BlackMail

UN CLIENTE DE GOOGLE MAIL

HECHO POR ANDRÉS BRUGAROLAS MARTÍNEZ

BlackMail

- **BlackMail** es un cliente de Gmail creado únicamente con tecnologías web (HTML, CSS y Javascript) y la API de Google.
- **BlackMail** funciona en todas las plataformas, pero está centrado en escritorio.
- Para tener una aplicación de escritorio independiente del navegador se ha utilizado node-webkit (más recientemente renombrado NW.js).

¿Por qué tecnologías web?

- No es extraño preguntarse por qué se han utilizado tecnologías web en lugar de otras alternativas, ya que éstas cuentan con numerosas desventajas, entre las que destaca el poco rendimiento que ofrecen. A continuación veremos algunos de los motivos por los que se decidió así.

Independencia de la plataforma

- El combo HTML, CSS y JS es completamente independiente de la plataforma, y por tanto goza de una excelente portabilidad, siendo el único factor limitante que la plataforma cuente con un navegador web.

¿Por qué tecnologías web?

Excelentes herramientas

- Se cuenta con excelentes herramientas para diseñar y depurar aplicaciones web, empezando por los propios navegadores. El diseño de aplicaciones es muy rápido y sencillo. La programación es, en general, más productiva que con otros lenguajes.

Diseño de interfaz

- Con HTML y CSS se pueden diseñar interfaces bonitas que se adaptan a cualquier tipo de pantalla, ofreciendo una gran versatilidad y flexibilidad en cuanto al diseño. Esto se puede hacer de forma muy sencilla utilizando las propias herramientas que los navegadores de escritorio nos ofrecen, como el Inspector o el Editor de estilos.

¿Por qué tecnologías web?

Librerías y frameworks

- Al tratarse de una tecnología relativamente madura, existen numerosos frameworks y librerías para ayudarnos en el desarrollo. Algunos ejemplos que hemos utilizado son AngularJS, Bootstrap, o Less.js.

Centrado en escritorio

- Aunque se ha tratado de darle un diseño adaptativo, BlackMail es una aplicación enfocada en el escritorio. Para eliminar la dependencia del navegador y disfrutar de otras ventajas, se ha utilizado NW.js. A continuación veremos algunas de las razones.

Escaso rendimiento en dispositivos móviles

- El principal motivo es que, al ser una aplicación que hace un uso intensivo de JavaScript, el rendimiento en dispositivos móviles es bastante pobre. Y aunque en terminales de gama alta funciona a la perfección, en terminales más modestos es inutilizable.

Centrado en escritorio

Falta de alternativas en escritorio

- Además, para dispositivos móviles existen otras alternativas gratuitas, mientras que para PC no hay ningún cliente que aproveche las características que nos ofrece la API de Google, sino que suelen protocolos más generales como IMAP o POP3.

¿Por qué la API de Google?

- ¿Por qué se ha utilizado la API de Google y no otros protocolos específicos para el correo como POP3 o IMAP? A continuación veremos algunas de las razones que me han llevado a tomar esa decisión en el diseño.

Demasiadas alternativas con POP3/IMAP

- Actualmente existen muchísimas alternativas de escritorio que se decantan por los protocolos de correo tradicionales. Sin embargo, no hay ni una sola alternativa que utilize la API de Google.

¿Por qué la API de Google?

Sencillez

- Se cuenta con excelentes herramientas para diseñar y depurar aplicaciones web, empezando por los propios navegadores. El diseño de aplicaciones es muy rápido y sencillo. La programación es, en general, más productiva que con otros lenguajes.

Integración con otros servicios

- Al utilizar la API de Google, es muy sencillo integrar nuestro cliente de Gmail con otros servicios de Google como Google Drive, Google+, Google Calendar o Hangouts para generar valor añadido.

Pero también tiene desventajas…

- La API de Google también tiene sus desventajas. Las principales las comentaremos a continuación.

Desarrollo temprano e inestabilidad

- La API de Google todavía se encuentra en alpha, una fase de desarrollo temprana. Por tanto es propensa a tener diversos errores siendo, en general, inestable. Además una parte de la funcionalidad todavía está por implementar.

Cuotas de uso

- La API de Google tiene cuotas de uso, a no ser que el desarrollador las aumente pagando, tanto en términos generales como en número de peticiones por segundo. Es por tanto necesario tener esto en cuenta cuando desarrollamos nuestra aplicación.

Pero también tiene desventajas…

Código ofuscado

- El código está ofuscado y por tanto es muy difícil para un desarrollador comprenderlo, extenderlo, corregir sus errores o en general modificarlo a su gusto. Esto puede jugar un papel muy negativo, pues el no comprender la librería que estamos utilizando puede llevarnos a cometer errores de diseño.

Cómo funciona la API de Google

- Creamos la petición con:

var func = gapi.client.**nombreAPI**.users.**nombreRecurso**.**funcion**

- Ejecutamos la petición:

func.execute(**nombreFuncionCallback**)

- La petición se convertirá en una objeto XMLHttpRequest, añadiendo automáticamente todos los metadatos necesarios como la autenticación, y se enviará al servidor de Google.
- El servidor de Google procesará la petición, y nos enviará la respuesta de vuelta. La API tratará la respuesta y la convertirá en un objeto JSON con los datos solicitados.
- La API de Google ejecutará entonces la función callback pasada como parámetro, pasando como parámetro la respuesta del servidor ya tratada y convertida a JSON.

