Contenido

[1. Autores del trabajo, planificación y entrega 2](#_Toc513404927)

[1.1 Autores 2](#_Toc513404928)

[1.2 Planificación 2](#_Toc513404929)

[1.3 Entrega 2](#_Toc513404930)

[2. Requisitos del prototipo a implementar 2](#_Toc513404931)

[2.1 Requisitos funcionales 3](#_Toc513404932)

[2.2 Otros requisitos 3](#_Toc513404933)

[3. Criterios de comparación en la implementación 4](#_Toc513404934)

[3.1 Criterio 1: Gráfico tarta 4](#_Toc513404935)

[3.2 Criterio 2: Gráfico de líneas 4](#_Toc513404936)

[3.3 Criterio 3: Gráfico múltiple eje x 4](#_Toc513404937)

[3.4 Criterio 4: Gráfico área stack 4](#_Toc513404938)

[3.5 Criterio 5: Gráfico de barras horizontales 4](#_Toc513404939)

[3.11 Criterio 11: Horas empleadas en el desarrollo 5](#_Toc513404940)

[3.12 Criterio 12: Velocidad de funcionamiento 5](#_Toc513404941)

[3.13 Criterio 13: Almacenamiento necesario para el desarrollo 5](#_Toc513404942)

[4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Chartjs 5](#_Toc513404943)

[4.1 Documentación de diseño 5](#_Toc513404944)

[4.2 Documentación de construcción 7](#_Toc513404945)

[4.3 Documentación de pruebas 9](#_Toc513404946)

[4.4 Documentación de instalación 10](#_Toc513404947)

[4.5 Manual de usuario 11](#_Toc513404948)

[5. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Echarts 11](#_Toc513404949)

[5.1 Documentación de diseño 11](#_Toc513404950)

[5.2 Documentación de construcción 12](#_Toc513404951)

[5.3 Documentación de pruebas 16](#_Toc513404952)

[5.4 Documentación de instalación 18](#_Toc513404953)

[5.5 Manual de usuario 18](#_Toc513404954)

[6. Comparación de las dos implementaciones 19](#_Toc513404955)

[6.1 Evaluación de los criterios en la implementación usando la tecnología Chartjs 19](#_Toc513404956)

[6.2 Evaluación de los criterios en la implementación usando la tecnología Echarts 19](#_Toc513404957)

[7. Comparación de la implementación de las tecnologías 20](#_Toc513404958)

[8. Conclusiones 22](#_Toc513404959)

# 1. Autores del trabajo, planificación y entrega

## 1.1 Autores

Componemos en grupo denominado como T4 y los integrantes son los siguientes:

* Urbano José Villanueva Rodríguez.
* Alberto Cabrera Plata.
* Fernando García Fernández.
* José Daniel Navarro Sierra.
* Martina Andrea Palomino Berrocal.

## 1.2 Planificación

En este apartado se debe incluir un enlace (URL) compartido a la planificación del trabajo utilizando una herramienta online de diagramación Gantt (por ejemplo, GanttPro, versión gratuita).

Hay que tener en cuenta que cada participante del grupo debe tener asignadas tareas que sumen al menos 45 horas. El peso de este trabajo en la calificación total de la asignatura es de un 30%, por tanto requiere de una dedicación de 45 horas del total de 150 horas de la asignatura.

## 1.3 Entrega

Enlace a GitHub: <https://github.com/UrbanoJVR/TG3>

* Informe del trabajo: con el nombre TG3\_final.docx
* Presentación del trabajo: TG3\_final.pptx

Con los archivos contenidos en el repositorio de GitHub y las instrucciones de instalación, se implementan los modelos de las dos tecnologías.

# 2. Requisitos del prototipo a implementar

El objetivo del proyecto es comparar la implementación de mismos prototipos utilizando las tecnologías Chartjs y Echarts.

En primer lugar, comparamos algunos de los gráficos más significativos que existen en ambas librerías, que son los siguientes: gráficos tarta, gráfico de barras, gráfico de líneas, gráficos múltiples, gráfico de ejes-x, gráficos de barras horizontales y gráficos radar.

En segundo lugar, realizamos la implementación de gráficos de árbol y mapa de densidad, que solo existen para la segunda librería estudiada, Echarts.

Finalmente, y como punto fuerte de la comparación entre ambas librerías, mostramos la implementación del gráfico de líneas de cada una de ellas a fin de mostrar datos de una API REST sobre el precio del bitcoin en USD, EUR y GBP.

## 2.1 Requisitos funcionales

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| RF01 | La aplicación permite mostrar el gráfico tarta en ambas tecnologías |
| RF02 | La aplicación permite mostrar el gráfico de barras en ambas tecnologías |
| RF03 | La aplicación permite mostrar el gráfico de líneas en ambas tecnologías |
| RF04 | La aplicación permite mostrar el gráfico múltiple en ambas tecnologías |
| RF05 | La aplicación permite mostrar el gráfico de ejes-x en ambas tecnologías |
| RF06 | La aplicación permite mostrar el gráfico de barras horizontales |
| RF07 | La aplicación permite mostrar el gráfico de tipo radar |
| RF08 | La aplicación permite mostrar un gráfico de líneas a fin de mostrar datos de una API REST sobre el precio de bitcoin en USD, EUR y GBP |
| RF09 | Mostrar leyenda de los gráficos |
| RF10 | Activar los datos que se muestran en los gráficos |
| RF11 | Desactivar los datos que se muestran en los gráficos |
| RF12 | Guardar gráfico en formato imagen |

## 2.2 Otros requisitos

USA: Usabilidad

MAN: Mantenibilidad

TR: Tiempo de respuesta

SEG: Seguridad

LAN: Lenguajes de alto nivel

REU: Reusabilidad

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| USA01 | La aplicación debe facilitar la navegación a través de sus páginas. |
| USA02 | La aplicación debe ser perfectamente manejable por cualquier tipo de usuario. |
| MAN01 | Cuidado en las fases de diseño, codificación y prueba. |
| MAN02 | Configuración adecuada del producto software. |
| MAN03 | Adecuada cualificación del equipo de desarrolladores del software. |
| MAN04 | Estructura del software fácil de comprender. |
| MAN05 | Estructura estandarizada de la documentación. |
| MAN06 | Documentación disponible de los casos de prueba. |
| TR01 | Todas las respuestas del sistema deben producirse en un tiempo máximo de 30 segundos. |
| SEG01 | La aplicación no perderá de forma accidental los datos. |
| SEG02 | El almacenamiento de información sensible, tanto propia de la lógica de la aplicación como las credenciales de acceso, debe de almacenarse cifrada en todos los servidores, y especialmente en el de base de datos. |
| LAN01 | Empleo de lenguajes de programación estandarizados. |
| LAN02 | Empleo de Javascript |
| REU01 | La aplicación reutilizará clases comunes. |

# 3. Criterios de comparación en la implementación

## 3.1 Criterio 1: Gráfico tarta

*Descripción: posibilidad de creación de un gráfico tarta.*

*Tipo de valor: booleano.*

## 3.2 Criterio 2: Gráfico de líneas

*Descripción: posibilidad de creación de un gráfico de líneas.*

*Tipo de valor: booleano.*

## 3.3 Criterio 3: Gráfico múltiple eje x

*Descripción: posibilidad de creación de un gráfico múltiple.*

*Tipo de valor: booleano.*

## 3.4 Criterio 4: Gráfico área stack

*Descripción: posibilidad de creación de un gráfico de eje-x.*

*Tipo de valor: booleano.*

## 3.5 Criterio 5: Gráfico de barras horizontales

*Descripción: posibilidad de creación de un gráfico de barras horizontales.*

*Tipo de valor: booleano.*

**3.6 Criterio 6: Gráfico de tipo radar**

*Descripción: posibilidad de creación de un gráfico de tipo radar.*

*Tipo de valor: booleano.*

**3.7 Criterio 7: Gráfico de mapa de densidad**

*Descripción: posibilidad de creación de un gráfico de mapa de densidad.*

*Tipo de valor: booleano.*

**3.8 Criterio 8: Gráfico de árbol**

*Descripción: posibilidad de creación de un gráfico de árbol.*

*Tipo de valor: booleano.*

**3.9 Criterio 9: Gráfico de datos recogidos de API**

*Descripción: conexión con una Api, lectura de datos y representación en un gráfico simple de barras. Los datos elegidos para representar es el valor del BTC.*

