PSet8: Mashup

1. Objetivos

- Introducirte a JavaScript, Ajax, JSON.
- Exponerte a objetos y métodos.
- Lanzarte a las librerías API del mundo real. (Te recomendamos leer https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide)

2. Honestidad Académica

Puede leer el código de conducta en: http://soporte.code-fu.net.ni/codigo-de-conducta-con-el-curso/

3. Evaluación

- Su trabajo en este PSet se evaluará a lo largo de cuatro ejes principalmente.

3.1. Alcance

- ¿En qué medida su código implementa las características requeridas por nuestra especificación?

3.2. Exactitud

- ¿Hasta qué punto su código está libre de errores?

3.3. Diseño

- ¿En qué medida está su código bien escrito (claramente, eficientemente, elegantemente, y/o lógicamente)?

3.4. Estilo

- ¿En qué medida es su código legible (comentado, sangrado, con variables adecuadamente nombradas)?

4. Preparándonos

Tu misión (final) de este problem set es implementar un "mashup" que integre a Google Maps con Google News con una base de datos MySQL conteniendo miles de códigos postales, coordenadas de GPS, y más. Algo así como esta versión del staff

https://mashup.cs50.net/
No solo podés buscar lugares a través de la caja de texto que está en la parte superior, podés hacer click en el mapa y arrastrarlo a cualquier otro lugar. Por otro lado, esparcidos por el mapa hay íconos de periódicos que al hacer click sobre ellos, nos dan enlaces de noticias locales.

Podrás notar que algunos marcadores (y etiquetas) se tapan entre sí o están en las coordenadas incorrectas. La base de datos geográfica GeoNames, la cual estamos usando, es de hecho, imperfecta, por tanto, las coordenadas de algunos lugares están apagadas. Por ejemplo, East Boston no están en Back Bay. Y Readville no está en Boston Harbor. No tenés que preocuparte si ves esos mismos síntomas en tu mashup, asumiendo que el origen del problema son los datos mismos.

De cualquier forma, ¡qué genial! Pero, ¿por dónde empezar?

4.1. Google Maps

De seguro ya te es familiar, pero surfeá con Google Maps a cualquier lugar en `https://www.google.com/maps`. Ingresá `42.374490,-71.117185` en la caja de búsqueda en la parte superior, y deberías ver que te lleva a Harvard. Ingresá `41.3163284, -72.9245318`, y deberías terminar en Yale.

¡Interesante! Parece que Google Maps entiende coordenadas GPS (latitud y longitud). Resulta que Google Maps ofrece una API que te permite incrustar los mapas de Google en tu propias aplicaciones web. Hey, ¡esos son uno de los ingredientes que necesitamos!. Familirizate con con la API Javascript de Google Maps[1] al examinar detenidamente las tres secciones debajo de su Guía de Desarrolladores. Lee todo código de muestra cuidadosamente, haciendo click en **View example** debajo de él, de estar presente, para ver el código en acción.

- Getting Started[2]
- Dibujando en el mapa

[1] `https://developers.google.com/maps/documentation/javascript/`
[2] `https://developers.google.com/maps/documentation/javascript/tutorial`

- Markers[3]
- Info Windows[4]

4.2. Google News

Ahora, necesitamos algunas noticias. Si resulta que tenés una cuenta de Google (Gmail) andá a https://news.google.com/ y hacé click en el botón **Pernalizar** en la parte superior a la derecha. Debajo de ese ícono debería aparecer **Personalizar Google Noticias**, debajo del cual debería aparecer **Avanzado**. Hacé click en este último, y **Agregar una sección local** debería aparecer a la derecha. Ingresá, digamos, **Cambridge, Massachusetts** o **New Haven, Connecticut** en esa caja, luego presiná **Agregar**. ¿Deberías... encontrarte en algunos de estos URLs?

- `https://news.google.com/news/section?cf=all&pz=1&geo=Cambridge,+Massachusetts&ned=us&redirect=true`
- `https://news.google.com/news/section?cf=all&pz=1&geo=New+Haven,+Connecticut&ned=us&redirect=true`

No te preocupés si no tenés una cuenta de Google. Solo dirigite a cualquier URL. Interesante, parece que nuestro input es ahora el valor de un parámetro HTTP, `geo`, aunque hay un montón de otros parámetro también. (Notá que `+` es una forma en que un buscador puede codificar un espacio en una URL. Otra forma es con `%20`.) Una a la vez, borrá cada una de esas otras parejas de valores claves más un ampersand (por ejemplo, primero `cf=all&`. luego `pz=1&`, luego `&ned=us`, luego `&redirect=true`, presionando Enter luego de cada vez que corrés para recargar la página a través de un URL cada vez más corto. Deberías descubrir que Google aún retorna noticias de Cambridge or New Haven incluso cuando la URL es cualquiera de las siguientes.

- https://news.google.com/news/section?geo=Cambridge,+Massachusetts
- https://news.google.com/news/section?geo=New+Haven,+Connecticut

¡Excelente! +1 por prueba y error. Ahora intentá cambiar el valor de `geo` a, digamos, `02138` o `06511` y luego presioná Enter de nuevo. ¿Deberías... encontrarte en la URL que se te muestra a continuación? Los artículos podrían cambiar (dado que Cambridge y New Haven tienen más de un código postal cada uno), pero las noticias aún deberían ser sobre Cambridge o New Haven.

