

# Fase 1: crear app, implementar enrutamiento y desarrollar la 1ª ruta

## Enunciado.

Vamos a desarrollar una aplicación de Angular con **enrutamiento** (*routing*) que se llame **WalletWatcher**. La aplicación permitirá consultar los precios actuales de 15 de las criptomonedas más populares. También permitirá que un usuario especifique su cartera de criptomonedas, para que la aplicación calcule automáticamente el balance de la cartera en función de los precios actuales.

## Origen de datos

Los datos van a ser obtenidos de las siguientes fuentes por orden de prioridad, es decir, se intentará la fuente (1) y si hay algún obstáculo insalvable pasaremos a la fuente (2) y así sucesivamente:

1. La **API** de **Nomics** en <https://nomics.com>.
2. Si fallara el acceso a este último servicio entonces el profesor os proporcionaría un archivo **JSON** para integrar en nuestra app como recurso local.

La URL base para consultar precios es:

<https://api.nomics.com/v1/currencies/ticker>

A la ruta base se le concatenarán al menos 2 parámetros:

- key, cuyo valor debe ser nuestra API Key;
- ids, cuyo valor debe ser una lista de IDs de monedas separados por comas. Por ejemplo, "ids=BTC,ETH,USDT,BNB". Llegaremos a consultar hasta 15 criptomonedas simultáneamente y sus IDs son:  
BTC,ETH,USDT,BNB,USDC,XRP,LUNA,SOL,ADA,AVAX,DOT,DOGE,SHIB,MATIC,ATOM

## Primer servicio

Tienes libertad para implementar servicios a tu criterio. Sin embargo, si te sirve de guía, aquí tienes los detalles del primer servicio que necesitas para obtener datos del origen de datos:

- **MarketData**. Conoce la *URL* del servidor, tu *API Key* y la lista de IDs de las 15 criptomonedas principales. Es capaz de hacer llamadas *HTTP* para obtener los datos en formato *JSON* necesarios para alimentar a la aplicación. También conviene que este servicio mantenga esas consultas cacheadas, para no saturar la *API*.

Los precios se consultan por primera vez en el constructor de este servicio. Una vez que se obtiene esta información, el servicio es capaz de aportar al resto de componentes de la aplicación consultas sencillas, como:

- *getMarketData*: devuelve información detallada de la lista completa de las 15 criptomonedas más populares;
- *fullUpdate*: actualiza la caché del servicio para tener todos los precios más actuales que los obtenidos en consultas anteriores;
- *update*: necesita que le proporciones una lista de IDs de monedas, ya que consulta al origen de datos por dichas monedas y devuelve información detallada y actualizada de ellas;
- Otras funciones necesarias para implementar objetivos opcionales.

## Hoja de estilos

De manera general, la aplicación usará **Bootstrap**, por lo que es necesario enlazar los archivos requeridos. Los estilos que sean exclusivos de un componente estarán en el CSS de dicho componente.

## Componente raíz

A continuación, se indica una lista de los 3 componentes que será necesario crear para dar contenido al componente raíz:

- **Head.** Muestra la imagen logotipo de la aplicación y el nombre de la aplicación.
- **NavMenu.** Ofrecerá la posibilidad de clicar en las secciones listadas en el apartado **Navegación y enrutamiento** más abajo.
- **Foot.** Muestra el nombre del autor o autora de la aplicación, es decir, tu nombre. También cita la fuente de los datos.

## Navegación y enrutamiento



Justo debajo del encabezado aparecerá una barra de navegación horizontal con varios elementos y, en esta Fase 1, debemos:

