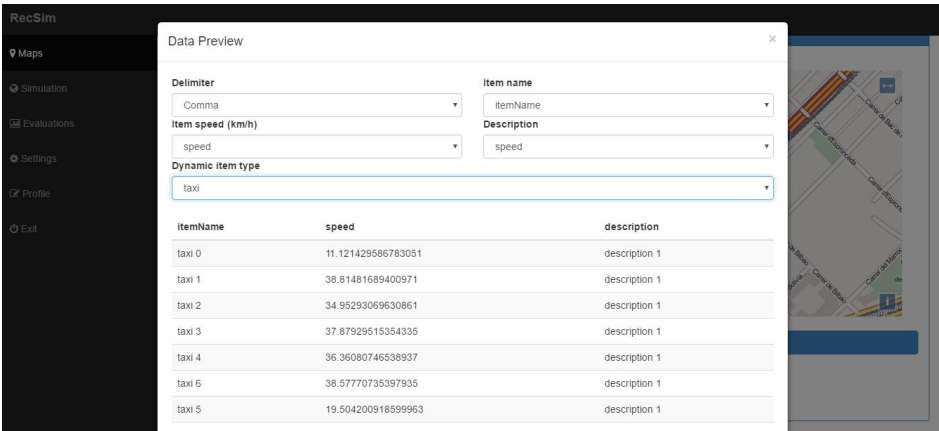


Figura A.13: Paso 4.1: importar los objetos dinámicos

Una vez que hayamos importado los ficheros vemos una tabla parecida que tabla del paso 3.1. La diferencia es que aquí disponemos de más opciones. Entre estas opciones son las de generar las rutas a un subconjunto a todos los datos, cambiar el tipo de objeto dinámico y borrarlos objetos seleccionados.

Paso 4.2: definir los objetos dinámicos manualmente

Para definir los objetos dinámicos manualmente seleccionamos la opción set manually nos aparecer un formulario en el cual tenemos que elegir el tipo de objeto dinámico, el nombre que queremos asignarle y su velocidad. Para definir la trayectoria de este tenemos que seleccionar los nodos del grafo de movimiento directamente en el mapa y nos aparecerá una tabla que contiene dichos nodos:

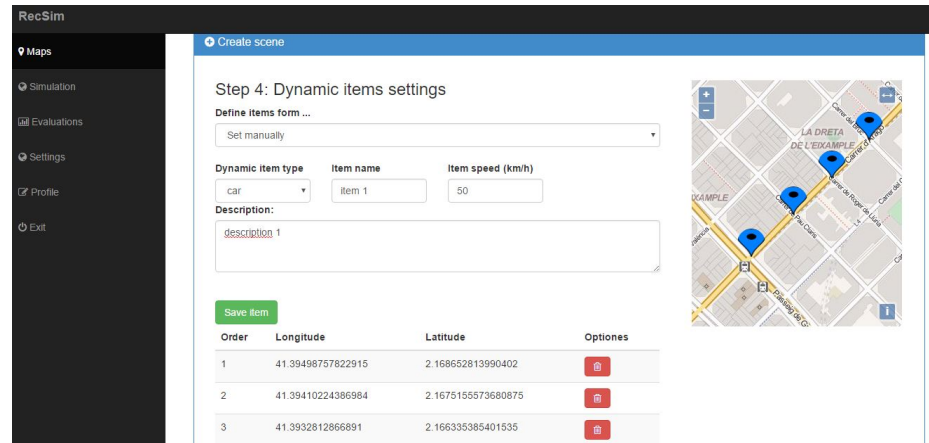


Figura A.14: Paso 4.2: definir los objetos dinámicos manualmente

Una vez que hayamos definido el trayecto pulsamos sobre el botón Save ítem y nos saldrá la lista con todos los objetos dinámicos en la escena:

Item name	Item type	Speed	Icon	Description	Options
Item 1	car	50		description 1	

Figura A.15: Paso 4.2: definir los objetos dinámicos manualmente

Paso 4.3: generar los objetos dinámicos aleatoriamente

Para generar los trayectos de forma aleatoria tenemos que seleccionar la opción Random trajectory y aparecerá un formulario donde tenemos que indicar el tipo de objeto dinámico que queremos generar, la cantidad de objetos que queremos generar y el tipo de vía en la que queremos que aparezcan. A continuación pulsamos sobre el botón Generate random way y tenemos que esperar que se generen las rutas. Este proceso puede ser bastante costoso y tenemos que ser pacientes. Esto es debido porque se baja el grafo de la escena y este pesa algunos cuantos megas.

Figura A.16: Paso 4.3: generar los objetos dinámicos aleatoriamente

A.10. Edición de mapas y escenas

Para modificar un mapa o escena primero tenemos que realizar una búsqueda de mapas en Maps → maps search y en la lista de resultados pulsamos sobre el icono de editar imagen. Solo podemos editar un mapa si la hayamos creado nosotros. De lo contrario el icono de editar imagen no aparecerá. Una vez que hayamos pulsado el icono de editar imagen entonces veremos la siguiente pantalla:

RecSim

- Maps
- Simulation
- Evaluations
- Settings
- Profile
- Exit

Basic data

Name: Type: State:

Create scene

Step 1: Set scene name and select Recommender settings

Scene name: Recommender settings:

List of scenes for this map

Scene name	Creation date	Recommender settings	City
------------	---------------	----------------------	------

Figura A.17: Edición de mapas y escenas

En esta pantalla disponemos de distintos tipos de opciones entre los cuales modificar datos del mapa, crear o modificar escenas.

A.11. Simulación

Para ejecutar una simulación lo primero que tenemos que hacer es realizar la búsqueda de un mapa explicado en el capítulo A.7. Una vez que hayamos realizado la búsqueda pulsamos el botón “Play” y a continuación se nos muestra una pantalla en la cual tenemos que elegir la escena a la cual queremos conectarnos.

