Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

APLICACIONES BASADAS EN INTERCAMBIO DE SERVICIOS

Alumno: Francisco Javier Lendínez Tirado

Tutores: Prof. Dña. María Teresa Martín Valdivia
Prof. Dña. Salud María Jiménez Zafra
Dpto: Departamento de informática

Septiembre, 2017

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Departamento de Informática

Doña María Teresa Martín Valdivia y Doña Salud María Jiménez Zafra, tutoras del Proyecto Fin de Carrera titulado: Aplicaciones basadas en intercambio de servicios, que presenta Francisco Javier Lendínez Tirado, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2017

El alumno:

Francisco Javier Lendínez Tirado

Los tutores:

María Teresa Martín Valdivia

Salud María Jiménez Zafra

Índice

1. Introducción

1.1. Introducción al proyecto

Basándonos en la cita de Nathaniel S. Borenstein:

"Es importante destacar que ningún ingeniero software con ética consentiría escribir un procedimiento llamado DestruirBaghdad. Su ética le obligaría a escribir un procedimiento DestruirCiudad, al que se pasaría el parámetro Baghdad"

Fuera del tono jocoso de esta, podemos ver que es parte de este campo tratar de abstraer procedimientos e ideas que serán llevados al software. Con esta idea en mente y viendo el éxito de las aplicaciones *sociales* que actualmente están en el mercado, nos hace ver que todas comparten una base común y pueden ser desarrolladas bajo un mismo sistema.

De aquí parte este proyecto, de abstraer la base de las aplicaciones basadas en intercambio de servicios y comprobar la viabilidad de un sistema que las genere.

1.2. Propósito

Este proyecto pretende crear una herramienta para desarrolladores que genere un esqueleto de aplicación basada en intercambios de servicios. Como el tiempo y las capacidades de equipo son limitadas, el proyecto buscará sólo crear aplicaciones para la Web y con una herramienta offline. Aunque la idea puede ser extrapolada a cualquier plataforma.

Al margen de la herramienta, se hará un breve análisis de aplicaciones basadas en intercambio de servicios con ejemplos de uso.

1.3. Objetivos

Los objetivos que se pretenden alcanzar con el desarrollo de este proyecto son:

- Realizar un estudio sobre aplicaciones basadas en intercambio de servicios.
- Identificar los roles necesarios en este tipo de aplicaciones.
- Determinar cómo garantizar un funcionamiento seguro.
- Identificar las características fundamentales de estas aplicaciones.

- Desarrollar una herramienta que permita generar el esqueleto de una aplicación web para el intercambio de servicios.
- Redactar una memoria que recoja todo el trabajo realizado, así como los manuales de instalación y de uso.

1.4. Resultados esperados

Como resultado de este proyecto se pretende mostrar una breve investigación sobre aplicaciones basadas en intercambios de servicios, dando a conocer las características comunes y los elementos necesarios para generar una aplicación basada en esta idea. Además, se ofrecerá una herramienta que permita generar el esqueleto de una aplicación web para el intercambio de servicios y que será creada usando la información obtenida en dicho análisis.

1.5. Planificación

1.5.1. Planificación temporal

Visto que el proyecto es algo ambicioso, la planificación en tiempo ha sido uno de los factores limitantes para poder haberlo hecho más detallado. Se han repartido 300 horas de trabajo en días de 5 horas, por lo que tuvimos 60 días a repartir para todas las tareas. En la tabla 1.1 se muestran las tareas realizadas y el tiempo empleado en cada una de ellas.

Número de tarea	Descripción de la tarea	Duración de la tarea	Tarea anterior
1	Informarse de las herramientas disponibles	5 días	-
2	Crear una aplicación web de ejemplo	1 día	1
3	Hacer análisis de aplicaciones	4 días	-
4	Extraer esquemas para realizar la aplicación web	2 días	3
5	Crear controladores	10 días	2, 4

6	Crear vistas funcionales (sin estética)	2 días	5
7	Crear modelos	4 días	5
8	Enlazar modelos, vistas y controladores	2 días	6, 7
9	Probar distintos CSS	5 días	8
10	Crear vistas estéticas	4 días	9
11	Crear funciones dinámicas para extraer datos de los modelos y los formularios	4 días	8
12	Revisar el funcionamiento con modelos y formularios estáticos	2 días	11
13	Decidir la organización de los módulos	1 día	-
14	Crear sistema que enlace modelos dinámicos con formularios	6 días	12, 13
15	Añadir internacionalización	3 días	12
16	Revisar y documentar	5 días	*

Tabla 1.1: Planificación temporal

1.5.2. Planificación de costes

Gracias a las herramientas libres del mercado, a que es un proyecto personal y a que no se ha necesitado ningún componente hardware, el coste real de este proyecto ha sido de 0€. Aunque, dado el tiempo invertido, se puede hacer un cálculo

de gasto haciendo una estimación por meses usando datos del salario medio mensual en España en 2016.[1]

2 Meses * 2.226€/Mes = 4.452€

Podemos asumir entonces, que este proyecto tiene un valor aproximado de 4.452 €.

2. Aplicaciones basadas en intercambio de servicios

2.1. Introducción

La primera acepción de la Real Academia Española nos dice que un servicio es la *acción y efecto de servir.*

Gracias al amplio significado de servir, este proyecto tratará de focalizar e indagar cuáles son las bases para poder hacer un sistema amplio y sólido que se adapte a una gran variedad de necesidades.

En este capítulo podremos entender mejor qué es un servicio y cómo podemos abstraer y definir ese tipo de aplicaciones.

2.2. ¿Qué son las aplicaciones basadas en intercambio de servicios?

Las aplicaciones basadas en intercambio de servicios son aplicaciones en las que cualquiera puede ser oferente o receptor de una oferta concreta. Estas ofertas (que en adelante llamaremos eventos), suponen la temática principal de la aplicación y sobre esa acción se requieren unos datos u otros. El evento puede ser de cualquier índole. Vamos a usar la aplicación más conocida de las que analizaremos posteriormente, Blablacar, para entender qué características tienen estas aplicaciones.

El servicio de Blablacar se define así:

“Blablacar es un servicio francés de vehículo compartido que hace posible que las personas que quieren desplazarse al mismo lugar en el mismo momento puedan organizarse para viajar juntos. Permite compartir los gastos puntuales del viaje (combustible y peajes) y también evitar la emisión extra de gases de efecto invernadero, al permitir una mayor eficiencia energética en el uso de cada vehículo.”

[1] http://www.datosmacro.com/mercado-laboral/salario-medio/espana

(Wikipedia)

En este caso, el evento de la aplicación es compartir un vehículo. Quien ofrece el coche es el creador de la oferta, mientras que quien se monta en él es el receptor de dicho evento.
Con la misma filosofía podría darse una aplicación cuyo evento sea, por ejemplo, pasear al perro de tu vecino, hacerle la compra o cuidar de sus hijos y sacar un beneficio de ese favor u objetivos más amplios como buscar freelancers para un proyecto.

2.3. Aplicaciones analizadas

Tras una revisión de aplicaciones similares, se ha decidido publicar el análisis de Blablacar por la simpleza de interfaz y analogía con el resto.

2.4. Estudio de roles

Para hablar de este tipo de aplicaciones debemos empezar por los roles que las definen. Por lo general, y dado el grueso de los casos analizados, suelen existir dos roles principales: Oferente y Receptor.

Estos roles son parte de una misma cuenta y suelen darse de forma simultánea.

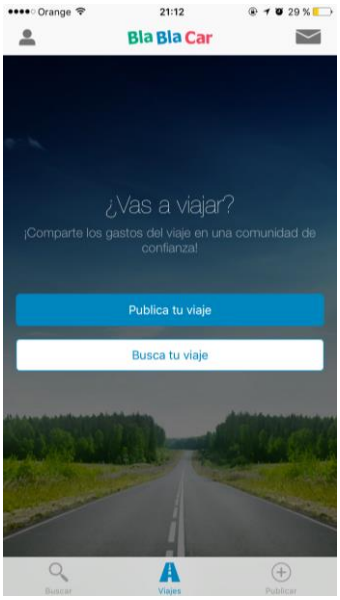

Ilustración 2.1: Vista inicial de la aplicación

Como se puede apreciar en la ilustración 2.1., usando la misma cuenta podemos publicar un viaje o buscarlo, actuando como oferente del evento o receptor, respectivamente, lo que nos hace ser siempre posibles oferentes o receptores del servicio.