*Tipo de valor: booleano.*

**3.10 Criterio 10: Líneas de código**

*Descripción: cantidad de líneas de código que han sido necesarias para la implementación de los gráficos.*

*Tipo de valor: numérico.*

## 3.11 Criterio 11: Horas empleadas en el desarrollo

*Descripción: horas necesarias que se han empleado en el desarrollo de los gráficos.*

*Tipo de valor: numérico.*

## 3.12 Criterio 12: Velocidad de funcionamiento

*Descripción: velocidad de funcionamiento de los gráficos.*

*Tipo de valor: numérico.*

## 3.13 Criterio 13: Almacenamiento necesario para el desarrollo

*Descripción: cantidad de memoria necesaria del ordenador para la realización y funcionamiento de los gráficos.*

*Tipo de valor: numérico.*

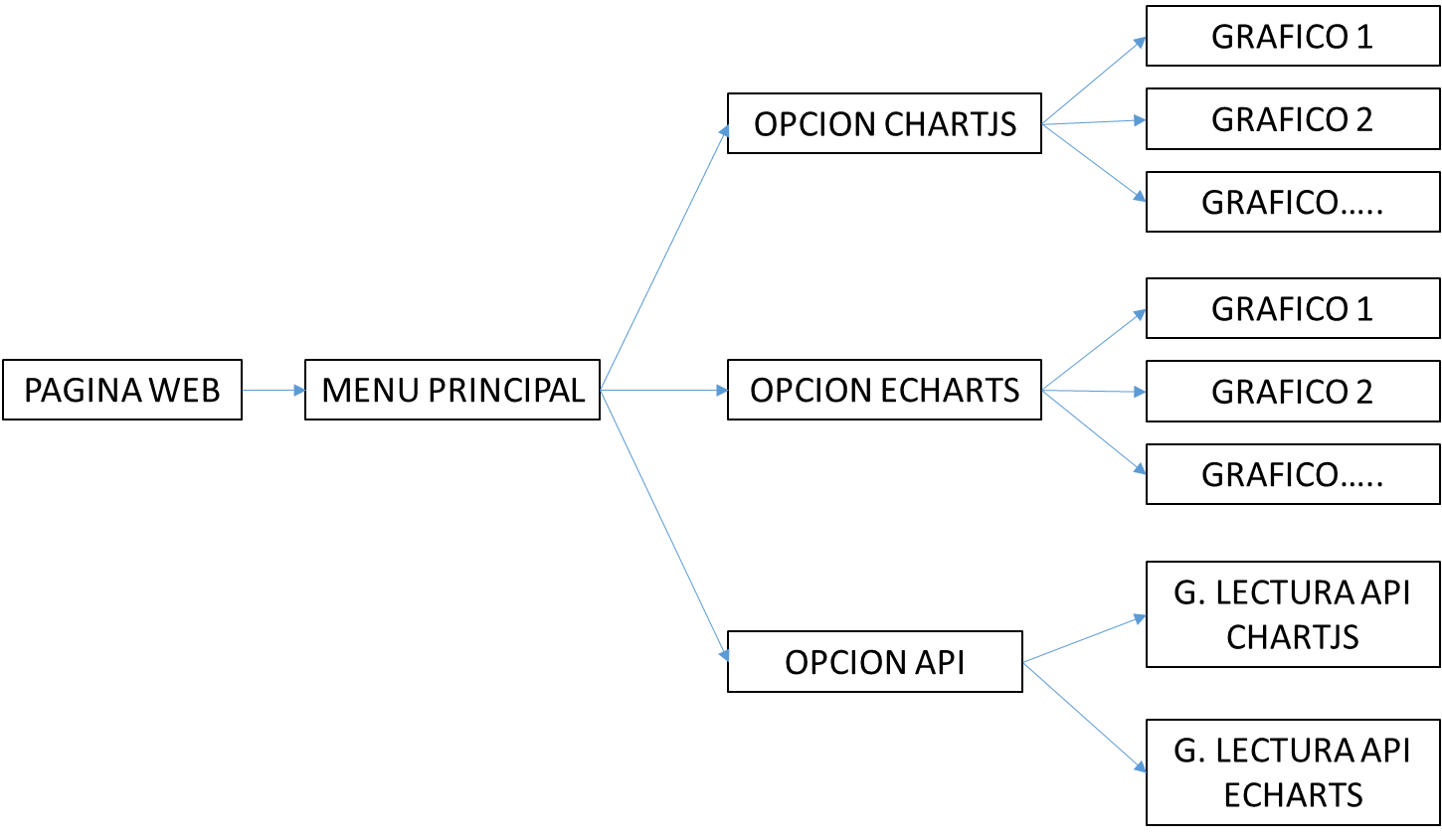
# 4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Chartjs

## 4.1 Documentación de diseño

**DESCRIPCIÓN**

Se pretende realizar un página web en la cual se permita la visualización de gráficos con la tecnología Chartjs, como se va a tratar de un entorno web, queremos incluir dentro del mismo entorno también la tecnología Echarts, por lo que se implementará una página Web, con un menú principal, en el que se presentarán 3 opciones, una para cada tecnología y ver tipos de gráficos que se pueden implementar en cada una y una tercera en la que se implementará una solución real, en la cual se leerán los datos de una API, y dibujará el mismo tipo de gráfico en ambas tecnologías.