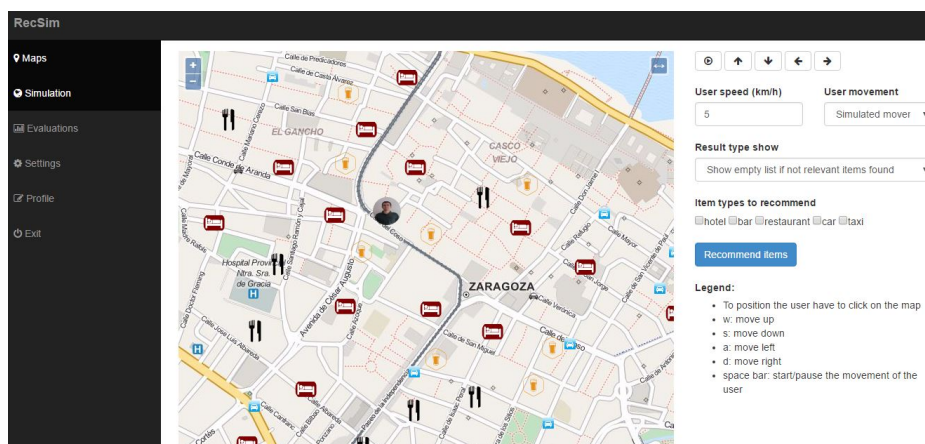


Figura A.18: Simulación

Para iniciar la simulación y que todos los ítems empiecen a moverse tenemos que pulsar el botón play. Hay que tener en cuenta que existe la posibilidad que de ya haya usuario que estén ejecutando una simulación sobre este escenario. Por esto cuando nos conectemos a la escena veremos que los objetos ya se estén moviendo. En cualquier momento podemos pausar la simulación. Entonces la simulación se pausa en todos los usuarios que están conectados en la escena. En cualquier momento cualquier usuario puede reanudar la simulación. Para salir de la simulación lo único que tenemos que hacer es salir de la pantalla de simulación.

A.11.1. Controles del usuario

Una vez cargada la escena podemos elegir entre usar la posición GPS del dispositivo con el cual nos estamos conectado y simular los movimientos del usuario.

Coordenadas GPS

Para usar el posicionamiento por GPS tenemos que elegir GPS positioning en el select con nombre User movement y nuestro icono aparecerá en el mapa. A medida que nos vamos moviendo vemos que nuestra posición se va actualizando.

Movimiento simulado

Para usar el movimiento simulado tenemos que elegir Simulated movement en el select con nombre User movement y a continuación tenemos que

elegir donde queremos situarnos en el mapa haciendo click en el mapa. A continuación podemos empezar a movernos por el mapa con los siguientes controles:

- tecla w: movimiento hacia arriba
- tecla s: movimiento hacia abajo
- tecla a: movimiento hacia la izquierda
- tecla d: movimiento hacia la derecha
- espacio: pausar el movimiento del usuario

A.11.2. Recomendaciones

Para obtener recomendaciones tenemos que seguir los siguientes pasos:

- paso 1: seleccionamos los tipos de ítems sobre los cuales queremos obtener resultados (ítems types to recommend)
- paso 2: en Result type show elegimos como queremos que se nos muestren los resultados. Por defecto si el recomendador no tiene ítems que recomendar mostrará una lista vacía.
- paso 3: pulsamos el botón Recommend ítems para obtener resultados

A.11.3. Votaciones

Para realizar las votaciones disponemos de las siguientes opciones:

- opción 1: pulsamos sobre el ítem concreto en el mapa y se nos abrirá un desplegable donde podemos votar
- opción 2: una vez que hayamos obtenido una lista con ítems recomendados disponemos de un desplegable donde podemos realizar la votación

A.12. Evaluación de un recomendador

Para realizar la evaluación de un recomendador vamos en el menú Evaluations:

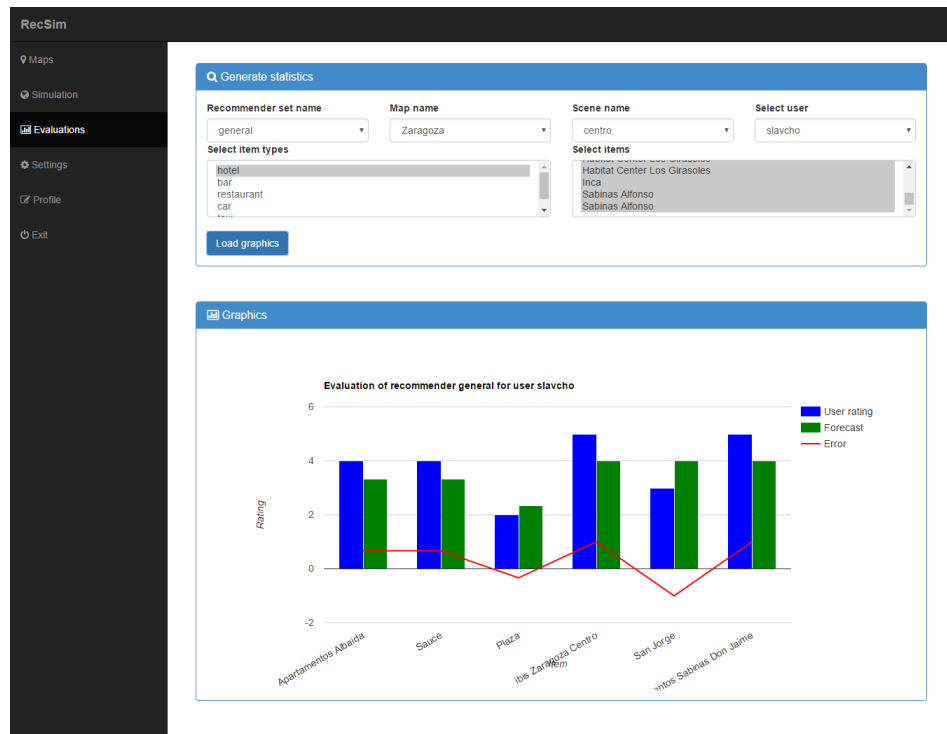


Figura A.19: Evaluación de un recomendador

Lo primero que tenemos que hacer es seleccionar el tipo de recomendador que queremos evaluar. Una vez que hayamos seleccionado el tipo de recomendador nos aparecen los mapas y escenas donde se utiliza este. A continuación tenemos que seleccionar el usuario que queremos evaluar y escoger una lista de ítems. Al final pulsamos el botón Load Graphics y se muestra un gráfico con el ítem, la valoración que ha dado el usuario, el valor estimado en este momento y el error. Si no disponemos de datos de algún ítem este no saldrá en el gráfico. Si no disponemos de ningún dato se nos indicara con un mensaje.

Anexos B

Análisis

B.1. Análisis de requisitos

B.1.1. 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 B.1: Requisitos no funcionales

B.1.2. 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, ciudad 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 listará 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 creados
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 límites 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
RF-33	La aplicación permitirá al usuario generar de forma aleatoria los grafos de movimiento de los vehículos

Cuadro B.2: Requisitos funcionales

B.2. Objetivos de Usabilidad

La Usabilidad, según el estándar ISO 9241-11, se define como la medida en la que un producto se puede usar por determinados usuarios para conseguir objetivos específicos de efectividad, eficiencia y satisfacción de un contexto de uso específico.

