Problemas encontrados

La documentación oficial (https://developers.google.com/finance/) no hace referencia alguna a cualquiera de los datos que son exigidos en los requisitos del bloque 1. La API únicamente sirve para tener acceso a los valores guardados en el 'portfolio' de tu cuenta personal (http://www.google.com/finance). Utilicé la API para comprobar todos los casos posibles, prueba de ello es la clase /nasdaq/application/util.php junto con el token de sesión proporcionado para poder autenticarme contra el servicio y obtener los XML.

Solución: desde http://www.google.com/finance se puede tener acceso a los datos de cada valor en formato JSON a través de la siguiente ruta: http://www.google.com/finance?q={símbolo de la compañía}&output=json.

De este modo sí tengo acceso a todos los datos necesarios, pero esto supone una gran limitación: no es una API documentada, no sé si en el futuro dejará de funcionar, de sí lo datos obtenidos pueden ser usados con fines comerciales o pueden bloquearme el acceso al servicio por su uso indiscriminado. He modificado el user-agent para que el servidor me reconozca como un browser. Se podían haber tomado muchas otras medidas al respecto.

Limitaciónes

Haciendo uso de la pseudo API anteriormente comentada, no resulta razonable obtener los datos de las más de 600 compañías que cotizan en NASDAQ, primero por una cuestión de rendimiento, es decir: para obtener los 30 valores con un precio de mercado mayor, es necesario descargarse los datos de cada uno de los 600 valores. El servicio no ofrece (que yo sepa) otro modo de acceder a esa información sin tener que descargarse los datos de todos los valores. Segundo, por una cuestión de riesgo, puesto que nos pueden bloquear el acceso por un uso abusivo.

Por lo tanto, se han escogido 30 compañías al azar (las 30 compañías que ofrecían un precio mayor en un día determinado) a modo de ejemplo. Realizar el informe tarda alrededor de unos 10 segundos.

Puesto que el reporte que realizo es obtenido de datos en tiempo real, cabría la posibilidad de hacer una mejora cacheando los datos durante un periodo de tiempo, pudiendo mostrar datos actualizados, por ejemplo cada 5 minutos, tiempo suficiente para poder haber obtenido los datos del resto de valores que no se han considerado inicialmente.

Es posible cambiar o añadir los valores que son consultados por la aplicación desde el fichero *php.ini*.

Explicación de los valores (CAPM)

Para poder calcular la estimación acorde a la fórmula del CAPM y envase a ella calcular el valor real de cada acción, he necesitado los siguientes datos (la fórmula es obtenida de http://es.wikipedia.org/wiki/Capital_asset_pricing_model):

$$E(r_i) = r_f + \beta_{im}(E(r_m) - r_f)$$

PRECIO (stock market): Coste de adquisición de una acción cotizada en el mercado de referencia a lo largo de la sesión bursátil. Este dato es obtenido directamente de la API.

EPS: Beneficio por cada acción obtenido el ejercicio anterior por la compañía, es decir, el resultado de dividir el capital activo de la compañía por el número de acciones emitidas de dicha compañía. Este dato es obtenido directamente de la API.

RENTABILIDAD: resultado de hacer la división EPS/PRECIO, es la tasa de beneficio por acción que espero obtener en el caso de que adquiera dicho titulo y se repita el mismo beneficio obtenido en el año precedente. Este dato es necesario para posteriormente calcular E(M) (ver la descripción de E(M)).

RF: La tasa de rentabilidad de un activo libre de riesgo. Es parte de la formula del modelo CAPM. Se ha elegido en este análisis el rendimiento del bono del Tesoro USA a 10 años igual a 1,66%. Este dato no ha sido obtenido de la API, sino del siguiente enlace: http://finance.yahoo.com/q?s=^TNX. Este valor puede ser actualizado "a mano" a través del fichero *app.ini* (el campo se llama *american_bond_rate*).

Bi: Es la Beta, indica la cantidad de riesgo de un valor individual respecto a la cartera del mercado donde se negocia. Indica la variabilidad de la rentabilidad de la acción respecto al resto del mercado. Betas menores que 1 indican menos variabilidad que la media del mercado, (la beta del mercado es 1), y menor riesgo a la hora de invertir. Betas mayores que 1 indican mayor variabilidad que el mercado y mas riesgo al invertir. Beta igual a 0 indica ausencia de riesgo. Este dato es obtenido directamente de la API y es utilizad directamente en la formula del modelo CAPM.

E(M): Es la tasa de rentabilidad del mercado. En nuestro modelo se ha obtenido calculando la media de las rentabilidades (precio de una acción en un momento dado por el número total de acciones que forman el capital social, como se ha comentado anteriormente) de entre nuestra muestra de 30 valores de las compañías de mayor capitalización bursátil del NASDAQ. Interviene igualmente en la fórmula del modelo CAPM.

E(Ri) o CAPM: Es la tasa de rendimiento esperada de capital sobre una acción concreta. Es la variable dependiente de la fórmula e interviene en la tasa de actualización de los flujos de retorno de la inversión que dan lugar al cálculo del precio del activo sobre el que se planea invertir.

FLUJOS DE CAJA: En nuestro modelo es la suma del PRECIO más EPS. Esta suma es la que se esperaría recuperar al cabo de un año.