**DIAGRAMAS**

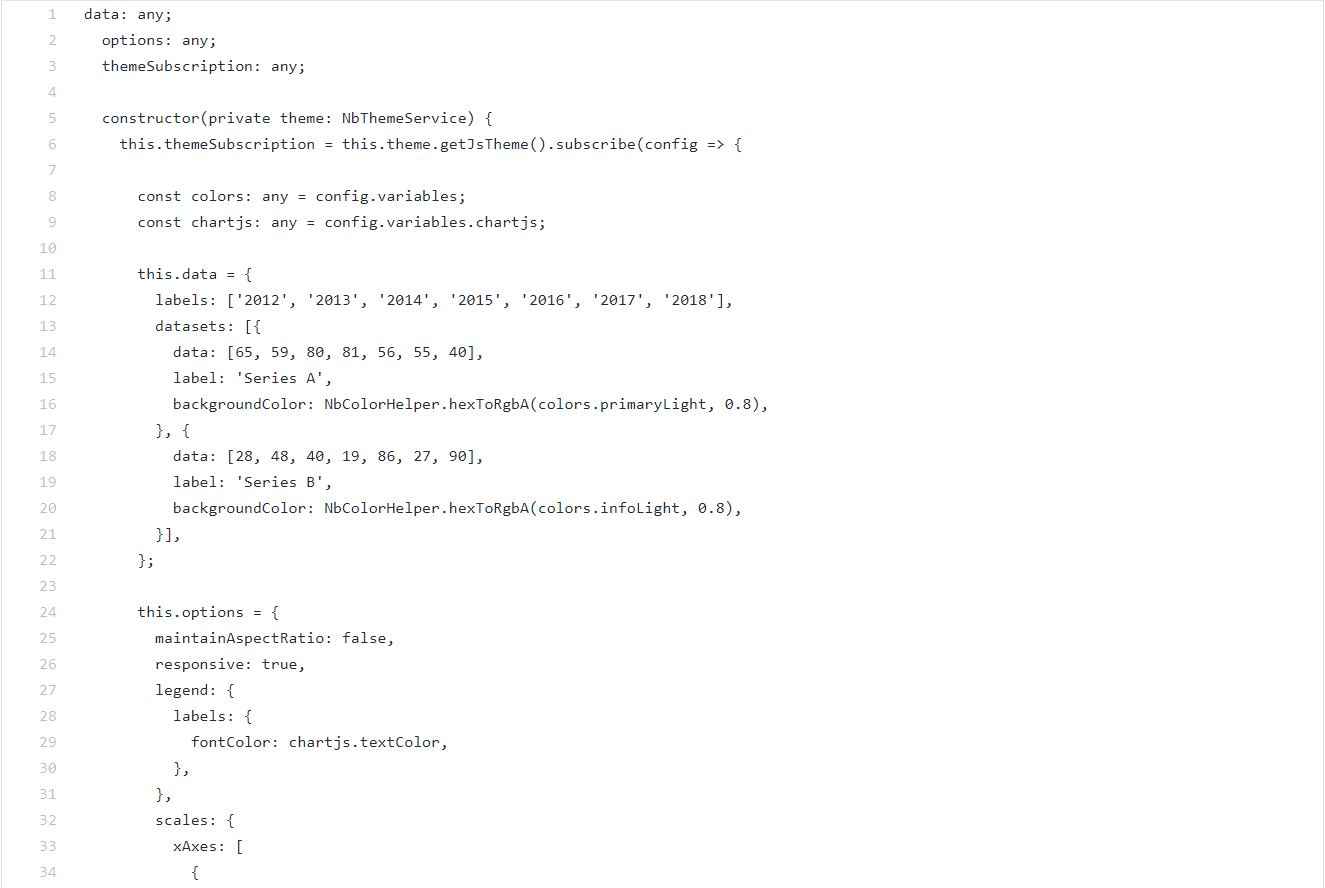
****

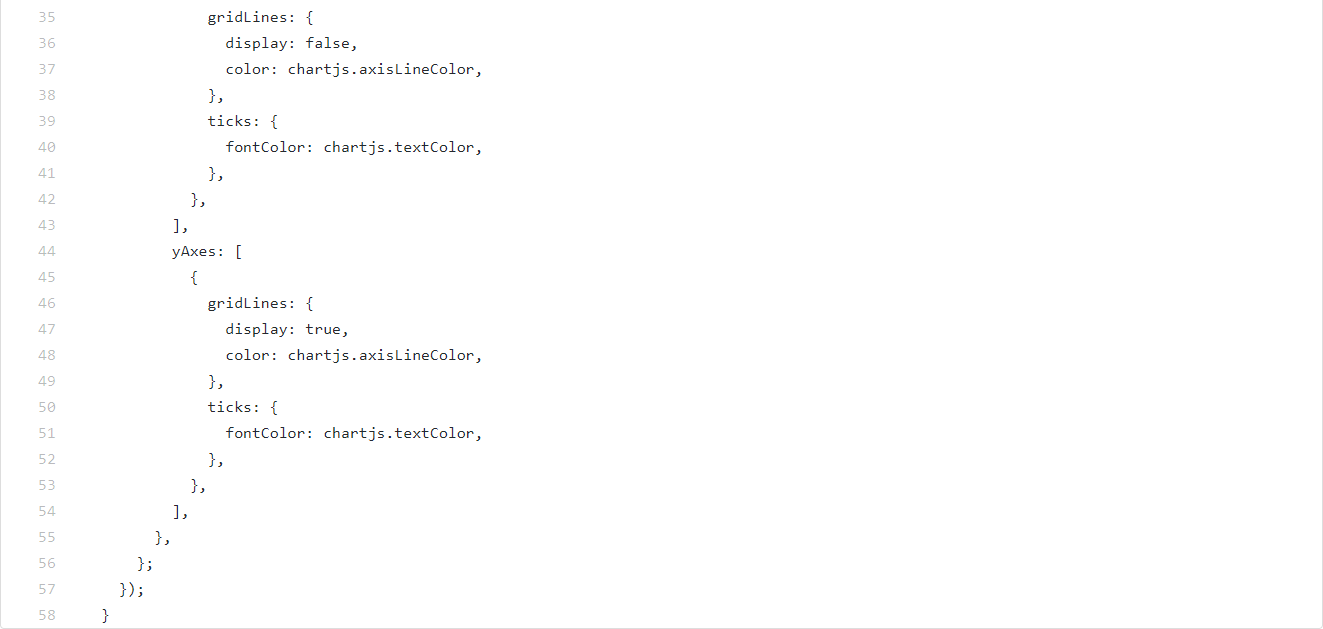
**INTERFAZ DE USUARIO**

****

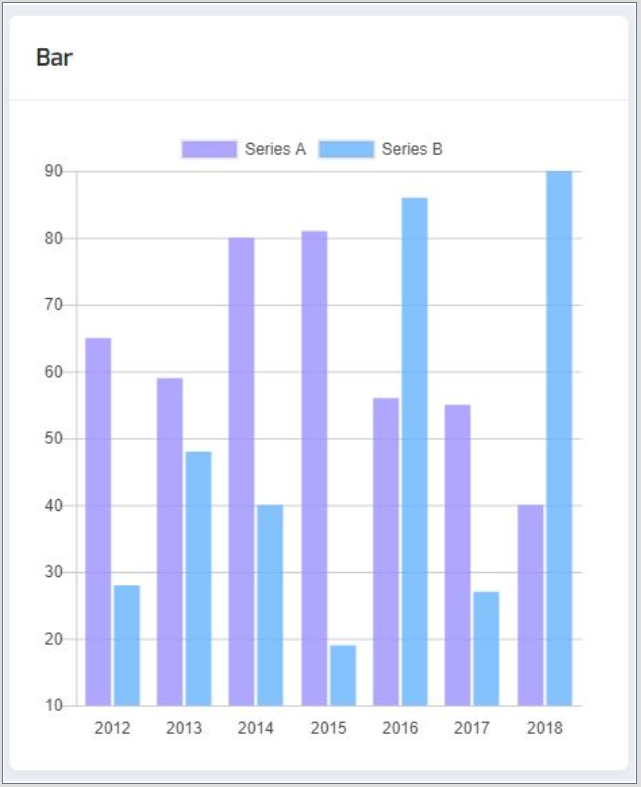
## 4.2 Documentación de construcción

A continuación, mostramos una captura de pantalla del código referente a la construcción del gráfico que hemos elegido para mostrar de Chartjs, que es el gráfico de barras:





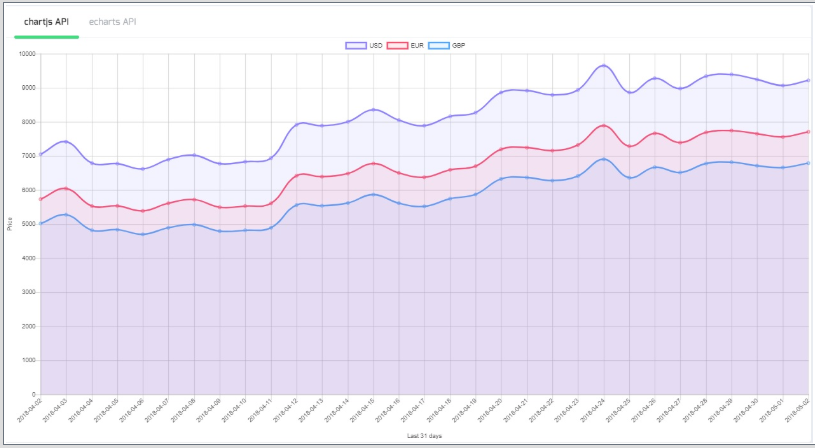
Y como resultado obtenemos:



Por otro lado, mostramos el código correspondiente a la realización del gráfico de líneas modo “live” que lee de una API REST los valores del Bitcoin diarios y los muestra en USD, EUR y GBP:



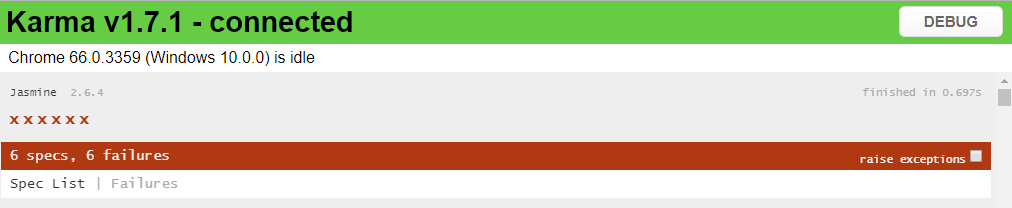
Y como resultado obtenemos:



## 4.3 Documentación de pruebas

Para comprobar de la mejor manera las pruebas, ejecutamos en la consola de Visual Studio el comando “ng llint”, el cual nos mostrará los errores de sintaxis en TypeScript que existen en nuestro código. Esto nos evitará que aparezcan errores relacionados con la sintaxis y nos centremos en los errores relacionados con los gráficos.

