

MORNING COFFEE



Arquitectura e Integración de Sistemas Software

Grado de Ingeniería del Software

Curso 2016-17

Santiago Borge Sánchez (santiaguito975@gmail.com)

Ángel María Mármol Fernández (angelmarmolfernandez@gmail.com)

Ángel Manuel Calzado Llamas (angelcalzadollamas@gmail.com)

Francisco Javier Abreu Vázquez (fran.abreu.3@gmail.com)

Tutor: José Antonio Parejo Maestre

Número de grupo: 1

Enlace de la aplicación: <http://proyecto-morning-coffee.appspot.com>

HISTORIAL DE VERSIONES

Fecha	Versión	Detalles	Participantes
26/03/2017	0.0	- Incluye introducción, prototipos de las interfaces de usuario y diagramas UML de componentes y despliegue.	Santiago Borge Ángel Mármol Ángel Calzado Fran Abreu
30/04/2017	1.0	- Incluye introducción, prototipos de las interfaces de usuario y diagramas UML de componentes y despliegue de bajo y alto nivel (provisionales). También posee una pseudo-documentación escrita de una API REST.	Santiago Borge Ángel Mármol Ángel Calzado Fran Abreu
21/05/2017	2.0	- Incluye documento plantilla al completo y corregido, así como una API REST con operaciones CRUD y el mashup funcional.	Santiago Borge Ángel Mármol Ángel Calzado Fran Abreu

Índice

1	Introducción	4
1.1	Aplicaciones integradas	4
1.2	Evolución del proyecto	5
2	Prototipos de interfaz de usuario	5
2.1	Vista A	5
2.2	Vista B	¡Error! Marcador no definido.
2.3	Vista C	¡Error! Marcador no definido.
2.4	Vista D	7
2.5	Vista E.....	7
3	Arquitectura	8
3.1	Diagrama de componentes.....	8
3.2	Diagrama de despliegue	8
3.3	Diagrama de secuencia de alto nivel	9
3.4	Diagrama de clases	11
3.5	Diagramas de secuencia	13
4	Implementación	19
5	Pruebas.....	21
6	Manual de usuario	25
6.1	Mashup	25
6.2	API REST	28
	Referencias	36

1 Introducción

Con este trabajo buscamos poner fin a un problema que muchas personas sufren: Reunir la información que necesitan para organizar su día todas las mañanas (leer los emails, revisar su lista de tareas, su calendario de eventos, etc). En lugar de tener que visitar numerosos sitios proponemos una solución que sirva como nexo de todas las funciones además de enlazar unas con otras para comodidad del usuario. Para ello vamos a crear un mash-up que integre las siguientes aplicaciones: Gmail (para visualizar los emails), Wunderlist (para la lista de tareas) y Google Calendar (para revisar los eventos de los usuarios y su disposición en el calendario). También combinaremos las funcionalidades de estas aplicaciones para facilitar distintas tareas al usuario como crear un evento a partir de un correo, gestionar de forma automática los emails, etc.

1.1 Aplicaciones integradas

Gmail: Aplicación de correos electrónicos de google que nos permitirá recibir correos del usuario y enviarlos. Además, podremos crear tareas y eventos a partir de estos mensajes.

Wunderlist: Aplicación que nos proporcionará la lista de tareas sobre la que el usuario podrá añadir y eliminar eventos de correos o personalizados.

Google Calendar: Aplicación que mostrará el calendario de eventos y tareas del usuario. También permitirá crear directamente un evento o un recordatorio.

Nombre aplicación	URL documentación API
Gmail	https://developers.google.com/gmail/api/v1/reference/?hl=es-419
Wunderlist	https://developer.wunderlist.com/documentation
Google Calendar	https://developers.google.com/google-apps/calendar/v3/reference/

TABLA 1. APLICACIÓN INTEGRADAS

1.2 Evolución del proyecto

El prototipo de interfaz de usuario que pensamos en un principio consistía en colocar toda la información de forma organizada en una sola pantalla. Sin embargo nos dimos cuenta de que iba a ser muy agobiante para el usuario tener tanta información en una sola pantalla, por eso pensamos un prototipo con 2 vistas diferentes (Correo y Eventos) donde teníamos más espacio para colocar la información, además de que estaría bastante mejor organizada.

2 Prototipos de interfaz de usuario

2.1 Vista A

Aquí el usuario inicia sesión con sus cuentas de Gmail, Wunderlist y Google Calendar



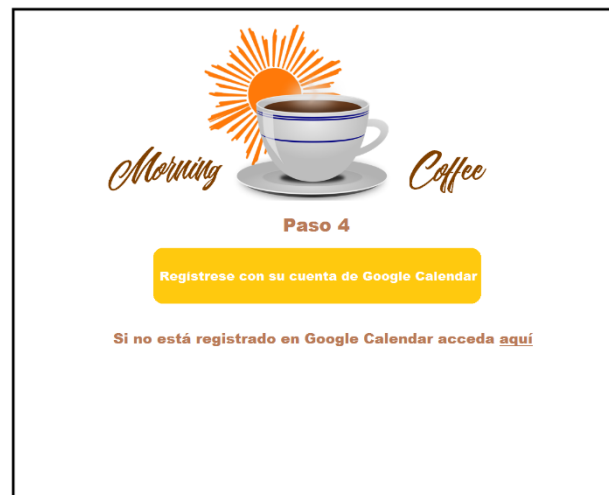


FIGURA 1. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA A

2.2 Vista B

Aquí el usuario podrá ver su lista de correos, responder a los que quiera y crear recordatorios y tareas a partir de dicha lista.



(Morning Coffee logo)		N	G	E	
N G E		De:			
		Asunto:			
		De:		MENSAJE	
		Asunto:			
				Responder Recordar Tarea	

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

2.3 Vista C

Aquí el usuario tendrá su lista de tareas junto con el calendario correspondiente y podrá administrar los eventos que ha creado a partir de la aplicación e incluso crear sus propios eventos directamente.

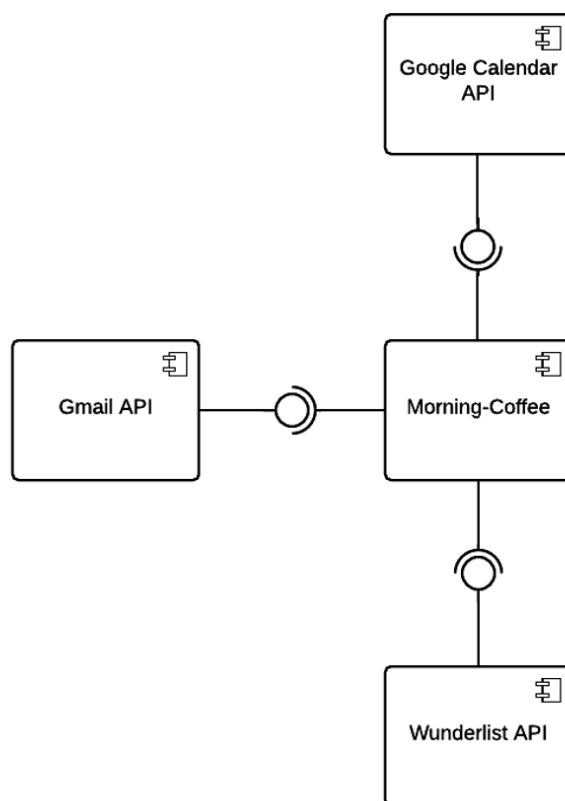
(Morning Coffee logo)		N	G	E				
		CALENDARIO						
<u>Tareas</u> _____ _____ _____ _____ _____ <div>Nueva Tarea</div>								
		<div>Nuevo Evento</div>						

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

3 Arquitectura

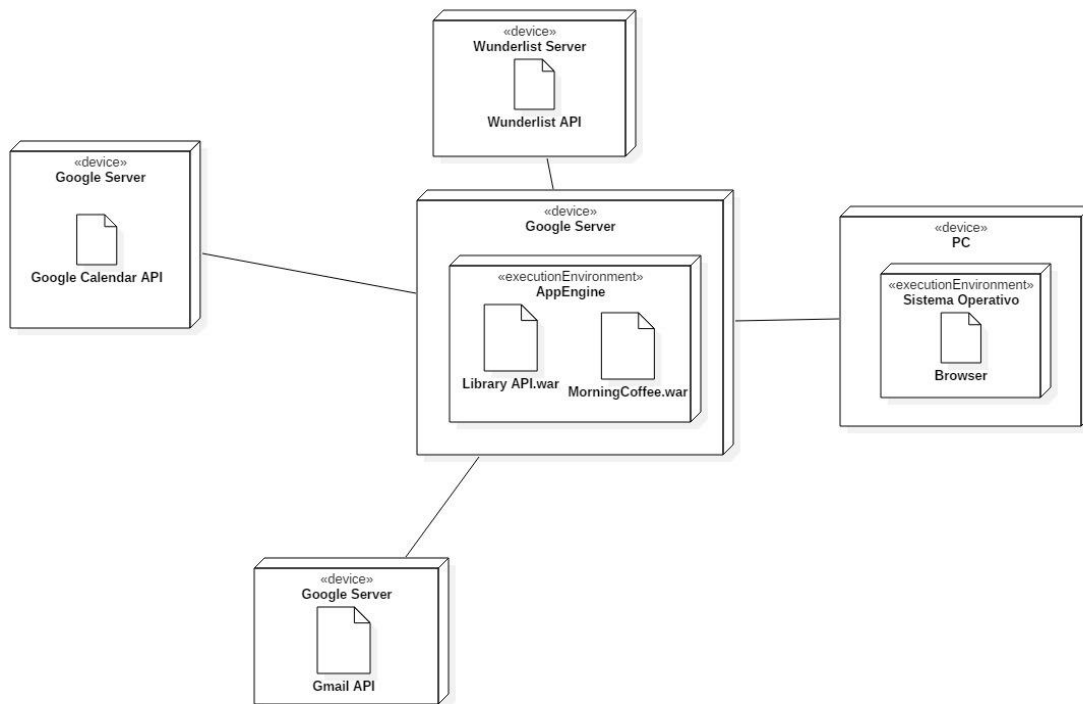
3.1 Diagrama de componentes

Nuestra aplicación realiza su funcionalidad usando los datos proporcionados por las API's de Google Calendar, Gmail y Wunderlist; mediante sus respectivas interfaces.



