

Preferiría que no



Arquitectura e Integración de Sistemas Software

Grado de Ingeniería del Software

Curso 2017/2018

Alonso Valenzuela, Juan Carlos (juaaloval@alum.us.es)
Cadenas Sánchez, María Jesús (marcadsan2@alum.us.es)
Gil Baeza, David (davgilbae@alum.us.es)

Tutor: Alfonso Eduardo Márquez Chamorro
Número de grupo: 2

Enlace de la aplicación: <http://www.prefeririaqueno-aiss.appspot.com>
Enlace de proyecto en projETSII, GitHub o similar:

HISTORIAL DE VERSIONES

Fecha	Versión	Detalles	Participantes
22/03/2018	1.0	-Incluye introducción, prototipos de las interfaces de usuario y diagramas UML de componentes y despliegue.	Juan Carlos Alonso M.ª Jesús Cadenas David Gil
06/05/2018	2.0	-Incluye revisión del primer entregable, diagrama de clases, diagramas de secuencia, manual de API Rest y despliegue	Juan Carlos Alonso M.ª Jesús Cadenas David Gil
27/05/2018	3.0	-Incluye revisión del segundo entregable, implementación del mashup completo siguiendo el patrón MVC, desarrollo de una API propia y documentación de las pruebas realizadas.	Juan Carlos Alonso M.ª Jesús Cadenas David Gil

Índice

1	Introducción	4
1.1	Aplicaciones integradas	4
1.2	Evolución del proyecto.....	5
2	Prototipos de interfaz de usuario.....	6
2.1	Vista Search.....	6
2.2	Vista Results	7
2.3	Vista Business.....	8
2.4	Vista Save.....	9
2.5	Vista Success	10
2.6	Vista Error	11
3	Arquitectura.....	12
3.1	Diagrama de componentes	12
3.2	Diagrama de despliegue.....	12
3.3	Diagrama de secuencia de alto nivel	13
3.4	Diagrama de clases	14
3.5	Diagramas de secuencia.....	15
3.5.1	Obtener Restaurantes	15
3.5.2	Obtener Restaurante	15
3.5.3	Guardar evento en Google Calendar	16
3.5.4	Confirmar guardar evento en Google Calendar	16
4	Implementación.....	17
5	Pruebas.....	21
6	Manual de usuario.....	27
6.1	Mashup	27
6.2	API REST	31
	Referencias	33

1 Introducción

Una de las costumbres más establecidas en nuestra sociedad es la innovación a la hora de salir a comer. Visitar siempre los mismos restaurantes y convertirlo en una rutina suele suponer un problema, sobre todo para los jóvenes. Las frases “¿Qué cenamos hoy?”, “¿Dónde vamos?” se repiten una y otra vez, sumado a todo ello el trajín actual, el cual puede llevar a olvidar citas importantes.

Por ello proponemos una solución compuesta por cuatro APIs: Yelp!, Geocoding, Embed Maps y Google Calendar. El cliente accede a una página en la cual escoge una ubicación y opcionalmente el tipo de comida que desea consumir. Una vez proporcionadas estas condiciones, se le muestra un listado de opciones disponibles provenientes de Yelp!. Cuando el usuario selecciona una de las opciones, accede a la información de dicho restaurante (tipo de comida, horario, distancia, valoraciones), etc. Junto a toda esta información, se puede apreciar un mapa con el recorrido desde la ubicación introducida en la página principal o de la ubicación actual del usuario hasta el restaurante elegido.

Desde esta página podemos decidir entre “¡Anótalo en Google Calendar” o “Preferiría que no”, dando la opción de crear un evento en Google Calendar (esta página te redirige hacia el inicio de sesión de Google) y de volver a la página de opciones disponibles, respectivamente. Cuando presionamos en “¡Anótalo en Google Calendar!", el usuario accede a una página en la cual tiene la opción de crear un evento que se añadirá automáticamente a su calendario, dando la opción a modificar el título (siendo por defecto el nombre del restaurante), añadir una descripción del evento (lugar, acompañantes, etc), elegir una fecha distinta a la de hoy, así como la hora.

Con ellos creamos un mashup capaz de mostrarnos otras opciones distintas a lo habitual, ayudando a conocer sitios nuevos, al igual que nos muestra el camino y nos permite guardar el evento para que no se nos olvide. Y con todo esto, ya solo queda... ¡disfrutar de la comida!

1.1 Aplicaciones integradas

Nombre aplicación	URL documentación API
Yelp!	https://www.yelp.com/developers
Embed Maps	https://developers.google.com/maps/documentation/embed/?hl=es-419
Geocoding	https://developers.google.com/maps/documentation/geocoding/intro?hl=es-419
Google Calendar	https://developers.google.com/calendar/

TABLA 1. APLICACIÓN INTEGRADAS

1.2 Evolución del proyecto

Inicialmente nuestro proyecto estaba basado en otras aplicaciones Cabify, Zomato y Foodspotting. Tras comenzar a investigar las APIs, nos dimos cuenta de que la aplicación Zomato no operaba en España, por lo que preferíamos alguna que pudiera estar operativa en territorio nacional. Por esta razón cambiamos a TripAdvisor.

Comenzando de nuevo nuestra búsqueda, y al intentar solicitar la key para acceder a la API de TripAdvisor, nos notificaron que no otorgaban permisos para investigación, por lo que migramos a Yelp!.

Centrándonos después en encontrar una aplicación en la cual compartir la experiencia, optamos en primer lugar por Foodspotting, pero rápidamente fue descartada al darnos cuenta de las limitaciones de su API, y que no poseía muchos usuarios que la utilizaran. Por estas razones, concluimos que lo más adecuado sería aplicar Vero, una nueva red social en auge.

Por razones de poder probar nuestro mashup en clase, tuvimos que descartar la idea de usar Uber. También tuvimos que desechar la API de Vero, ya que, al consultar la información, ésta dejaba mucho que desear, por lo que no era muy recomendable.

Por tanto, finalmente, nuestro mashup está integrado por Yelp!, Geocoding, Embed Maps y Google Calendar, permitiendo al usuario elegir entre varios restaurantes, mostrando el camino hacia el restaurante. Una vez elegido el restaurante, el usuario puede crear la cita en Google Calendar, evitando así que se olvide del evento.

2 Prototipos de interfaz de usuario

2.1 Vista Search

En esta vista, el usuario establece una ubicación y elige un tipo de comida (Opcional) y clickando en “¡Comienza la búsqueda!” se accederá a la lista de opciones de restaurante.



www.preferirlaqueno-appspot.com/search

Hoy quiero comer en:

Q Sevilla

Elige un tipo de comida ▼

¡Comienza la búsqueda!

FIGURA 1. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA SEARCH

2.2 Vista Results

En esta vista el usuario observa la lista de restaurantes de acuerdo con lo seleccionado en vista Search. Al clicar sobre el nombre de una de las opciones, le redirigirá a la vista Business.