2.5. Estudio de seguridad

Para que este tipo de aplicaciones tengan éxito, debe de haber ciertas garantías de seguridad entre el oferente y el receptor.

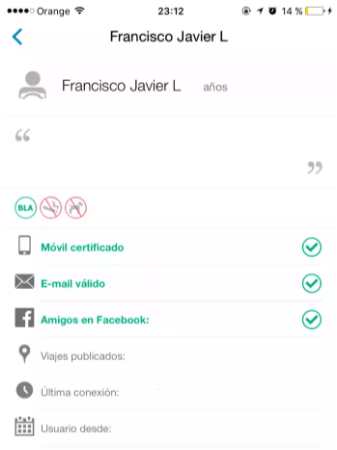

Ilustración 2.2: Perfil de usuario

La solución principal a este problema suele ser verificando la identidad de los usuarios por varias vías. Un usuario estándar confiará más en un usuario cuanto más verificado esté. Aquí las empresas suelen aprovechar estándares como OAuth para registrar e iniciar sesión mientras verifican una red social conocida, agilizando así el registro y la autenticación y tercerizando así este componente de seguridad.

“OAuth permite a un usuario del sitio A compartir su información en el sitio A (proveedor de servicio) con el sitio B (llamado consumidor) sin compartir toda su identidad. Para desarrolladores de consumidores, OAuth es un método de interactuar con datos protegidos y publicarlos. Para desarrolladores de proveedores de servicio, OAuth proporciona a los usuarios un acceso a sus datos al mismo tiempo que protege las credenciales de su cuenta. Este mecanismo es utilizado por compañías como Google, Facebook, Microsoft, Twitter y Github para permitir a los

usuarios compartir información sobre sus cuentas con aplicaciones de terceros o sitios web.” (Wikipedia)

2.6. Estudio de características

Viendo el inicio de la aplicación Blablacar, podemos extraer las partes necesarias de la aplicación y, además, seccionar estas en partes más genéricas. En dicha vista de inicio podemos ver que en la parte superior se mantienen siempre los componentes relativos al perfil, como son el editor/visor del perfil y el visor de notificaciones.

3.5.1. Secciones

Por otra parte, vemos que abajo se mantiene toda la temática de trabajo de la aplicación, como son el creador de eventos, el buscador de eventos y el visor de eventos.

Tal como se puede apreciar en el siguiente diagrama, la página de inicio quedaría al margen de estas dos secciones.

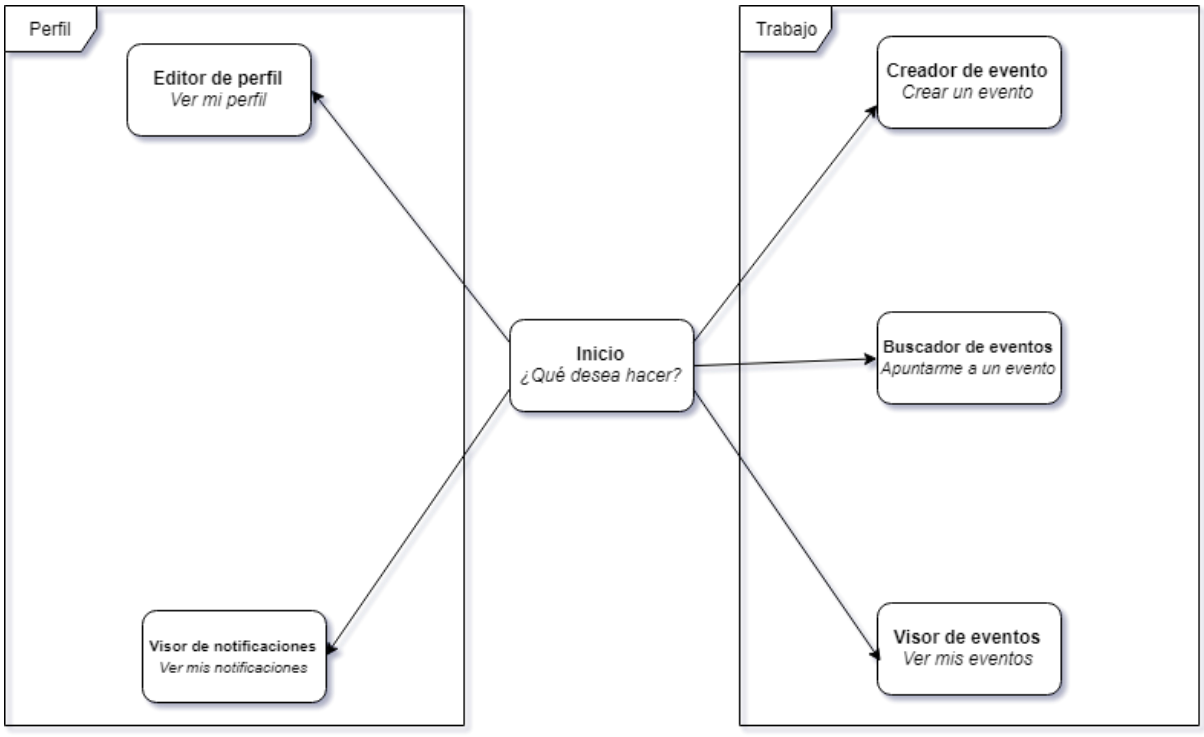

Ilustración 2.3: Esquema de vistas y secciones

Al hacer esta división, en la vista de la aplicación, el contenido siempre quedaría en la parte central permitiendo al usuario tener flexibilidad y control de sus acciones, ya que podrá moverse entre vistas cuando sea necesario.

Ilustración 2.4: División de secciones

3.5.2. Formularios

El formulario del evento de Blablacar contiene 2 localizaciones, dos checkbox y dos fechas. Estos tipos serán básicos para crear eventos más complejos, aunque existen otros tipos más complejos como pueden ser mapas donde definir rutas dentro de una ciudad.

Ilustración 2.5: Vista de publicación de la aplicación

El formulario de búsqueda tiene un subconjunto de los parámetros de creación. Tiene 2 localizaciones y una fecha que se corresponden directamente a parámetros específicos del formulario de creación.

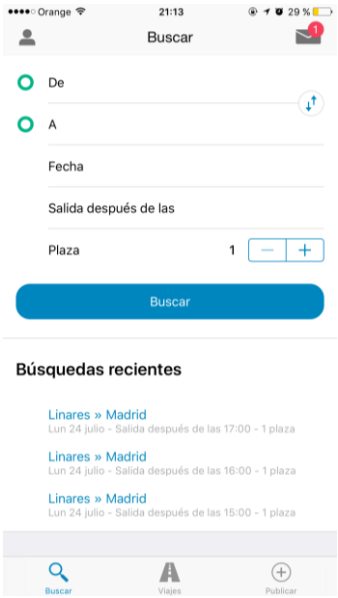

Ilustración 2.6: Vista de búsqueda de la aplicación

El formulario restante es el de editar perfil. Este formulario contiene dos campos de texto, una fecha y un bloque de texto. Aparte permite cambiar el e-mail, pero no se considera parte del mismo formulario.

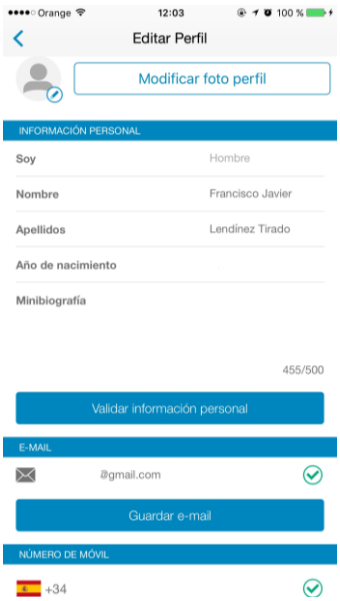

Ilustración 2.7: Vista de edición de perfil de la aplicación

2.7. Estudio técnico

Visto todo lo anterior, podemos entrar a hablar de las vistas necesarias para el desarrollo de la aplicación web.