TASA DE ACTUALIZACIÓN: Es la tasa de descuento que se utiliza para calcular el valor actual de los flujos de caja esperados a lo largo del año siguiente. Se determina a efectos de cálculo sumando a la unidad la tasa E(Ri), calculada en el modelo.

VALOR DE LA ACCIÓN: Es el resultado de aplicar la TASA DE ACTUALIZACION a los FLUJOS DE CAJA, dividiendo estos por la anterior. Si como resultado de esta actualización el valor actualizado esperado por la inversión es inferior al precio de la cotización del mercado se dice que dicha acción esta sobrevalorada. Si por el contrario el valor actualizado es superior al de cotización se dice que la acción está infravalorada.

MongoDB

El nombre de la base de datos utilizada es la utilizada por defecto en la instalación, la llamada 'test'. Dentro de esta base de datos, la colección utilizada se llama 'users'.

La base de datos ha sido utilizada para guardar el nombre de los usuarios y sus emails. El sistema controla que no puedan haber direcciones de correo repetidas.

He definido un campo adicional para cada usuario, el campo llamado *timestamp*. Lo utilizo para guardar la marca de tiempo de la última vez que se le envió un reporte con los datos bursátiles a su correo. No se permite volver a enviar un correo al mismo usuario hasta pasadas 12 horas.

Se aporta en /doc el fichero con los datos importados de mi base de datos correspondiente a la base de datos 'test' y a la colección 'users'.

Uso de librerías externas

Por un parte he tenido que añadir el módulo de cURL en su última versión a mi instalación de PHP. También he tenido que utilizar el módulo Pear Mail en su versión 1.2.0, puesto que la que venía por defecto no funcionaba correctamente.

MVC Framework

Apenas tengo experiencia en PHP así que no he tenido oportunidad de probar ningún framework.

Sin embargo mi diseño de la aplicación implementa el patrón MVC. Este patrón y las clases utilizadas serán explicadas en el apartado de diseño.

Envío de emails

Se puede ejecutar desde consola por medio de la siguiente orden:

```
php -f send_mail.php subject="the subject",
from_email="An@email.com", to_email="another@email.com",
from_name="A name", to_name="Another name"
```

Requisitos sin implementar

PHPUnit: por falta de tiempo y por desconocimiento sobre el uso de pruebas de unidad sobre PHP.

AJAX en la vista de reporte: por falta de tiempo. Sin embargo su implementación habría sido bastante sencilla. Como ya comentaré en el apartado de diseño, a la vista de reportes le paso desde mi clase controladora la parte del cuerpo que conforma el HTML de cada uno de los cuerpos de las tablas de cada informe. Lo único que debería haber hecho es haber creado otro recurso .php el cual devolviera un objeto JSON con 3 elementos, que son, cada una de las estructuras HTML que le paso a la vista de reporte que he comentado antes (siempre es mejor cuando modificas el DOM de la página escribir directamente un bloque de HTML, que operar manualmente con los nodos del DOM, puesto que el coste resulta mayor). Tras esto, bastaría con hacer una llamada XHR ha dicho recurso e insertar los datos directamente en el informe.

Capturas

En /doc hay 4 capturas que muestran cada una de las páginas que conforman la aplicación (la home, el formulario de registro, la lista de mails, y la página con el informe).

Diseño

Clase Controller

Contiene los objetos modelo (clase Model) y vista (clase Load). La clase Controller obtiene y actualiza los datos de Model, que posteriormente serán pasados a la vista, a través de la clase Load que buscará y mostrará los datos renderizados en HTML. La clase Controller también es la encargada de capturar los eventos que provengan del cliente, para poder iniciar el flujo de datos a través del modelo y cargar la vista solicitada, así de como filtrar los datos de entrada.

Es recomendable delegar ciertas funcionalidades de la clase controladora a otras clases auxiliares o al propio modelo. En mi caso, las funciones encargadas de traducir los datos obtenidos a HTML para posteriormente pasarlo a la vista correspondiente podrían haber ido en una clase estática aparte con dicho fin.

Clase Load

Clase con una única función que por un lado, expande las variables que provienen de una colección para que estén disponibles en la vista, y por otro lado, carga el fichero de la vista que no tendría que ser más (en principio) que simple HTML sin ningún tipo de lógica.

Clase Model

Clase encargada de interactuar con la API de MongoDB a modo de CRUD. Una mejora podría haber sido extender de esta clase para haber realizado un DAO de 'Users'.

Clase MyMail

Clase que utiliza el módulo Pear Mail para el envío de correos. La he llamado 'MyMail' puesto que el nombre 'Mail' entraba en conflicto con el de la propia librería de Pear. Desconozco si habrá algún modo de desambigüar el nombre de las clases en estos casos.

Nasdaq.php

Script que se encarga de cargar todas las librerías e iniciar la clase controladora para que esté a la escucha de los eventos de usuario.

Send_mail.php

Script para enviar mails que puede ser invocado desde linea de comandos.

Clase Util

Clase que utilicé para invocar al Google Finance API que posteriormente, desestimé.

/Views

Directorio con cada una de las 4 vistas del sistema.

Clase StockQuotes

Clase que se encarga de obtener, parsear y calcular todos los datos financieros que es invocada por la clase controladora. Dada su extensión y complejidad, para entender su funcionamiento es mejor leer los comentarios en el código de dicha clase.

Por otra parte, la aplicación ha sido probada en Chrome y Firefox.