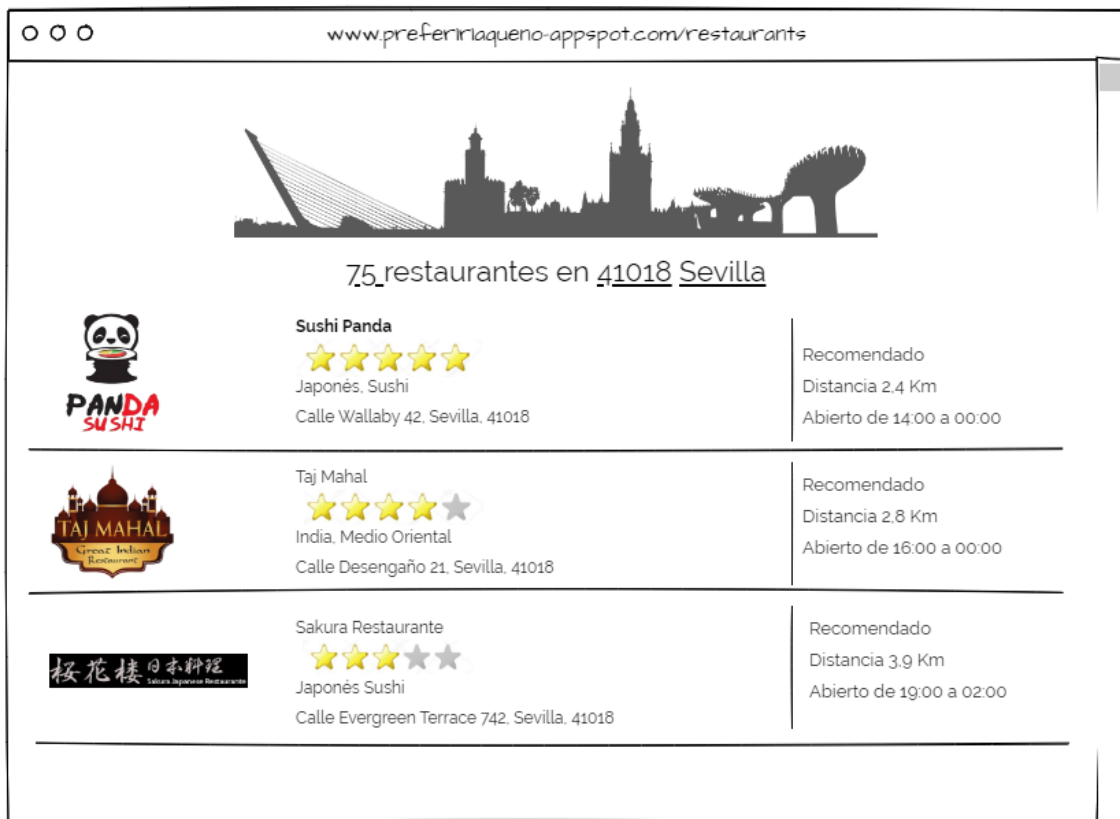


FIGURA 2. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA RESULTS

2.3 Vista Business

En esta vista el usuario accede a la información más detallada del restaurante seleccionado, así como un mapa que muestra el recorrido desde la ubicación establecida o la ubicación actual del usuario hasta el restaurante. En esta vista, el usuario tiene la opción de presionar sobre “¡Anótalo en Google Calendar!” y “Preferiría que no”, la cual le llevara hasta la vista Save en la que el usuario creará un evento en Google Calendar o, por el contrario, le redirige de nuevo a la vista Results respectivamente.

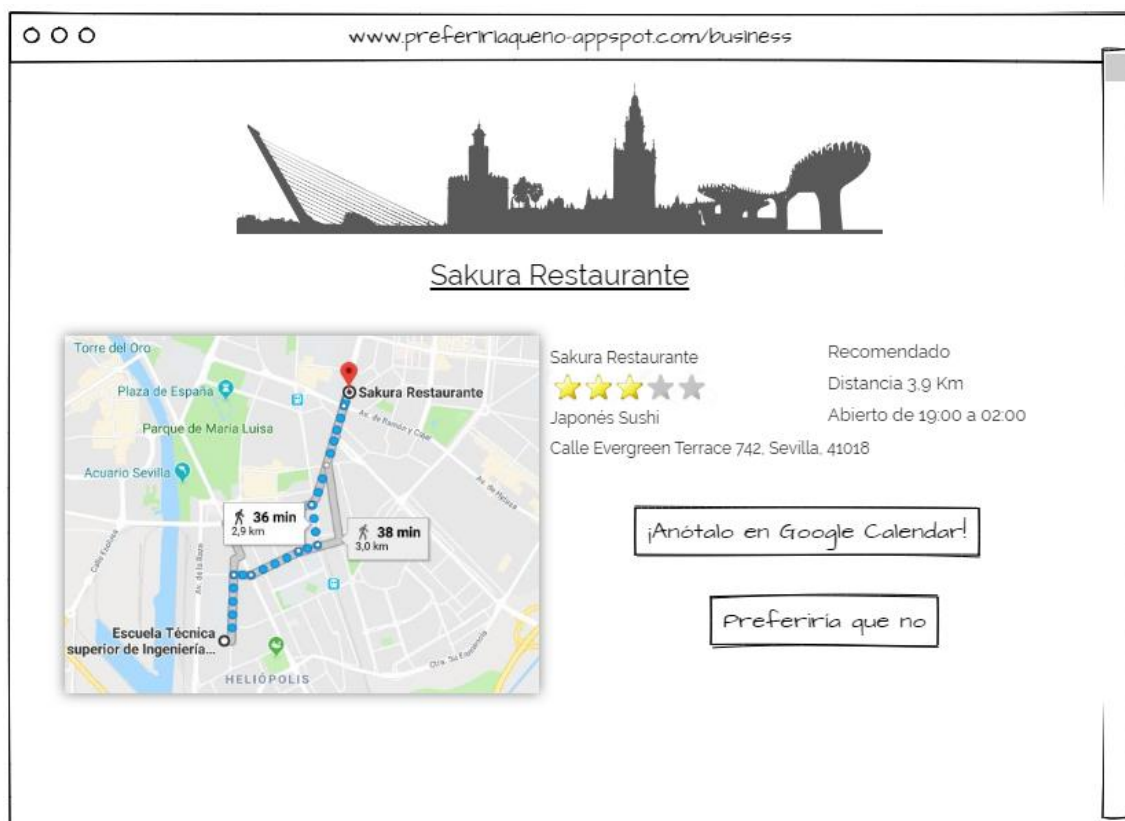


FIGURA 3. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA BUSINESS

2.4 Vista Save

En esta vista, el usuario crea un evento en su cuenta de Google Calendar. El título del evento vendrá predefinido por el nombre del restaurante, pudiendo ser modificado. El usuario podrá añadir una descripción (acompañantes, ubicación, información relevante), elegir una fecha posterior distinta a la actual, al igual que elegir la hora del evento. Al clicar sobre “¡Anótalo en Google Calendar!” se guardará la cita en su cuenta.

www.preferirlaqueno-appspot.com/save

Titulo: Sakura Restaurante

Descripción: Añadir información adicional...