Así que para poder garantizar la calidad y la satisfacción de los usuarios tenemos que tener en cuenta los objetivos de usabilidad descritos:

- **Efectividad:** asegurar que la aplicación desempeñe correctamente todos los objetivos de la aplicación.
- **Eficiencia:** asegurar que cada objetivo de aplicación sea realizado en el menor tiempo posible desempeñando correctamente su tareas .

- **Utilidad:** para que el sistema pueda hacer todas las tareas que el usuario deba hacer, la aplicación tendrá conexión a Internet ya que establecerá conexión con un servidor en el que se encuentra toda la información de los productos. Siempre y cuando la conexión sea satisfactoria, el usuario podrá realizar las tareas descritas en el apartado de análisis de requisitos funcionales.
- **Seguridad:** asegurar que la aplicación evite situaciones de pérdida de información, evitar que se cuelgue y garantizar la confidencialidad de la información ya que la aplicación tiene acceso a Internet.

Objetivos	Eficacia	Eficiencia	Satisfacción
Utilizabilidad global	Usuarios que terminan la tarea con éxito: 99 % de usuarios	Tiempo de realización de tareas: 8 seg.	Frecuencia de quejas: 2 - 4 de cada 100
Satisface las necesidades de los usuarios habituales	Tareas terminadas con éxito: 95 % de tareas	Tiempo de realización de tareas: 5 seg.	Evaluación de satisfacción en el uso de las funciones: 9/10
Satisface las necesidades de los usuarios noveles	Tareas terminadas con éxito en el primer intento: 90 % de tareas	Tiempo de realización de las tareas: 15 seg.	Tiempo de uso no obligatorio: 10 seg. - 20 seg.
Facilidad de aprendizaje	Número de funciones aprendidas: 100 % de las funciones	Número de usos para aprendizaje: 2 - 3 usos	Evaluación de la facilidad de aprendizaje: 8/10
Tolerancia a errores	Errores registrados o corregidos por el sistema: 100 % de errores	Tiempo empleado en corregir errores: 30 seg.	tratamiento de errores: 9/10
Legibilidad	Palabras leídas correctamente a distancia normal: 100 % de palabras	Tiempo necesarios par leer la pantalla: 10 seg. - 15 seg.	Evaluación de las molestias visuales: 1/10 (menos nota implica menor molestia)

Cuadro B.3: Objetivos de usabilidad

Anexos C

Implementación

En este capítulo se mostraran los fundamentos mas importantes sobre la implementación tanto del front-end como del back-end y de como montar un instalador para Windows.

C.0.1. Implementación del front-end

En esta sección se explicaran cuales son las configuraciones más importantes de Angular.js porque este es el más importante del front-end.

Organización y carpetas

La capeta public es la que contiene todo lo relacionado con el front-end del simulador de escenarios. Dento de este encontramos la siguiente estructura de directorios:

```
Simulator
├── public
│   ├── images
│   ├── javascript
│   │   ├── angular
│   │   ├── bootstrap
│   │   ├── jquery
│   │   ├── openlayers
│   │   └── socketsIO
│   ├── stylesheets
│   └── views
```

A continuación vamos a ver cual es la función de cada uno de estos directorios:

- images: contiene imágenes estáticas que se están utilizando en la apli-

cación.

- javascript: contiene todos los frameworks de javascript que se están utilizando en el front-end. Hay una carpeta para cada framework.
- javascript/angular: contiene todas las configuraciones de Angular.js. Este tiene 4 subdirectorios: config (contiene el router de Angular.js), controllers (contiene los controladores de Angular.js), factory (contiene el factory de Angular.js) y services (contiene los servicios de Angular.js).
- javascript/bootstrap: contiene el sistema de componentes de bootstrap.
- javascript/openlayers: contiene el framework de OpenStreetMap.

El router de Angular.js

El router de Angular.js es el encargado del direccionamiento dentro de la página. En las configuraciones de este vinculamos una URL con una vista y esta con un controlador. Podemos encontrar las configuraciones de este en la carpeta `/public/javascript/angular/config/routes.js`:

```
1 var app = angular.module("app", ['ngRoute', 'ngStorage', 'checklist-model', 'xeditable']);
2
3 app.config(['$routeProvider', '$httpProvider', function($routeProvider, $httpProvider) {
4   $routeProvider.when('/', {
5     templateUrl: 'views/index.html',
6     controller: 'LoginController'
7   }).when('/register', {
8     templateUrl: 'views/register.html',
9     controller: 'RegisterController'
10  }).when('/map', {
11    templateUrl: 'views/maps/index.html',
12    controller: 'MapController'
13  }).when('/map/newMap', {
14    templateUrl: 'views/maps/newMap.html',
15    controller: 'NewMapController'
16  }).when('/map/editScene', {
17    templateUrl: 'views/maps/editScene.html',
18    controller: 'EditSceneController'
19  }).when('/map/editMap', {
20    templateUrl: 'views/maps/editMap.html',
21    controller: 'EditMapController'
22  }).when('/simulator', {
23    templateUrl: 'views/simulator/index.html',
24    controller: 'SimulatorController'
25  }).when('/settings', {
26    templateUrl: 'views/settings/recommenderSettings.html',
27    controller: 'RecommenderSettingsController'
28  }).when('/settings/staticItemSettings', {
29    templateUrl: 'views/settings/staticItemSettings.html',
```

```
30 controller: 'StaticItemSettingsController'
31 }).when('/settings/dynamicItemSettings', {
32   templateUrl: 'views/settings/dynamicItemSettings.html',
33   controller: 'DynamicItemSettingsController'
34 }).when('/profile', {
35   templateUrl: 'views/configurations/index.html',
36   controller: 'ConfigurationController'
37 }).otherwise({
38   redirectTo: '/'
39 });
40
41 $httpProvider.interceptors.push(['$q', '$location', '$localStorage',
42   function($q, $location, $localStorage) {
43     return {
44       'request': function (config) {
45         config.headers = config.headers || {};
46
47         if ($localStorage.token) {
48           config.headers.Authorization = 'Bearer ' + $localStorage.token;
49         }
50
51         return config;
52       },
53       'responseError': function(response) {
54         if(response.status === 401 || response.status === 403) {
55           $location.path('/signin');
56         }
57
58         return $q.reject(response);
59       }
60     };
61   }]);
62 }]);
63 }]);
```

En el mismo fichero tenemos configurado también un interceptor. El interceptor es un método que se ejecuta cada vez que llega o sale una petición. En nuestro caso el interceptor es utilizado para encriptar y desencriptar los datos involucrados en la comunicación. También es el encargado de poner la cabecera Authorization que se utiliza para autentificar el usuario en cada petición.