Aprovechando el diagrama anterior podemos contar las vistas necesarias por partes:

- Editor de perfil
- Visor de perfil público
- Listado de notificaciones con acciones
- Creador de eventos
- Buscador de eventos
- Visor de eventos

Con estas 6 vistas básicas podemos tener cubierto prácticamente todo el esqueleto de la aplicación.

Tanto el editor de perfil como el creador de eventos son formularios específicos de cada aplicación, habrá que definir los campos previamente y qué campos serán opcionales u obligatorios. Para la herramienta que vamos a crear, se asume que el listado de notificaciones no sufre cambios significativos, por lo que no serán alteradas mediante configuración.

El buscador de eventos se podrá definir mediante los campos que se indiquen en la configuración.

El listado de eventos depende en gran parte de los atributos del evento por lo que esta parte tendrá que ser definida en la configuración.

3. Desarrollo del proyecto

3.1. Descripción del problema

Entrando ya en la parte técnica, el problema al que nos enfrentamos es crear una aplicación web que nos sirva de base, que cuente con componentes reusables y que sea totalmente funcional. Al margen de esta, crear un sistema que monte dicha aplicación web base y le añada los componentes creados o componentes nuevos y siga siendo totalmente funcional.

3.2. Objetivos de la herramienta

El principal objetivo del sistema es crear el esqueleto de una web totalmente funcional para trabajar sobre ella. Como la web será creada de manera automática, existen elementos que deben ser modificados a mano (textos, traducción, estilos…).

Una decisión importante sobre los objetivos del sistema era si este debía ser montado como una herramienta web o que se ejecutase offline. Se decidió la ejecución offline por la naturaleza estática del contenido de la web y para desacoplar la herramienta de la web resultante.

3.3. Especificación de requisitos

Los requerimientos describen el comportamiento que tiene, o debería tener, el sistema. Por lo tanto, debemos hablar de qué hace nuestro sistema, no de cómo.

Requerimientos funcionales

Este tipo de requerimiento marca los servicios a cumplimentar por el software.

1. Al finalizar una ejecución correcta del sistema, este debe dar como salida una web ejecutable y válida.
2. La web resultante de la salida podrá registrar usuarios, dar acceso a estos, registrar eventos de cada usuario, hacer visibles los eventos para todos los usuarios, apuntarse a eventos y aceptar/rechazar a gente en ellos.
3. La web, además, debe ser adaptada a móviles.

Requerimientos no funcionales

Este tipo de requerimiento no es fácilmente medible y no afecta al funcionamiento del sistema, aunque afecta a la calidad de este.

El sistema debe:

1. Ser multiplataforma.
2. Ser fácilmente usable.
3. Estar bien autodocumentado.

Aparte:

4. Debe tener una documentación para la instalación, uso y ejecución que sea directa y clara.

3.4. Metodología para el desarrollo de software y método a seguir

La metodología en la que se ha basado el desarrollo de este sistema ha sido la programación extrema.

"Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo en el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software." (Wikipedia)

En primer lugar, se crearon varios prototipos de aplicación web, con datos estáticos y funcionalidad limitada. Entre esos prototipos, se seleccionó el prototipo más adecuado a los requerimientos funcionales iniciales. Ahí empezamos a añadir de forma iterativa las funcionalidades requeridas, teniendo en cuenta los requerimientos no funcionales iniciales.

Con el modelo inicial creado, se generó un sistema que crease de manera adecuada los modelos de la base de datos, las secciones de eventos y las secciones relativas a los usuarios.

3.5. Historias de usuario

Este apartado puede ser un poco más tedioso ya que habrá que hacer dos secciones, historias propias del sistema y de cómo este gestiona los módulos y por otra parte las historias de la web que genera y las características que la definen.

3.5.1. Historias de la herramienta

Historia de usuario				
Número:	1	Nombre:	Generador de contenido	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Baja	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Alto	
Descripción:				
El sistema debe generar el contenido de la web de forma ordenada y funcional. En caso de que el software esté encapsulado debe generar el contenido visible y útil para el usuario. En cualquier otro caso el usuario deberá tener acceso a una web funcional sin configuración.				

Tabla 3.1: Historia de usuario 1 del sistema

Historia de usuario				
Número:	2	Nombre:	Modificador de contenido	
Usuario:	Administrador			
Modificación de la historia:	1	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Alta	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Alto	
Descripción:				
El sistema tiene que generar contenido modificado automáticamente dados unos parámetros de configuración. Este contenido debe ser extraído de los módulos preparados.				

Tabla 3.2: Historia de usuario 2 del sistema

Historia de usuario				
Número:	3	Nombre:	Creación de módulos	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Media	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Bajo	
Descripción:				
Crear módulos para enriquecer la configuración y, por tanto, el sistema.				

Tabla 3.3: Historia de usuario 3 del sistema

Historia de usuario				
Número:	4	Nombre:	Asistente de internacionalización	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Media	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Bajo	
Descripción:				
Modificar el sistema para aceptar varios idiomas y ofrecer ayuda al usuario para poder incluirlos.				

Tabla 3.4: Historia de usuario 4 del sistema

3.5.2. Historias de la aplicación web

Como la web y el sistema tienen unos requerimientos totalmente desacoplados, el número de historia de estos se cuenta aparte.

Historia de usuario				
Número:	1	Nombre:	Registro de usuarios	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Alta	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Alto	
Descripción:				
La web debe registrar usuarios, que son los que la usarán.				

Tabla 3.5: Historia de usuario 1 de la web

Historia de usuario				
Número:	2	Nombre:	Ingreso de usuarios	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Alta	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Alto	
Descripción:				
Los usuarios correctamente registrados deben tener acceso a la aplicación.				

Tabla 3.6: Historia de usuario 2 de la web

Historia de usuario				
Número:	3	Nombre:	Modificación de datos de usuario	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Baja	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Bajo	
Descripción:				
La web debe permitir que se modifiquen datos de usuario. Los datos modificables serán los que formen parte de los módulos, si este lo permite.				

Tabla 3.7: Historia de usuario 3 de la web

Historia de usuario				
Número:	4	Nombre:	Creación de eventos	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Alta	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Alto	
Descripción:				

Tabla 3.8: Historia de usuario 4 de la web

Historia de usuario				
Número:	5	Nombre:	Ingreso en eventos	
Usuario:	Administrador			
Modificación de la historia:	X	Iteración asignada:		
Prioridad de Negocio: (Alta/Media/Baja)	Alta	Riesgo en Desarrollo: (Alto/Medio/Bajo)	Alto	
Descripción:				
Los usuarios pueden contactar con el creador del evento y, en caso de que este dé su consentimiento, el usuario podrá formar parte del evento.				