June 2018

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Hora

¡Anótalo en Google Calendar!

FIGURA 4. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA SAVE

2.5 Vista Success

En esta vista, se avisa al usuario de que el evento se ha creado correctamente en su cuenta de Google Calendar. Te da la opción de volver a la página de inicio para hacer otra búsqueda.



FIGURA 5. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA SUCCESS

2.6 Vista Error

La vista Error es una vista genérica que se mostrará si ocurre algún error. Desde esta vista se ofrece la opción de volver a la página de inicio (vista Search).



FIGURA 6. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA ERROR

3 Arquitectura

3.1 Diagrama de componentes

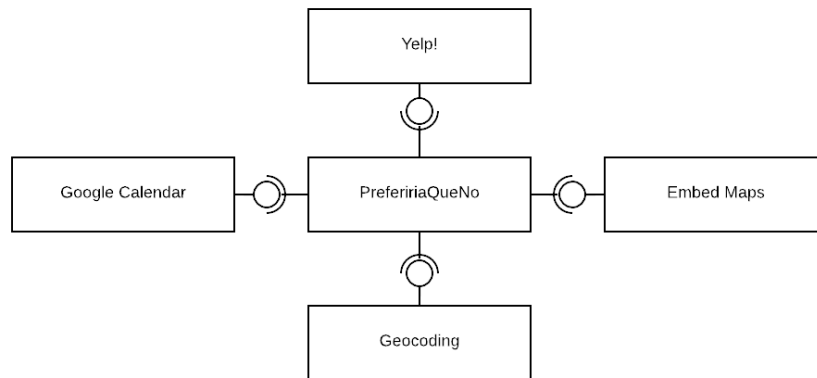


FIGURA 7. DIAGRAMA DE COMPONENTES

3.2 Diagrama de despliegue

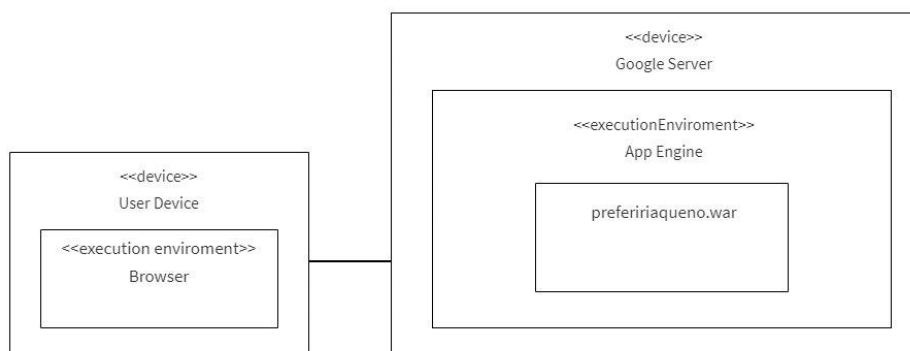


FIGURA 8. DIAGRAMA DE DESPLIEGUE

El nodo de la izquierda representa al usuario. Está compuesto por el dispositivo del usuario y el entorno en el que ejecuta la aplicación (en este caso su Navegador Web). El usuario (Cliente) realiza peticiones a la aplicación (Servidor).

El nodo de la derecha representa el punto de vista del servidor, compuesto por la aplicación que vamos a desarrollar, que será ejecutada sobre AppEngine, que funciona a través del Servidor de Google. El Servidor se encargará de atender las peticiones del cliente.

3.3 Diagrama de secuencia de alto nivel

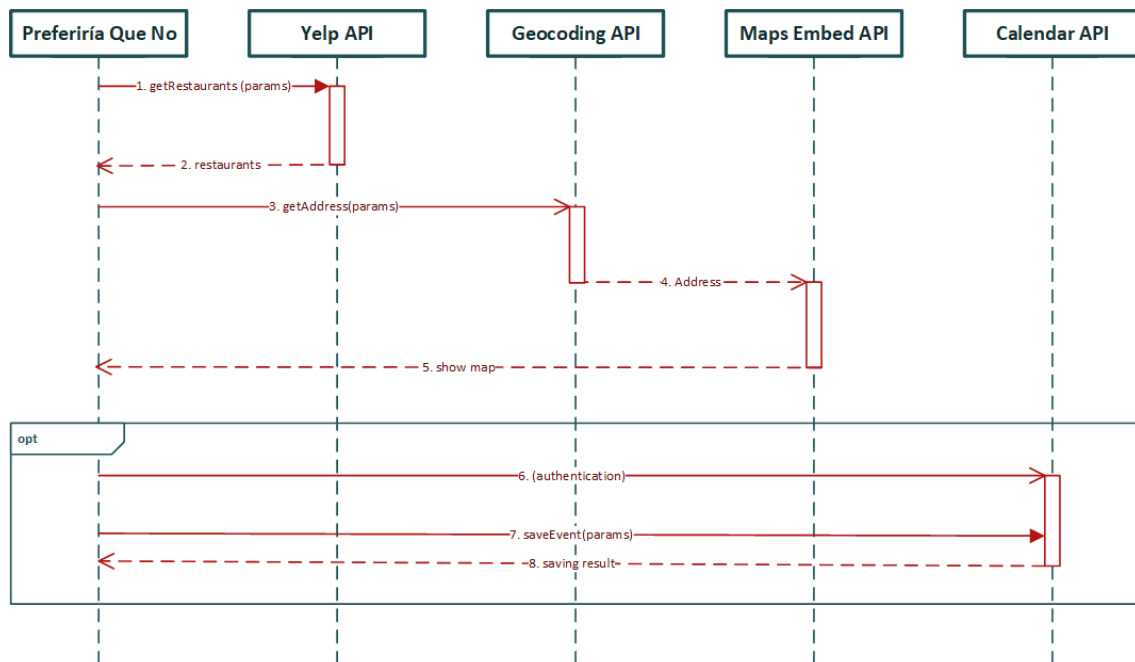


FIGURA 9. DIAGRAMA DE SECUENCIA DE ALTO NIVEL

Empleando nuestra aplicación, el usuario solicita una lista de restaurantes pasando unos parámetros. La aplicación accede a la API de Yelp! y obtiene una lista a partir de la ubicación y del tipo de restaurante escogido. De esta forma, el usuario tiene ante sí una lista de posibles restaurantes.