Los controladores de Angular.js

Los controladores de Angular.js contienen la lógica de las vistas y establecen la comunicación con el back-end mediante los servicios de Angular.js. Existe un controlador por cada página y contiene funciones y datos propias de la vista. Como regla general se ha establecido que el nombre de controlador es el mismo que la vista a la que va asociada pero acabado con la palabra clave Controller. Podemos encontrar todos los controladores en la carpeta /public/javascript/angular/controllers.

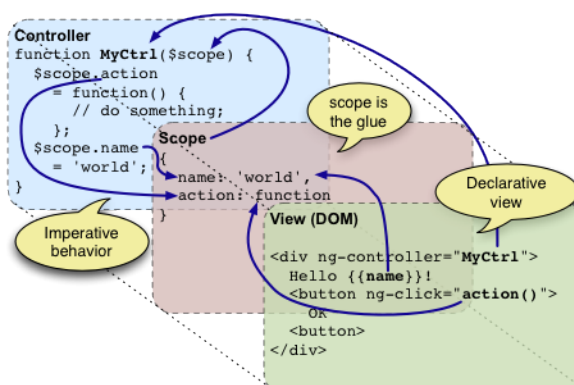


Figura C.1: Ejemplo de un controlador

Según la filosofía de Angular.js las vista se actualizan de forma automática cuando actualizamos el modelo de datos del controlador. A continuación podemos ver un diagrama a cerca de como funciona:

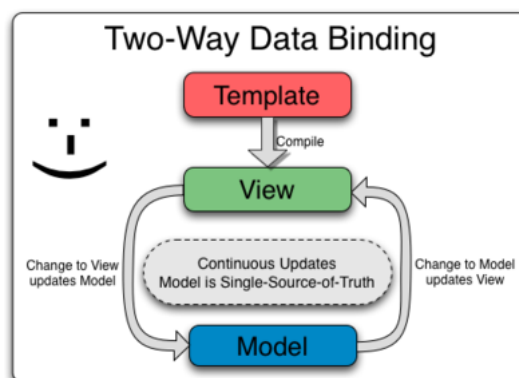


Figura C.2: Two-way data binding

Los servicios de Angular.js

Tener toda la lógica metida en los controladores y en el modelo de datos no es buena idea porque cuando la aplicación empieza a crecer se complican las cosas.

Los servicios son un concepto importante en Angular.js porque nos permiten agrupar funcionalidades que luego estarán disponible en los controladores, mejorando la claridad del código y favoreciendo la reutilización.

En nuestro caso hemos agrupado las llamadas al back-end y accedemos a la información del usuario autenticado descriptada por los interceptores. Podemos encontrar todas las configuraciones de los servicios en el fichero

/public/javascripts/angular/services/Services.js.

C.0.2. Implementación del back-end

En esta sección se explican cuales son las configuraciones más importantes de Node.js y Express.

Organización y carpetas

El back-end se basa en la filosofía de desarrollo de aplicaciones con Node.js Express. A continuación podemos ver cuales son los ficheros y directorios mas importantes:

```
Simulator
├── app.js
├── package.json
├── routes
│   ├── configurations.js
│   ├── evaluation.js
│   ├── maps.js
│   └── user.js
└── models
```

El primer fichero que vemos es el app.js. Ahí es donde se encuentran todas las configuraciones del back-end como por ejemplo como se realiza el enrutamiento dentro de la REST API, la conexión con la base de datos y las configuraciones de sockets.io.

El fichero package.json forma parte del gestor de paquetes npm y contiene todos los modulos que estamos usando en la aplicación.

Por ultimo nos encontramos con los directorios routes y models. En el directorio routes podemos encontrar todo lo relacionado con el enrutamiento interno de la REST API y en models encontramos el modelo de datos usado en la base de datos.

C.0.3. ¿Cómo crear un instalador de la aplicación?

Para crear un instalador de la aplicación se ha utilizado el programa Inno Setup v5.5.9. Dispone de un asistente de configuración que podemos ver en las secciones siguientes. Una vez que hayamos finalizado el asistente se crea un script que tenemos que compilar. Durante la compilación se comprimen todos los ficheros y directorios de nuestro proyecto y nos pide cual es el programa que tenemos que ejecutar cada vez que se inicie la aplicación.

Pasos previos

Para la aplicación pueda instalarse y ejecutarse como cualquier programa en Windows lo que necesitamos es que Maven, Node.js y mongoDB se nos instalen y configuren de forma automática con el instalador. El problema que esto supone es que el instalador no permite instalar programas externos porque lo único que haces es descomprimir el directorio de nuestra aplicación y crear un acceso directo para script que arranca los diferentes componentes del sistema: arrancar mongoDB, simulador y el recomendador. A continuación podemos ver el contenido del script que arrancar los distintos componentes del sistema:

```
1 @echo off
2 SET M2_HOME=../applications/maven
3 SET M2=%M2_HOME%/bin
4 SET PATH=%M2%; %M2_HOME%
5
6 if "%PROCESSOR_ARCHITECTURE%" == "x86" (
7 SET PATH=%PATH%;applications/mongodb32bits/bin;applications/nodejs32bits
8 ;
9 ) ELSE (
10 SET PATH=%PATH%;applications/mongodb/bin;applications/nodejs;
11 )
12
13 start mongod.exe --dbpath=applications/data/db
14
15 start npm start
16
17 cd Recommender
18 start mvn exec:java -Dexec.mainClass=Server
19 cd ..
```

Para solucionar el problema de la instalación de programas externos se han incluido los programas externos en la carpeta `applications` que se encuentra en el directorio raíz del simulador. En esta carpeta se han incluido Maven, Node.js y mongoDB. Hay que tener en cuenta que Node.js y mongoDB tienen distintas implementaciones para 32 y 64 bits. Por esto se han incluido ambas implementaciones en esta carpeta. Por esto durante el arranque se comprueba el tipo de sistema donde se este instalando el simulador y se configura en el PATH en función de esto.