Tabla 3.9: Historia de usuario 5 de la web

3.6. Propuesta y planificación software

El plan principal para construir la web y definir el sistema se puede entender con el siguiente diagrama:

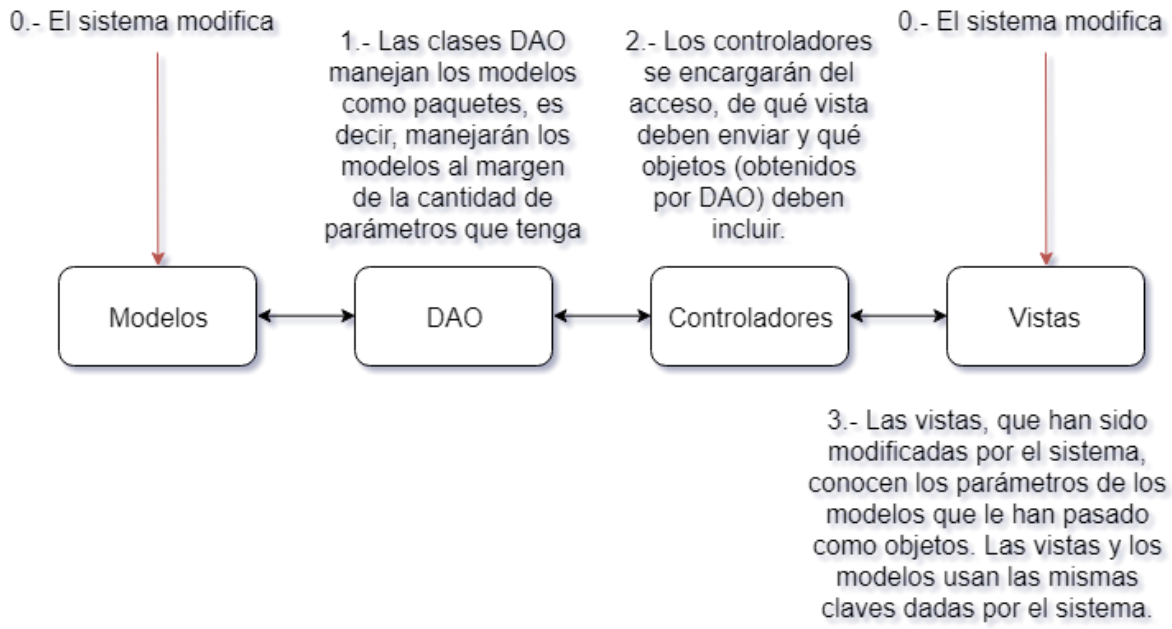

Ilustración 3.1: Esquema conceptual del conjunto

La web sigue la arquitectura MVC (modelo-vista-controlador), incluyendo una capa intermedia entre los modelos y el controlador, usando el patrón DAO (Data Access Object).

La arquitectura MVC tiene como objetivo separar los datos (modelos), la lógica (controladores) y la interfaz de usuario (vistas) en tres componentes independientes.

El patrón DAO sirve para manejar el acceso y utilización de los datos de forma segura. Así podemos separar la lógica de negocio y el acceso a los datos, algo fundamental para proyectos de gran envergadura.

Haciendo un recorrido lineal, el plan del sistema es:

0. El sistema modifica la web (una vez).
1. El patrón DAO gestiona las funciones CRUD (Create-Read-Update-Delete).
2. El controlador llama a esas funciones para pedir los modelos como objetos, o para modificarlos y eliminarlos.
3. Las vistas reciben los objetos que ha enlazado el controlador, el servidor hará un renderizado de la página y se la enviará al cliente.

La solución construida cumple, a priori, con todos los requisitos básicos necesarios para montar una base sólida sobre la que extender el desarrollo. Y mantiene la siguiente estructura.

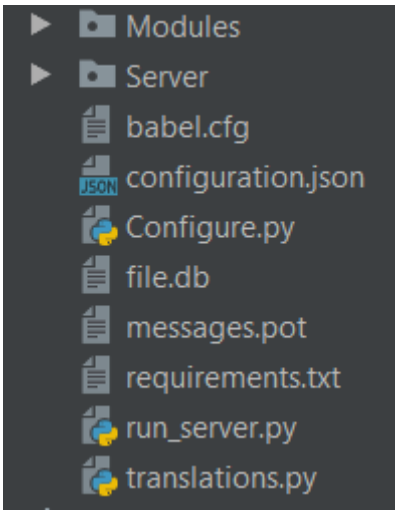

Ilustración 3.2; Organización de archivos

Como se puede apreciar en la figura, se ha estructurado el sistema de esta forma:

- Carpeta *Modules*: Carpeta donde se situarán los módulos de la configuración.
- Carpeta Server: Carpeta con las vistas, modelos, controladores y archivos estáticos necesarios y básicos para hacer funcionar el servidor.
- babel.cfg: Archivo de configuración de la extensión Flask-Babel para la internacionalización.
- configuration.json: Es el archivo de configuración para construir la aplicación web. Este es el único archivo que debe ser modificado por el usuario, siguiendo las normas básicas de la documentación.
- Configure.py: Este será el ejecutor del sistema, desde él se iniciará la configuración.
- file.db: Base de datos SQLite para desarrollos rápidos. Puede ser cambiado con facilidad sin alterar la aplicación gracias a la abstracción del ORM.
- messages.pot: Archivo base de traducción para la aplicación. Necesario para poder insertar nuevos idiomas.
- requirements.txt: Archivo de texto con las dependencias necesarias en un formato entendible por el gestor Pip.
- run_server.py: Archivo que inicia la aplicación.

- translations.py: Gestor de traducciones. Siguiendo la documentación, a través de la línea de comandos se pueden extender los idiomas necesarios.

3.7. Justificación de la solución

Para hacer el proyecto que tenemos en mente, hemos de elegir las herramientas que nos faciliten el trabajo.

Para el backend se ha pensado en la posibilidad de usar distintos lenguajes de programación, como puede ser PHP, Ruby, Java, Javascript… pero nos hemos decantado por Python.

Ventajas de Python sobre el resto de competidores:

- Código claro y limpio.
- Gran cantidad de librerías y frameworks para trabajar.
- Fácil de entender y montar.

Sobre Python hemos elegido el microframework flask por estas razones:

- Es sencillo de entender y usar.
- Se tiene mucho control.
- Es liviano, por lo que funciona bien para APIs

Para el frontend nuestra decisión ideal fue en un principio usar tecnologías que llevan los controladores al lado del cliente ya que quitaría carga de trabajo al servidor, que sólo tendría que intercambiar datos mediante una API y pasar los archivos requeridos cuando fuese necesario. Lamentablemente, por falta de tiempo, se decidió recurrir al concepto clásico de arquitectura MVC, con los controladores en el cliente y las vistas renderizadas por el servidor. Igualmente se hicieron pruebas para los frameworks Angular y Angular2 y la librería React.

3.8. Descripción de la solución

Para describir la solución al completo tendremos que dividir las partes como ya hemos hecho anteriormente. Por un lado, se describirá la herramienta, que es la encargada de montar el esqueleto de la aplicación web para el intercambio de servicios y, por otro lado, se describirá la web, que será la aplicación con la que el usuario final interactúe.

3.8.1. Descripción de la herramienta

La herramienta extrae de cada módulo dos partes:

- data.json: Donde se encontrarán los fragmentos de texto relativos a cada archivo que será modificado. Estos fragmentos de texto serán parte del html o código de Python que se insertará en la aplicación. Dentro de estos fragmentos de texto hay unas palabras clave que serán modificadas. (Para más detalle, ver el anexo C)
- vars: Conjunto de palabras clave que serán modificadas del texto, suelen ser para crear palabras claves para la base de datos y formularios, descripciones, etc.

La herramienta sustituirá cada palabra clave dentro del texto y lo añadirá al archivo correspondiente.

3.8.2. Descripción de la web

Como se entiende, estas aplicaciones social-colaborativas funcionan principalmente en dispositivos móviles por la disponibilidad y dinamismo de estos. Sabiendo esto, la web se ha hecho adaptada principalmente a dispositivos móviles, tal como se mostrará en las capturas. Como anotación importante, las capturas mostrarán la web sin configuración aplicada, es decir, vacía.

Toda página de este ámbito necesita una página de entrada y registro. La base de esta página de inicio es una plantilla de W3School, ya que se ha usado W3.CSS para crear la interfaz.

Ilustración 3.3; Vista inicial de nuestra aplicación web

Ilustración 3.4: Ejemplo de error de nuestra aplicación web

A excepción de la palabra *BRAND* (Marca) que aparece en varias vistas, todo el texto de la aplicación está adaptado para ser traducido. Además, se ha hecho un tratamiento de errores y seguridad básico ya que, como reiteramos, es un esqueleto.

El formulario de inicio de sesión requiere un usuario y una contraseña. El formulario de registro, en cambio, pide además una cuenta de correo. Esto se debe a que esta aplicación usa el servicio de Gravatar para los avatares de usuario. Como se explicó anteriormente, esta parte podía ser tercerizada gracias a protocolos como OAuth.