3.4 Diagrama de clases

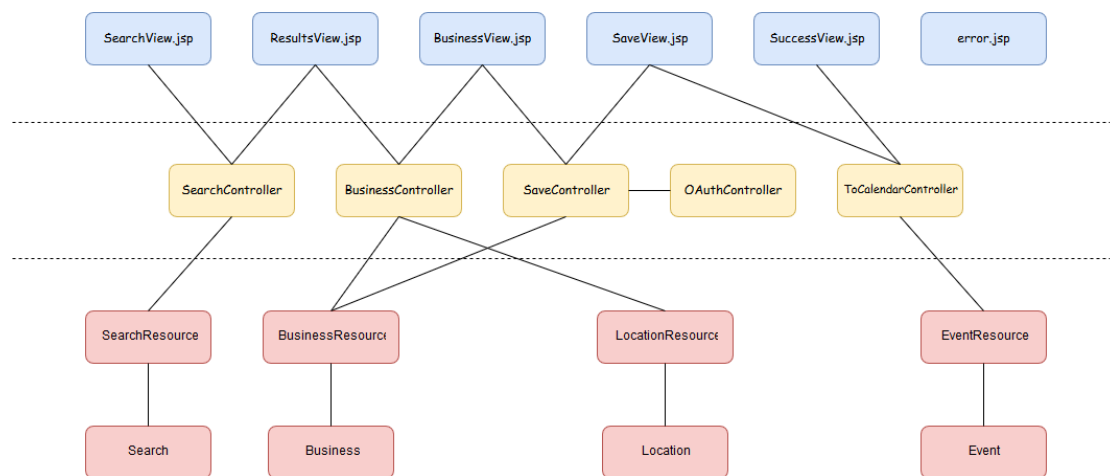


FIGURA 10. DIAGRAMA DE CLASES

Diagrama UML de clases indicando la distribución de las clases entre las distintas capas, según el patrón MVC.