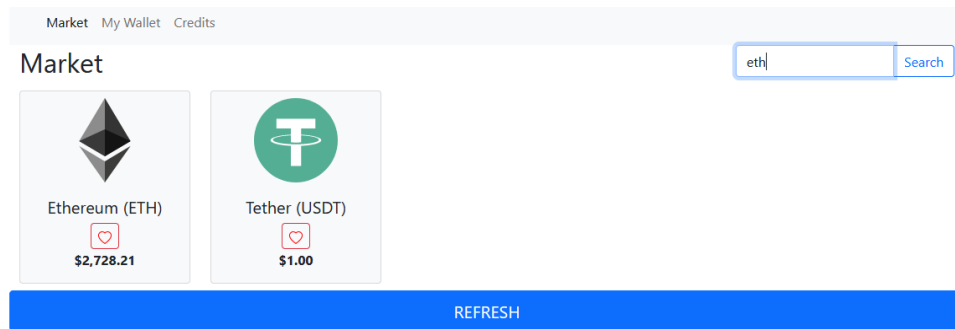
- a. implementar el enrutamiento (que el usuario pueda cambiar de sección, aunque estén vacías)
- b. desarrollar la 1ª sección dándole contenido y funcionalidad.

Si a la URL base de nuestra aplicación se le añade alguno de los siguientes sufijos, se mostrará lo que se indica a continuación:

- `"/market"`: lleva a la sección **Market**.
- `"/wallet"`: lleva a la sección **My Wallet**.
- `"/credits"`: lleva a la sección **Credits**.
- `"/"`: lleva a la sección **Market**.
- Cualquier otro sufijo: muestra el componente **Page404**.

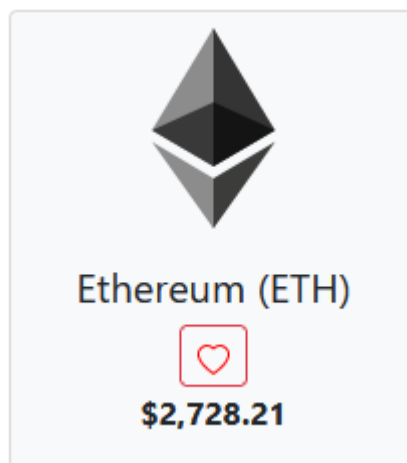
Esta primera sección, **Market**, mostrará el componente **Market**. Este componente contiene:

- Un título y un campo para buscar por nombre o ID. Para filtrar las 15 criptomonedas y que sólo se muestren las buscadas, implementaremos un filtro (*pipe*) de Angular llamado **FilterByString**. Este filtro recibe un *array* de monedas y una cadena de texto escrita por el usuario en el campo de búsqueda, y devuelve otro *array* de monedas que sólo contiene aquellas del *array* original cuyo nombre o ID incluya la cadena de texto escrita por el usuario.
- Un listado de las 15 criptomonedas más populares en formato cuadrícula de cinco columnas. Con el buscador mencionado antes, podremos filtrar esos 15 resultados para que se muestren menos elementos.
- Un botón en la parte inferior que permitirá actualizar los precios listados con una nueva consulta al origen de datos.



Para mostrar en formato cuadrícula este listado, nos apoyaremos en el siguiente componente:

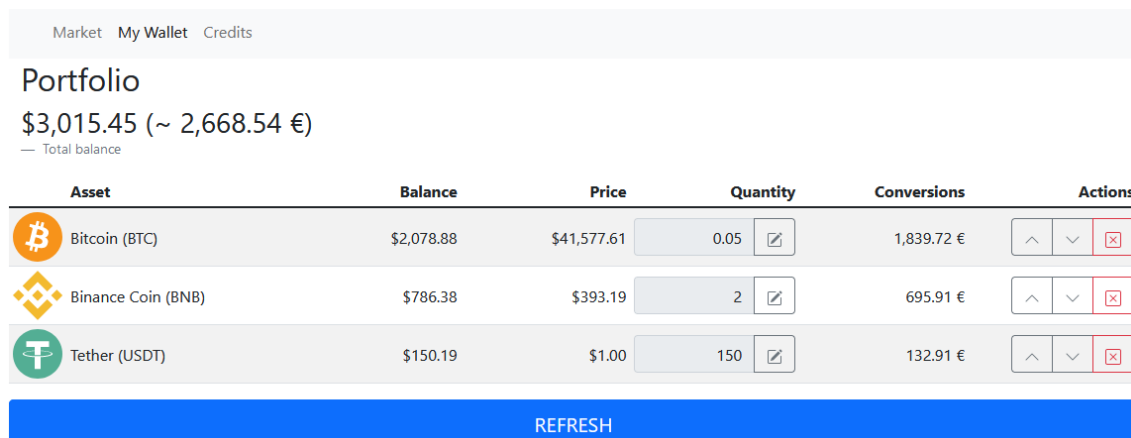
- **CoinCard**. Muestra en formato tarjeta información básica de una criptomoneda (logotipo, nombre e ID y precio). El botón de corazón se puede implementar más tarde, cuando pasemos a desarrollar la sección "My Wallet".