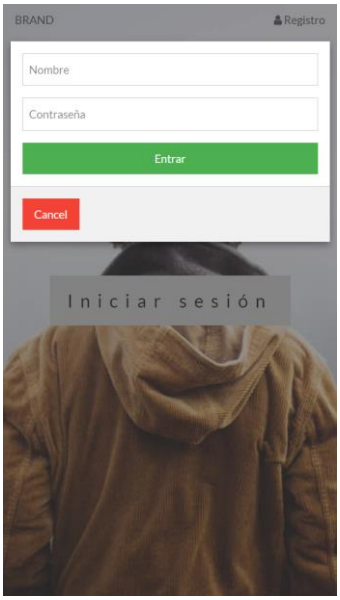

Ilustración 3.5: Formulario de inicio de sesión de nuestra aplicación

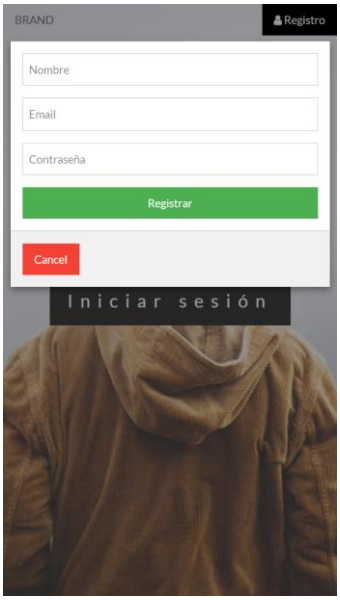

Ilustración 3.6: Formulario de registro de nuestra aplicación

Una vez registrados y dentro de la aplicación, se ha aprovechado el estudio anterior para distribuir las secciones de la aplicación. Tal como indicamos, el espacio de trabajo (publicar, buscar, y ver eventos) se ha situado en la parte inferior, mientras que el perfil y las notificaciones se han dejado en la parte superior.

La solución ha aprovechado mucho la influencia de la aplicación analizada anteriormente en el modo de operar. En la parte de publicar y buscar tenemos un formulario que se adaptará a las necesidades según la configuración. En la parte de ver tendremos cada evento en bloque, dispuestos según indican los módulos.

Es importante saber que en el apartado *ver* no sólo aparecerán nuestros eventos, también aparecerán los eventos ajenos en los que hayamos sido aceptados. En definitiva, en ver aparecerán todos los eventos que realmente nos importan.

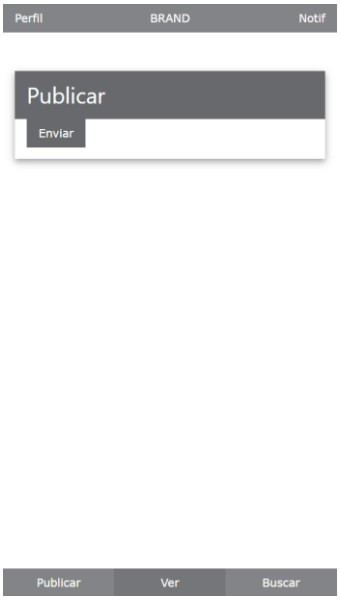

Ilustración 3.7: Creador de eventos de nuestra aplicación (sin configuración)

Ilustración 3.8: Visor de eventos de nuestra aplicación (sin configuración)

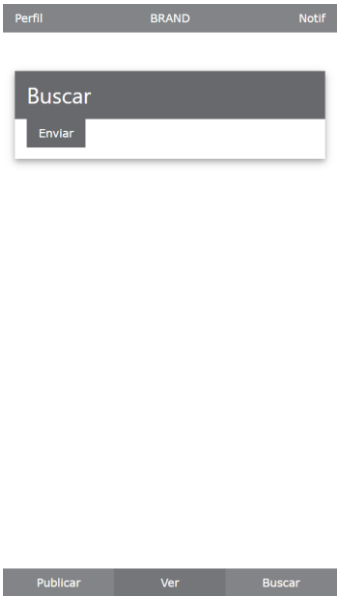

Ilustración 3.9: Buscador de eventos de nuestra aplicación (sin configuración)

En la parte superior, nos encontramos con dos botones, uno para el perfil y acciones relativas a este como es editarlo o cerrar sesión, y uno de notificaciones, que mostrará ahí la gente que ha decidido contactar con el usuario para un evento concreto.

Ilustración 3.10: Visor de perfil (sin configuración)

Ilustración 3.11: Editor del perfil (sin configuración)

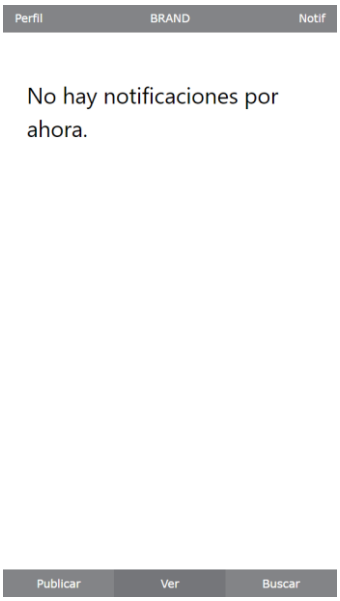

Ilustración 3.12: Visor de notificación vacío

3.8.3. Descripción de la web configurada con la herramienta

Una vez correctamente configurada la web con nuestra herramienta siguiendo el manual de uso e instalación, obtendremos una web como la del siguiente ejemplo. En él se puede apreciar cómo el formulario de publicar ha cambiado, añadiendo distintos checkboxes, dos selectores, un campo fecha, un campo hora y un campo de texto, que enriquecen el evento que, en este caso, pretende ser análogo a la aplicación analizada. Cada sección se muestra de una forma particular pudiendo complicarse tanto como sea necesario.

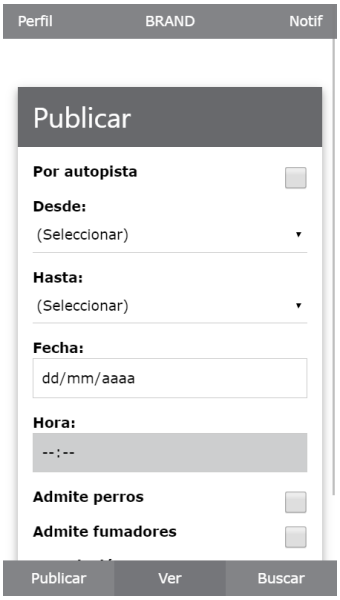

Ilustración 3.13: Creador de eventos (configurado)

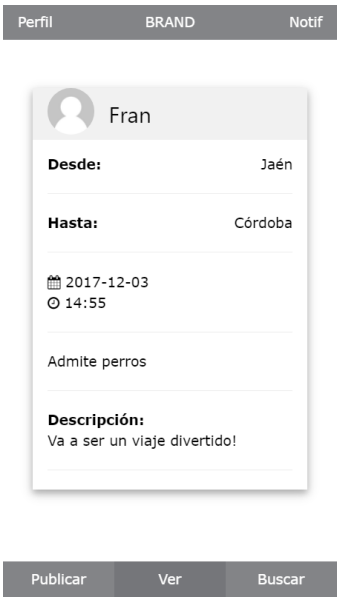

Ilustración 3.14: Visor de eventos con un evento (configurado)

El apartado de búsqueda es algo más complejo, ya que en la configuración se debe decidir qué parámetros se pueden buscar y qué parámetros no. En este ejemplo hemos elegido un subconjunto de ellos. El resultado de la búsqueda mostrará los eventos en los que uno puede participar y dará la opción de contactar con el creador.

Ilustración 3.15: Buscador de eventos (configurado)

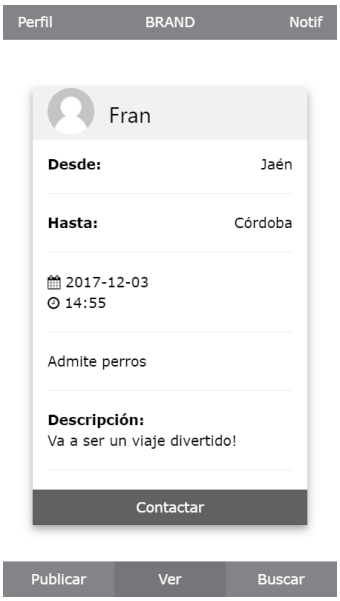

Ilustración 3.16: Resultado del buscador de eventos

De igual forma que el evento, el usuario también admite características para definir ciertos parámetros. En el caso del ejemplo hemos puesto un checkbox para saber si es fumador o no.