- https://news.google.com/news/section?geo=02138
- https://news.google.com/news/section?geo=06511

[3] `https://developers.google.com/maps/documentation/javascript/markers`
[4] `https://developers.google.com/maps/documentation/javascript/infowindows`

Muy bien. Aunque la página a la que estás viendo, por supuesto, está escrita en HTML. Y todo lo que queremos, si es que la solución del Staff no te dio alguna indicación, es una lista con viñetas de títulos de artículos y enlaces. ¿Cómo hacer eso sin hacer "scraping" (seguramente complicado) del HTML de esta página? Movete hacia la parte inferior de la página y buscá **RSS**. Hacé click en ese enlace, y deberías encontrarte en alguno de los URLs siguientes.

- `https://news.google.com/news?cf=all&hl=en&pz=1&ned=us&geo=02138&output=rss`
- `https://news.google.com/news?cf=all&hl=en&pz=1&ned=us&geo=06511&output=rss`

A como antes, borrá todos los parámetros que no se sean vitales para la misión encomendada, dejando solamente, digamos, `geo` y, ahora, `output`. Deberías encontrarte en una de las siguientes URLs (aún enteramente funcionales).

- `https://news.google.com/news/feeds?geo=02138&output=rss`
- `https://news.google.com/news/feeds?geo=06511&output=rss`

Borrar esos parámetros probablemente no es necesario (quién sabe, su ausencia podría quebrar eventualmente algunas cosas), al escarbar hacia la esencia de una URL se siente mejor como que tiene mejor diseño, así que apeguémonos a lo simple.

Ahora, ¿qué es todo ese etiquetado que tenés en pantalla? Se parece un poco a HTML, pern en realidad estás viendo un "RSS feed", un sabor de XML (a tag-based markup language). Por cierto tiempo, RSS causó el furor pues habilitaba a los sitios web manejar y controlar ("syndicate") artículos en un formato estándar que puedena leer los "lectores RSS". RSS ya no es la última Coca Cola del desierto en estos días, pero es aún una hallazgo estupendo dado que las máquinas pueden leerlo. Dado que se adhiere a un formato estándar, podemos analizarlo (¡muy fácilmente!) con software. Aquí te mostramos cómo un feed RSS se ve generalmente (sin datos reales):

```
<rss version="2.0">
    <channel>
        <title>...</title>
        <description>...</description>
        <link>...</link>
        <item>
            <guid>...</guid>
            <tittle>...</tittle>
            <link>...</link>
            <description>...</description>
            <category>...</category>
            <pubDate>...</pubDate>
        </item>
        ...
    </channel>
</rss>
```

En otras palabras, un feed RSS contiene un elemento root llamado `rss`, el hijo del cual es un elemento llamado `channel`. Dentro de `channel` hay elementos llamados `title`, `description`, y `link`, seguido por uno o más elementos llamados `item`, cada uno de los cuales representa un artículo (o blog post o cosas similares). Cada `item`, a su vez, contiene elementos llamados `guid`, `title`, `link`, `description`, `category`, y `pubDate`. Claro, entre la mayoría de estas etiquetas de inicio y final debería haber datos reales (por ejemplo, el título real de un artículo). Para más detalles, mirá https://cyber.harvard.edu/rss/rss.html.

En última instancia, analizaremos feeds RSS de Google News usando PHP y luego retornaremos los títulos de los artículo y los links a nuestra app web a través de Ajax como JSON. Pero hablaremos más de eso en un rato.

4.3. jQuery

Recordá que jQuery[5] es una librería de Javascript súper popular que "hace cosas como la habilitación de documentos especializados HTML y manipulación, manejo de eventos, animación, y Ajax muchos más simple con una API fácil de usar que funciona en un montón de navegadores." Para ser precisos, esto no es sin una curva de aprendizaje. Leé

[5] http://jquery.com/

alguna funciones de la documentación de jQuery.

- $(document).ready()[6]
- Selecting Elements[7]
- jQuery's Ajax-Related Methods[8]

La documentación de jQuery no es la más amigable para el usuario, sin embargo, apostamos por que encontrarás a Google y Stack Overflow[9] como recursos más útiles.

Recordá que `$` es usulamente (aunque no siempre) un alias para un objeto global que es, de otra forma, llamado `jQuery`.

4.4. typehead.js

Nota echale un vistazo a un demo de la librería typehead.js de Twitter, un "plugin" jQuery que agrega apoyo para que los campos de texto HTML sean autocompletados. Mirá **The Basics** específicamente:

http://twitter.github.io/typeahead.js/examples/

Y ahora mirá la documentación para la versión 0.10.5 de esa misma librería, el cual (sorpresa, sorpresa) no es tan amigable para el usiario a como sería lo ideal. Pero de nuevo, no te preocupés.

https://gist.github.com/dmalan/8abe1025cfe5121614b8

Por cierto, la última versión[10] de la librería de Twitter es de hecho 0.11.1, pero está con bugs. :-(

4.5. Underscore

Finalmente, revisá la documentación de Underscore[11], otra librería JavaScript popular que ofrece funciones que puede que algunos desearan que estuvieran construídas en JavaScript misma. En particular, tomá notas de `template`. Hay que admitir que esta

[6] http://learn.jquery.com/using-jquery-core/document-ready/
[7] http://learn.jquery.com/using-jquery-core/selecting-elements/
[8] http://learn.jquery.com/ajax/jquery-ajax-methods/
[9] https://stackoverflow.com/
[10] Este documento es originalmente del 2015, así que el comentario quizá no sea válido
[11] http://underscorejs.org/

documentación tampoco es muy amigable para el usuario, así que no te preocupés si el uso no es tan obvio por el momento.

- http://underscorejs.org/#template

Así como jQuery usa `$` como su símbolo (porque se mira cool), Underscore utiliza `_` como su símbolo. Por ejemplo, `_.template` significa que Underscore tiene un método (función) llamado `template`.