## Fase 2: desarrollar la 2ª ruta

La sección **My Wallet** mostrará el componente **Wallet**. La idea es que el usuario podrá incorporar criptomonedas a su *wallet* o *portfolio*. Para ello, en la primera sección, pulsará el botón de corazón de alguna moneda. Si la moneda ya existe en el *wallet* entonces no se vuelve a añadir.

Hasta aquí podría parecer una lista de favoritos, pero es algo más, tiene una funcionalidad extra. Una vez que el usuario ha añadido una o más monedas al *wallet*, podrá indicar la cantidad que posee de cada moneda. De esta forma, la aplicación podrá calcular el valor total del *wallet*. Observa la siguiente captura:



The screenshot shows a web application interface for a wallet. At the top, there are tabs for 'Market', 'My Wallet', and 'Credits'. Below the tabs, the title 'Portfolio' is followed by the total balance: '\$3,015.45 (~ 2,668.54 €)'. Below this, there is a table with columns: 'Asset', 'Balance', 'Price', 'Quantity', 'Conversions', and 'Actions'. The table lists three assets: Bitcoin (BTC), Binance Coin (BNB), and Tether (USDT). Each row shows the asset's icon, name, balance, price, quantity (with a disabled text input and an edit button), and conversion rate. The 'Actions' column contains up, down, and delete buttons. Below the table is a blue 'REFRESH' button.

Asset	Balance	Price	Quantity	Conversions	Actions
 Bitcoin (BTC)	\$2,078.88	\$41,577.61	0.05 <input type="text"/>	1,839.72 €	<input type="button" value="↑"/> <input type="button" value="↓"/> <input type="button" value="✕"/>
 Binance Coin (BNB)	\$786.38	\$393.19	2 <input type="text"/>	695.91 €	<input type="button" value="↑"/> <input type="button" value="↓"/> <input type="button" value="✕"/>
 Tether (USDT)	\$150.19	\$1.00	150 <input type="text"/>	132.91 €	<input type="button" value="↑"/> <input type="button" value="↓"/> <input type="button" value="✕"/>

REFRESH

En este ejemplo, el usuario ha incorporado BTC, BNB y USDT a su *wallet*. En el momento de la incorporación de una moneda, la cantidad de dicha moneda es cero. A continuación, ha especificado que tiene 0.05 BTC, 2 BNB y 150 USDT. La aplicación muestra las cantidades en campos de texto deshabilitados, pero el botón junto a ellos habilita la edición. Cuando el usuario ha introducido una cantidad, pulsa la tecla *Intro* o de nuevo dicho botón para que se almacene la cantidad y se deshabilite el campo.

Para generar el listado del *wallet* no nos apoyaremos en ningún componente adicional encargado de renderizar una sola fila. Sin embargo, al generar cada una de las filas sí que nos apoyaremos en dos componentes auxiliares:

- **AssetQuantity.** Incluye un campo de texto y el botón de al lado. El botón sirve para alternar el estado habilitado/deshabilitado del campo de texto. Pulsar la tecla *Intro* en el campo de texto también sirve para deshabilitar el propio campo. Cuando el usuario introduce una cantidad (entera o con decimales), esa moneda verá actualizado su *Balance*. El balance es el resultado de multiplicar la cantidad de moneda por el precio de la moneda.
- **AssetActions.** Incluye tres botones. Los dos primeros sirven para reordenar la lista de monedas en el *wallet*. El tercero sirve para eliminar la moneda del *wallet*.