Ilustración 3.17: Perfil de usuario (configurado)

Ilustración 3.18: Editor de perfil (configurado)

Las únicas vistas que no serán modificadas serán la de inicio/registro y la de notificaciones, quedando igual que en las ilustraciones 3.6, 3.7 y 3.13. En este caso, al poder producir eventos, podemos mostrar cómo se verá un evento para cualquier configuración.

Ilustración 3.19: Visor de notificaciones con una notificación

3.9. Herramientas para la implementación de la solución

3.9.1. Python

"From the earliest version in the late 1980s to the current version, it has evolved with the same philosophy: providing a multiparadigm programming language with readability and productivity in mind.

People used to see Python as yet another scripting language and wouldn't feel right about using it to build large systems. However, over the years and thanks to some pioneer companies, it became obvious that Python could be used to build almost any kind of system."

(Michał Jaworski y Tarek Ziadé, 2008)

Python fue una elección segura hecha antes incluso de atacar este proyecto. Este lenguaje ha demostrado ser la elección perfecta para hacer desarrollos multiplataforma, claros y concisos.

3.9.2. Flask

"Flask is a lightweight Web framework written in Python. Flask started out as an April fool's joke that became a highly popular underdog in the Python web framework world. It is now one of the most widely used Python web frameworks for start-ups, and is becoming commonly accepted as the perfect tool for quick and simple solutions in most businesses. At its core, it provides a set of powerful libraries for handling the most common web development tasks."

(Copperwaite, M. y Leifer, C. 2015)

Flask, al igual que Python, fue una elección tomada desde el primer momento que se decidió comenzar este proyecto. Gracias a la libertad y a su ligereza, permite hacer desarrollos rápidos y bien organizados.

3.9.3. SQLAlchemy (Flask-SQLAlchemy)

"*SQLAlchemy is a powerful object relational mapper that allows us to abstract away the complexities of multiple database engines, to work with the database directly from within Python.*"

(Copperwaite, M. y Leifer, C. 2015)

Cuando se empezó el proyecto, se usaron scripts SQL para manejar las bases de datos, complicando el desarrollo muchísimo más de lo necesario por desconocimiento de la existencia de los ORMs (Object Relational Mappers). Los ORM gestionan las bases de datos directamente desde su código objetivo, como SQLAlchemy es para Python, Hibernate es para Java, etcétera. Gracias a ellos podemos crear modelos y gestionarlos directamente como objetos, creando funciones específicas para ellos, haciendo prácticamente invisible la base de datos.

Otros de los principales puntos positivos es que se puede conectar una base de datos SQLite (por defecto de la herramienta), MySQL, entre otras, sin realizar grandes modificaciones.

3.9.4. Flask-Babel

Flask-Babel es una extensión de Flask que añade soporte para la internacionalización y localización de cualquier aplicación Flask con ayuda de babel, pytz y speaklater.

3.9.5. W3.CSS

W3.CSS fue el framework CSS elegido tras muchas pruebas e intentos con otros frameworks más completos. Se acabó eligiendo este por distintas razones:

- Es más ligero y rápido.
- Es realmente fácil cambiar la gama de color. Ofreciendo, en teoría, gamas de colores seguros.

- Está pensado para adaptarse a móviles.
- No necesita javascript.
- Está muy bien documentado, es sencillo de usar y encaja muy bien con el desarrollo rápido que se proponía.

4. Conclusiones y trabajos futuros

4.1. Conclusiones

Este proyecto no trata de ser más que la constatación de que prácticamente todo el software es susceptible de englobarse dentro de un conjunto similar y de que la metaprogramación (programar programas que escriben programas) puede tener una gran utilidad en este aspecto.

El principal problema de este tipo de aplicaciones es la ausencia de normativa a favor o en contra de ellas. Mucho se ha hablado sobre las diferencias entre aplicaciones tales como *Uber* y *Blablacar*, donde, según parece, la principal diferencia es que el objetivo de una es lucrarse mientras que la otra es compartir gastos, igual que lo mucho que se ha hablado del daño que han hecho a taxistas y transportistas en general. De forma análoga, cualquier aplicación creada con esta mentalidad podrá ser el centro de muchas polémicas ya que son aplicaciones centradas por y para usuarios. En la siguiente ilustración podemos ver ejemplos en los que la tecnología ha desbancado al negocio tradicional.

Ilustración 4.1: Imagen "tecnología y diseño centrado en el consumidor"

Ilustración 4.1

4.2. Trabajos futuros

En este campo hay muchísimo por hacer. Este proyecto no deja de ser un primer encuentro o un toque de atención sobre cómo unas pocas aplicaciones, con el mismo trasfondo, han cambiado la forma de pensar y de proceder del mundo.

Quizá algún día haya aplicaciones para todo, o quizá un día con una aplicación podamos adquirir cualquier servicio, pero lo que es innegable es que el mundo ya está conectado y la gente puede sacar provecho de él.

Este proyecto es fácilmente extensible por la comunidad, ya que los módulos pueden ser creados de forma mecánica. El primer paso para poder hacer realmente útil la herramienta es hacer un repaso de toda su funcionalidad en busca de elementos faltantes. Por ejemplo, no hay vista de editar eventos, pero no por olvido, sino por decisión de diseño. Los eventos son de creación y de acceso rápido y editarlos podría suponer un problema contra dicha filosofía, además de una complicación innecesaria.

El siguiente paso sería añadir secciones para incluir javascript de la misma manera y generar filtros preproceso y postproceso para los controladores, dando así una gran cantidad de personalización manteniendo una complejidad relativamente baja.

Otra vía posible de extensión o actualización sería generar distintos temas y estructurar los módulos por temas, así podríamos tener el mismo sistema con un tema usando W3.CSS que con Bootstrap.

Por último, se podría rediseñar completamente la aplicación para llevar los controladores al lado del cliente y tan solo mantener una API, pudiendo generar aplicaciones híbridas con Ionic, por ejemplo.

Cabe destacar que se ha intentado tener en cuenta desde un primer momento criterios como la internacionalización y el uso de colores seguros, sin embargo, queda muchísimo trabajo por hacer en cuestión de accesibilidad.

Bibliografía

Jaworski, M. Ziadé, T. (2016). *Expert Python Programming. Second Edition.*

Kasampalis, S. (2015). *Mastering Python Design Patterns.*

F. Lott, S. (2016). *Modern Python Cookbook.*

Copperwaite, M. Leifer, C. (2015). *Learning Flask Framework.*

Enlaces

https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

http://chuwiki.chuidiang.org/index.php?title=Patr%C3%B3n_DAO

https://www.w3schools.com/w3css/default.asp

http://flask-sqlalchemy.pocoo.org/2.1/

https://pythonhosted.org/Flask-Babel/

Anexo A: Manual de instalación

Instalar Python y Pip

Antes de comenzar a instalar el software, debemos preparar el entorno y las dependencias necesarias. Para ello ejecutaremos:

- Linux

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install python python-dev python-virtualenv
```

- Windows

  Para Windows iremos a la web python.org y bajaremos la última versión ejecutable (actualmente 3.6.1).

Instalar Virtualenv, crear y activar un entorno

Una vez esté instalado Python y pip, ejecutaremos en un terminal:

```
$ pip install virtualenv
```

Teniendo Python, Pip y Virtualenv instalado, nos dirigimos a la carpeta CreadorDeServicios y ahí ejecutaremos desde un terminal:

```
$ virtualenv flask
```

Para activar el entorno en Linux:

```
$ source flask/bin/activate
```

En Windows:

```
$ flask\Scripts\activate.bat
```

Instalar dependencias

Una vez preparado el entorno, toca instalar las dependencias, para ello debemos ejecutar la siguiente orden:

```
$(flask) pip install -r requirements.txt
```

** Si alguna dependencia falla en sistemas Windows, hay que seguir los siguientes pasos.

El principal error que puede dar en la instalación es un pequeño fallo de pip en Windows que se corrige fácilmente siguiendo estos pasos.

- Vamos a la ruta …\CreadorDeServicios\flask\Lib\site-packages\pip\compat
- Abrimos el archivo __init__.py con un editor de texto
- Localizamos sys.__stdout__.encoding
- Sustituimos por “latin-1”

Como se puede apreciar en las siguientes ilustraciones:

```
if sys.version_info >= (3,):
    def console_to_str(s):
        try:
            return s.decode(sys.__stdout__.encoding)
        except UnicodeDecodeError:
            return s.decode('utf_8')
```

Ilustración A.1: Fragmento de código sin modificar

```
if sys.version_info >= (3,):
    def console_to_str(s):
        try:
            return s.decode("latin-1")
        except UnicodeDecodeError:
            return s.decode('utf_8')
```

Ilustración A.2: Fragmento de código modificado

Anexo B: Manual de uso de la herramienta

Configurar parámetros

Cada módulo viene con un archivo llamado vars. Ese archivo nos dirá las variables de ese módulo para ser configurado.

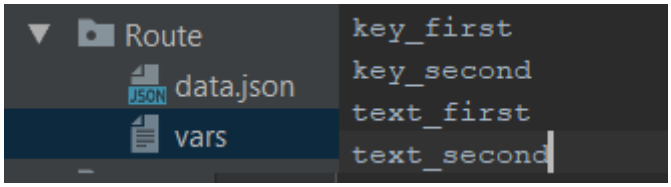

Ilustración B.1: Ejemplo de variables del módulo "Route"

Usando el módulo *Route* como ejemplo, vemos que tiene cuatro variables, llamadas *key_first, key_second, text_first, text_second*.

Los módulos deberían venir con una nota aclaratoria de qué es cada clave. Una vez sabemos qué es cada clave y qué parámetros configurar, nos vamos a *configuration.json*

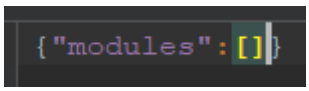

Ilustración B.1: Configuración de módulos vacío.

y añadimos el módulo a la lista.

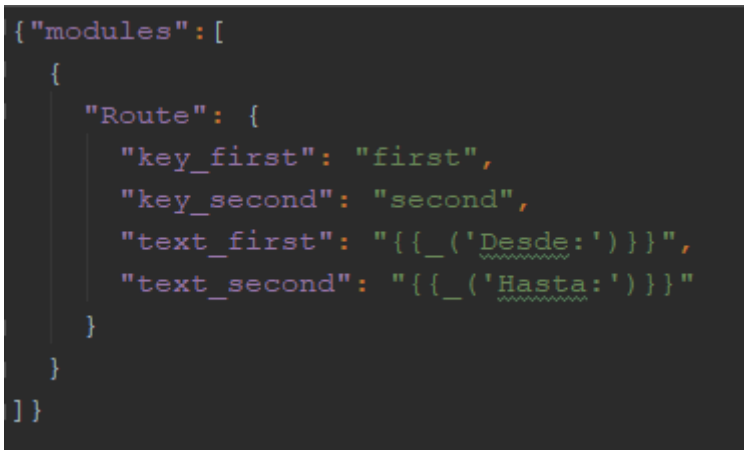

Ilustración B.2: Configuración con un módulo

Se pueden añadir todos los módulos que sean necesarios evitando que sus claves se repitan.

Si se escriben los textos usando *{{_('<texto>')}}*, ese texto quedará marcado como internacionalizable. Si el texto no cambia en otro idioma, se puede escribir directamente.

Una vez configurado al gusto. Dentro de nuestro entorno ejecutamos:

```
$(flask) Configure.py
```

Si los parámetros son correctos, la ejecución terminará con éxito y tendremos la herramienta lista para ser ejecutada.

Añadir internacionalización

Tras una ejecución correcta de la configuración, obtendremos un archivo “messages.pot” en el que se deberían encontrar todos los fragmentos de texto a traducir.

Dentro de nuestro entorno ejecutamos:

```
$(flask) translate.py add <tag del idioma>
```

Por ejemplo, *es* para español, *en* para inglés…

Eso nos generará en “Server/translations” una carpeta para cada idioma. Traduce cada texto al idioma respectivo y ejecuta:

```
$(flask) translate.py compile
```

Ejecutar servidor

Una vez configurado y añadido al path, ejecutamos:

```
 $ run_server.py
```

Con la configuración descrita, se generará una aplicación con las siguientes vistas:

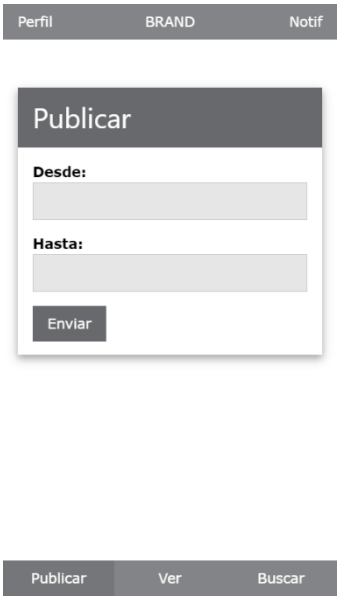

Ilustración B.3: Creador de eventos con el módulo Route

Ilustración B.5: Evento con módulo Route

Anexo C: Manual para construir módulos

Cada módulo se compone de dos archivos, *data.json* y *vars.*

Crear el código HTML del módulo

Lo primero de todo será crear el fragmento sobre las vistas y comprobar que las vistas se ven correctamente. Para que sea más claro se hará la creación de un módulo checkbox para eventos.

En la vista create, creamos los inputs necesarios con todo, tal como lo queramos mostrar, teniendo en cuenta que usamos W3.CSS y Jinja2. Es **imprescindible** que el nombre coincida con la clave del modelo tal como se explicará más adelante.

```
    <!-- AutoCreate -->

<!-- CHECKBOX -->
<p>
<label><b>{{_('Por autopista')}}</b></label>
<input class='w3-check w3-right w3-theme-l4' name='road' type='checkbox'>
</p>
<!-- END CHECKBOX -->



    <!-- End AutoCreate -->
```

Ilustración C.1: Ejemplo de fragmento HTML

En la vista search actuamos de forma análoga. En la vista see, mostramos el evento tal como queramos que aparezca. Nótese que en la ilustración siguiente usamos **ev** que es el objeto del evento y **road** que es el nombre del input y del atributo del modelo.

```
      <!-- AutoSee -->
<!-- CHECKBOX -->
{% if ev.road %}
<p>{{_('Por autopista')}}</p>
<hr>
{% endif %}
<!-- END CHECKBOX -->


      <!-- End AutoSee -->
```

Ilustración C.2: Ejemplo de uso de "ev" y Jinja2

Crear el código de los modelos

Al igual que con el código HTML, lo único que hay que hacer es añadir los campos que necesite el modelo, en *models.py*.

Es necesario que el modelo y los atributos del input tengan el mismo nombre en clave, ya que el controlador no podrá enlazar correctamente dichos atributos si no es así.

```
    # [AutoEvent]
    # CHECKBOX
    road = db.Column(db.Boolean, nullable=False, default=False)
    # END CHECKBOX

    # [End]
```

Ilustración C.3: Fragmento de código Python

Comprobación del código

Tras hacer esto, al ejecutar el servidor, debería mostrar un evento con un checkbox tal como lo hemos planteado. Si hay algún problema con la comunicación entre los datos, se tendrá que revisar dónde ha podido fallar.

Pasar el código a un módulo

Para ello, nos vamos a la carpeta *Modules* y creamos una carpeta con el nombre que hayamos decidido ponerle. Ahí creamos un archivo *data.json* que será donde estará incluido el código. El archivo *data.json* debe ir estructurado de la siguiente forma y con las siguientes palabras clave. Si alguna vista o modelo no es modificado por el módulo, tan solo hay que quitarlo.