5. Empezando

Uf, eso fue un montón. Pero pensá de esta forma: hay muchas funcionalidades que no necesitarás implementar por tu cuenta. De hecho, solo implementar el autocompletado podría ser un proyecto por sí mismo. Solo necesitamos imaginarnos cómo conectar todos estos componentes juntos para contruir nuestra propia y asombrosa app web.

Iniciá sesión en CS50 IDE y, abrí una terminal, y ejecutá `update50` para asegurarte de que tu workspace esté actualizado.

Luego, dentro del CS50 IDE, descargá el código de distribución de este problem set desde `http://cdn.cs50.net/2015/fall/psets/8/pset8/pset8.zip`. Descomprimilo en `~/workspace`, de manera que tengás un directorio `pset8` en `~/workspace`. Luego borrá `pset8.zip`. Y luego descargá `http://cdn.cs50.net/2015/fall/psets/8/pset8/pset8.sql` en `~/workspace` también.

¿Te acordás de cómo hacerlo?

A continuación, ejecutá `ls` dentro de `~/workspace/pset8`, y deberías ver tres subdirectorios: `bin`, `includes`, y `public`. Asegurate de que los permisos son a como siguen:

- `700`
  - `bin`
  - `bin/import`
  - `includes`
  - `vendor`
- `711`

    - `public`
    - `public/css`
    - `public/fonts`
    - `public/img`
    - `public/js`
- `600`
    - `includes/*.php`
    - `public/*.php`
- `644`
    - `public/css/*`
    - `public/fonts/*`
    - `public/img/*`
    - `public/index.html`
    - `public/js/*`

¿Recordás cómo?, ¿recordás por qué?
Como siguiente paso, asegura de que Apache no esté ya corriendo (con alguna otra root) al ejecutar lo siguiente.

```
apache50 stop
```

Entonces (re)inicie Apache con el comando a continuación para que utilice `~/workspace/pset8/public` como root

```
apache50 start ~/workspace/pset8/public
```

A continuación, asegura de que MySQL esté corriendo al ejecutar lo siguiente.

```
mysql50 start
```

Muy bien, hora de una prueba. En otra pestaña, visitá `https://ide50-username.cs50.io/`, donde `username` es tu propio nombre de usuario.

Deberías encontrarte en un mapa (no con mucho ocurriendo en él). (Si en vez de eso ves Forbidden, de seguro te saltaste algún paso anterior, mejor intentá todo esos pasos de chmod de nuevo.) Sentite en la libertad de hacer click en el mapa y moverte en él. O intentá buscar tu ciudad natal a través de la caja de texto en la parte superior. De hecho, el mashup aún no hace muchas cosas.
Ahora, dirigute a `https://ide50-username.cs50.io/phpmyadmin`, donde `username` es de nuevo tu nombre de usuario, para ingresar a phpMyAdmin.Iniciá sesión si se te pide. Deberías estar en la página principal de phpMyAdmin.

Dentro del CS50 IDE, abrí `pset8.sql`, el cual descargaste más temprano. Deberías ver un monton de declaraciones de SQL. Seleccionalo todo, seleccioná **Edit >Copy** (o presioná control-c), luego regresá a phpMyAdmin. Hacé click en la pestaña SQL de phpMyAdmin, y pegá todo lo que copiaste la caja de texto de de esa página (la cual está debajo de **Run SQL queriy/queries on server "127.0.0.1"**. Mirá lo que acabás de pegar para tener un sentido de los comandos que estás a punto de ejecutar. Luego deberías ver una bandera verte indicando éxito (esto es, **1 fila afectada**). En la esquina superior izquierda de phpMyAdmin, deberías ver un enlace a una base de datos llamada **pset8**, debajo del cual hay un enlace a una tabla llamada **places**. (De no ser asi, tratá de volver a cargar la pagina.) Si hacés click en **place**, encontrarás que esta tabla está vacia. Pero hemos definido su "esquema" (estructura) por vos. Hacé click en la pestaña **Structure** de phpMyAdmin para ver.

Ahora descarguemos los datos que importaremos a esta tabla. En una pestaña separada, dirigite a http://download.geonames.org/export/zip/, donde verás un montón de archivos ZIP, "data dumps" (en formato `.txt`) de la base de datos GeoNames, la cual "cubre todos los paises y contiene más de ocho millones de topónimos que están disponibles para descargar libres de cargo." Control[o hacé click derecho en `US.zip` y seleccioná **Copy Link Address** (o el equivalente de tu navegador), y luego descargá ese ZIP en `~/workspace` dentro del CS50 IDE, al escribir `wget` en una terminar y luego pegando la dirección que acabás de copiar. Alterativamente, eres bienvenido a descargar datos de otro país, aunque asumiremos a los Estado Unidos por efectos de la discusión. Mirá https://en.wikipedia.org/wiki/Main_Page si no estás seguro de cómo interpretar las dos letras de archivos ZIP de nombre base.

A continuación, descomprimí `US.zip`, el cual debería tener a `US.txt`. (¿Recordás cómo?) Y luego borrá `US.zip`.

Por http://download.geonames.org/export/zip/readme.txt, `US.txt` es como un archivo CSV excepto que lo campos estén delimitados en vez con `\t` (un caracter de pestaña) en vez de una coma. Para ver los contenidos del archivo, se te invita a abrirlo dentro del CS50 IDE, pero teniendo cuidado de no cambiarlo.