Debido a que los programas externos disponen de ficheros demasiado grandes no era posible subirlos a ningún repositorio. Además que los repositorios de git no permite subir ejecutables por razones de seguridad. Por esto se han comprimido en un fichero .zip de fragmentado de 5MB cada fichero. Este fichero .zip fragmentado lo podemos encontrar dentro del directorio `applications` y tenemos que descomprimirlo dentro de este directorio.

El mismo problema ha surgido al subir el instalador en los repositorios Github y Gitlab. Por esto también se ha comprimido el instalador en un

fichero .zip fragmentado. Podemos encontrarlo dentro del directorio instalador.

Una vez descomprimidos estos dos ficheros comprimidos podemos proceder a eliminarlos porque si hacemos un nuevo instalador estos ficheros se comprimirán dentro del nuevo instalador y este ocupará mucho más.

Paso 1

En la primera pantalla solo vemos un texto informativo y lo único que tenemos que hacer es pulsar en Next:

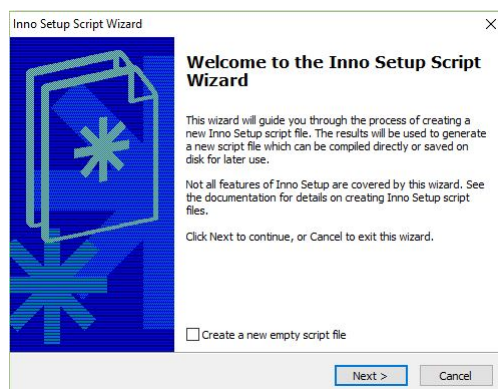


Figura C.3: Paso 1

Paso 2

En el segundo paso tenemos que rellenar un formulario con la siguiente información: el nombre de la aplicación, la versión e información de autor.

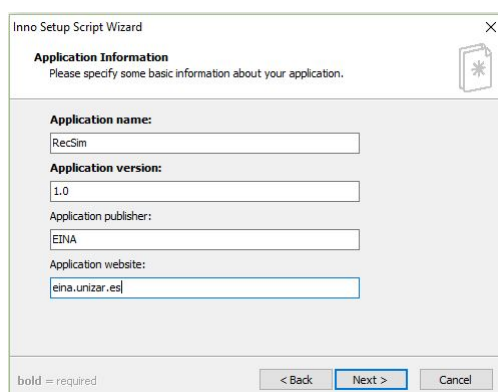


Figura C.4: Paso 2

Paso 3

En este paso indicamos cual es el nombre de la carpeta donde se van a descomprimir todos los ficheros tras la instalación:

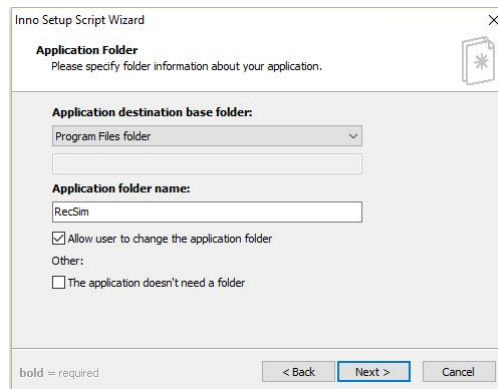


Figura C.5: Paso 3

Paso 4

En este paso tenemos que indicar cual es el script o ejecutable que arranca la aplicación e incluir el directorio del proyecto que sera comprimido en el instalador.

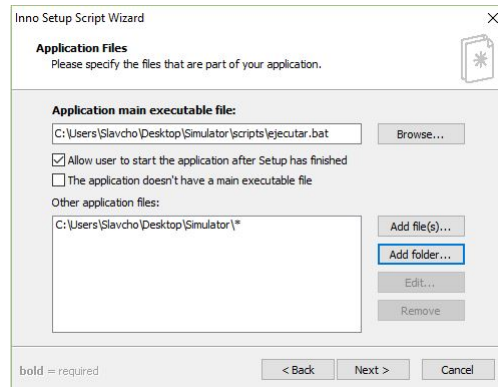


Figura C.6: Paso 4

Paso 5

En este paso dejamos seleccionadas las opciones de crear un acceso directo en el menú de inicio e indicar que queremos un desinstalador:

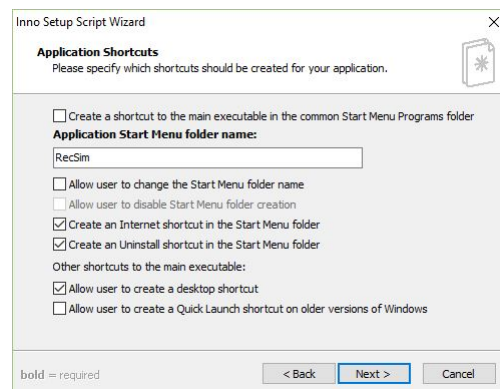


Figura C.7: Paso 5

Paso 6

En este paso disponemos de opciones de licencias de software. En nuestro caso las dejamos vacías:

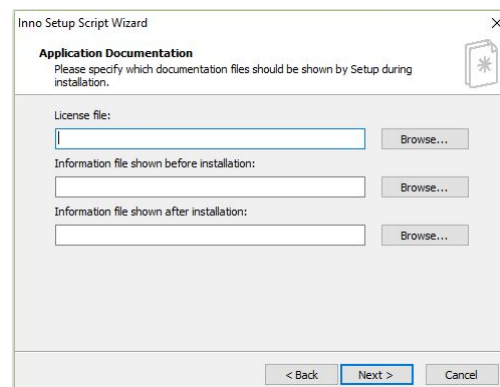


Figura C.8: Paso 6

Paso 7

En este paso indicamos cuales son los idiomas que se nos mostraran durante la instalación. Los dejamos todos seleccionados:

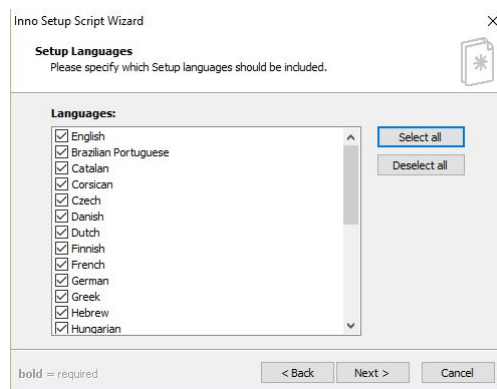


Figura C.9: Paso 7

Paso 8

En este paso tenemos que indicar cual es el directorio de salida donde se dejara el ejecutable, el nombre del instalador y su icono:

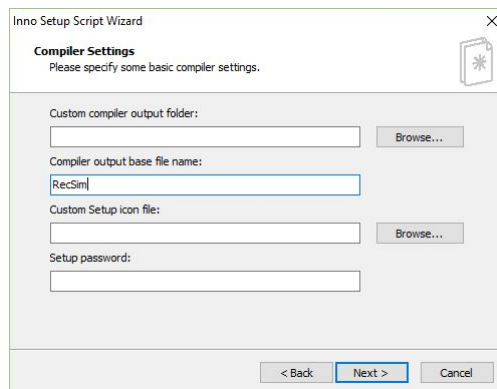


Figura C.10: Paso 8

Paso 9

En este paso nos preguntan si queremos incluir directivas en el script que se nos va a crear. Estas directivas tienen la función de variables globales con información que se reutiliza en el script como por ejemplo el nombre del instalador etc. En este caso dejamos el check marcado:

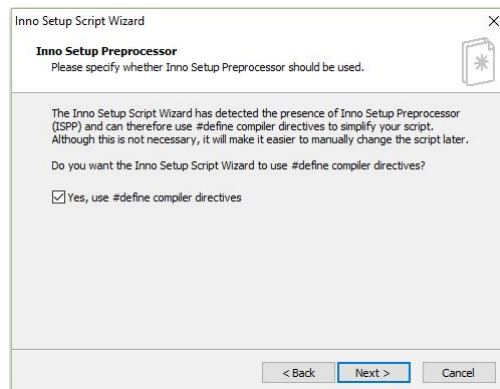


Figura C.11: Paso 9

Paso 10

Este es el último paso y lo único que tenemos que hacer es pulsar en Finish para finalizar el asistente:

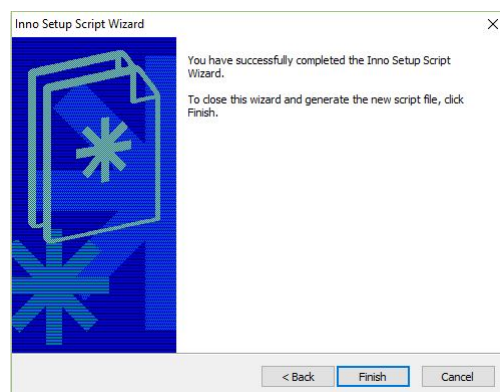


Figura C.12: Paso 10

Observamos que se genera un script como este:

```

1 ; Script generated by the Inno Setup Script Wizard.
2 ; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT
  FILES!
3
4 #define MyAppName "Simulator"
5 #define MyAppVersion "1.0"
6 #define MyAppPublisher "EINA"
7 #define MyAppURL "eina.unizar.es"
8 #define MyAppExeName "ejecutar.bat"
9
10 [Setup]
11 ; NOTE: The value of AppId uniquely identifies this application.
12 ; Do not use the same AppId value in installers for other applications
13 ; (To generate a new GUID, click Tools | Generate GUID inside the IDE
  .)
14 AppId={{23676DOA-DCBC-41E5-B959-47753F7EE277}
15 AppName={#MyAppName}
16 AppVersion={#MyAppVersion}
17 ; AppVerName={#MyAppName} {#MyAppVersion}
18 AppPublisher={#MyAppPublisher}
19 AppPublisherURL={#MyAppURL}
20 AppSupportURL={#MyAppURL}
21 AppUpdatesURL={#MyAppURL}
22 DefaultDirName={pf}\{#MyAppName}
23 DefaultGroupName={#MyAppName}
24 DisableProgramGroupPage=yes
25 OutputBaseFilename=Simulator
26 Compression=lzma
27 SolidCompression=yes
28 AlwaysShowDirOnReadyPage=yes
29
30 [Languages]
31 Name: "english"; MessagesFile: "compiler:Default.isl"
32 Name: "brazilianportuguese"; MessagesFile: "compiler:Languages\
  BrazilianPortuguese.isl"
33 Name: "catalan"; MessagesFile: "compiler:Languages\Catalan.isl"
34 Name: "corsican"; MessagesFile: "compiler:Languages\Corsican.isl"
35 Name: "czech"; MessagesFile: "compiler:Languages\Czech.isl"
36 Name: "danish"; MessagesFile: "compiler:Languages\Danish.isl"
37 Name: "dutch"; MessagesFile: "compiler:Languages\Dutch.isl"
38 Name: "finnish"; MessagesFile: "compiler:Languages\Finnish.isl"
39 Name: "french"; MessagesFile: "compiler:Languages\French.isl"
40 Name: "german"; MessagesFile: "compiler:Languages\German.isl"
41 Name: "greek"; MessagesFile: "compiler:Languages\Greek.isl"
42 Name: "hebrew"; MessagesFile: "compiler:Languages\Hebrew.isl"
43 Name: "hungarian"; MessagesFile: "compiler:Languages\Hungarian.isl"
44 Name: "italian"; MessagesFile: "compiler:Languages\Italian.isl"
45 Name: "japanese"; MessagesFile: "compiler:Languages\Japanese.isl"
46 Name: "norwegian"; MessagesFile: "compiler:Languages\Norwegian.isl"
47 Name: "polish"; MessagesFile: "compiler:Languages\Polish.isl"
48 Name: "portuguese"; MessagesFile: "compiler:Languages\Portuguese.isl"
49 Name: "russian"; MessagesFile: "compiler:Languages\Russian.isl"
50 Name: "scottishgaelic"; MessagesFile: "compiler:Languages\
  ScottishGaelic.isl"
51 Name: "serbiancyrillic"; MessagesFile: "compiler:Languages\
  SerbianCyrillic.isl"
52 Name: "serbianlatin"; MessagesFile: "compiler:Languages\SerbianLatin.
  isl"

```



```
53 Name: "slovenian"; MessagesFile: "compiler:Languages\Slovenian.isl"
54 Name: "spanish"; MessagesFile: "compiler:Languages\Spanish.isl"
55 Name: "turkish"; MessagesFile: "compiler:Languages\Turkish.isl"
56 Name: "ukrainian"; MessagesFile: "compiler:Languages\Ukrainian.isl"
57
58 [Tasks]
59 Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}";
    GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked
60
61 [Files]
62 Source: "C:\Users\Slavcho\Desktop\Simulator\scripts\ejecutar.bat";
    DestDir: "{app}"; Flags: ignoreversion
63 Source: "C:\Users\Slavcho\Desktop\Simulator\*"; DestDir: "{app}";
    Flags: ignoreversion recursesubdirs createallsubdirs
64 ; NOTE: Don't use "Flags: ignoreversion" on any shared system files
65
66 [Icons]
67 Name: "{group}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
68 Name: "{group}\{cm:ProgramOnTheWeb,{#MyAppName}}"; Filename: "{#
    MyAppURL}"
69 Name: "{group}\{cm:UninstallProgram,{#MyAppName}}"; Filename: "{
    uninstall.exe}"
70 Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName
    }"; Tasks: desktopicon
71
72 [Run]
73 Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,{#
    StringChange(MyAppName, '&', '&&')}}"; Flags: shellexec
    postinstall skipifsilent
```


Anexos D

Integración con un recomendador externo

En este capítulo se mostrará como se realiza la integración con un recomendador externo mediante un sistema bidireccional dirigido por eventos y el intercambio de mensajes JSON.