3.2 Diagrama de despliegue

Aquí podemos ver como nuestra aplicación está corriendo sobre el entorno de ejecución App Engine, el cual corre sobre el dispositivo que es el servidor de Google. Además nuestra aplicación se comunica con un dispositivo que puede ser un PC, Smartphone, Tablet, etc.



3.3 Diagrama de secuencia de alto nivel

El usuario entra en la página por primera vez y se carga su lista de correos, de tareas y su calendario de eventos.

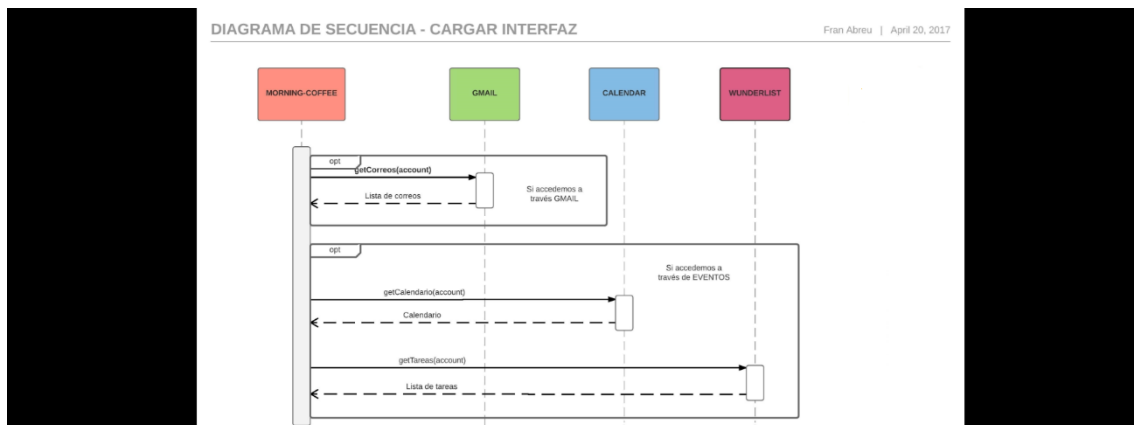
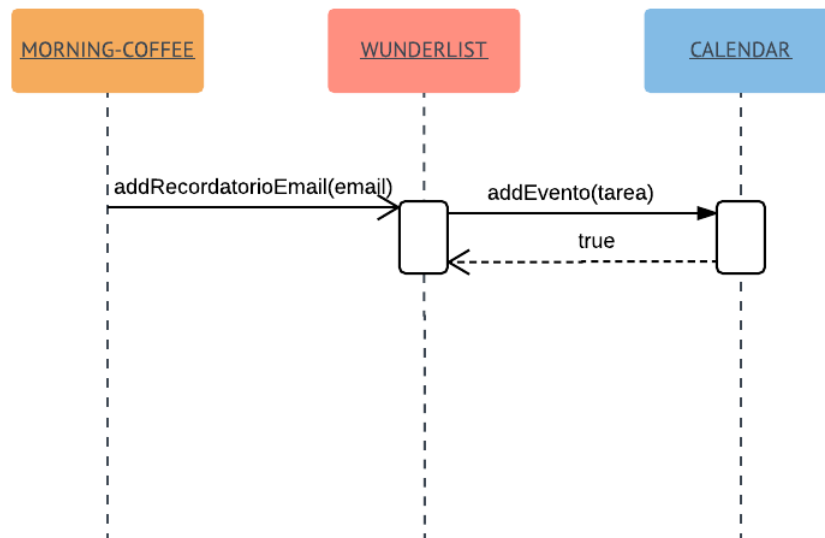
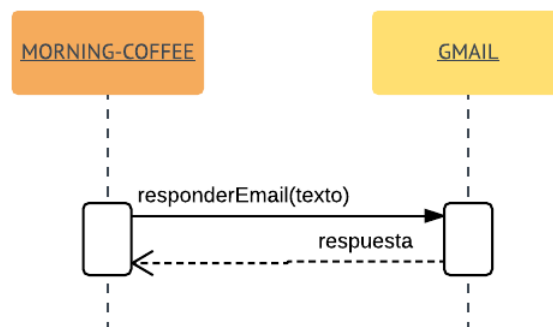


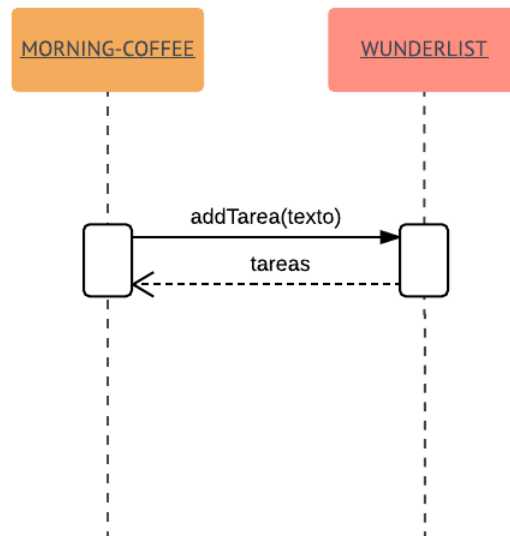
DIAGRAMA DE SECUENCIA DE RECORDATORIO



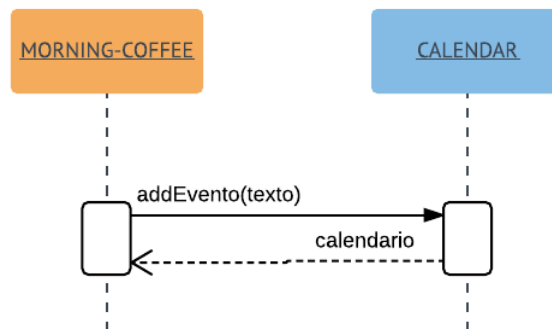
El usuario crea un recordatorio a partir de un correo y se registra en el calendario.



El usuario responde a un correo electrónico.

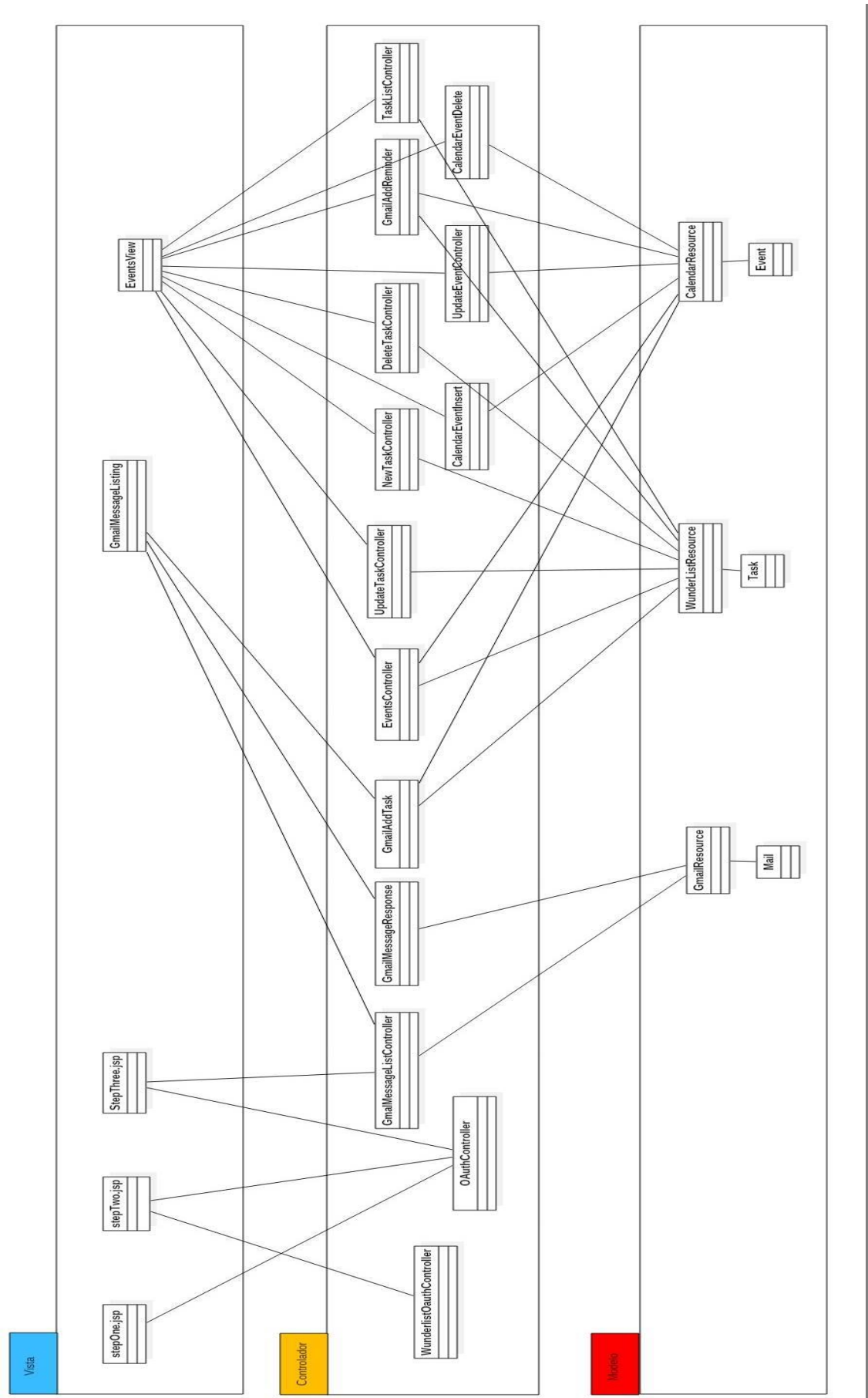


El usuario crea una tarea personalizada.



El usuario crea un evento personalizada en el calendario.