Tras ello, ejecutamos el comando “ng test”, el cual nos lanzará los test unitarios de cada componente del proyecto. El resultado del test será mostrado en una nueva pestaña de nuestro navegador por defecto:



Por otro lado, para testear funcionalidades complejas o flujos que engloban varios métodos, no basta con tests unitarios, sino que se hacen necesarios los tests de integración. Éstos permiten simular el uso de la aplicación y chequear distintos parámetros que se debiesen cumplir.

En nuestro caso, no es necesario generar test de integración, ya que no existen funcionalidades complejas en este módulo de chartjs y chartjs-live. Si tuviéramos, por ejemplo, la funcionalidad de login, podríamos crear la siguiente función dentro de su correspondiente archivo .spec.ts:

it('Should log in', () => {

browser.get('/login');

element(by.css('input[name=email]')).sendKeys('test@email.com');

element(by.css('input[name=password]')).sendKeys('testpassword');

element(by.css('button[name=submit]')).click();

expect(element(by.css('input[name=email]')).isPresent()).toBeFalsy();

});

Para llevar a cabo las pruebas, deberíamos ejecutar el comando “ng e2e” en la terminal de Visual Studio Code.

En cuanto a las pruebas realizadas a los gráficos de chartjs, podemos destacar que fue necesario declarar las variables de “data”, “options”, y demás variables necesarias dependiendo del gráfico, antes de rellenar los objetos necesarios en cada caso. De lo contrario, el gráfico no se mostrará en nuestro navegador. Por otro lado, en el momento de mostrar los datos de una API, debemos llamar a un servicio, el cual realiza la petición HTTP get. Pues bien, para que ello funcione correctamente, ha sido necesario rellenar el array de “providers” con el nombre del servicio dentro del componente de dicho gráfico.

@Component({

selector: 'chartjs-live',

templateUrl: './chartjs-live.component.html',

styleUrls: ['./chartjs-live.component.scss'],

providers: [LiveService,],

})

## 4.4 Documentación de instalación

Describiremos paso a paso el proceso para instalar el prototipo:

1. Instalar en la computadora Visual Studio Code como IDE.
2. Descargar Node.js.
3. Descargar la carpeta “charts-comparison” de <https://github.com/UrbanoJVR/TG3>.
4. Una vez descargada la carpeta anterior, abrirla con VS Code.
5. Ejecutar CTRL + Ñ para abrir la terminal en el IDE.
6. Ejecutar en la terminal el comando “npm install –g @angular/cli”.
7. Ejecutar en la terminal el comando “npm install -g” para instalar las dependencias node que necesita el proyecto.
8. Ejecutar en la termina el comando “ng serve -o” para que la aplicación corra en localhost y se abra directamente la web en el navegador por defecto.

## 4.5 Manual de usuario

Una vez realizados los pasos en el apartado 4.4, y la aplicación se muestre en el navegador que tengamos establecido por defecto en nuestra computadora, observaremos que la aplicación consta de un menú en la parte izquierda dispuesto con tres pestañas:

* Chartjs: nos redirige a los gráficos de Chartjs.
* Echarts: nos redirige a los gráficos de Echarts.
* Live: nos redirige a los gráficos de líneas utilizados para la representación de los datos del API REST sobre bitcoin.

Tanto con Chartjs como con Echarts hemos implementado los siguientes gráficos: gráficos tarta, gráfico de barras, gráfico de líneas, gráficos múltiples, gráfico de ejes-x, gráficos de barras horizontales y gráficos radar.

Adicionalmente Echarts, presenta dos tipos de gráficos más: gráficos de árbol y mapa de densidad; ya que Chartjs no los dispone.

Dentro de cada gráfico existe un botón referente a la leyenda del gráfico en cuestión, con el fin de activar o desactivar los datos que se desean mostrar.

Alguno de los gráficos, además de la leyenda posee un icono para guardar el gráfico en formato imagen.

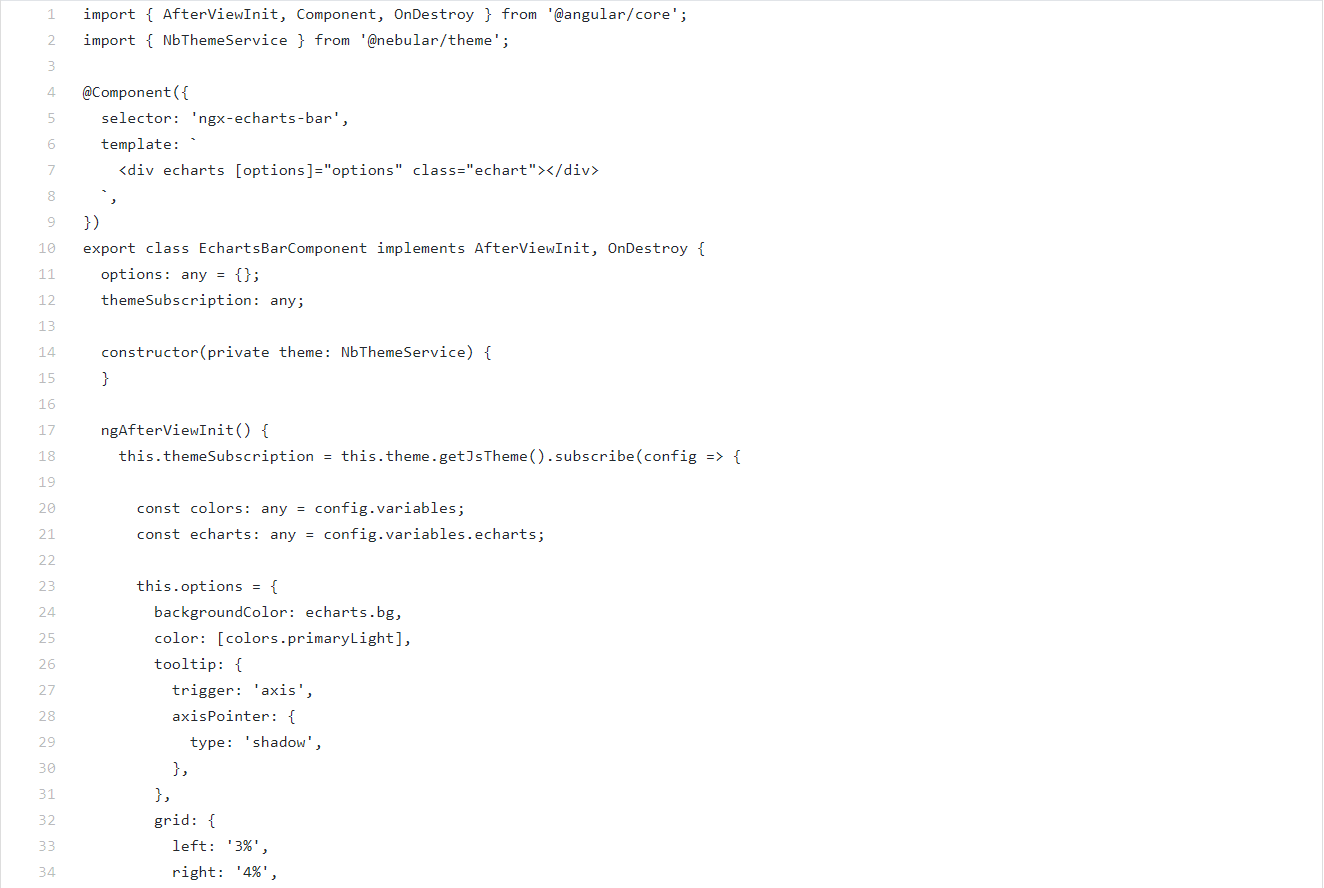
# 5. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Echarts