D.1. Introducción

Tal y como hemos mencionado anteriormente el simulador y el recomendador están integrados mediante un sistema bidireccional dirigido por eventos y el intercambio de mensaje en JSON. Actualmente está desarrollando un recomendador de ejemplo basado en usuarios de tipo pull mediante la librería Apache Mahout.

La razón por la que se han integrado los dos sistemas mediante un sistema bidireccional dirigido por eventos es que existen recomendadores que realizan recomendaciones sin que el usuario las haya solicitado. En protocolos como el HTTP no podemos enviar respuesta al navegador sin que este haya lanzado una petición previamente mientras que en el sistema bidireccional dirigido por evento si que podemos enviar datos al cliente cuando sea necesario y sin que este los haya solicitado.

El recomendador de ejemplo desarrollado está pensado en lanzar un hilo para cada tipo de recomendador, es decir, un hilo para el servidor de tipo pull y otro para el recomendador de tipo push. Actualmente solo existe una implementación de un recomendador de tipo pull y por lo tanto solo hay un hilo. Este recomendador incorpora un patrón de diseño de tipo Strategy combinado con un patrón de diseño de tipo Factory que nos permiten cambiar el tipo de implementación del recomendador de tipo pull en tiempo de

ejecución.

De esta manera podemos cambiar la implementación del recomendador de tipo pull entre peticiones y dar servicio a los distintos recomendadores configurados en las distintas escenas.

A continuación se mostrará cual es el modelo de eventos para cada tipo de recomendador y cual es el forma de los mensajes intercambiados.

D.2. Recomendadores pull

En esta sección se mostrará la arquitectura del recomendador pull existente y como expandirlo para crear una nueva implementación para este tipo de recomendador.

D.2.1. ¿En que consiste el patrón de diseño de tipo Strategy?

El patrón Strategy es un patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. Permite mantener un conjunto de algoritmos de entre los cuales el cliente puede elegir cual le conviene usar. Tiene los siguientes participantes:

- Contexto: es el elemento que usa los algoritmos y delega en la jerarquía de estrategias. Configura una estrategia mediante una referencia a la estrategia concreta.
- Estrategia: una interfaz común para todos los algoritmos. Es usada por el contexto para invocar la estrategia concreta.
- Estrategia concreta: implementa el algoritmo utilizando la interfaz definida por la estrategia.

En nuestro caso el contexto es el objeto Recommender ubicado en el paquete recommender.context, la interfaz estrategia se corresponde la interfaz Strategy ubicada en el paquete recommender.strategy y la estrategia concreta es la clase UserBasedStrategy ubicada en el paquete recommender.strategy.

En la implementación actual se ha usado el patrón de diseño de tipo Factory para crear y gestionar las instancias de las estrategias concretas. Cuando llega una solicitud de recomendación por un usuario se crea una instancia de la estrategia concreta y se almacena en una tabla hash. Si la estrategia concreta ya está almacenada en la tabla hash entonces solo se devuelve la instancia de esta.

D.2.2. Pasos para crear un nuevo recomendador pull

Paso 1: Crear una estrategia concreta

Tal y como hemos mencionado anteriormente la estrategia concreta implementa el algoritmo de recomendación concreto. Para implementar un nuevo tipo de algoritmo de recomendación tenemos que crear una clase en el paquete `recommender.strategy` que implemente la interfaz `Strategy` del mismo paquete.

Esta interfaz tiene dos métodos: `recommend` e `itemForecast`. El método `recommend` es el que se invoca cuando el usuario solicita una recomendación y devuelve una lista de ítems. El método `itemForecast` realiza una predicción para un ítem dado un usuario y devuelve un `float` que se corresponde a la predicción realizada.

Los parámetros de entrada de ambos métodos son iguales y tienen el mismo significado:

- `JSONObject data`: contiene todos los datos en formato JSON que enviar el simulador.
- `List<Item> itemList`: contiene un array list con todos los ítems a recomendar.
- `List<Ratings> ratingList`: contiene un array list con las valoraciones de los usuarios para cada ítem.
- `RecommenderConfig recommenderConfig`: contiene los parámetros de configuración del recomendador como número de ítems a recomendar etc.

Todos los datos vienen listos para ser usados por el algoritmo de recomendación que estamos implementando.

Paso 2: expandir el enumerado StrategyType

StrategyType es un tipo enumerado que contiene todas las estrategias concretas implementadas. Es utilizado por patrón Factory para determinar que tipo de estrategia concreta crear.

```
1 package recommender.models;
2
3 public enum StrategyType {
4     UserBasedStrategy("User based recommender");
5
6     private String type;
7
8     StrategyType(String type) {
9         this.type = type;
10    }
11
12    public String getType() {
13        return this.type;
14    }
15
16    public static StrategyType fromString(String type) {
17        if (type != null) {
18            for (StrategyType b : StrategyType.values()) {
19                if (type.equalsIgnoreCase(b.type)) {
20                    return b;
21                }
22            }
23        }
24        return null;
25    }
26 }
27 }
```

En la implementación de la figura anterior podemos ver el siguiente enumerado UserBasedStrategy("User based recommender") que se corresponde a la implementación actual del algoritmo de recomendaciones basados en usuarios. La cadena entre comillas es el nombre que se va mostrar en el simulador cuando configuramos un nuevo recomendador.