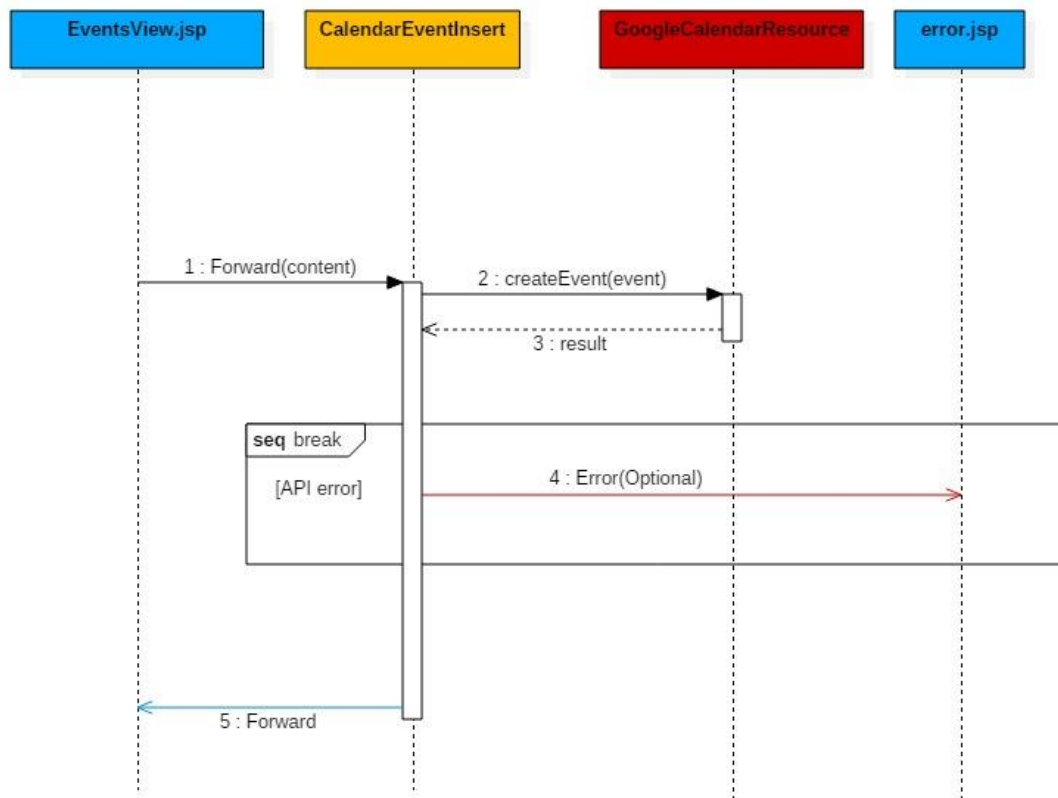
3.4 Diagrama de clases



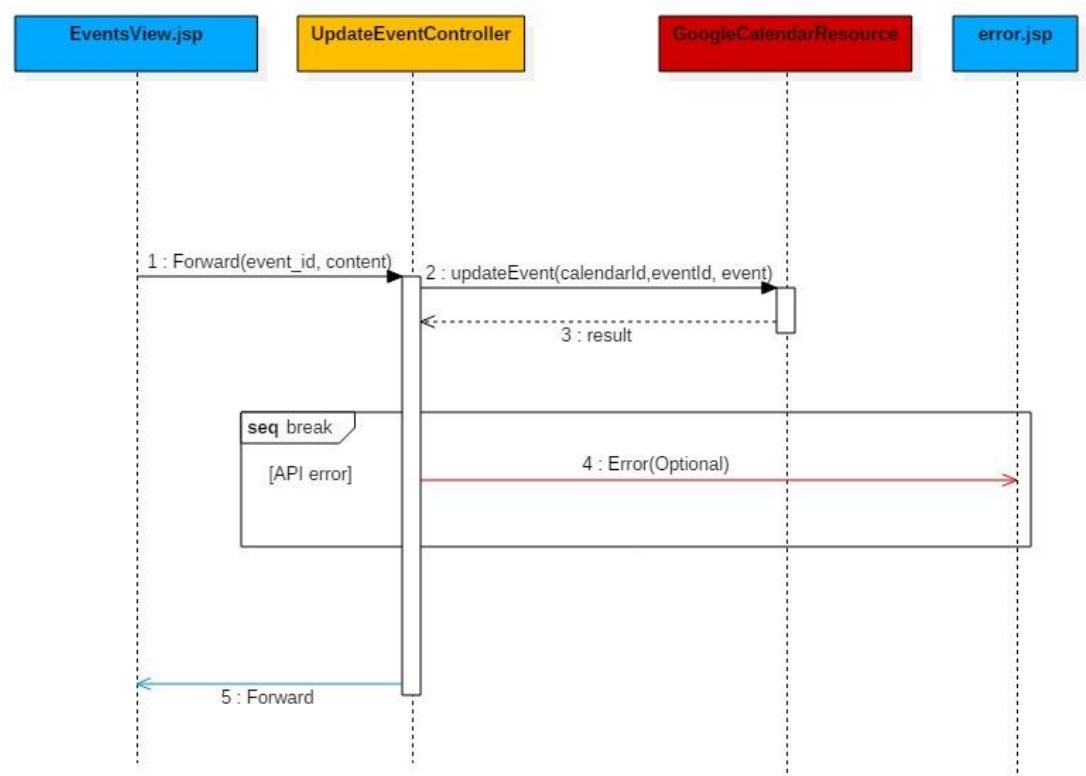
3.4 Diagramas de secuencia

Diagramas UML de secuencia ilustrando la comunicación entre vistas, controladores y clases del modelo

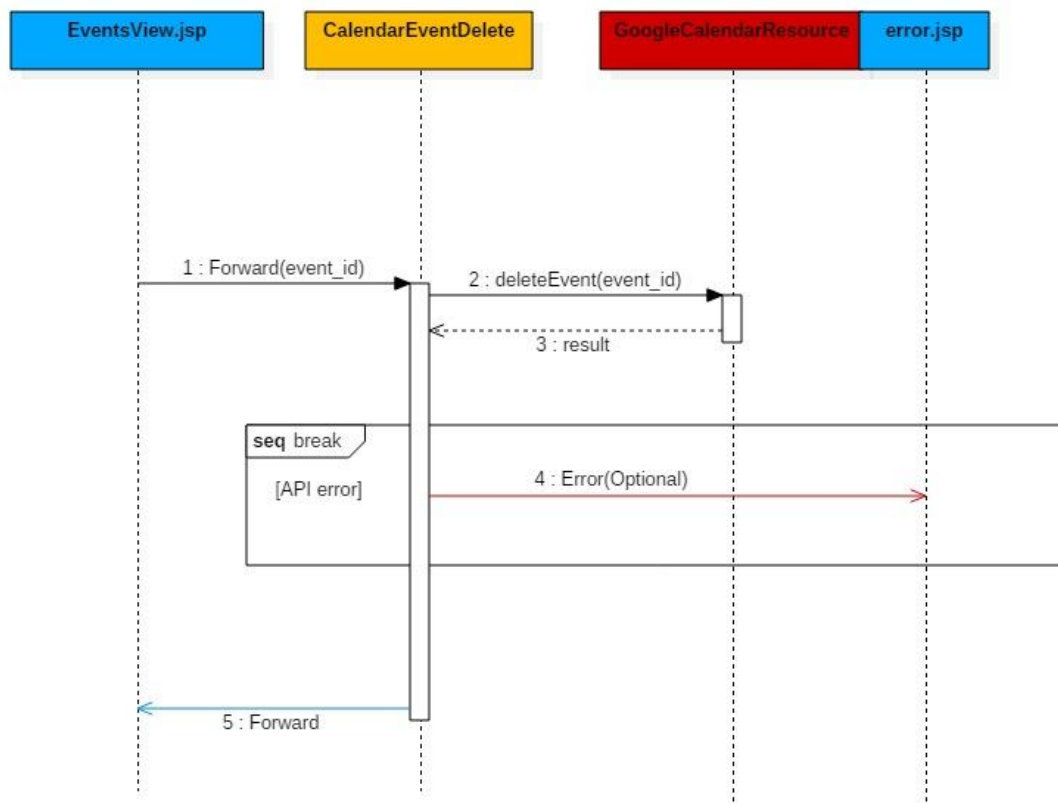
Añadir evento:



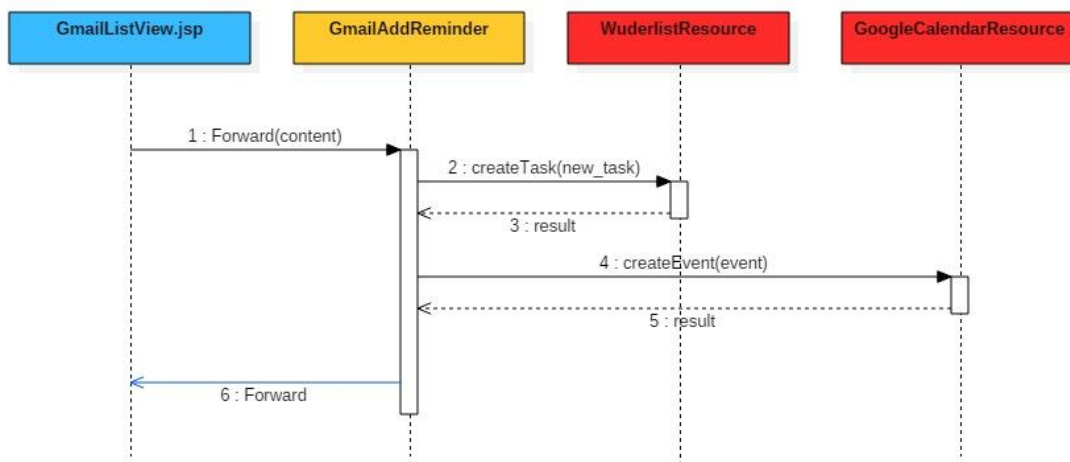
Editar Evento:



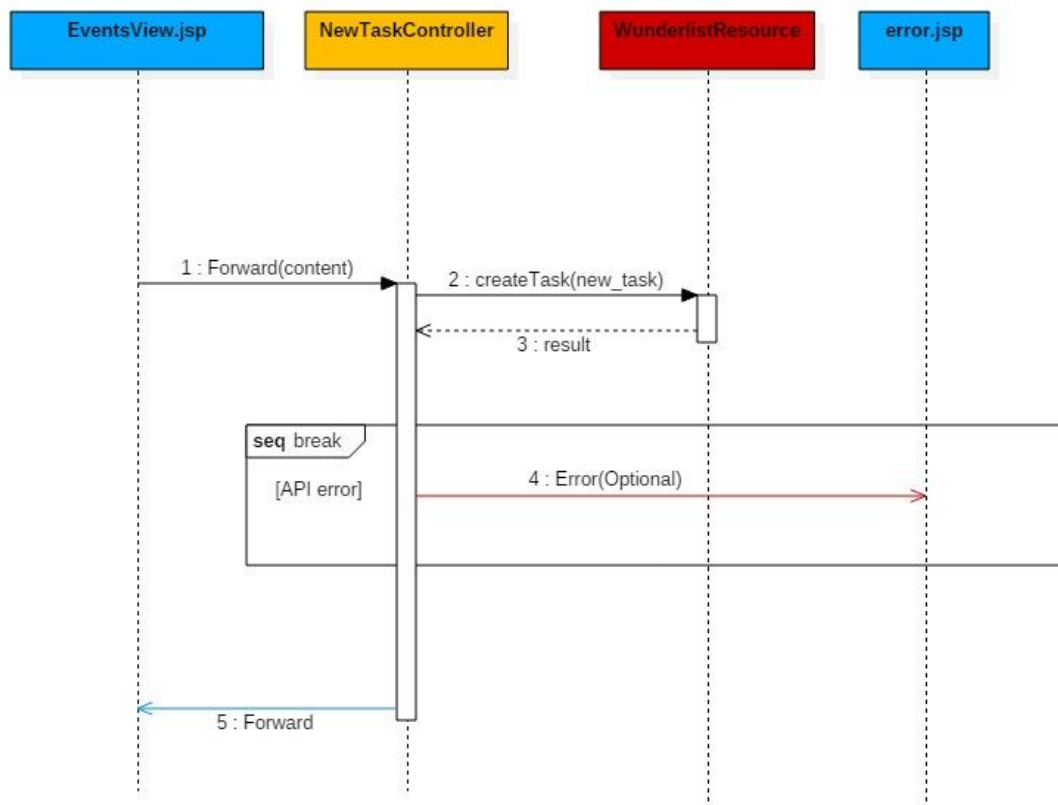
Eliminar Evento:



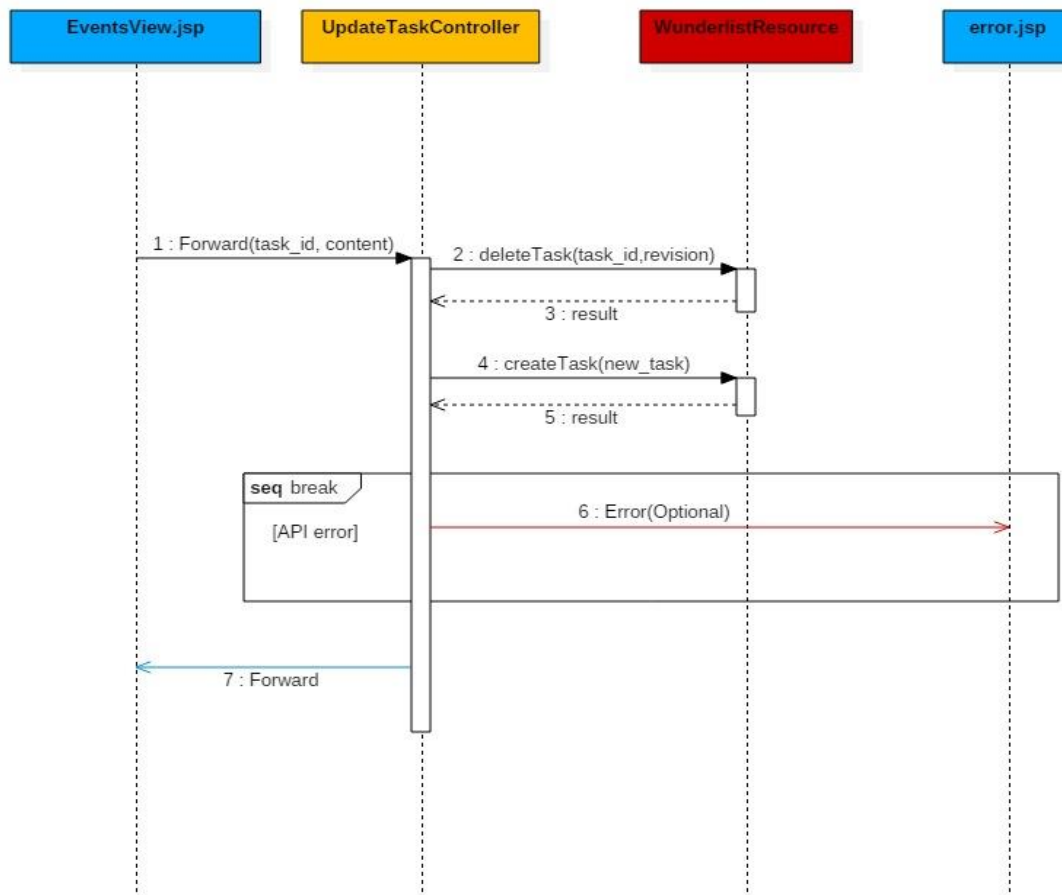
Añadir recordatorio desde un correo:



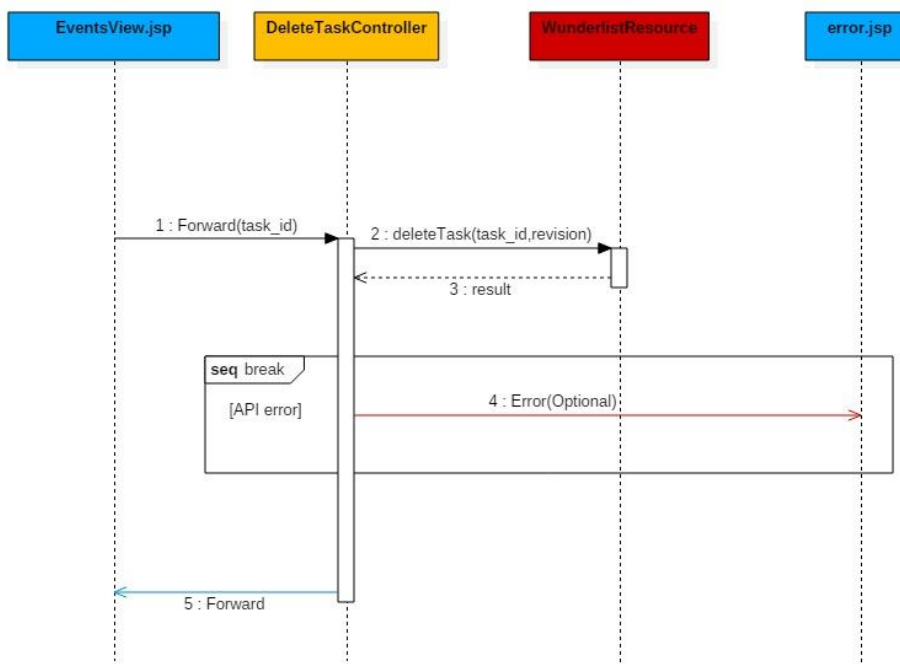
Añadir tarea:



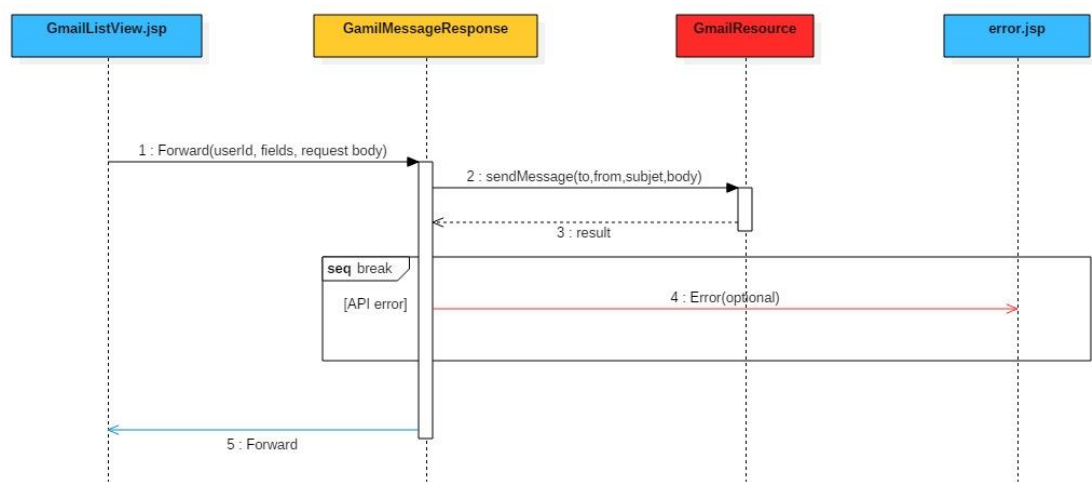
Editar tarea:



Eliminar tarea:



Responder correo:



4 Implementación

Nuestro trabajo posee unas características que consideramos importantes de tratar.

En primer lugar, nos encontramos con la dificultad de que todas las APIs usan OAuth2, y dos de ellas operan en la misma página. Por ello, a lo largo del desarrollo de la aplicación hemos tenido que tener en cuenta la coexistencia e interacción de los posibles problemas, así como de las posibles interacciones. Para evitar problemas de acceso, hemos tomado la decisión de tener que hacer pasar al usuario por un proceso de registro (los 3 pasos) que se repetirán en caso de que uno de los token de acceso caduque.

Por otro lado, hemos tenido que implementar la API de Wunderlist, de la que descubrimos demasiado tarde que no seguía los estándares de OAuth2 que nos proporcionaba la librería usada en la Facultad. Por tanto, hemos tenido que crear dos controladores personalizados para la autenticación, y redirigir al usuario a páginas distintas.