Cómo funciona la API de Google

Un sencillo ejemplo

```
gapi.client.gmail.users.threads.get({
    'userId': this.email,
    'id': id,
    'format': 'full'
}).execute(function (response) {
      console.log(response);
});
```

Cómo funciona la API de Google

Es posible agrupar hasta 100 solicitudes, así como especificar qué campos no queremos obtener con el atributo 'fields' para conseguir un rendimiento mejor.

```
var batchRequest = gapi.client.newBatch();
for (var i in threadIds) {
    batchRequest.add(
        gapi.client.gmail.users.threads.get({
            'userId': this.email,
            'id': threadIds[i],
            'format': 'metadata'
        }), {'id': threadIds[i]}
    );
}
batchRequest.execute(callback, error);
```

Recursos de Gmail

La API de Gmail tiene una serie de recursos de los cuales ofrece una serie de funciones tales como listar todos sus elementos, obtener un elemento concreto, eliminarlo, o modificarlo. A continuación veremos cuáles son estos recursos con los que podemos trabajar.

Hilos (*threads*)

- Es el elemento principal con el que trabajaremos. Un hilo engloba todos los mensajes de un mismo asunto.

Mensajes (messages)

- Su nombre lo indica todo. Un mensaje es un correo electrónico enviado a uno o más destinatarios.

Recursos de Gmail

Borradores (*drafts*)

- Es un mensaje que aún no ha sido enviado, probablemente se encuentre a medio escribir.

Adjunto (*attachment*)

- Es un fichero de hasta 25MB que se envía junto a un mensaje. Para que no se necesiten tantos recursos al visualizar un mensaje con adjuntos, estos cuentan con su propio recurso.

Recursos de Gmail

Etiquetas (*labels*)

- Metainformación de un hilo que determina a las categorías a las que pertenece (enviados, recibidos, social, notificaciones, etcétera).

Historia (*history*)

- Recurso asociado a las últimas actividades que has efectuado (leer un hilo, enviar un mensaje, etcétera). Cada elemento de historia tiene un tiempo de vida de entre unas horas y dos o tres semanas.

Cómo funciona BlackMail

A continuación veremos, de forma muy resumida, cómo funciona nuestra aplicación. Se ha tratado de hacer un diseño por capas.

Plantilla HTML y CSS

- Lo principal es señalar que la interfaz de la aplicación está realizada utilizando un fichero HTML y su hoja de estilos CSS. En realidad, para facilitar el desarrollo de los estilos se ha utilizado LESS.

Cómo funciona BlackMail

Plantilla HTML y CSS

- Para rellenar la plantilla con datos útiles hemos utilizado Angular.js, el cual solicita datos a la capa inferior, que hemos denominado 'sistema', y los actualiza en la interfaz cuando es necesario.

Capa de sistema

- Consta de dos subsistemas: almacenamiento y red. Su objeto es gestionar ambas capas de forma invisible al controlador a la hora de gestionar los diversos recursos.

Cómo funciona BlackMail

Capa de sistema: ejemplo

- Por ejemplo, si queremos leer un hilo concreto, llamaremos a la capa 'sistema' preguntándole por ese hilo. La capa sistema comprobará si se encuentra almacenado, y si lo está lo devolverá desde la capa de almacenamiento. Si no lo está, hará una llamada utilizando la capa de red, y antes de devolver los datos al controlador, los almacenará.

Cómo funciona BlackMail

Capa de sistema: otro ejemplo

- Otro ejemplo: si queremos eliminar o modificar un hilo, la capa sistema hará la petición por medio de la capa de red, y si se recibe una respuesta positiva del servidor, entonces el elemento se eliminará o modificará también en la capa de almacenamiento.

Cómo funciona BlackMail

Capa de red

- Encapsula las llamadas con la API de Google para hacerlas más sencillas de utilizar desde la capa de sistema.

Capa de almacenamiento

- Ofrece estructuras de datos eficientes para almacenar y gestionar en memoria la mayoría de datos de la aplicación, así como métodos para persistir la información necesaria y no tener que sincronizar el cliente con cada ejecución.

Otras librerías utilizadas

Además de las librerías mencionadas (LESS.js, AngularJS, API de Google, reveal.js), BlackMail utiliza otras librerías y frameworks que también merecen ser mencionadas.

- Bootstrap (por Twitter): Framework CSS que ayuda mucho en la elaboración de un diseño adaptativo.
- Prefixfree (por Lea Verou): Librería JavaScript que elimina la necesidad de incluir sentencias CSS específicas para cada navegador.
- Base64 (por Dan Kogai): Librería JavaScript para codificar y descodificar en base64 *web safe*, codificación utilizada en mensajes y adjuntos en la API de Google.
- Perfect Scrollbar (por Noraesae): Sustituye las horrorosas barras de desplazamiento nativas por unas mucho más bonitas y con un aire más moderno, unificando la experiencia en los distintos sistemas operativos y navegadores.

Otras librerías utilizadas

- Lo-Dash (por el equipo de Lo-Dash): Ofrece distintas funciones muy bien optimizadas para manipular grandes cantidades de datos o ayudar a desarrollar algoritmos complejos. Enfocada en el rendimiento, prometen ser más rápidas que las funciones nativas a las que sustituye.
- Lz-String (por Pieroxy): Comprime y descomprime elementos JSON para un almacenamiento óptimo en localStorage, utilizando un algoritmo de compresión sin pérdidas, con una compresión de alrededor del 75% y bastante rápido para su uso en navegadores.