6. Walkthrough

¿Damos un paseo? Tené en mente que hemos hecho algunos retoques al código de distribución desde que este walkthrough se grabó:

- Hemos removido `constants.php` y `functions.php` de `includes`.
- Hemos agregado `helpers.php`, el cual implementa una función, `lookup`, la cual es llamada en `articles.php`.
- Hemos agregado `vendor`, el cual contiene la librería PHP de CS50.
- Hemos agregado `config.json` con el cual podés configurar la librería PHP de CS50 para conectarse a una base de datos.

Muy bien, ¡qué comience el paseo!
https://www.youtube.com/watch?v=ASA8fAEerNo
Y ahora, un vistazo más cercano al código de dsitribución.

6.1. import

Navegá a `~/workspace/pset8/bin` y abrí `import`. No hay mucho ahí todavía. Solo un asunto y un `TODO`. Es en este archivo que escribirás un script PHP que itera sobre las líneas en `US.txt`, usando `INSERT` para insertar datos de cada uno en `places`, esa tabla MySQL. Pero hablaremos más de eso en un rato.

6.2. index.html

Navegá a `~/workspace/pset8/public` y abrí `index.html`. Y aquí vamos. Si mirás el `head` de la página, mirarás esas librerías CSS y JavaScript que estaremos usando (con lagunas otras). Incluídos en los comentarios HTML hay URLs para la documentación de cada librería.

A continuación dale un vistazo al `body` de la página, dentro del cual hay un `div` con un único `id` de `map-canvas`. Es dentro de ese `div` que inyectaremos un mapa. Debajo de ese `div`, mientras tanto, está un `form`, dentro del cual hay un `input` del tipo `text` con un `id` único de `q` que usaremos para tomar inputs de los usuarios.

6.3. style.css

Ahora, nagvegá a `~/workspace/pset8/public/css` y abrí `style.css`. Ahí dento hay un montón de CSS que implementa el UI predefinido de mashup. Sentite libre de manosearlo (hacer cambios, guardar el archivo, y recargar la página en Chrome) para ver cómo funciona todo, pero será mejor que deshagás todos esos cambios antes de seguir adelante.

6.4. script.js

Navegá a `~/workspace/pset8/public/js` y abrí `scripts.js`. Ah, el archivo más interesante hasta ahora. Este es el archivo que implementa el UI "front-end" de mashup, descansando en Google Mapsy algunos scripts "back-end" para obtener datos (que pronto exploraremos). Demos una vuelta por este.

En la parte superior del archivo hay algunas variables globales:

- `map`, el cual contendrá una referencia (esto es, un puntero de clases) al mapa que estaremos pronto instanciando;
- `markers`, un arreglo que contendrá referencia a cualquier marcador que agreguemos en la parte superior del mapa; y
- `info`, una referencia a una "info window" (ventana de información) en la cual desplegaremos los enlaces a los artículos.

Debajo de esas variables locales hay una función anónima la cual será llamada automáticamente por jQuery cuando el DOM de mashup esté completamente cargado (esto es, cuando `index.html` y todo lo que tenga, especialmente CSS y JavaScript, hayan sigo cargados en memoria).
En la parte superior de esta función está una definición de `styles`, un arreglo de dos objetos que usaremos para configurar nuestro mapa, por https://developers.google.com/maps/documentation/javascript/styling. Recordá que `[` y `]` denotan un arreglo, mientras `{` y `}` denotan un objeto. Las sangrías que ves (muy bonitas) es solo una convención de estilo, la cual es probablemente ideal para usar en tu código tambien.

Debajo de `styles` está `options`, otra colección de claves y valores que será usada para confirar el mapa más adelante, por https://developers.google.com/maps/documentation/javascript/reference#MapOptions.

Ahora, definimos `canvas`, usando un poco de jQuery para obtener el nodo DOM cuya `id` única es `map-canvas`. Por tanto `$("#map-canvas")` retorna un objeto jQuery (al que se le ha construido un montón de funcionalidad), `$("#map-canvas").get(0)` retorna el verdadero nodo subyacente al que jQuery está conteniendo.
Quizá la línea más poderosa hasta ahora es la siguiente, en la cual le asignamos a `map` un valor. Con

```
new google.maps.Map(canvas, options);
```

le estamos diciendo al navegador que instancee un nuevo mapa, inyectándolo en el nodo DOM especificado por `canvas`, configurado según `options`.

La línea debajoe de esa, le dice al navegador que llame a `configure` (otra función que hemos escrito) tan pronto el mapa sea cargado.

6.4.1. addMarker

Oh, un `TODO`. Dado un `place` (código posta, etc.), esta función necesitará agregar un marcador (ícono) en el mapa.

6.4.2. configure

Esta función, continúa donde la fución anónima sale. Recordá que `configure` es llamado tan pronto el mapa ha sido cargado. Dentro de esta función hemos configurado un numero de "oyentes", especificando lo que debería ocurrir cuando "oigamos" ciertos eventos. Por ejemplo,

```
google.maps.event.addListener(map, "dragend", function() {
    update();
});
```

indica que queremos escuchar un evento `dragend` en el mapa, callamando al a función anónima que nos fue provista cuando lo escuchemos. Esa función anónima, a su

vez, simplemente llama a `update` (otra función que pronto veremos). Según https://developers.google.com/maps/documentation/javascript/reference#Map, `dragend` es "disparado"(transmitido) "cuando el usuario deja de arrastrar el mapa."

De igual forma, escuchamos a `zoom_changed`, el cual es disparado "cuando la propiedad de zoom del mapa cambia" (el usuario usa el zoom).
O por el otro lado, al oír `dragstart`, llamamos a `removeMarkers` para que todos los marcadores desaparezcan temoralmente mientras el usuario arrastra el mapa, evitando de esta forma la aparición de un parpadeo que podría ocurrir mientras los marcadores son removidos y luego re-añadidos luego de que los bordes (esquinas) hayan cambiado.