3.5 Diagramas de secuencia

3.5.1 Obtener Restaurantes

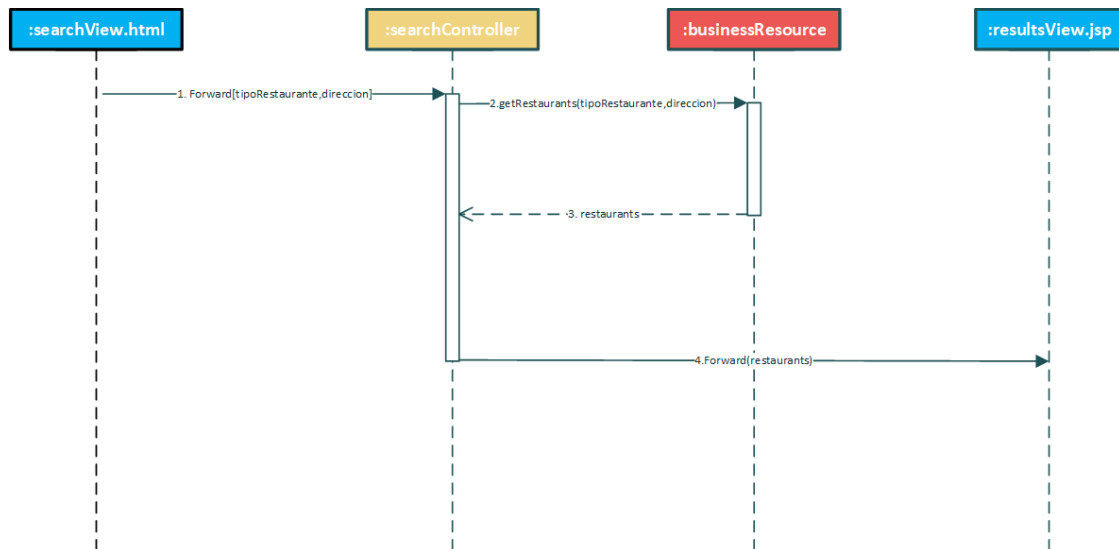


FIGURA 11. DIAGRAMA DE SECUENCIA OBTENER RESTAURANTES

3.5.2 Obtener Restaurante

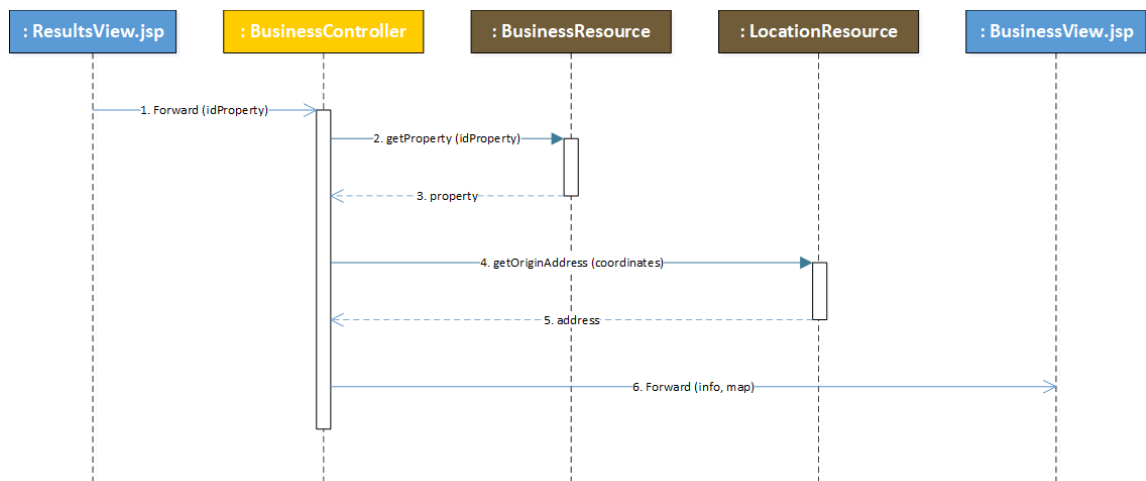


FIGURA 12. DIAGRAMA DE SECUENCIA OBTENER RESTAURANTE

3.5.3 Guardar evento en Google Calendar

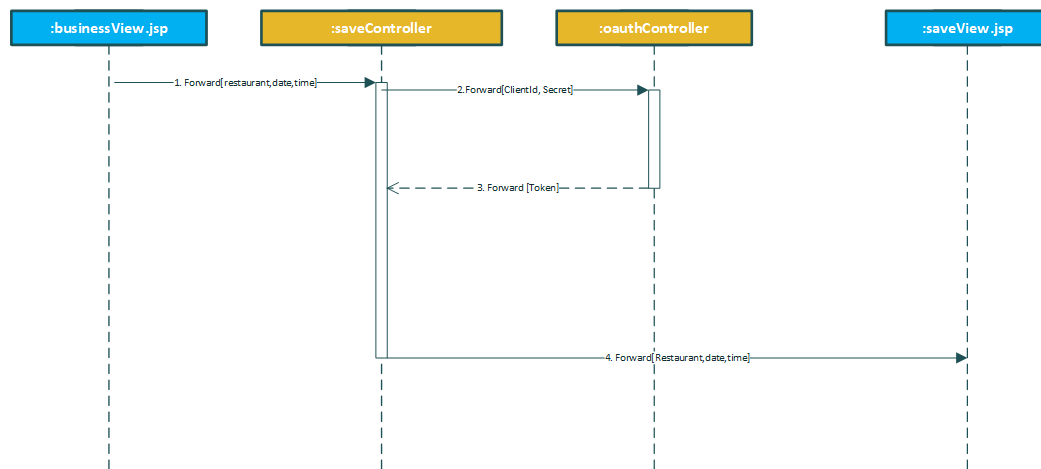


FIGURA 13. DIAGRAMA DE SECUENCIA GUARDAR EVENTO EN GOOGLE CALENDAR

3.5.4 Confirmar guardar evento en Google Calendar

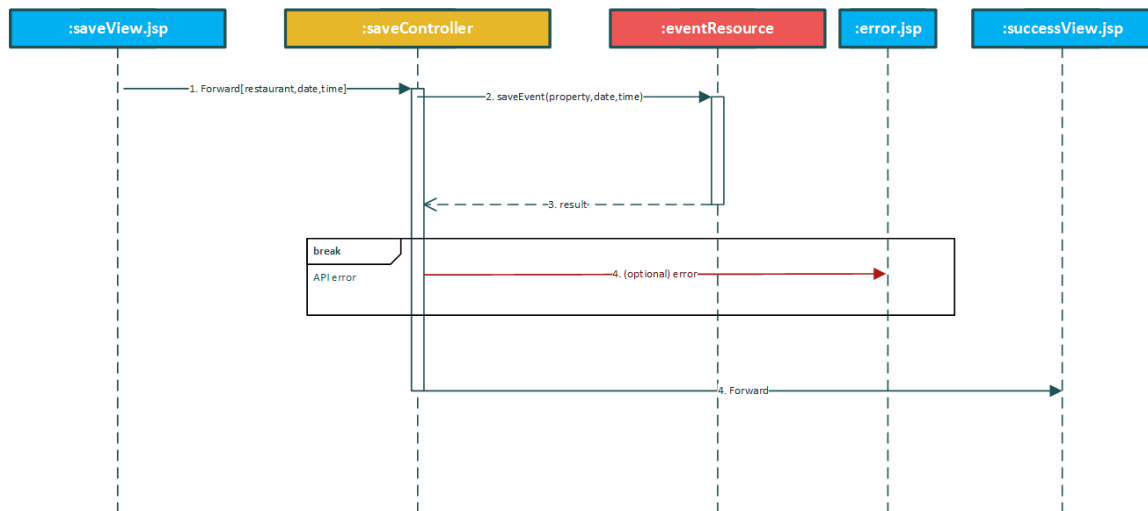


FIGURA 14. DIAGRAMA DE SECUENCIA CONFIRMAR GUARDAR EVENTO EN GOOGLE CALENDAR

4 Implementación

Durante el desarrollo de la aplicación nos hemos encontrado con diferentes problemas o dificultades que han ido suponiendo retos a superar.

En nuestro mashup usamos la geolocalización de HTML5 para obtener las coordenadas del usuario y gracias a la API de Geocoding poder conseguir su dirección exacta. Para poder hacer esto, necesitábamos de forma obligada usar https, por lo que tuvimos que añadir esto al web.xml:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>all</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

A la hora de implementar los controladores, el más complejo es el BusinessController, el cual se encarga de mostrar un mapa de Google con la localización del restaurante que hemos seleccionado y todos sus datos. Además, se brinda al usuario la opción de poder determinar una ruta desde su ubicación al restaurante. Para ello, el navegador solicita permiso para obtener la ubicación. Una vez concedido, el usuario pulsará el botón “¡Obtener ruta!” y de nuevo el controlador genera la vista con el nuevo mapa. Para ello necesitamos la geolocalización de HTML5 y la API de Geocoding. Este es el código del controlador:

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String id = request.getParameter("id");

    if(id!=null) {
        log.log(Level.FINE, "ID: " + id + " cargado con
        éxito.");
    } else {
        log.log(Level.WARNING, "No se ha realizado la carga
        del parámetro ID");
        request.setAttribute("message", "No se ha realizado
        la carga del parámetro ID");

        request.getRequestDispatcher("/ErrorView.jsp").forward(request,
        response);
    }

    request.setAttribute("id", id);
    BusinessResource resource = new BusinessResource();
    BusinessDetails business = resource.getBusiness(id);
```

```

        log.log(Level.INFO, "Objeto restaurante creado con éxito");

        request.setAttribute("business", business);

        String destination = business.getName() + " ";
        for(String s:business.getLocation().getDisplayAddress()) {
            destination = destination + " " + s;
        }
        String place = URLEncoder.encode(destination, "UTF-8");

        if(request.getParameter("locationForm")==null) {
            String map = "place?q=" + place;
            request.setAttribute("map", map);
        } else {
            String lat = request.getParameter("lat");
            String lon = request.getParameter("lon");

            if(lat!=null && lon!=null) {
                log.log(Level.FINE, "Latitud: " + lat + " y longitud: " + lon + " cargadas con éxito");
            } else {
                log.log(Level.WARNING, "No se ha realizado la carga de los parámetros latitud y longitud");
                request.setAttribute("message", "No se ha realizado la carga de los parámetros latitud y longitud");
            }

            request.getRequestDispatcher("/ErrorView.jsp").forward(request, response);
        }

        LocationResource resource2 = new LocationResource();
        CurrentLocation location = resource2.getLocation(lat, lon);

        log.log(Level.INFO, "Objeto localización creado con éxito");

        String origin = URLEncoder.encode(location.getResults().get(0).getFormattedAddress(), "UTF-8");

        String map = "directions?origin=" + origin + "&destination=" + place;
        request.setAttribute("map", map);
    }

    request.getRequestDispatcher("/BusinessView.jsp").forward(request, response);
}

```

Por otro lado, para hacer un POST de un evento en Google Calendar, necesitamos indicar la fecha y hora de inicio y fin del evento en un formato RFC3339, lo que nos planteó una gran dificultad en principio. Finalmente hemos usado Joda Time, que nos proporciona los métodos que nos ayudan a realizar esto. Este es el fragmento del código incluido dentro del controlador ToCalendarController que genera el String con la fecha y hora de comienzo del evento:

```
DateTime fechal = new DateTime(año, mes, dia, hora, minutos,
    DateTimeZone.forID("Europe/Madrid"));
    DateTimeFormatter dateFormatter =
    ISODateTimeFormat.dateTime();
    String startRFC = dateFormatter.print(fechal);
```

Por otra parte, la API de Yelp nos pedía incluir la API_KEY en la cabecera HTTP de la siguiente manera: "Authorization: Bearer <API_KEY>". Intentamos hacerlo de diferentes maneras, pero no conseguíamos dar con la solución. Finalmente, con este código (incluido en SearchResource y en BusinessResource) lo pudimos solucionar:

```
String pAccessToken = "Bearer " + API_KEY;
    ChallengeResponse challengeResponse = new
    ChallengeResponse(
        new ChallengeScheme("", ""));
    challengeResponse.setRawValue(pAccessToken);
    cr.setChallengeResponse(challengeResponse);
```

Por último, a la hora de realizar pruebas con JUnit, no teníamos claro como hacer el test de Google Calendar, ya que para crear un evento necesitamos un Token de OAuth 2. Al final lo que hicimos fue implementar este clase JUnit, en la cual hay que introducir un Token de un usuario de Google para poder realizarla:

```
package aiss.model.repository;

import static org.junit.Assert.*;
import org.joda.time.DateTime;
import org.junit.Test;
import aiss.model.calendar.CalendarEvent;
import aiss.model.calendar.End;
import aiss.model.calendar.Start;
import aiss.model.resource.EventResource;

public class EventResourceTest {

    private static final String TOKEN
        = "TOKEN_GOOGLE_USUARIO";
```

```

@Test
public void testCreateEvent() {
    EventResource r = new EventResource(TOKEN);
    CalendarEvent ev = new CalendarEvent();
    ev.setDescription("Prueba: " + DateTime.now().toString());
    ev.setSummary("Test JUnit");
    ev.setLocation("ETSII");
    Start start = new Start();
    start.setDateTime("2018-06-05T21:30:00+02:00");
    End end = new End();
    end.setDateTime("2018-06-05T22:30:00+02:00");
    ev.setStart(start);
    ev.setEnd(end);
    ev.setStatus("confirmed");
    r.createEvent(ev);
    assertTrue("No se ha creado el evento",
        ev.getSummary().equals("Test JUnit"));
}
}

```

5 Pruebas

Hemos realizado pruebas incrementales ascendentes. Lo primero que desarrollamos de nuestra aplicación fue la parte del modelo con sus correspondientes recursos. Una vez desarrollada y probada con JUnit toda esta parte (ya que pensamos que la base de la aplicación y lo que primero debe funcionar debe ser el acceso a los recursos), nos dedicamos ya a desarrollar las vistas y los controladores y a comprobar y testear el correcto funcionamiento e integración con la capa de modelo.

Resumen	
Número total de pruebas realizadas	22
Número de pruebas automatizadas	22 (100%)

ID	Prueba 1
Descripción	Prueba para la detección de errores al realizar llamadas a la API de Yelp, en este caso llamadas GET.
Entrada	Se hace una llamada a la API con el ID concreto de un restaurante.
Salida esperada	Los datos obtenidos en forma de JSON hacen referencia al restaurante deseado, cuyo nombre es "HOB – House of Burger".
Resultado	EXITO
Automatizada	Sí

ID	Prueba 2
Descripción	Prueba para la detección de errores al realizar llamadas a la API de Yelp, en este caso llamadas GET.
Entrada	Se hace una llamada a la API con el ID concreto de un restaurante.
Salida esperada	Existe un restaurante con el ID indicado, y por tanto la respuesta es no nula.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 3
Descripción	Prueba para la detección de errores al realizar llamadas a la API de Geocoding, en este caso llamadas GET.
Entrada	Se hace una llamada a la API con latitud y longitud asociadas a un lugar conocido.
Salida esperada	Las coordenadas son correctas y la respuesta a la llamada es un JSON con los resultados asociados a esas coordenadas.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 4
Descripción	Prueba para la detección de errores al realizar llamadas a la API de Geocoding, en este caso llamadas GET.
Entrada	Se hace una llamada a la API con latitud y longitud inventadas, las cuales sabemos que son incorrectas.
Salida esperada	Las coordenadas son incorrectas y el JSON no contiene respuestas a dichas coordenadas.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 5
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se añade una nueva ciudad al repositorio.
Salida esperada	Tras añadir la ciudad el número de ciudades del repositorio aumenta.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 6
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se obtienen todas las ciudades del repositorio.
Salida esperada	Existen ciudades en el repositorio.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 7
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se obtiene una ciudad concreta del repositorio, mediante su ID, de la cual conocemos su nombre.
Salida esperada	Existe una ciudad en el repositorio con el nombre "Sevilla".
Resultado	EXITO
Automatizada	Sí

ID	Prueba 8
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se actualiza una ciudad del repositorio, mediante su ID, de la cual conocemos su información actual.
Salida esperada	Los datos de la ciudad se han actualizado correctamente.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 9
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se elimina una ciudad del repositorio, mediante su ID.
Salida esperada	La ciudad se elimina correctamente.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 10
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se obtienen todos los restaurantes del repositorio asociados a una ciudad, mediante el ID de la ciudad.
Salida esperada	Existen restaurantes en el repositorio asociados a la ciudad indicada.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 11
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se añade un restaurante a una ciudad, mediante sus IDs.
Salida esperada	El restaurante aparece dentro de la lista de restaurantes de la ciudad que hemos indicado.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 12