Paso 3: expandir el factory StrategyFactory

Una vez que hayamos expandido StrategyType vamos a expandir el factory StrategyFactory para que pueda usar este nuevo tipo de enumerado y crear la estrategia concreta.

```
1 package recommender.strategy;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import recommender.models.StrategyType;
6
7 public class StrategyFactory {
```

```

8      static Map<StrategyType, Strategy> map = new HashMap<
          StrategyType, Strategy>();
9
10     public static Strategy createStrategy(StrategyType type){
11         Strategy strategy = map.get(type);
12
13         if(strategy==null){
14             switch(type){
15                 case UserBasedStrategy: strategy = new
                     UserBasedStrategy(); break;
16                 default: throw new IllegalArgumentException("
                     The recommender type " + type + " is not
                     recognized.");
17             }
18
19             map.put(type, strategy);
20         }
21
22         return strategy;
23     }
24 }

```

Para expandirlo vamos al método `createStrategy` y expandimos el `switch` para que use el nuevo enumerado que hemos creado en el paso 2 y en función a ellos cree una nueva instancia de la estrategia concreta que hemos creado en el paso 1.

D.2.3. Eventos y mensajes

En el diagrama siguiente podemos ver cuales son todos los eventos en el recomendador de tipo pull:

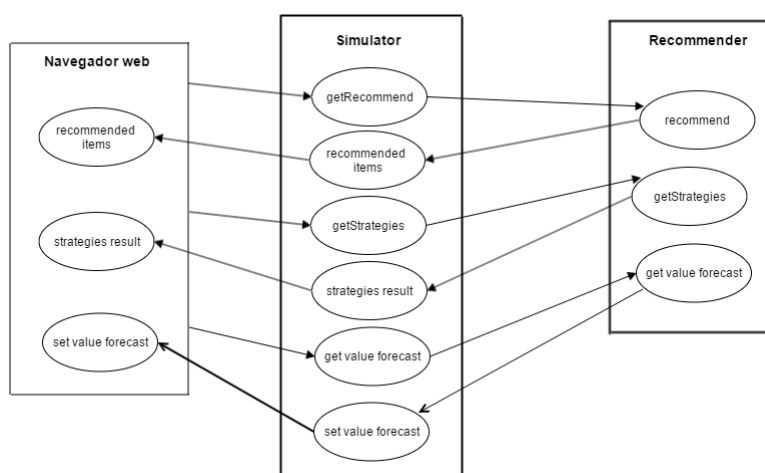


Figura D.1: Diagrama de eventos del recomendador de tipo pull

Los eventos importantes del recomendador son:

- `recommend`: es el evento que invoca el simulador para solicitar una recomendación.
- `getStrategies`: es el evento que se invoca para recuperar la lista de estrategias concretas implementadas. No tiene parámetros de entrada.
- `get value forecast`: es el evento que se invoca para solicitar una predicción para un ítem

Los eventos importantes del simulador son:

- `recommended items`: es el evento que invoca el recomendador para enviar la lista de ítems recomendados.
- `strategies result`: es el evento que invoca el recomendador para enviar la lista de estrategias concretas implementadas.
- `set value forecast`: es el evento que invoca el recomendador para enviar la predicción para un ítem.

A continuación podemos ver los parámetros de entrada de los eventos importantes para una integración con un recomendador externo:

Nombre	Tipo	Descripción
<code>userId</code>	String	token del usuario
<code>strategyType</code>	String	tipo de estrategia concreta que queremos usar para la recomendación. Contiene uno de los valores del enumerado <code>StrategyType</code>
<code>mapId</code>	String	identificador del mapa
<code>sceneId</code>	String	identificador de la escena
<code>rating</code>	number	Valoración del usuario sobre ítem con identificador <code>itemId</code>
<code>itemId</code>	String	identificador del ítem
<code>recommenderId</code>	String	identificador de la configuración del recomendador

Cuadro D.1: Evento `get value forecast` del recomendador

Nombre	Tipo	Descripción
<code>itemList</code>	Array	lista con ítems que recomendamos al usuario
<code>userList</code>	Array	lista de tokens de usuarios a los que recomendamos los ítems

Cuadro D.2: Evento `recommended items` del simulador

Nombre	Tipo	Descripción
strategyType	String	tipo de estrategia concreta que queremos usar para la recomendación. Contiene uno de los valores del enumerado StrategyType
mapId	String	identificador del map en el que está conectado el usuario
sceneId	String	identificador de la escena en la que está conectado el usuario
token	String	token del usuario conectado
recommender	String	identificador de la configuración del recomendador que estamos usando
itemTypesToRecommend	Array	contiene una lista con los identificadores de los tipos de ítems sobre los que el usuario desea recibir recomendaciones
resultTypeShow	String	el tipo de resultado que quiere ver el usuario. Puede contener los siguientes valores: - nonPersonalizedRecommendation: para mostrar recomendaciones no personalizadas en caso que de que el recomendador no tiene ítems que recomendar - emptyList: para mostrar una lista vacía cuando el recomendador no tiene ítems que recomendar
location	Object	localización actual del usuario. Es un objeto que contiene los siguientes atributos: - longitude: longitud actual - latitude: latitud actual

Cuadro D.3: Evento recommend del recomendador

D.3. Recomendadores push

Para crear un recomendador de tipo push tenemos que editar el objeto Server y lanzar un nuevo hilo que se corresponde al recomendador de tipo push y pasarle en el constructor los parámetros configs y socket igual que el PullServer en el siguiente ejemplo:

```
1 public Server() throws URISyntaxException, InterruptedException,
   IOException{
2     try {
3         Configurations configs = readBaseConfig();
4         String url = configs.getHost()+":"+configs.getPort();
5
6         socket = IO.socket(url);
7         socket.connect();
8
9         //Inicializamos un servidor de tipo pull
10        PullServer pullServer = new PullServer(configs, socket
11        );
12        pullServer.run();
13    } catch (URISyntaxException e) {
14        System.out.println(e);
15    }
16 }
```

La variable config contiene datos a cerca de la dirección de la base de datos y la dirección del simulador. La variable socket es el sistema bidireccional dirigido por eventos.

En cuanto a la arquitectura tenemos total libertad para el diseño del recomendador push. Lo único es que tenemos que hacer es crear el evento get strategies push. Este evento no tiene parámetros de entrada y devuelve una lista de enumerados con las implementaciones del recomendador push (ver implementación de la clase StrategyType). Para devolver la lista de recomendaciones tenemos que invocar el evento get strategies push result.

Para enviar las recomendaciones tenemos que usar el evento recommend items del simulador indicando la lista de ítems que estamos recomendando y la lista de usuarios (tokens) a los que realizamos las recomendaciones.