Debajo de esos oyentes esá nuestra configuración de ese plugin typeahead. Tomá otro vistazo de https://github.com/twitter/typeahead.js/blob/master/doc/jquery_typeahead.md si no estás seguro de lo que hacen aquí `autoselect`, `highlight` y `minLength`. Aun más importante, date cuenta que el valor de `source` (esto es, `search`), es la función a la que el plugin llamará tan pronto como el usuario comience a escribir para que la función pueda responder con un arreglo de resultados de búsqueda basados en el input del usuario. Por ejemplo, si el usuario escribe `foo` en la caja de texto, la función debería retornar un arreglo un arreglo de todos los lugaes en tu base de datos que de alguna forma concuerdan con `foo`. La forma de gacer esas correspondencias se te dejan a vos. Por otro lado, el valor de `templates` es un objeto con dos claves: `empty`, cuyo valor es el HTML que debería ser desplegado cuando `search` regrese vacío (retorna una arreglo de longitud 0), y `suggestion`, cuyo valor es una "plantilla" que será usada para darle formada a cada entrada en el menú dropdown del plugin. Justo ahora, la plantilla es simplemente `<p>TODO</p>`, lo cual significa que cada entrada en ese dropdown literalmente dirá `TODO`. Querrás cambiar ese valor a algo como

```
<p><%- place_name %>, <%- admin_name1 %></p>
```

para que ese plugin inserte dinámicamente esos valores (`place_name` y `admin_name1`) o algunos otros para vos. En contraste con `<%=`, el cual Underscore también permite su operación, el `<%-` asegura que el valor será evacuado, al estilo de `htmlspecialchars`, según http://underscorejs.org/#template.
Luego, notá estas líneas, admito que se ven algo crípticas a primera vista:

```
$("#q").on("typeahead:selected", function(eventObject, suggestion, name) {
     map.setCenter({lat: parseFloat(suggestion.latitude), lng:
     parseFloat(suggestion.longitude)});
     update();
});
```

Estas líneas están diciendo que el elemento HTML cuyo `id` único es `q` dispara un evento llamado `typhead::selected`, a como ocurrirá cuando el usuario seleccione una entrada de menú dropdown del plugin, queremos que jQuery llame una función anónima cuyo segundo argumento, `suggestion`, será un objeto que represente la entrada seleccionada. Dentro de ese objeto debe de haber al menos dos propuedades: `latitude` y `longitude`. Luego llamaremos a `setCenter` para re-centrar el mapa a esas coordenadas, luego de lo cual llamaremos a `update` para actualizar a todos los marcadores.

Debajo de esas líneas, están estas:

```
$("#q").focus(function(eventData) {
     hideInfo(); });
```

Si consultás http://api.jquery.com/focus/, ¿podrás ver el sentido de esas líneas?

Debajo de esas se encuentran:

```
document.addEventListener("contextmenu", function(event) {
     event.returnValue = true;
     event.stopPropagation && event.stopPropagation();
     event.cancelBubble(); }, true);
```

Desafortunadamente, Google Maps deshabilita ctrl- y clicks derechos en los mapas, lo cual interfiere con el uso de la herramienta **Inspect Element** de Chrome (sorprendetemente útil), así que estás líneas re-hablitan esa funcionalidad.

Por último en `configure` hay una lllamada a `update` (la cual pronto veremos) y una llamada a `focus`, esta vez sin argumentos. Mirá http://api.jquery.com/focus/ para ver la razón.

6.4.3. hideInfo

Afortunadamente, una función corta. Esta llama a `close` en nuestra ventana de información global.

6.4.4. removeMarkers

Hm, un `TODO`. A la larga, esta función necesitará remover todos los marcadores del mapa.

6.4.5. search

Esta función es llamada por el plugin typeahead cada vez que el usuario cambio la caja de texto del mashup, al escribir o borrar un caracter. El valor de la caja de texto (esto es, lo que sea que el usuario haya escrito) es pasado a `search` como `query`. Y el plugin también pasa a `search` un segundo argumento, `cb`, una "callback" la cual es una función que `search` debería llamar tan pronto haya terminado de buscar. En otras palabras, el pasar `cb` empodera a `search` para ser "asincrónica", por lo que solo llamará a `cb` en cuanto está listo, sin bloquear ninguna de las otras funcionalidades de mashup. En consecuencia, `search` usa el método `getJSON` de jQuery para contactar a `search.php` asincrónicamente, pasando un parámetro, `geo`, el valor del cual es `query`. Una vez que `seach.php` responda (sin importar cuántos milisegundo o segundos más tarde), la función anónima pasada a `done` será llamada y le será pasada `data`, cuyo valor será cualquier JSON que `search.php` haya emitido. (Aunque si algo resulta mal, se llama a `fail`). Finalmente es llamada `cb`, al cual `search` pasa esa misma `data` para que el plugin pueda iterar sobre los lugares ahí (asumiendo que se hallaron resultados a la búsqueda) para actualizar el dropdown del plugin. Uf.

Notá que estmoa usando la interfaz "Promise" de `getJSON`, según `http://api.jquery.com/jquery.getjson/`. En vez de pasar un función anónima directamente a `getJSON` (para que se le llame si se tiene éxito), estamos "encadenando" juntas llamadas a `getJSON`, `done` (cuyo argumento, una función anónima, será llamado si hay éxito), y `fail` (cuyo argumento, otra función anónima, será llamada en caso de fracaso). Mirá http://api.jquery.com/jquery.ajax/ para algunos detalles adicionales. Y mirá https://davidwalsh.name/write-javascript-promises para una explicacion de las promesas mismas.