Primero redirigimos al usuario a la URL de autorización de Wunderlist (a la hora de desplegar la aplicación habría que cambiarla):

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {

    response.sendRedirect("https://www.wunderlist.com/oauth/authori
ze?client_id=4017379b2bb52043c811&redirect_uri=http://localhost:8090/W
underlist&state=RANDOM");

}
```

La URL de redirección es la que nos permite extraer el token de acceso y redirigirnos al paso 3 (inicio de sesión de Google Calendar).

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
    String code = request.getParameter("code");
    ClientResource cr = new
    ClientResource("https://www.wunderlist.com/oauth/access_token");

    WunderlistRequest json = new WunderlistRequest();
    json.setClient_id("4017379b2bb52043c811");
    json.setClient_secret("f943802799f6516084619132c993e5c4c65dd934
216bd52b2766a1b34f0a");
    json.setCode(code);
    WunderlistToken respuesta=cr.post(json,WunderlistToken.class);

    request.getSession().setAttribute("Wunderlist-
token",respuesta.getAccess_token());

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

Para acabar con los aspectos importantes de la implementación de Wunderlist, hay que decir que la actualización de tareas se hacía mediante el verbo PATCH, que la librería no nos dejaba usar. Por tanto, hemos “simulado” una actualización, en la que se elimina la tarea a actualizar y se crea una con los datos nuevos:

```
//Simulamos actualización: Primero se borra la tarea

boolean success = wlResource.deleteTask(task_id, revision);

//A continuación, creamos una nueva con el título a "editar"

Task task_To_Update = new Task();
task_To_Update.setTitle(task_title);
task_To_Update.setListId(inbox_id);

//Creamos la tarea

Task updated_task = wlResource.createTask(task_To_Update);
```

También hemos decorado la lista de correos para que al principio sólo muestre el asunto, el remitente y los botones de acción. Si el usuario hace click en “Mostrar cuerpo”, se aplica una función de JavaScript que muestra el div del cuerpo de ese correo (cuyo id se genera dinámicamente con JSTL):

```
function cuerpo(div_id){
    var div_inicio = "div_mensaje_";
    var res = div_inicio.concat(div_id);
    var x = document.getElementById(res);
    if (x.style.display === 'none') {
        x.style.display = 'block';
        x.style.wordBreak = 'break-all';
        x.style.width = '100%';
    } else {
        x.style.display = 'none';
    }
}

<div id= "div_mensaje_${message.id}" style="display:none;" >
    <c:forEach items="${message.payload.parts}" var="parte">
        <c:if test="${parte.mimeType == 'text/plain'}">

            <p><c:out value="${parte.body.data}"></c:out></p>
        </c:if>
    </c:forEach>
    <p><c:out value="${message.payload.body.data}"></c:out></p>

</div>
```

5 Pruebas

Siguiendo un orden ascendente en complejidad, hemos hecho primero las pruebas de los métodos del Resource de cada API en JUnit, para así saber si los métodos básicos (GET, POST, DELETE, PUT) funcionaban. Posteriormente, hemos comprobado las funcionalidades extra de la aplicación (como crear un recordatorio a partir de un correo) manualmente. Antes de mostrar los resultados, debemos comentar que aunque estamos contando las pruebas de JUnit como automáticas, en realidad antes de hacerlas tenemos que facilitar los token de acceso manualmente (porque todas nuestras llamadas a las APIs los necesitan).

Resumen	
Número total de pruebas realizadas	13
Número de pruebas automatizadas	11 (85%)

ID	Pruebas 1, 2 y 3
Descripción	Pruebas para la detección de errores al implementar la obtención de correos Gmail, la obtención de un correo y la respuesta a un correo usando servicios RESTful.
Entrada	Se usan los métodos getListMessages(null), getMessageOfList (String MessageId) y sendMessage (String to, String from, String Subject, String body) de la clase GmailResource, que hace una llamada a la API mediante la URI https://www.googleapis.com/gmail/v1/users/me/messages https://www.googleapis.com/gmail/v1/users/me/messages/send
Salida esperada	Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. La lista de correos no debe de ser null, así como el correo seleccionado. El envío de correo debe devolver true.
Resultado	ÉXITO, ÉXITO, ÉXITO
Automatizada	Sí

ID	Pruebas 4, 5, 6, 7
Descripción	Prueba para la detección de errores al implementar la obtención de la lista de eventos del calendario primario de Google Calendar, la creación de un evento, su actualización y su eliminación usando servicios RESTful.
Entrada	Se usa el método <code>getEvents(String calendarId)</code> , <code>deleteEvent(String id)</code> , <code>createEvent(Item event)</code> y <code>updateEvent(String calendarId, String eventId, Item event)</code> de la clase <code>CalendarResource</code> , que hace una llamada a la API mediante la URI https://www.googleapis.com/calendar/v3
Salida esperada	Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. La lista de eventos no debe de ser null, ni el evento devuelto en la creación. La actualización y eliminación devuelven un boolean que debe de ser true.
Resultado	ÉXITO, ÉXITO, ÉXITO, ÉXITO
Automatizada	Sí

ID	Pruebas 8, 9, 10, 11
Descripción	Prueba para la detección de errores al implementar la obtención de la lista de listas de tareas de Wunderlist, la obtención lista de tareas del inbox, la creación de una tarea, y su eliminación usando servicios RESTful.
Entrada	Se usan los métodos <code>getLists()</code> , <code>getTasks(Long list_id)</code> , <code>createTask(Task new_task)</code> y <code>deleteTask(Long task_id, Integer revision)</code> de la clase <code>WunderlistResource</code> , que hace una llamada a la API mediante la URI https://a.wunderlist.com/api/v1/tasks http://a.wunderlist.com/api/v1/lists
Salida esperada	Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. Ni la lista de listas ni la lista de tareas del inbox deben de ser null, la tarea creada tampoco. La eliminación de la tarea debe de devolver un boolean true.
Resultado	ÉXITO, ÉXITO, ÉXITO, ÉXITO
Automatizada	Sí

GmailTest.java

```

16 public class GmailTest {
17     private static String access_token;
18
19     @BeforeClass
20     public static void setUp() throws Exception {
21         try{
22
23             /*
24              Scanner ss=new Scanner(System.in);
25              System.out.print("Introduzca token de Wunderlist: ");
26

```

JUnit
Finished after 3,066 seconds
Runs: 3/3 Errors: 0 Failures: 0

```

project_name.GmailTest (Runner: JUnit 4) (3,026 s)
  testGetListMessages (1,533 s)
  testSendMessage (0,712 s)
  testGetMessageOfList (0,781 s)

```

Failure Trace

CalendarTest.java

```

1 package project_name;
2
3 import static org.junit.Assert.assertNotNull;
4
16
17 public class CalendarTest {
18     private static String access_token;
19
20     @BeforeClass
21     public static void setUp() throws Exception {
22         try{
23
24

```

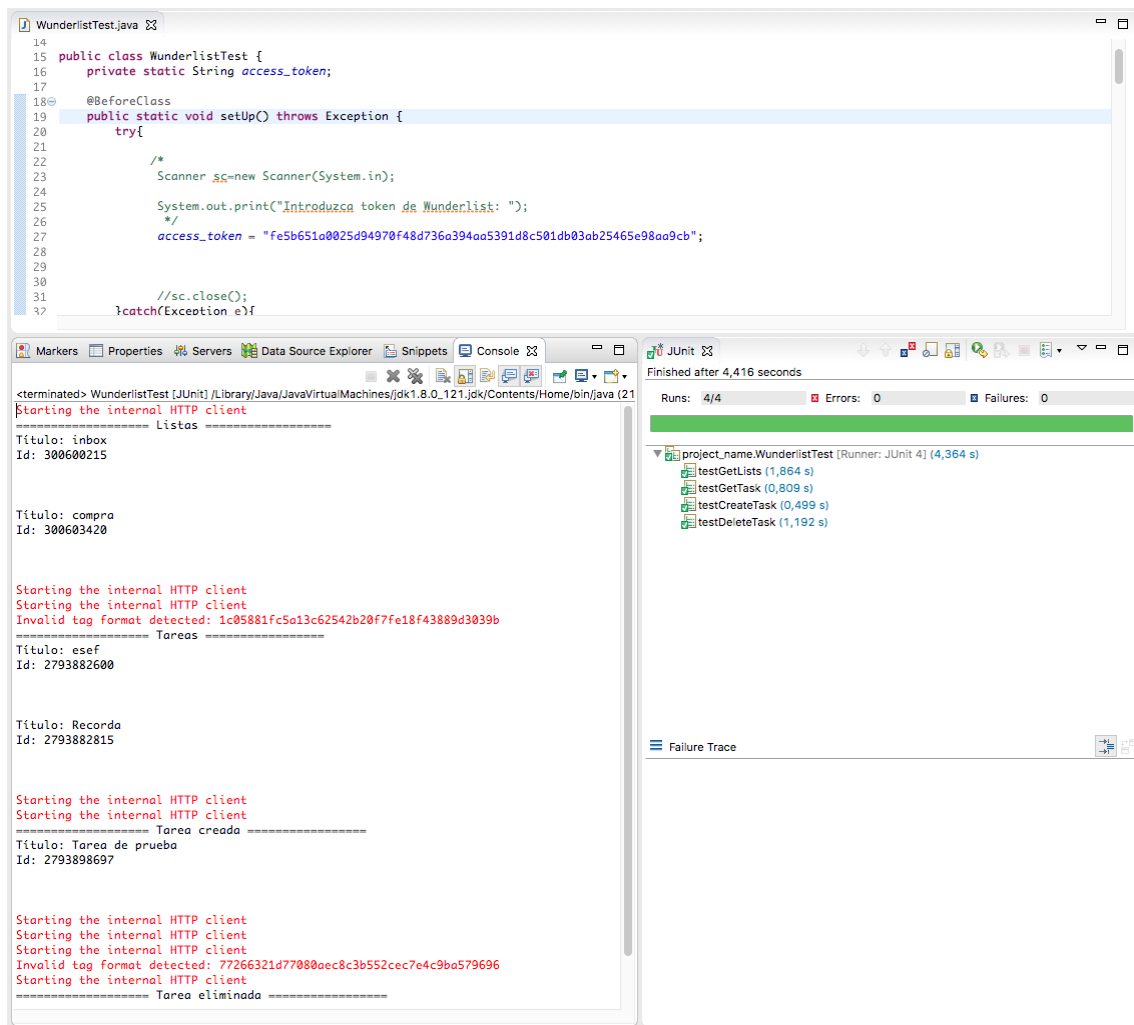
JUnit
Finished after 4,954 seconds
Runs: 4/4 Errors: 0 Failures: 0