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se elimina un restaurante perteneciente a una ciudad, mediante sus IDs.
Salida esperada	El restaurante ya no aparece dentro de la lista de restaurantes de la ciudad que hemos indicado.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 13
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se añade un restaurante al repositorio interno de nuestra aplicación.
Salida esperada	El número de restaurantes del repositorio aumenta tras añadirlo.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 14
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se obtienen todos los restaurantes del repositorio.
Salida esperada	Existen restaurantes en el repositorio.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 16
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se obtienen un restaurante del repositorio, mediante su ID, del cual conocemos sus datos.
Salida esperada	Existe el restaurante indicado en el repositorio, cuyo nombre es "Sushi Factory".
Resultado	EXITO
Automatizada	Sí

ID	Prueba 17
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se actualiza un restaurante, mediante su ID, del cual conocemos sus datos.
Salida esperada	El restaurante ha sido actualizado correctamente.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 18
Descripción	Prueba para la detección de errores al realizar operaciones CRUD sobre el repositorio interno de nuestra propia API.
Entrada	Se elimina un restaurante, mediante su ID.
Salida esperada	Tras eliminar el restaurante, el número de restaurantes del repositorio decrementa.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 19
Descripción	Prueba para la detección de errores al realizar llamadas a la API de Google Calendar, en este caso llamadas POST.
Entrada	Se crea un evento con unos datos concretos en una cuenta de Google Calendar.
Salida esperada	El evento aparece creado con los datos correctos y deseados.
Resultado	EXITO
Automatizada	Sí

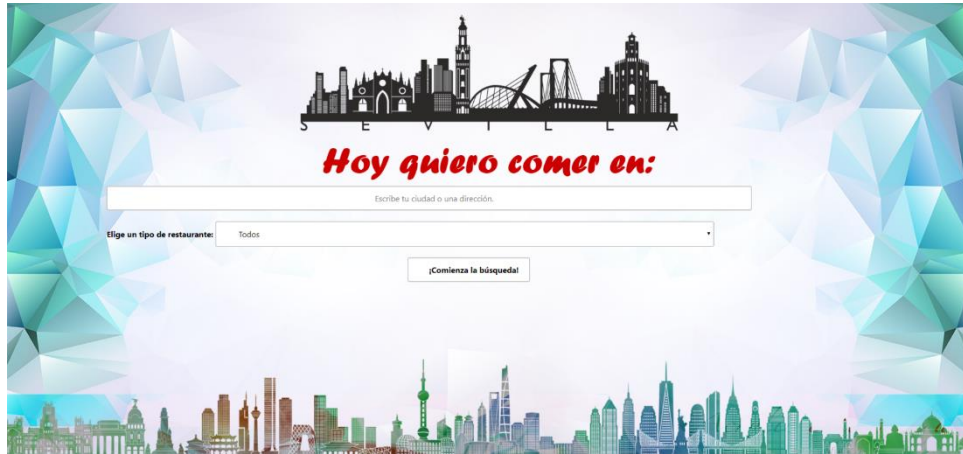
ID	Prueba 20
Descripción	Prueba para la detección de errores al realizar llamadas a la API de Yelp, en este caso llamadas GET.
Entrada	Se realiza una búsqueda de restaurantes en Sevilla.
Salida esperada	El JSON de respuesta contiene 20 resultados de restaurantes en Sevilla.
Resultado	EXITO
Automatizada	Sí

ID	Prueba 21
Descripción	Prueba para la detección de errores al realizar llamadas a la API de Yelp, en este caso llamadas GET.
Entrada	Se realiza una búsqueda de restaurantes en Sevilla, de la cual conocemos cual va ser el primer resultado.
Salida esperada	El JSON de respuesta contiene 20 resultados de restaurantes en Sevilla, y el primer resultado se corresponde con el restaurante "Eslava".
Resultado	EXITO
Automatizada	Sí

ID	Prueba 22
Descripción	Prueba para la detección de errores al formatear un objeto DateTime de JodaTime. La fecha formateada es necesaria para hacer un POST de un evento en Google Calendar.
Entrada	Se genera un DateTime y se formatea en un String con formato RFC3339.
Salida esperada	El String resultante tiene formato y datos correctos.
Resultado	EXITO
Automatizada	Sí

6 Manual de usuario

6.1 Mashup



Si escribimos prefeririaqueno-aiss.appspot.com en nuestro navegador web nos encontraremos con un formulario como el de la imagen superior, en el primer campo escribiremos una ciudad o una dirección donde queramos encontrar un restaurante, este campo es obligatorio. El segundo campo nos permite escoger el tipo de restaurante entre un repertorio de más de cien categorías, este campo es opcional.

Si pulsamos el botón "¡Comienza la búsqueda!" Tras haber rellenado el formulario, avanzaremos hasta la siguiente vista.

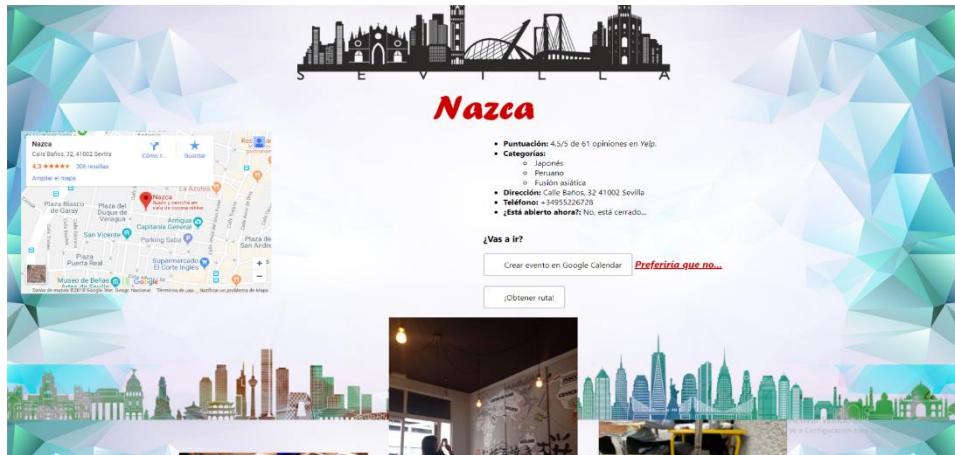


En esta vista se nos ofrecen un máximo de 20 Restaurantes en función de los parámetros de búsqueda insertados en la página anterior. Sobre cada restaurante se nos ofrecen los siguientes datos:

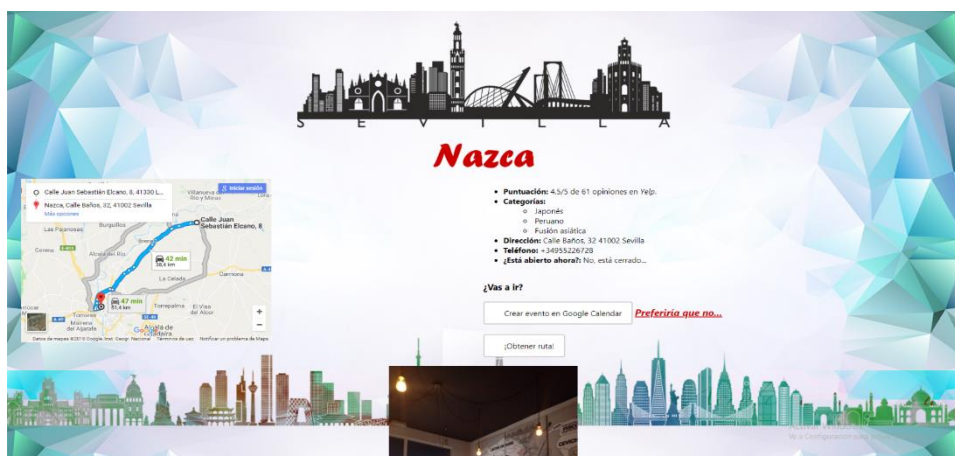
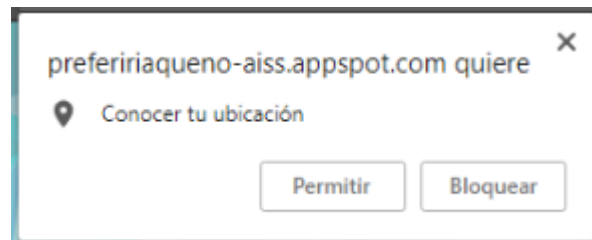
- ✓ Nombre
- ✓ Imagen
- ✓ Puntuación
- ✓ Categorías

✓ Dirección

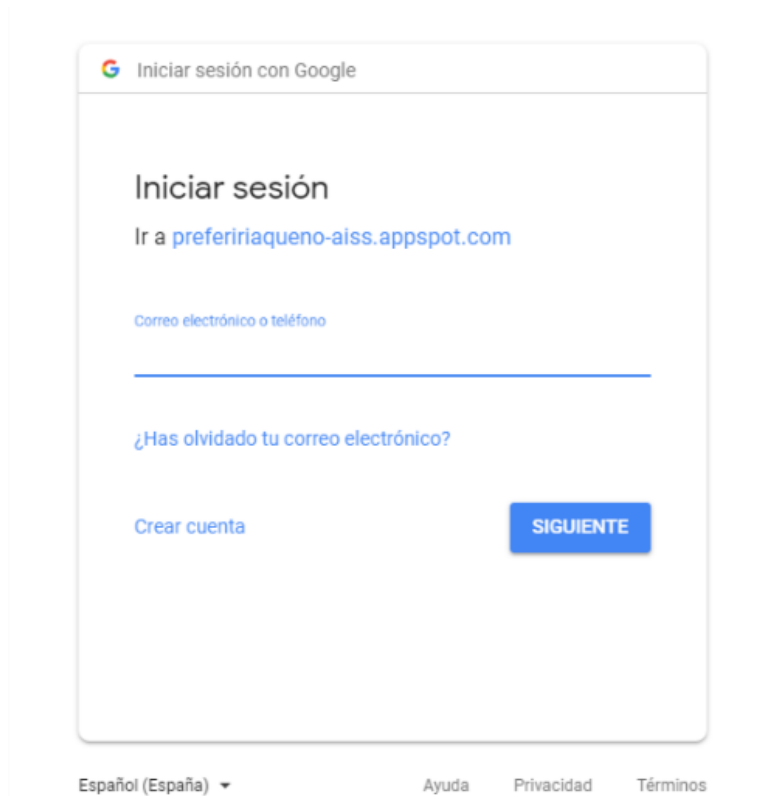
Si pulsamos sobre el nombre de un restaurante seremos redirigidos a la siguiente vista.



En esta vista se muestra un mapa con la ubicación del restaurante elegido anteriormente, además de contener más información sobre el mismo que en la vista anterior, como más imágenes. También nos dice si el restaurante está abierto en el momento de realizar la búsqueda. Si permitimos a la aplicación conocer nuestra ubicación actual podremos obtener una ruta hasta el restaurante pulsando en "¡Obtener ruta!". Si queremos crear un evento en Google Calendar podemos pulsar en "Guardar evento en Google Calendar". Por otra parte, si queremos seguir viendo restaurantes podemos pulsar en "Preferiría que no".



Si pulsamos en obtener ruta el mapa de la izquierda se actualizará, mostrando la ruta desde nuestra ubicación actual hasta el Restaurante en cuestión.