Notá también que estamos usando `console.log` de manera similar a como podríamos usar `printf` en C, para registrar errores por amor a la depuración. Podrías querer hacer

eso también! Solo date cuenta que `cosole.log` escribirá mensajes en la consola del navegador (esto es, la pestaña **Console** de las herramientas de desarrollador de Chrome), no en tu terminal. Mirá https://developer.mozilla.org/en-US/docs/Web/API/Console/log para tener sugerencias.

6.4.6. showInfo

Esta función abre la ventana de información en un marcador con un contenido particular (HTML). Aún si solo se provee de un argumento (`marker`), `ShowInfo` simplemente despliega un ícono giratorio (el cual es un GIF animado). Notá cómo esta función está creando una string de HTML dinámicamente, después de eso pasándola a `setContent`. Quizá debás mantener esa técnica en mente en otro lado.

6.4.7. update

Por último está `update`, la cual primero determina los bordes actuales, las esquinas superior derecha (Noreste) e inferior izquierda (Suroeste). Luego pasa esas coordenadas a `update.php` a través de una petición GET (debajo de la manga de `getJSON`) al estilo:

```
GET /update.php?ne=37.45215513235332%2C-122.03830380859375&q=&sw=37.3
9503397352173%2C-122.28549619140625 HTTP/1.1
```

eL `%2C` son solo comas que han sido "codificadas en URL". Date cuenta que nuestro uso de comas es arbitrario; esperamos que `update.php` analice y extraiga latitudes y longitudes de estos parámetros. Podríamos simplemente haber pasado cuatro parámetros distintos, pero sentimos que era más semánticamente limpio pasar solo un parámetro por esquina.

A como veremos pronto, `update.php` está diseñada para retornar un arreglo JSON de lugares que caen dentro de los bordes actuales del mapa (ciudades dentro de la vista). Después de todo, solo con esas dos esquinas podés definir un rectángulo, lo cual es exactamente lo que el mapa es.

Tan pronto como `update` responda, la función anónima pasada a `done` es llamada y pasó `data`, el valor del cual es el JSON emitido por `update.php`. (Aunque si algo sale mal, `fail` es llamada.) La función anónima primero quita todos los marcadores del mapa y luego iterativamente agrega nuevos marcadores, uno por cada lugar (ciudad) en el JSON.

6.5. update.php

Ahora navegá a `~/workspace/pset8/public` y abrí `update.php`. Ah, está bien, aquí está el script "back-end" que generó un arreglo JSON de hasta 10 lugares (ciudades) que caen dentro de bordes específicos (dentro del rectángulo definido por esas esquinas). No necesitás hacerle cambios a este archivo, pero leelo línea por línea, Googleando cualquier función con la que no te sintás familiar. `preg_match` te debería ser de particular interés, la cual te permite comparar strings contra "expresiones regulares". Aunque pueda verse críptica a primera vista, nuestras dos llamadas a `preg_match` en `update.php` son simplemente asegurándose que ambos `sw` y `ne` son latitudes y longitudes separadas por coma.

Oh, si, las peticiones SQL de este archivo asumen que el mundo es llano por simpleza.

6.6. search.php

A continuación, abrí `search.php`. Ah, no hay mucho por acá ahorita, solo un eventual `TODO`.

6.7. articles.php

Ahora abrí `articles.php`. Este lo hemos implementado por vos. Notá como espera un parámetro GET llamado `geo`, el cual pasa a `lookup` (lo cual está definido en `helper.php`) pidiendo noticias locales, luego retornan un arreglo de objetos JSON, cada uno de lo cuales tiene dos claves: `link` y `title`.

De hecho, podés ver esto en acción. Visitá URLs como

- https://ide50-username.cs50.io/articles.php?geo=Cambridge,+Massachusetts
- https://ide50-username.cs50.io/articles.php?geo=02138

o bien

- https://ide50-username.cs50.io/articles.php?geo=New+Haven,+Connecticut
- https://ide50-username.cs50.io/articles.php?geo=06511

donde `username` es tu propio nombre de usuario. Deberás ver (impreso bonito) arreglos JSON de artículos.

6.8. config.php

Tomemos un vistazo rápido del archivo que ha sido requerido por todos esos archivos PHP. Navegá a `~/workspace/pset8/includes` y abrí `config.php`. Ah, un archivo como el propio `config.php` del Problem Set 7, aunque más simple. Incluso carga la librería PHP de CS50, junto con `helper.php`.

6.9. helpers.php

En este archivo solo hemos definido una función, `lookup`. Aunque distinto del `lookup` del Problem Set 7, esta versión de `lookup` hace peticiones a Google News de noticias para una geografía en particular. No hay necesidad de entender cada una de las líneas de este archivo, pero revisá sus comentarios.

6.10. config.json

Ahora abrí `config.json`. AH, otra vista familiar, aunque para la base de datos llamada `pset8`.

7. What To Do

Bien, es hora de juntar dos APIs de Google.

7.1. config.json

Primero, atacá esos `TODO`s dentro de `cofig.json`, a como hiciste en Problem Set 7.