```

project_name.CalendarTest (Runner: JUnit 4) (4,883 s)
  testDeleteEvent (2,730 s)
  testGetEvents (0,334 s)
  testUpdateEvent (1,403 s)
  testCreateEvent (0,416 s)

```

Failure Trace



ID	Pruebas 12, 13
Descripción	Prueba para la detección de errores al implementar las acciones de añadir tarea a partir de correo y añadir recordatorio a partir de correo.
Entrada	Se usan los métodos <code>createTask(Task new_task)</code> y <code>CreateEvent(Item event)</code> de la clase <code>WunderlistResource</code> y <code>CalendarResource</code> , que hace una llamada a la API mediante la URI https://a.wunderlist.com/api/v1/tasks https://www.googleapis.com/calendar/v3
Salida esperada	Se espera en ambos casos una redirección a “event_success.jsp”, que nos redirige al controlador <code>EventsController</code> donde se mostrarán todas las tareas y eventos, junto a los creados en la prueba
Resultado	ÉXITO, ÉXITO
Automatizada	No

6 Manual de usuario

6.1 Mashup

En la página de inicio, tendremos dos opciones: “Ir al proceso de registro”, que nos lleva a los 3 pasos para autenticarnos en las páginas, y “Comenzar la mañana”, que nos llevará directamente a la lista de correos (si no nos hemos identificado, nos traerá de vuelta a ésta página).



En el proceso de registro, nos identificamos.





Paso 2

Inicie sesión con su cuenta de Wunderlist

Si no está registrado en Wunderlist acceda [aquí](#).

localhost:8080/kun/Controller/Wunderlist



Paso 3

Inicie sesión con su cuenta de Google Calendar

Si no está registrado en Google Calendar acceda [aquí](#).

Tras la identificación, la aplicación nos llevará a la lista de correos, donde podremos desplegar el cuerpo del correo que queramos. Como muchos correos usan html, al desplegarlos tenemos que mostrar los datos en bruto, dando lugar a correos con cuerpos formados por símbolos extraños. Si mostrásemos html, daríamos la posibilidad a que el html modificase la página, por lo que le damos la posibilidad al usuario de crear un recordatorio para leerlo más tarde, así como tareas según los asuntos de los

mensajes. Además, podremos responder a los correos.



[Gmail](#) [Eventos](#)

Buzón de entrada [>](#)

Asunto	Enviado por		Acciones
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Nuevo inicio de sesión en Chrome con Windows	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Re:	Bounce <nobody@gmail.com>	Mostrar cuerpo	
Nuevo inicio de sesión en Chrome con Linux	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Re:	Bounce <nobody@gmail.com>	Mostrar cuerpo	



[Gmail](#) [Eventos](#)

Buzón de entrada [>](#)

Asunto	Enviado por		Acciones
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
<p>Morning Coffee se ha conectado a tu cuenta de Google. Hola, Morning: Morning Coffee ya tiene acceso a tu cuenta de Google proyectomorningcoffee@gmail.com. Morning Coffee puede hacer lo siguiente: - Consulta y modifica tu correo pero no lo elimines. Si lo debes permitir este tipo de acceso a las aplicaciones de confianza. Puedes revisar o quitar las aplicaciones conectadas a tu cuenta cuando quieras en Mi Cuenta https://myaccount.google.com/permissions. Más información https://support.google.com/accounts/answer/3466521 sobre lo que implica conectar una aplicación a tu cuenta. El equipo de Cuentas de Google Esta dirección de correo electrónico no admite respuestas. Para obtener más información, visita el Centro de ayuda de Cuentas de Google https://support.google.com/accounts/answer/3466521. Te hemos enviado este correo electrónico de anuncio de servicio obligatorio para informarte sobre una serie de cambios importantes que afectan a tu cuenta o producto de Google. © 2017 Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA.</p>			
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
<p>Morning Coffee se ha conectado a tu cuenta de Google. Hola, Morning: Morning Coffee ya tiene acceso a tu cuenta de Google proyectomorningcoffee@gmail.com. Morning Coffee puede hacer lo siguiente: - Consulta y modifica tu correo pero no lo elimines. Si lo debes permitir este tipo de acceso a las aplicaciones de confianza. Puedes revisar o quitar las aplicaciones conectadas a tu cuenta cuando quieras en Mi Cuenta https://myaccount.google.com/permissions. Más información https://support.google.com/accounts/answer/3466521 sobre lo que implica conectar una aplicación a tu cuenta. El equipo de Cuentas de Google Esta dirección de correo electrónico no admite respuestas. Para obtener más información, visita el Centro de ayuda de Cuentas de Google https://support.google.com/accounts/answer/3466521. Te hemos enviado este correo electrónico de anuncio de servicio obligatorio para informarte sobre una serie de cambios importantes que afectan a tu cuenta o producto de Google. © 2017 Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA.</p>			
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Morning Coffee se ha conectado a tu cuenta de Google	Google <no-reply@accounts.google.com>	Mostrar cuerpo	
Nuevo inicio de sesión en Chrome con Windows	Google <no-reply@accounts.google.com>	Mostrar cuerpo	




[Gmail](#) [Eventos](#)

Asunto

Cuerpo

Enviar

Finalmente , podremos ver nuestras tareas y eventos del calendario, crearlos, editarlos y borrarlos en la misma página.



GmailEventos

Tareas:

Nueva tarea:

Recorda

Tarea de prueba

Leer correo -- Morning Coffee se ha conectado a tu cuenta de Google

Dentista

Hacer la compra

Eventos:

Nuevo evento

Inicio:

Fin:

Título:	Inicio:	Fin:	Editar	Borrar
Entrega Aiss	2017-05-20	2017-05-21	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
Recordatorio de correo -- Morning Coffee se ha conectado a tu cuenta de Google	2017-05-02	2017-05-26	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
Evento de prueba CSS	2017-05-09	2017-05-10	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
Convocatorias	2017-06-05	2017-06-30	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
Dentista	2017-05-16	2017-05-17	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
Evento prueba updated!			<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
Evento prueba			<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>

6.2 API REST

Nuestra api trata de la organización de libros de una biblioteca (Library Application), donde el usuario puede consultar los datos de los libros ya sean de todos o de algunos específicos. Además podrá añadir y consultar sus propias listas y añadir los que considere.

Propiedades de las clases:

Book:

String id -> Se asigna automáticamente

String title

String autor

Integer añoPublicacion

Integer numPaginas

ListaLibros:

String id

String name

List<Book> libros;

URLs y verbos HTTP:

-GET https://servicio-restfull.appspot.com/api/books

Devuelve todos los libros que se encuentran almacenados.

Formato: JSON

Código de Estado: 200-OK

Ejemplo:

```
[
  {
    "id": "s0",
    "title": "elSeñorDeLosAnillosPartII",
    "autor": "J.R.R Tolkien",
    "añoPublicacion": 1954,
    "numPaginas": 408
  },
  {
    "id": "s1",
    "title": "elNombreDelViento",
    "autor": "Angel Calzado",
    "añoPublicacion": 1988,
    "numPaginas": 800
  }
]
```

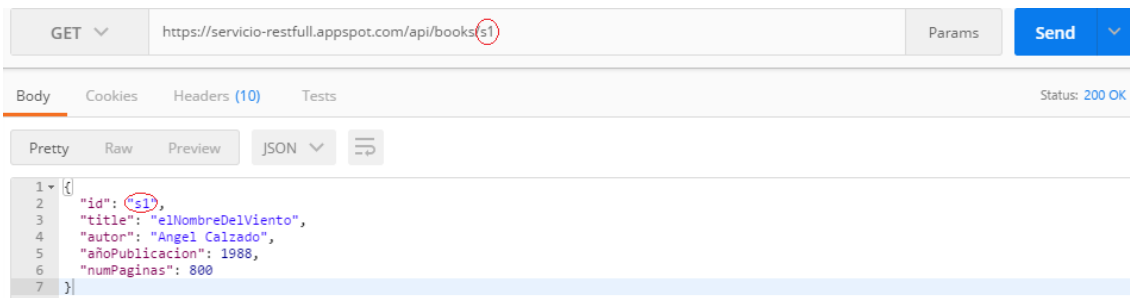
-GET https://servicio-restfull.appspot.com/api/books/{bookId}

Devuelve el libro con el mismo id que el indicado en la url (bookId). Es necesario por tanto indicar el id del libro que queremos que devuelva.

Formato: JSON

Código de Estado: 200-OK ó 404-Si no existe ningún libro con ese Id.

Ejemplo:



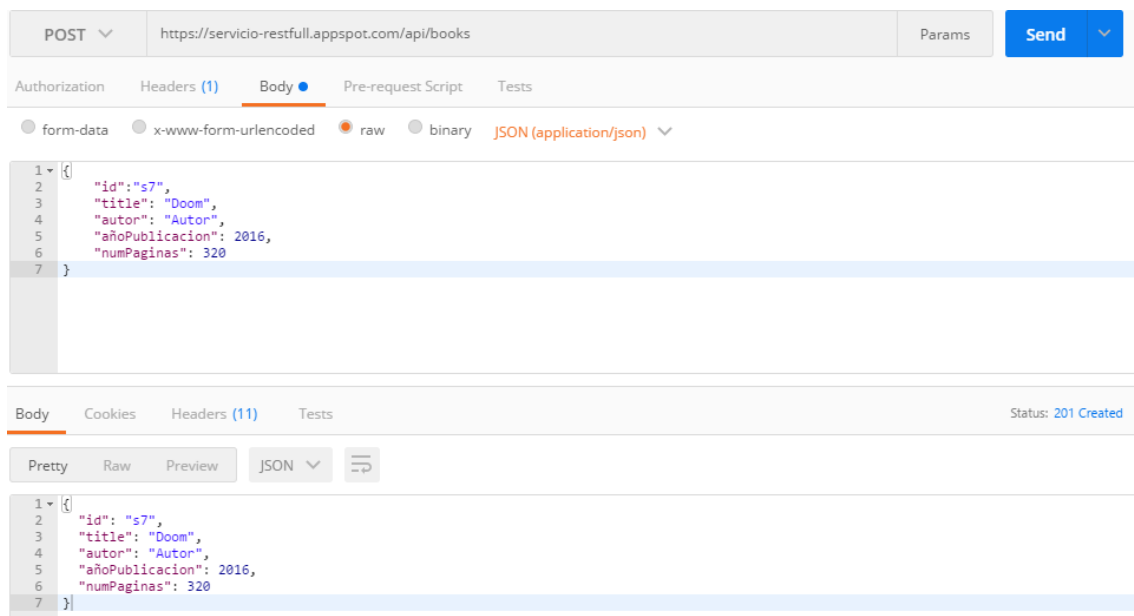
-POST https://servicio-restfull.appspot.com/api/books

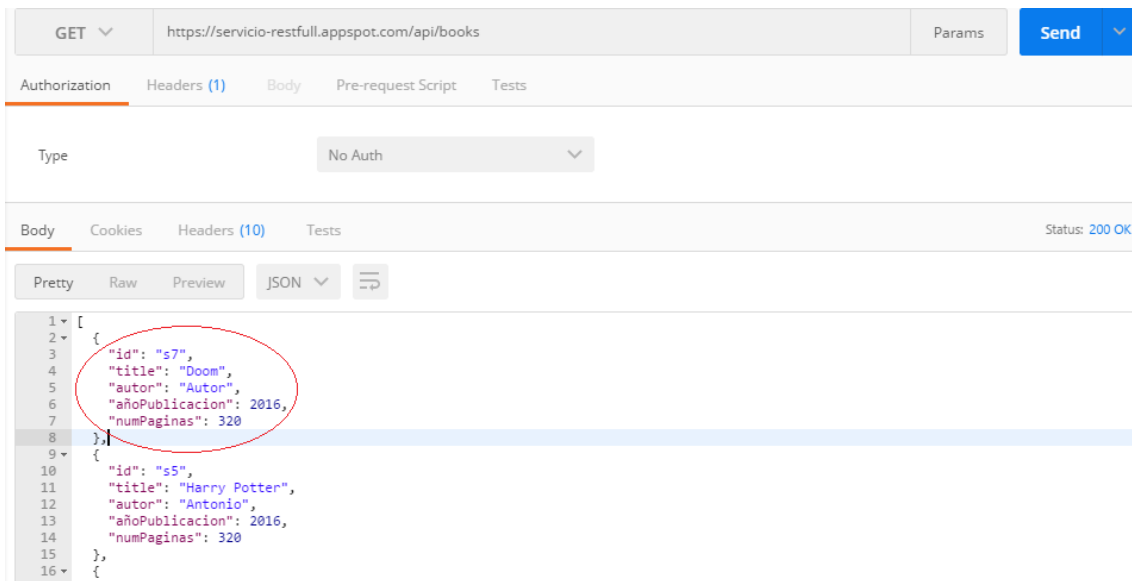
Se indican los atributos de la clase libro y sus respectivos valores en JSON, se envían y se añaden. Todos los campos son obligatorios excepto el id que se añade automáticamente.

Formato: JSON

Código de Estado: 201-Created ó 400-Solicitud Incorrecta.

Ejemplo:





-PUT <https://servicio-restfull.appspot.com/api/books>

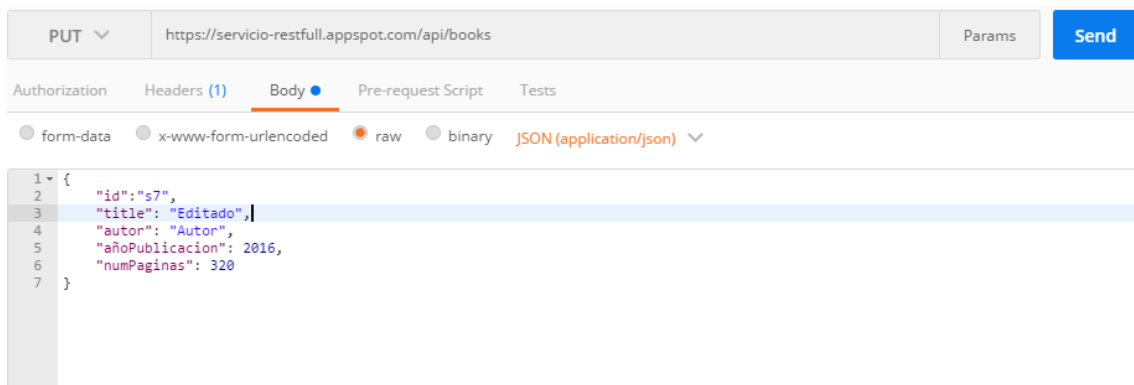
Edita el libro cuyo id es el mismo que el indicado en el JSON que se envía. Sólo puede editarse el título del libro, por lo que los demás campos deben permanecer igual.

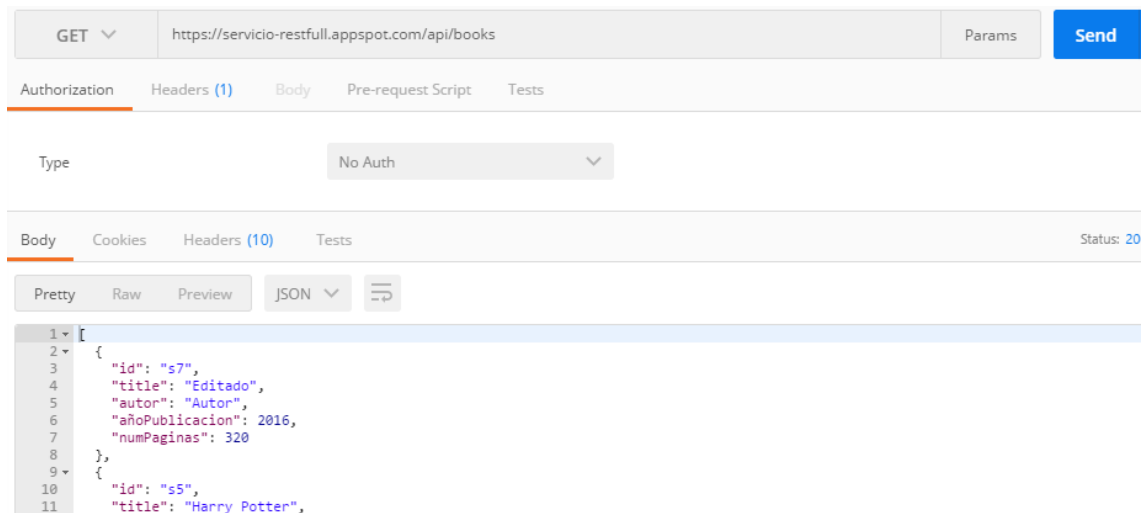
Formato: JSON

Código de Estado: 204-No content(Pero lo construye).

Nota: Si se modifica cualquier otro campo que no sea el título no aparece código, no identifica el libro y no edita.

Ejemplo (Fíjese en el libro con id s7 de la anterior URL):





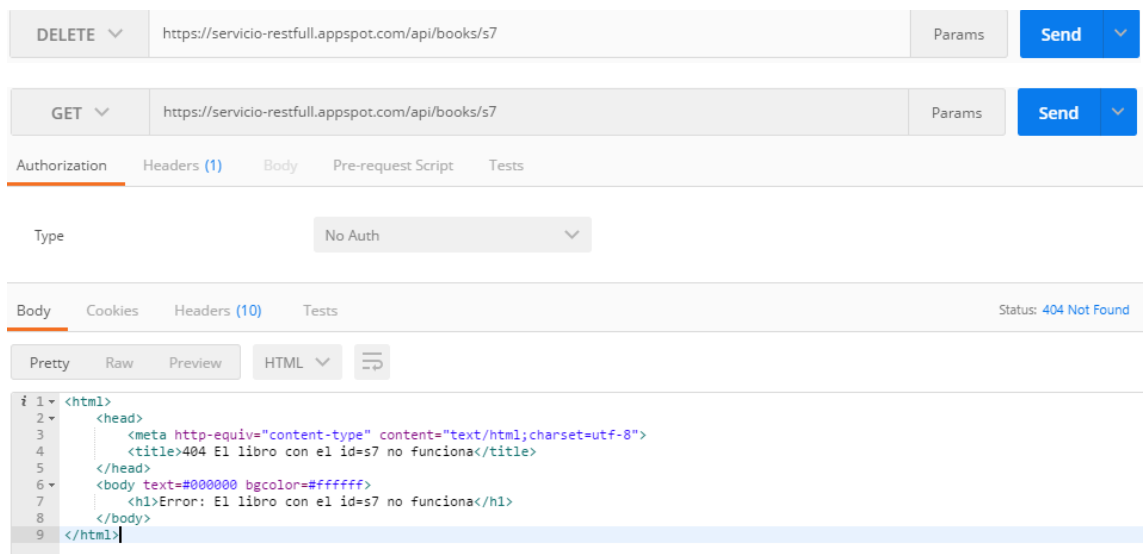
-DELETE https://servicio-restfull.appspot.com/api/books/{bookId}

Elimina el libro cuyo id es el mismo que el indicado en la url(bookId).

Formato: Text

Código de Estado: 204-no Content ó 404-Si el libro no existe.

Ejemplo:



-GET <https://servicio-restfull.appspot.com/api/bookList>

Devuelve todos los objetos de tipo lista que existen.

Formato: JSON

Código de Estado: 200-OK

Ejemplo:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** <https://servicio-restfull.appspot.com/api/bookList>
- Status:** 200 OK
- Response Format:** JSON
- Response Body:**