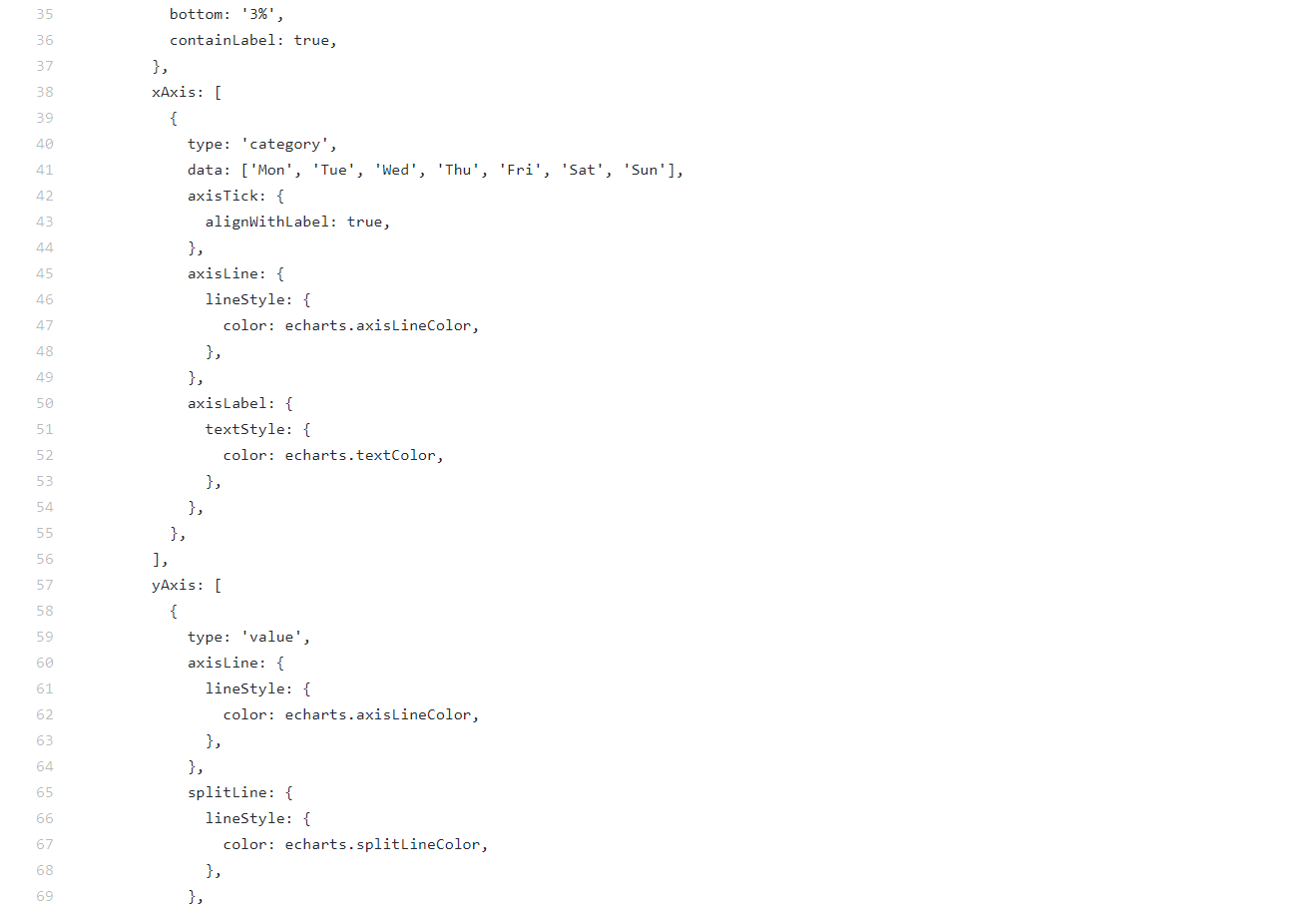
## 5.1 Documentación de diseño

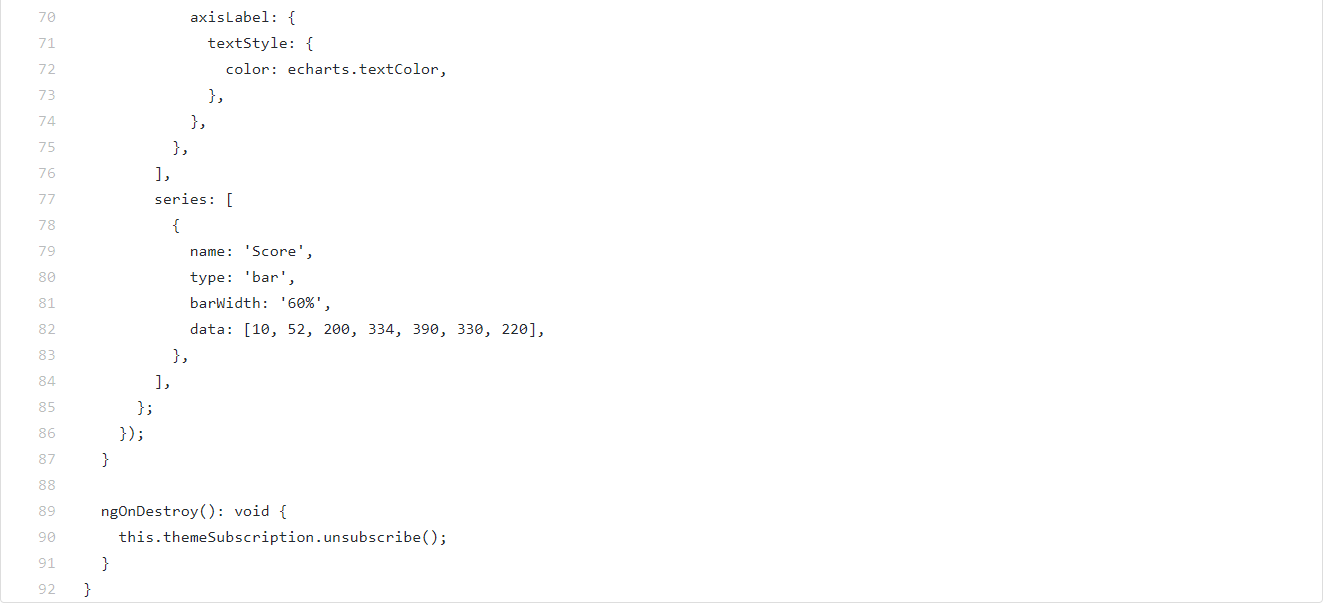
Como se ha comentado en el punto 4.1, se pretende desarrollar un entorno web para ver los distintos tipos de gráficos que se pueden implementar en la tecnología echarts, al tratarse de un entorno web en el que queremos implementar y comparar ambas tecnologías la descripción de diseño, diagramas e interfaz de usuario son los mismo que los ya comentados y diseñados para el punto 4.1