7.2. import

A continuación, recordá que `places`, esa tabla MySQL que importaste más temprano, está actualmente vacía. Los datos necesitan estar en ella, mientras tanto, está en `US.txt`.
Escribí en `import`, un script de línea de comando en PGP que acepte como argumento de línea de comando el camino a un archivo (el cual puede asumirse en un formato como `US.txt`) que itera sobre las líneas del archivo, insertando cada uno como una nueva fila en `places`. Te dejamos a vos el diseño general de este script, pero asegurate de hacer una revisión rigurosa, aprovechando `file_exists`[12], `is_readable`[13], y/o cosas

[12] http://php.net/manual/en/function.file-exists.php
[13] http://php.net/manual/en/function.is-readable.php

similares. De seguro encontrarás a fopen[14], fgetcsv[15], y fclose[16] particularmente útiles, junto con `CS50::query` de la librería PHP de CS50. Notá que `fgetcsv` toma un tercer argumento opcional que te permite anular el delimitador predefinido de una coma o alguna otra cosa.

Para correr este script, deserás ejecutar un comando como el siguiente

```
./import /path/to/US.txt
```

donde `/path/to/US.txt` es de hecho el camino (relativo o absoluto) a ese archivo.

De seguro las primeras veces que corrás tu script no estará bien, así que querrás vaciar `places` entre corridas, al ejecutar

```
TRUNCATE places
```

en la pestaña **SQL** de phpMyAdmin haciendo click en **Empty the table (TRUNCATE)** en la pestaña **Operations** de phpMyAdmin. Si tomás este último acercamiento, asegurate de haber primero seleccionado **places** (al hacer click en él en la columna al lado izquierdo de phpMyAdmin) para que no trunqués alguna otra tabla. Y asegurate de no hacer click en **Delete the table (DROP)**, de otra forma, tendrás que re-importar `pset8.sql` y re-crear cualquier cambio hecho.

Ya sea ahora o más tarde, deberías probablemente agregar uno o más índices adicionales a `places` para acelerar las búsquedas (para `search.php`. Mirá `https://dev.mysql.com/doc/refman/5.5/en/mysql-indexes.html` y `https://dev.mysql.com/doc/refman/5.5/en/fulltext-search.html` (y Google) para tener tips. (Nosotros definimos `places` en `pset8.sql` al usar un "motor" MyISAM para que un índice `FULLTEXT` sea una opción.)

Incluso aunque los datos puedan a veces ser importados en bulto a través de la pestaña **Import** de phpMyAdmin, debés implementar `import` a como fue prescrito.

[14] http://php.net/manual/en/function.fopen.php
[15] http://php.net/manual/en/function.fgetcsv.php
[16] http://php.net/manual/en/function.fclose.php

7.3. search.php

Implementá `search.php` de tal forma que genere un arreglo JSON de objetos, cada uno de los cuales representa una fila de `places` que de alguna forma concuerda con el valor de `geo`. El valor de `geo`, pasado en `search.php` como un parámetro GET, mientras tanto, podría ser una ciudad, estado y/o código postal. Te dejamos a vos decidir qué constituye una correspodencia y, por tanto, qué filas `SELECT` (seleccionar). De seguro encontrarás las palabras clave `LIKE` y/o `MATCH` de mucha ayuda.

Por ejemplo, considerá la siguiente petición

```
cs50::query("SELECT * FROM places WHERE postal_code = ?", $_GET["geo"])
```

Desafortunadamente, esa petición requiere que el input de un usuario sea exactamente igual a un código postal(según el `=`), lo cual no es tan obligatorio por la autocompletación. ¿Que tal si en vez de esa usamos esta otra?

```
cs50::query("SELECT * FROM places WHERE postal_code LIKE ?", $_GET["geo"] . "%")
```

Notá cómo esta petición *apela* al input del usuario, el cual resulta ser el caracter comodín de SQL que significa "matcheá cualquier número de caracteres". El efecto es que esta petición retornará filas cuyo código postal corresponderán con lo que sea que el usuario haya escrito seguido por cualquier número de caracteres. En otras palabras, cualquiera de entre `0`, `02`, `021`, `0213` y `01238` puede retornar filas, así como lo pueden cualquiera de entre `0`, `06`, `065`, `0651`, y `06511`.

Finalmente, considerá una petición como la siguiente.

```
cs50::query("SELECT * FROM places WHERE MATCH(postal_code, place_name)
AGAINST (?)", $_GET["geo"])
```

Esta petición no solo busca en `postal_code`, sino también en `place_name`, pero le deja a MySQL el pensar cómo hacerlo. Asume que has definido un índice `FULLTEXT` juntamente en `postal_code` y `place_name`, lo cual podés hacer a través de la pestaña **Structure** de phpMyAdmin. (Mirá si podés determinar cómo hacerlo)

De segura no querrás usar algunas de estas peticiones completamente, sino decidir por vos mismo qué tipo de búsquedas soportar y qué campos buscar.

A como se dijo antes, mirá https://dev.mysql.com/doc/refman/5.5/en/string-comparison-functions.html y https://dev.mysql.com/doc/refman/5.5/en/fulltext-search.html para cierta guianza, aunque Google y Stack Overflow pueden tener algunas sugerencias útiles.