A screenshot of the Google login interface. At the top, it says "Iniciar sesión con Google" with the Google logo. Below that, the heading "Iniciar sesión" is followed by the URL "Ir a prefeririaqueno-aiss.appspot.com". There is a text input field labeled "Correo electrónico o teléfono". Below the field is a link "¿Has olvidado tu correo electrónico?". At the bottom left is a link "Crear cuenta" and at the bottom right is a blue button labeled "SIGUIENTE". The footer contains "Español (España)", "Ayuda", "Privacidad", and "Términos".

Iniciar sesión con Google

Iniciar sesión

Ir a prefeririaqueno-aiss.appspot.com

Correo electrónico o teléfono

[¿Has olvidado tu correo electrónico?](#)

[Crear cuenta](#) [SIGUIENTE](#)

Español (España) Ayuda Privacidad Términos

Si pulsamos en "Guardar evento en Google Calendar" tendremos que iniciar sesión en Google para avanzar hacia la siguiente vista.

A screenshot of a web form titled "¿Cómo quieres guardar el evento?". The form has a decorative background with a city skyline at the top and bottom. The form fields are: "Restaurante:" (Nazca), "Notas:" (Japones, Peruano, Fusión asiática. +34955226726), "Dirección:" (Nazca, Calle Banos, 32. 41002 Sevilla), "Fecha:" (dd/mm/aaaa), and "Hora:" (20:30 (HH:MM)). A "Guardar en Calendar" button is at the bottom.

¿Cómo quieres guardar el evento?

Restaurante: Nazca

Notas: Japones, Peruano, Fusión asiática. +34955226726

Dirección: Nazca, Calle Banos, 32. 41002 Sevilla

Fecha: dd/mm/aaaa

Hora: 20:30 (HH:MM)

[Guardar en Calendar](#)

Una vez hemos iniciado sesión con nuestra cuenta de Google debemos concretar la fecha y la hora del evento en un formulario como el de la imagen superior, también tenemos la opción de modificar el resto de campos, que ya tienen asignados un valor por defecto. Por último, pulsamos en "Guardar en Calendar" para pasar a la vista final.



Si todo ha ido bien seremos redirigidos a una vista de éxito en la que se nos informará de que el evento ha sido añadido correctamente a nuestro calendario. Si pulsamos en el enlace volveremos a la primera vista.

6.2 API REST

	<u>RECURSO RESTAURANTE</u>	
HTTP	URI	Descripción
GET	/restaurants	Devuelve una lista con todos los restaurantes disponibles.
GET	/restaurants/{restaurantId}	Devuelve el restaurante con id=restaurantId. Si el restaurante no existe devuelve: "404 Not Found".
POST	/restaurants	Añade un nuevo Restaurante cuyos datos se pasan en el cuerpo de la petición en formato JSON (la id se genera automáticamente). Si el nombre del restaurante no es válido (null o vacío) devuelve "400 Bad Request" Si se añade satisfactoriamente, devuelve "201 Created" con la referencia a la URI y el contenido del restaurante
PUT	/restaurants	Actualiza el restaurante cuyos datos se pasan en el cuerpo de la petición en formato JSON (debe incluir el id del restaurante) Si el restaurante no existe, devuelve un "404 Not Found". Si se realiza correctamente, devuelve "204 No Content".
DELETE	/restaurants/{restaurantId}	Elimina el restaurante con id=restaurantId Si el restaurante no existe, devuelve "404 Not Found". Si se realiza correctamente, devuelve "204 No Content".

	<u>RECURSO CIUDAD</u>	
GET	/ciudades	Devuelve una lista con todas las ciudades existentes
GET	/ciudades/{ciudadId}	Devuelve la ciudad con id=ciudadId. Si la ciudad no existe devuelve un "404 Not Found".
POST	/ciudades	Añadir una nueva ciudad. Los datos de la ciudad (nombre y descripción) se proporcionan en el cuerpo de la petición en formato JSON. Los restaurantes de la ciudad no se pueden incluir aquí, para ello se debe usar la operación POST específica para añadir un restaurante a una ciudad.
PUT	/ciudades	Actualiza la ciudad cuyos datos se pasan en el cuerpo de la petición en formato JSON (deben incluir el id de la ciudad). Si la ciudad no existe, devuelve "404 Not Found". Si se intenta actualizar los restaurantes de la ciudad, devuelve "400 Bad Request". Para actualizar los restaurantes se debe usar el recurso restaurant. Si se realiza correctamente, devuelve "204 No content"
DELETE	/ciudades/{ciudadId}	Elimina la ciudad con id=ciudadId. Si la ciudad no existe, devuelve "404 Not Found". Si se realiza correctamente, devuelve "204 No Content".
POST	/ciudades/{ciudadId}/{restaurantId}	Añade el restaurante con id=restaurantId a la ciudad con id=ciudadId. Si la ciudad o el restaurante no existen, devuelve "404 Not Found" Si el restaurante ya está en la ciudad devuelve un "400 Bad Request" Si se añade satisfactoriamente, devuelve "201 Created"
DELETE	/ciudades/{ciudadId}/{restaurantId}	Elimina el restaurante con id=restaurantId de la ciudad con id=ciudadId. Si la ciudad o el restaurante no existen, devuelve "404 Not Found". Si se realiza correctamente, devuelve "204 No Content".

Referencias

- [1] Librería Joda-Time. <http://www.joda.org/joda-time/>
- [2] Mock-ups. <https://www.mockflow.com/>
- [3] Documentación para la API. <https://app.swaggerhub.com/g>