### Segundo servicio

Aunque todavía no se ha mencionado, para almacenar la información del *wallet* haremos uso de un segundo servicio. El servicio **Wallet** almacenará el contenido del *wallet* del usuario. A diferencia del mercado, aquí cada moneda también debe guardar una cantidad numérica. Es decir, el *wallet* guarda los nombres y códigos de las monedas, los precios y las cantidades.

Además, nos aportará las siguientes funcionalidades:

- `addToWallet()`: permite añadir al *wallet* una nueva moneda con cantidad cero;
- `askForWalletPricesUpdate()`: solicita al origen de datos que se le proporcionen precios actualizados de las monedas que incluya el *wallet*;
- `getWalletAssets()`: devuelve el *wallet* con todas sus monedas;
- `getWalletTotalBalance()`: calcula y devuelve el valor total de todas las monedas que posee el *wallet*;
- `moveAssetDown()` y `moveAssetUp()`: permiten reordenar la lista de monedas del *wallet* controlando si la moneda ya se encuentra arriba del todo o abajo del todo;
- `removeAssetFromWallet()`: borra una moneda del *wallet*;
- `setAssetQuantity()`: permite establecer un nuevo valor en el campo “cantidad” de una moneda.

El *wallet* del usuario no debe perderse, por lo que este servicio se encargará de gestionar la **persistencia** en el almacenamiento local del navegador (**local storage**).

## Modelos de *templates* entregados por el profesor

Para ahorrar tiempo, se entregan algunas *templates* con código HTML genérico que ayudan mucho, pero deben ser adaptadas previamente para funcionar bien:

- Encabezado y pie
- Tarjeta de criptomoneda
- Listado de monedas del *wallet*
- Etc.

## Objetivos

Objetivos obligatorios	OK
Los contenidos del componente raíz se ajustan a lo pedido	
La barra de navegación funciona	
Un servicio obtiene datos del servidor mediante HTTP y dichos datos son usados y mostrados correctamente en alguna parte de la aplicación	
El buscador muestra resultados correctos al buscar por nombre sin importar mayúsculas o minúsculas	
Las monedas pueden ser añadidas al <i>wallet</i>	
Se lista correctamente el contenido del <i>wallet</i>	
Se puede borrar individualmente una moneda del <i>wallet</i>	
Los precios son indicados con el filtro de unidades monetarias de Angular	
Objetivos opcionales	OK
El usuario puede establecer cantidades de cada moneda de su <i>wallet</i>	
Las cantidades decimales se pueden introducir usando la coma y el punto	
El <i>wallet</i> muestra su valor total y dicho valor se actualiza si cambia el <i>wallet</i>	
El usuario puede reordenar las monedas de su <i>wallet</i>	
La aplicación actualiza los precios automáticamente cada 5 o 10 segundos para evitar que el usuario tenga que pulsar botones	
La tasa de cambio de USD a EUR se obtiene de algún servidor externo con valores actualizados	
No se realizan llamadas HTTP innecesarias y el aspecto está cuidado y los errores en la consola controlados	

## Conocimientos examinados

En este examen se **requiere** que el alumno o la alumna sepa:

- Crear aplicaciones web con Angular
- Insertar nuevos componentes en la aplicación
- Capturar eventos producidos por el usuario sobre ciertos elementos
- Definir enrutamiento para navegar dentro de la aplicación
- Establecer vías de comunicación entre componentes
- Añadir servicios e inyectarlos en los componentes que los requieran
- Realizar llamadas HTTP asíncronas para interactuar con una base de datos

En este examen se **valora** que el alumno o la alumna sepa:

- Detectar y manejar los posibles errores en tiempo de ejecución
- Ahorrar ancho de banda
- Implementar aspectos avanzados en aplicaciones web para aumentar la funcionalidad de la misma

## Calificación del examen

La correcta resolución de todos los objetivos obligatorios otorga una calificación de 5 sobre 10. Solamente si todos los objetivos obligatorios han sido resueltos adecuadamente se procederá a sumar puntuación adicional por cada uno de los objetivos opcionales solucionados con éxito, hasta un máximo de 5 puntos adicionales si se implementan 5 de los objetivos opcionales.