```
{
  "DDBB":{
    "Event":"Código Python en línea\n sin tabular",
    "User":"Código Python en línea\n sin tabular"
  },
  "CREATE":"<p class='comillas simples'>\nTexto HTML en línea</p>",
  "SEE":"<p class='comillas simples'>\nTexto HTML en línea</p>",
  "PROFILE":"<p class='comillas simples'>\nTexto HTML en línea</p>",
  "PUB_PROFILE":"<p class='comillas simples'>\nTexto HTML en línea</p>",
  "SEARCH":"<p class='comillas simples'>\nTexto HTML en línea</p>"
}
```

Ilustración C.4: Distintas claves posibles para data.json

El texto debe ir en línea, añadiendo \n para los saltos de línea y \" para comillas dobles, aunque se recomienda usar siempre comillas simples.

Hacer el módulo dinámico

Con todo lo anterior, tenemos un módulo estático, sin ningún tipo de configuración por parte del usuario. Ahora vamos a dotar de un carácter dinámico al módulo añadiendo variables. Las variables serán añadidas entre dos barras **||** por delante y por detrás, como se puede apreciar en el ejemplo siguiente.

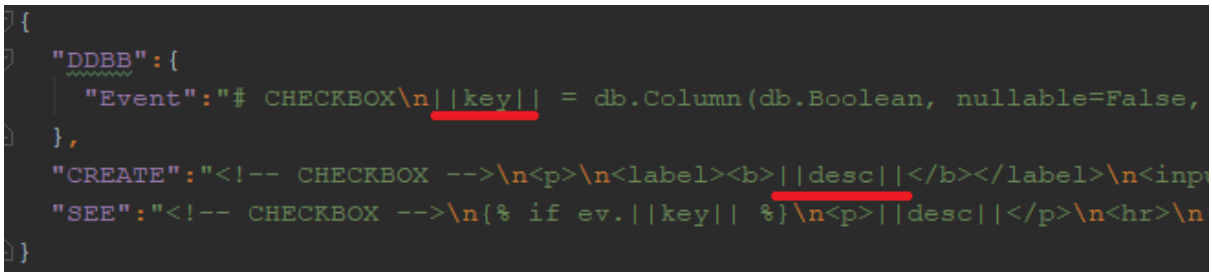

Ilustración C.5: Ejemplo de variable dentro de data.json

Siguiendo el ejemplo, sustituimos *road* por ||*key*|| y *{{_("Por autopista")}} por* ||*desc*||. Ahora creamos junto al archivo *data.json* un archivo llamado *vars* donde añadiremos las variables que existen dentro del módulo.

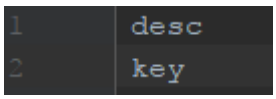

Ilustración C.6: Ejemplo de archivo vars

Con esto, el usuario ya podrá usar el módulo creado por nosotros tal como explicamos en el anexo B, con una configuración como este ejemplo.

Ilustración C.7: Ejemplo de uso del módulo construido

Anexo D: Manual de uso de la web

En esta parte se pretende explicar cómo funciona la web y qué puede hacer un usuario con ella.

Registro

Desde la página principal, un usuario podrá registrar una nueva cuenta rellenando el formulario pertinente, tal como se puede apreciar en la ilustración D.1

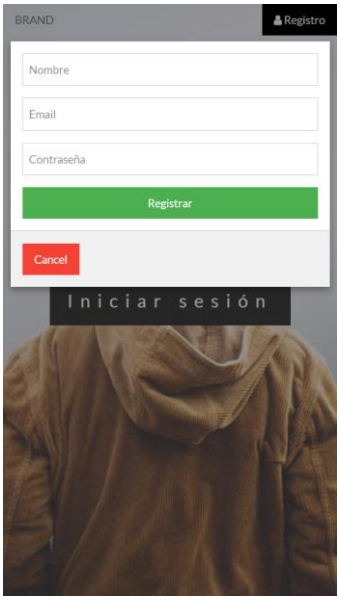

Ilustración D.1: Formulario de registro.

Inicio de sesión

Tras haber realizado un registro correcto, podremos iniciar sesión desde la página principal, como se puede ver en la ilustración D.2

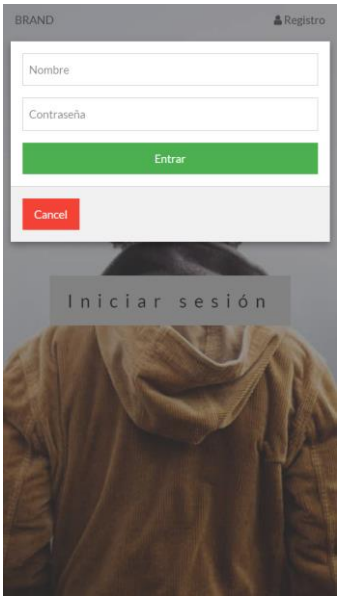

Ilustración D.2: Formulario de inicio de sesión

Crear un evento

Con una sesión iniciada, podremos crear eventos soportados por la aplicación.

Para ello debemos pulsar en el botón “Publicar”, situado en la parte inferior de la interfaz. Tras esto, nos saldrá el formulario que debemos cumplimentar correctamente y pulsar “Enviar”. Si todo ha ido bien, se podrá ver el evento creado en el apartado “Ver”, como en la siguiente ilustración.

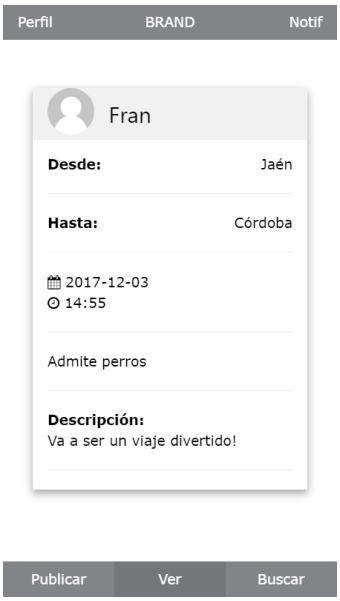

Ilustración D.3: Visor de eventos con un evento configurado

Buscar un evento

Basta con pulsar en el botón *Buscar* en la parte inferior de la interfaz y rellenar uno o más campos que sean de interés. Tras esto recibiremos los eventos disponibles con esos criterios de búsqueda como se puede apreciar en la siguiente ilustración.

Ilustración D.4: Buscador de eventos (configurado)

Apuntarse a un evento

Una vez encontrado un evento que sea de interés, el usuario tan solo debe pulsar en el botón *Contactar* situado en dicho evento. El creador recibirá una notificación y decidirá si acepta o no la participación del usuario en él.

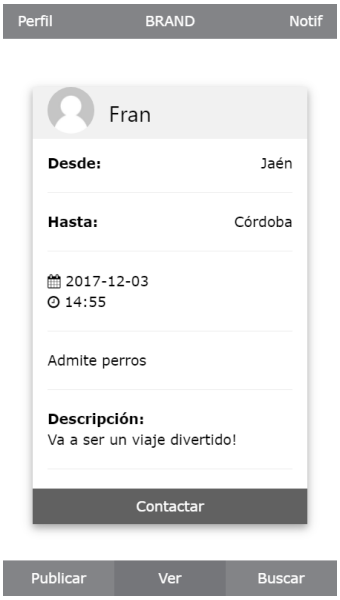

Ilustración D.5: Resultado de una búsqueda

Gestionar notificaciones

En la parte superior de la interfaz, donde pone *Notif.* podemos acceder a las notificaciones. Las notificaciones serán de personas que han intentado contactar con el usuario por un evento concreto. Debemos elegir si aceptar o no a dicha persona en el evento.

Ilustración D.6: Ejemplo de notificación

Ver/Editar perfil

Situado en la parte superior de la interfaz, podremos localizar el botón *Perfil*. Pulsando este seremos dirigidos a nuestro perfil, desde donde podremos cerrar sesión o editar.

Para editar el perfil hay que modificar los parámetros tal como queramos que se muestren y pulsar *Enviar.*

Ilustración D.7:Visor de perfil (configurado)

Ilustración D.8: Editor de perfil (configurado)

Anexo E: Índice de ilustraciones

Anexo F: Índice de tablas