```
[
  {
    "id": "12",
    "name": "Calzado",
    "libros": [
      {
        "id": "s0",
        "title": "elSeñorDeLosAnillosPartII",
        "autor": "J.R.R Tolkien",
        "añoPublicacion": 1954,
        "numPaginas": 408
      }
    ]
  },
  {
    "id": "13",
    "name": "Santi",
    "libros": [
      {
        "id": "s1",
        "title": "elNombreDelViento",
        "autor": "Angel Calzado",
        "añoPublicacion": 1988,
        "numPaginas": 800
      }
    ]
  }
]
```

-GET <https://servicio-restfull.appspot.com/api/bookList/{listId}>

Devuelve la lista cuyo id es el mismo que el indicado en la url.

Formato: JSON

Código de Estado: 200-OK ó 404-Si la lista no existe.

Ejemplo:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** <https://servicio-restfull.appspot.com/api/bookList/12>
- Status:** 200 OK
- Response Format:** JSON
- Response Body:**

```
{
  "id": "12",
  "name": "Calzado",
  "libros": [
    {
      "id": "s0",
      "title": "elSeñorDeLosAnillosPartII",
      "autor": "J.R.R Tolkien",
      "añoPublicacion": 1954,
      "numPaginas": 408
    }
  ]
}
```

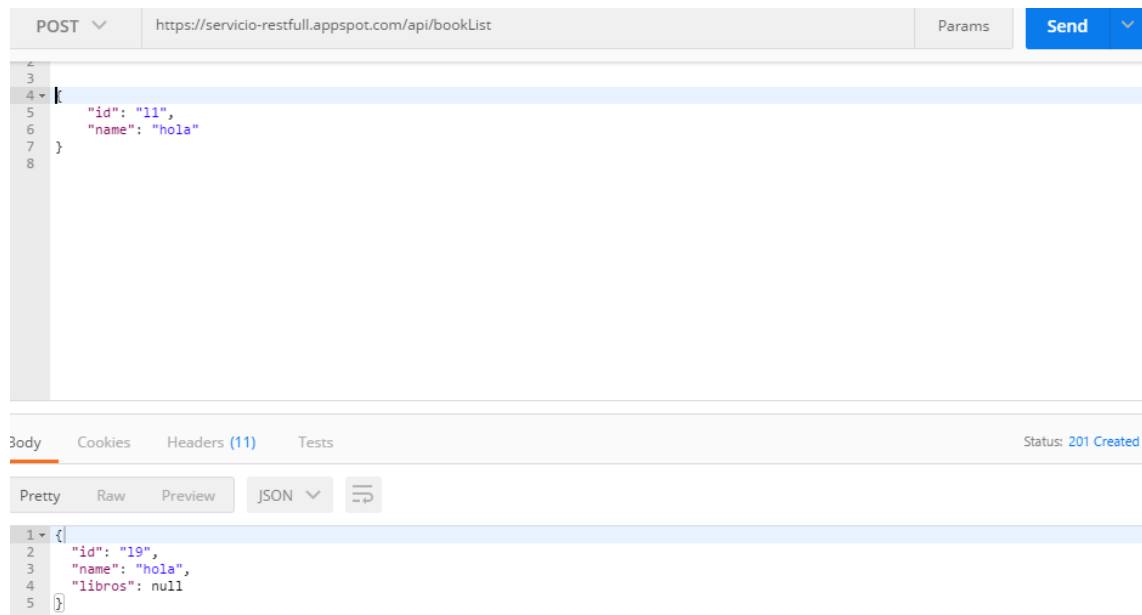
-POST <https://servicio-restfull.appspot.com/api/bookList>

Añade un nuevo objeto lista. El campo de “libros” debe ser nulo para después añadirlos. El id no es obligatorio ya que se genera automáticamente.

Formato: JSON

Código de Estado: 201-Created ó 400-Bad Request (al añadir la propiedad libros)

Ejemplo:



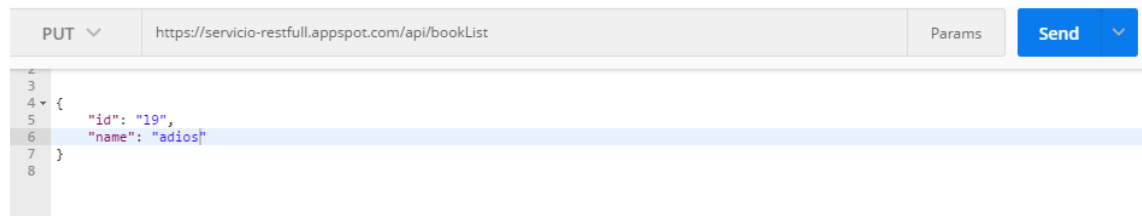
-PUT <https://servicio-restfull.appspot.com/api/bookList>

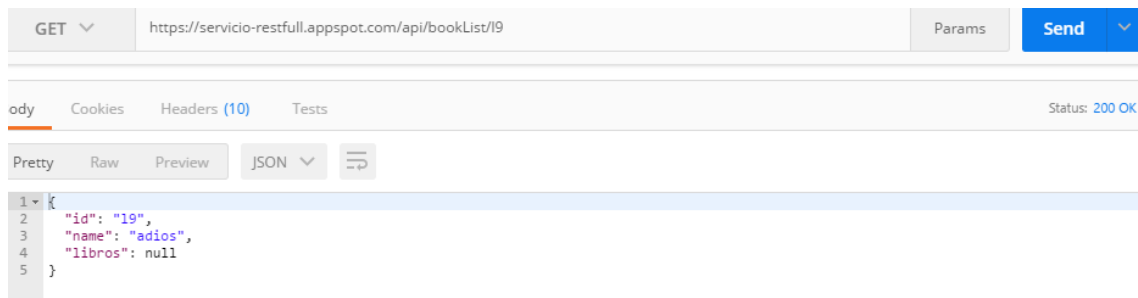
Actualiza el nombre de una lista, la indicada mediante su id en el cuerpo JSON.

Formato: JSON

Código de Estado: 204-NoContent (Construye el nuevo objeto) ó 404-La lista no se encuentra.

Ejemplo (Fíjese en el Anterior Post al crear el objeto con name=hola e id=19)





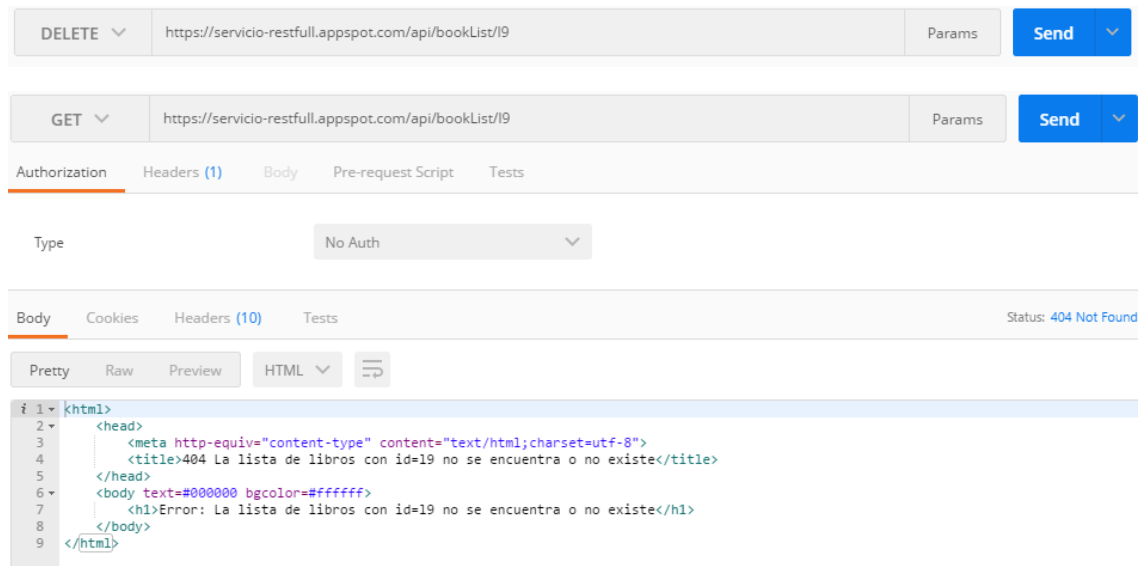
-DELETE https://servicio-restfull.appspot.com/api/bookList/{listId}

Elimina la lista cuyo id es el indicado en la url.

Formato: Text

Código de Estado: 204-NoContent(Lo Borra)

Ejemplo(Fíjese en la lista con id I9 del apartado anterior):



-POST https://servicio-restfull.appspot.com/api/bookList/{listaLibrosId}/{bookId}

Añade un libro específico (con bookId) a una lista determinada (indicada con el id).

Formato: Text

Código de estado: 201-Created, 404-O no se encuentra el libro o no se encuentra la lista y 400-El libro ya está incluido en la lista.

Ejemplo:

```
{
  "id": "18",
  "name": "hola",
  "libros": null
},|
```

The screenshot shows a REST client interface. At the top, a POST request is configured to `https://servicio-restfull.appspot.com/api/bookList/18/s10`. Below the request bar, the response body is displayed in JSON format, showing a list of books. The response status is 201 Created.

```
POST https://servicio-restfull.appspot.com/api/bookList/18/s10
Status: 201 Created

{
  "id": "18",
  "name": "hola",
  "libros": [
    {
      "id": "s10",
      "title": "Libro",
      "autor": "Autor",
      "añoPublicacion": 2017,
      "numPaginas": 300
    }
  ]
}
```

-DELETE `https://servicio-restfull.appspot.com/api/bookList/{listaLibrosId}/{bookId}`

Añade un libro específico (con bookId) a una lista determinada (indicada con el id).

Formato: Text

Código de Estado: 204-No Content(Lo elimina), 404-No se encuentra la lista o el libro.

Ejemplo(Eliminamos el libro que hemos añadido anteriormente):

The screenshot shows a REST client interface. At the top, a DELETE request is configured to `https://servicio-restfull.appspot.com/api/books/s7`. Below the request bar, the response body is displayed in JSON format, showing a list of books. The response status is 201 Created.

```
DELETE https://servicio-restfull.appspot.com/api/books/s7
Status: 201 Created

{
  "id": "18",
  "name": "hola",
  "libros": null
},|
```

Como al principio.

Referencias

[2] J. Webber, S. Parastatidis y I. Robinson. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly Media. 2010.