Para probar `search.php`, incluso antes de que tu caja de texto sea operacional, simplemente visitá URLs como las siguientes:

- https://ide50-username.cs50.io/search.php?geo=Cambridge,Massachusetts,US
- https://ide50-username.cs50.io/search.php?geo=Cambridge,+Massachusetts
- https://ide50-username.cs50.io/search.php?geo=Cambridge,+MA
- https://ide50-username.cs50.io/search.php?geo=Cambridge+MA
- https://ide50-username.cs50.io/search.php?geo=02138

o bien

- https://ide50-username.cs50.io/search.php?geo=New+Haven,Connecticut,US
- https://ide50-username.cs50.io/search.php?geo=New+Haven,+Massachusetts
- https://ide50-username.cs50.io/search.php?geo=New+Have,+MA
- https://ide50-username.cs50.io/search.php?geo=New+Haven+MA
- https://ide50-username.cs50.io/search.php?geo=06511

y otras variantes, donde `username` es tu propio nombre de usuario, para ver si conseguís de vuelta el JSON que esperás (y no un gran error naranja). De nuebo, dejamos a tu criterio cuán tolerante será `seach.php` de abreviaciones, puntuación, y cosas similares. Mientras más flexible, mejor. Tratá de implementar características que esperarías como usuario.

Sentite en la libertad de ver la solución del staff en https://mashup.cs50.net/, inspeccionando sus peticiones HTTP a través de la pestaña **Network** de Chrome a como sea necesario, en caso de que estés inseguro de cómo tu propio código debería funcionar.

7.4. scripts.js

Primero, en la parte superior de `scripts.js` verás una función anónima, dentro de la cual hay una definición de `options`, un objeto, una de cuyas claves es `center`, el valor del cual es un objeto con dos claves propias, `lat`, y `lng`. Según el comentario al lado de ese objeto, el mapa de tu mashup está actualmente centrado en Stanford, California. Cambiá las coordenadas del centro de tu mapa a Cambridge (42.3770, -71.1256) o New Haven (41.3184, -72.9318) o cualquier otro lugar. (Aunque tenés que asegurar de usar coordenadas en EUA si descargaste `US.txt`. Una vez que guardés tus cambios y recargués el mapa, deberías encontrarte ahí. Alejá el zoom tanto como lo necesités para confirmar visualmente.

Como dicho antes, sentite en la livertad de jugar con la solución del staff en https://mashup.cs50.net/, inspeccionando sus peticiones HTTP a través de la pestaña **Network** de Chrome a como sea necesario, en caso de que estés inseguro de cómo tu propio código debería funcionar.

7.4.1. configure

Ahora que `search.pgp` y tu caja de texto están (ojalá) trabajando, modificá el valor de `suggestion` en `configure`, la función en `scripts.js`, para que despliegue las coincidencias (esto es, `place_name`, `admin_name1`, y/o otros campos) en vez de `TODO`. Recordá que un valor como

```
<p><%- place_name%>, <%- admin_name1%></p>
```

podría hacer el truco, quizá acoplado con algo de CSS.

7.4.2. addMaker

Implementá `addMarker` en `scripts.js` de forma tal que agregue un marcador para `place` en el mapa, donde `place` es un objeto JavaScript que representa una fila desde `places`, tu tabla MySQL. Mirá https://developers.google.com/maps/documentation/javascript/markers para ver tips. Pero, también mirá
http://google-maps-utility-library-v3.googlecode.com/svn/tags/markerwithlabel/1.1.9/
para una alternativa a los marcadores propios de Google, los cuales agregan soporte para las etiquetas debajo de los marcadores. (Recordá que ya estamos cargando `markerwithlabel_packed.js`) para vos en `index.html`.)

Al hacer click en un marcador, debería disparar la ventana de información de mashup para que se abra, anclada en el mismo marcador, los contenidos de los cuales deberían ser una lista sin orden de enlaces a un artículo para la locación de ese artículo (a menos que `articles.php` genere un arreglo vacío).

De nuevo, no te preocupés si alguno de tus marcadores (o etiquetas) se sobreponen entre sí, asumiendo que tal es el resultado de las imperfecciones de `US.txt` y no de tu propio código.

Si quisieras personalizar los íconos de tus marcadores, mirá https://developers.google.com/maps/documentation/javascript/markers#simple_icons. Para las URLs de íconos contruídos en Google Maps, mirá http://www.lass.it/Web/viewer.aspx?id=4. Para íconos de terceros, mirá https://mapicons.mapsmarker.com/

7.4.3. removeMakers

Implementá `removeMarkers` de tal forma que remueva todos los marcadores del mapa. De segurás necesitarás que `addMarker` modifique esa variable global llamada `markers` para que `removeMarkers` haga su propia magia.

7.4.4. personal touch

Por último, pero no menos importante, agregá al menos un toque personal a tu mashup, alterando su estética o agregando algunas características que (idealmente) no tenga ningún compañero de clases. Cualquier toque que te obligue a aprende (o Googlear) al menos una nueva técnica es de un alcance razonable.

8. Entrega

Debe de compartir su espacio de trabajo en su cuenta de cs50.io por lo cual deberá de seguir los siguientes pasos (por favor omitir los que muestran cómo crear la cuenta desde cero si ya tiene una. Estas instrucciones ya estaban descritas también desde el PSet1. Si ya compartió su espacio de trabajo no es necesario repetir el proceso):

- Acceda a edx.org y dé clic en el botón Registrer (parte superior derecha del sitio).
- Rellene todos los datos que se le solicitan o acceda con una de sus redes sociales (es preferencial).
- Una vez que se registró diríjase a cs50.io y será redirigido a una página donde deberá escoger la opción **edX** como **CS50 ID** (dé clic en **Submit**).

- Se le redirigirá a un formulario de acceso del sitio de **edx.org** donde deberá ingresar su correo electrónico y contraseña.
- Una vez que ha accedido ya podrá compartir su **Workspace** dando clic en el botón **Share** ubicado en la **parte superior derecha.**
- Una vez que creó sus cuentas en **edX** y en el **IDE de CS50**, al mismo tiempo que compartió el **Workspace** con el usuario **silfdv** ya podremos revisar su código.

Esto fue PSet8.