## 5.2 Documentación de construcción

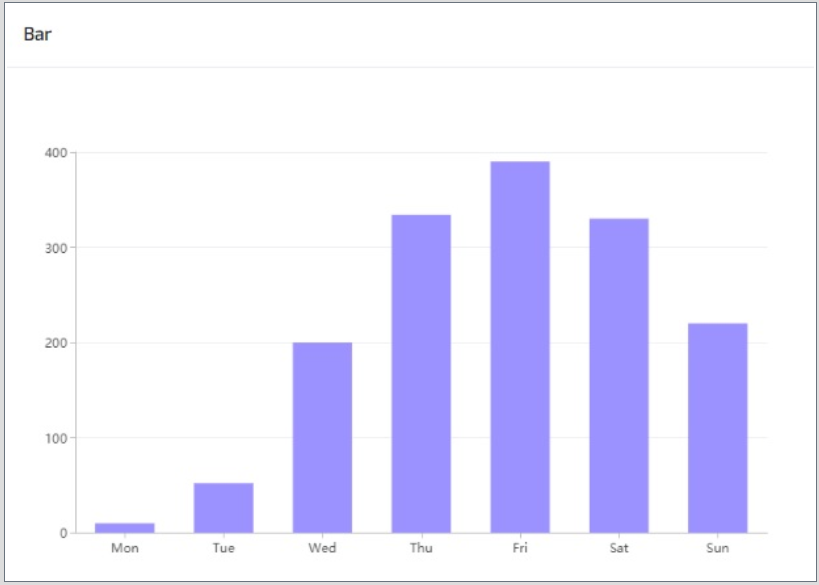
A continuación, mostramos una captura de pantalla del código referente a la construcción del gráfico que hemos elegido para mostrar de Echarts, que es el gráfico de barras:



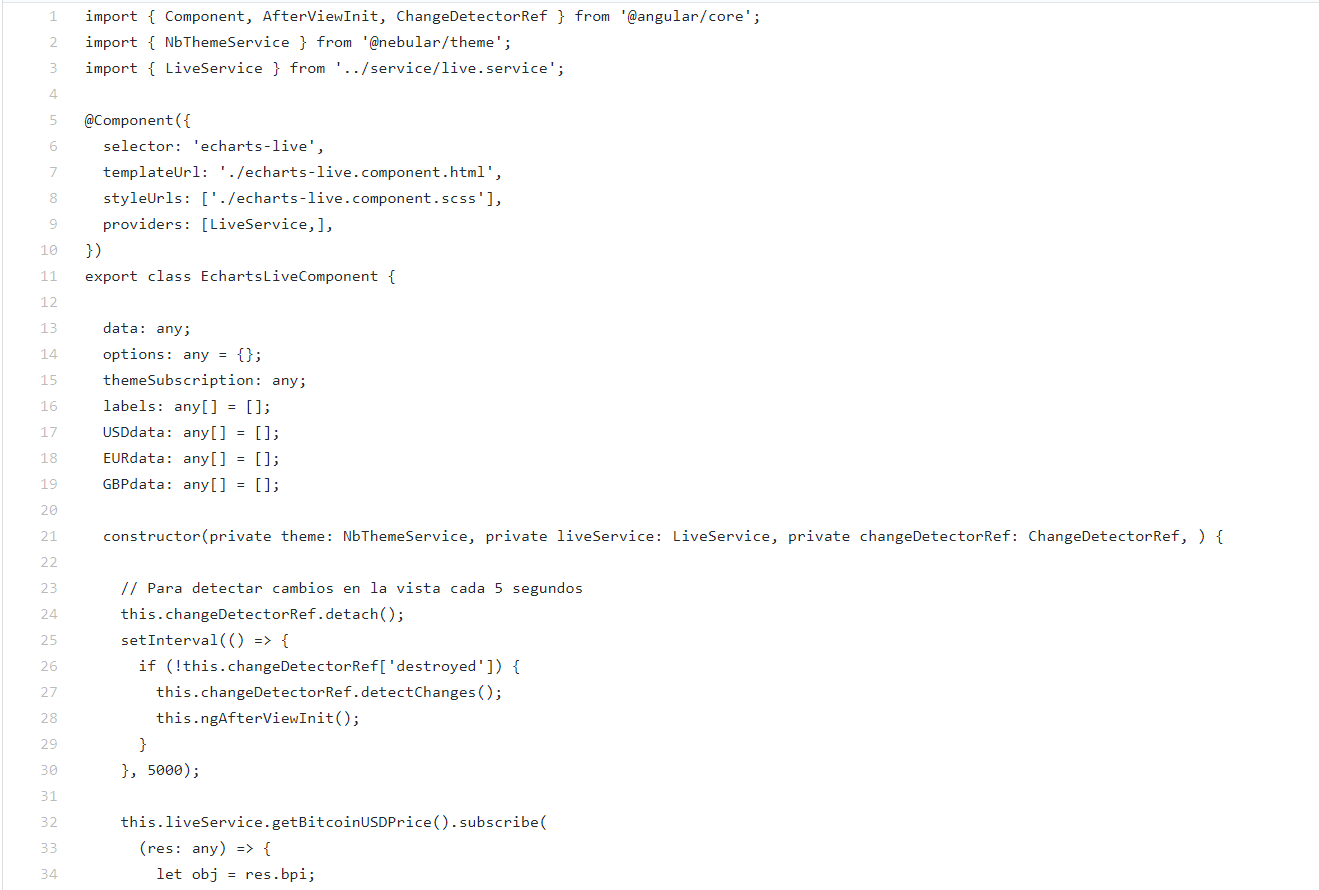


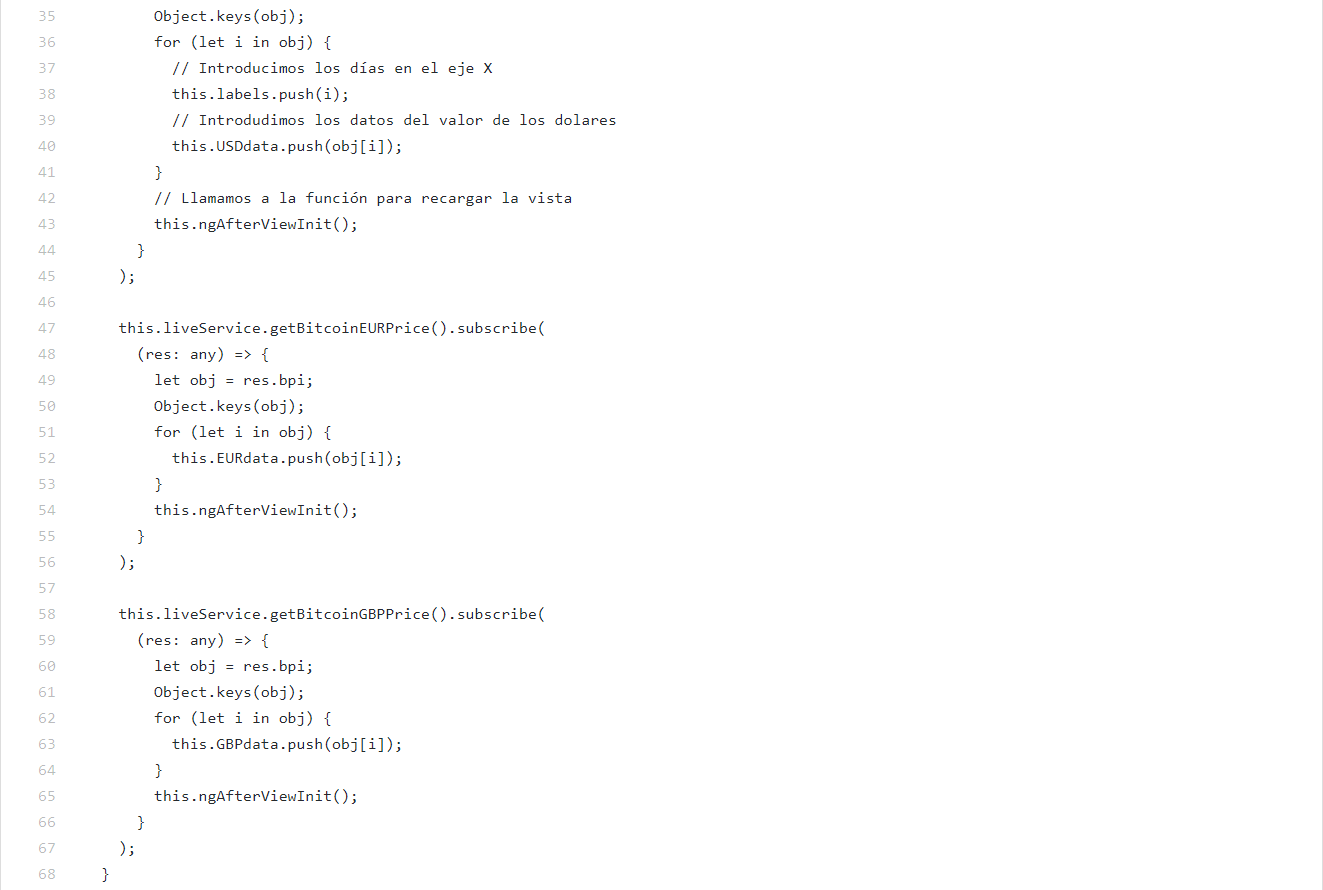


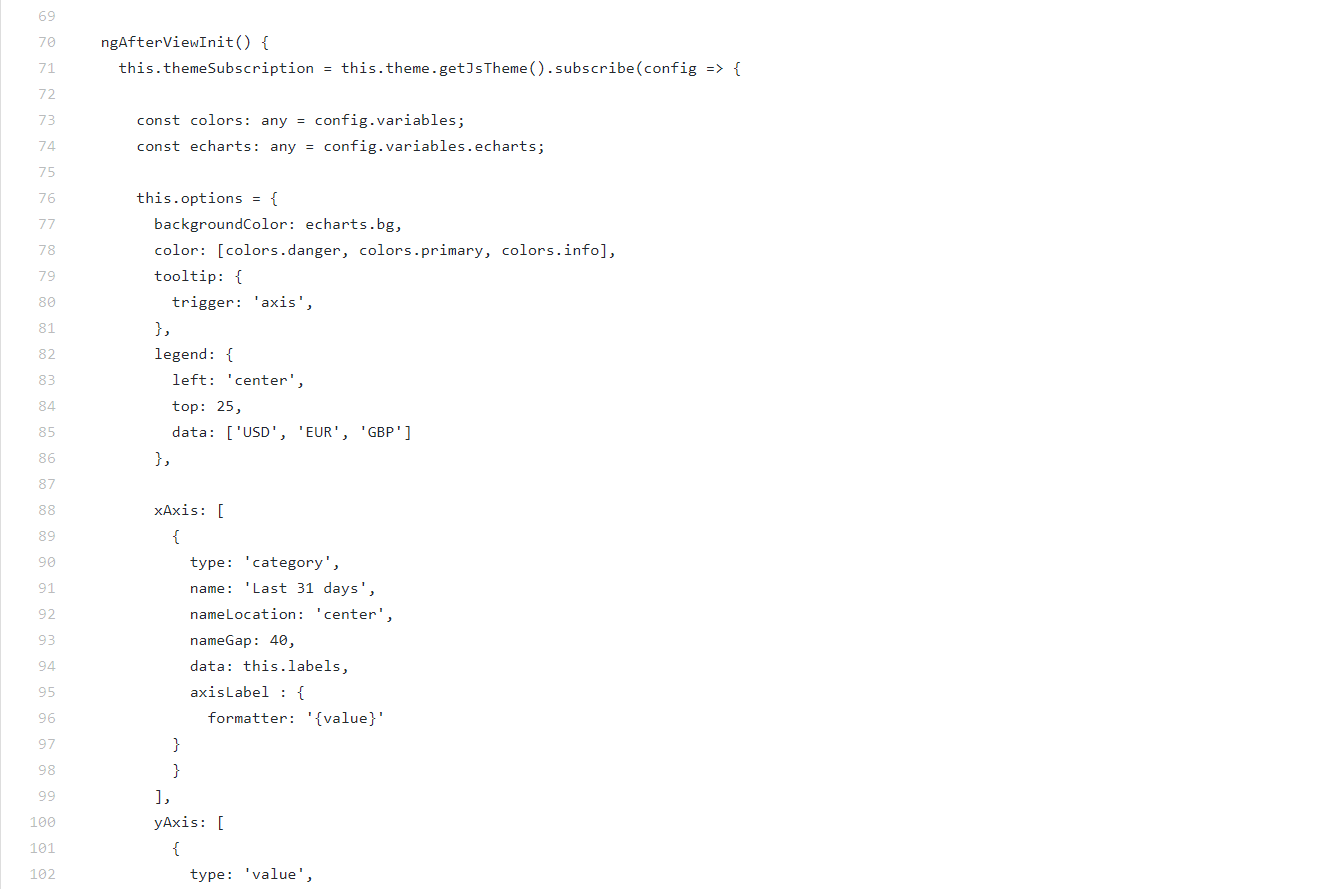
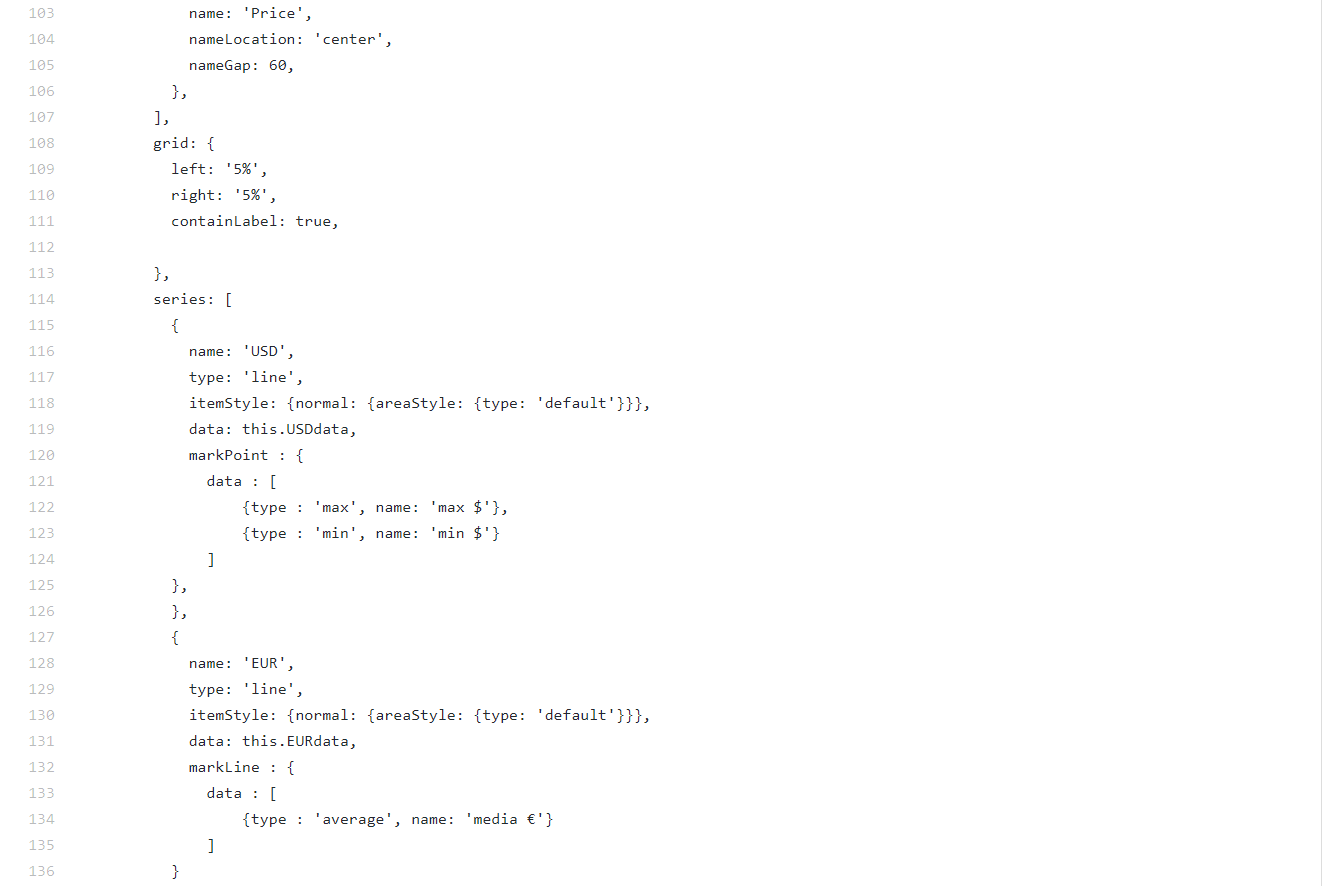
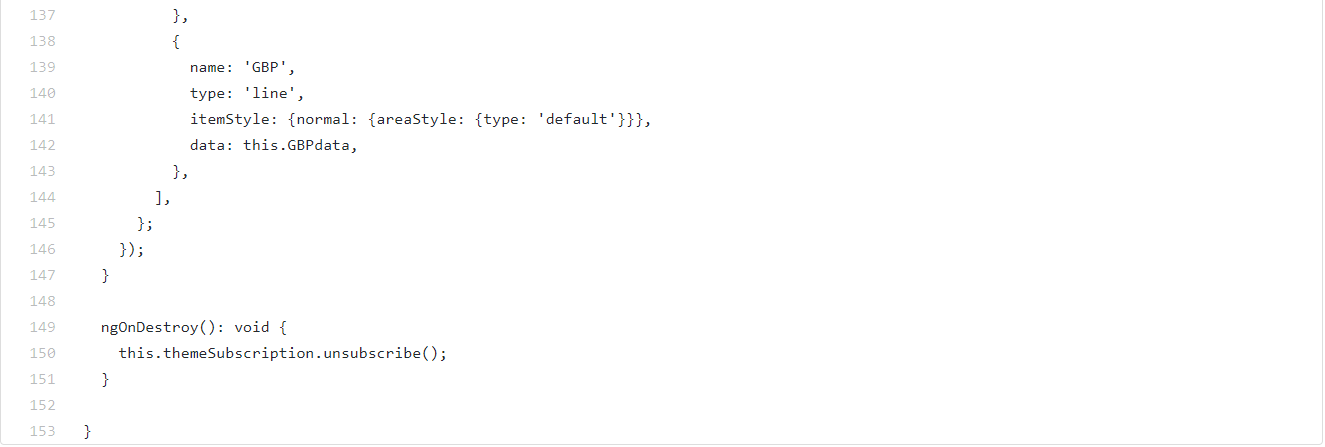
Y como resultado obtenemos:



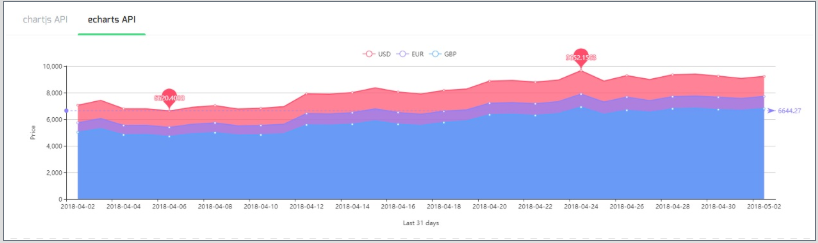
Por otro lado, mostramos el código correspondiente a la realización del gráfico de líneas modo “live” que lee de una API REST los valores del Bitcoin diarios y los muestra en USD, EUR y GBP:





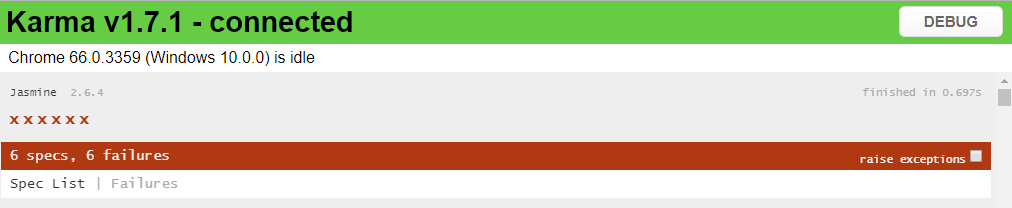
Y como resultado obtenemos:



## 5.3 Documentación de pruebas

De igual modo que con chartjs, para comprobar de la mejor manera las pruebas, ejecutamos en la consola de Visual Studio Code el comando “**ng llint**”, el cual nos mostrará los errores de sintaxis en TypeScript que existen en nuestro código. Esto nos evitará que aparezcan errores relacionados con la sintaxis y nos centremos en los errores relacionados con los gráficos.

Tras ello, ejecutamos el comando “**ng test**”, el cual nos lanzará los test unitarios de cada componente del proyecto. El resultado del test será mostrado en una nueva pestaña de nuestro navegador por defecto:



Por otro lado, para testear funcionalidades complejas o flujos que engloban varios métodos, no basta con tests unitarios, sino que se hacen necesarios los tests de integración. Éstos permiten simular el uso de la aplicación y chequear distintos parámetros que se debiesen cumplir.

En nuestro caso, no es necesario generar test de integración, ya que no existen funcionalidades complejas en este módulo de echarts y echarts-live. Si tuviéramos, por ejemplo, la funcionalidad de login, podríamos crear la siguiente función dentro de su correspondiente archivo .spec.ts:

it('Should log in', () => {

browser.get('/login');

element(by.css('input[name=email]')).sendKeys('test@email.com');

element(by.css('input[name=password]')).sendKeys('testpassword');

element(by.css('button[name=submit]')).click();

expect(element(by.css('input[name=email]')).isPresent()).toBeFalsy();

});

Para llevar a cabo las pruebas, deberíamos ejecutar el comando “**ng e2e**” en la terminal de Visual Studio Code.

En cuanto a las pruebas realizadas a los gráficos de chartjs, podemos destacar que fue necesario declarar las variables de “data”, “options”, y demás variables necesarias dependiendo del gráfico, antes de rellenar los objetos necesarios en cada caso. De lo contrario, el gráfico no se mostrará en nuestro navegador. Por otro lado, en el momento de mostrar los datos de una API, debemos llamar a un servicio, el cual realiza la petición HTTP get. Pues bien, para que ello funcione correctamente, ha sido necesario rellenar el array de “providers” con el nombre del servicio dentro del componente de dicho gráfico.

@Component({

selector: 'echarts-live',

templateUrl: './echarts-live.component.html',

styleUrls: ['./echarts-live.component.scss'],

providers: [LiveService,],

})

Además, tuvimos problemas con el refresco de los datos traídos de la API, ya que había ocasiones en las que no recargaba por defecto. Por ello, fue necesario implementar un “timer” que refresque el contenido del gráfico cada 5 segundos:

// Para detectar cambios en la vista cada 5 segundos

this.changeDetectorRef.detach();

setInterval(() => {

if (!this.changeDetectorRef['destroyed']) {

this.changeDetectorRef.detectChanges();

this.ngAfterViewInit();

}

}, 5000);

## 5.4 Documentación de instalación

La documentación de instalación del prototipo para Echarts es la misma que para Chartjs ya que se presentan conjuntamente en la misma aplicación:

Describiremos paso a paso el proceso para instalar el prototipo:

1. Instalar en la computadora Visual Studio Code como IDE.
2. Descargar Node.js.
3. Descargar la carpeta “charts-comparison” de <https://github.com/UrbanoJVR/TG3>.
4. Una vez descargada la carpeta anterior, abrirla con VS Code.
5. Ejecutar CTRL + Ñ para abrir la terminal en el IDE.
6. Ejecutar en la terminal el comando “npm install –g @angular/cli”.
7. Ejecutar en la terminal el comando “npm install -g” para instalar las dependencias node que necesita el proyecto.
8. Ejecutar en la termina el comando “ng serve -o” para que la aplicación corra en localhost y se abra directamente la web en el navegador por defecto.

## 5.5 Manual de usuario

El manual para el usuario es el mismo para Chartjs que para Echarts ya que ambas tecnologías se han utilizado para la creación de los mismos gráficos, y adicionalmente dos más de Echarts, en una misma aplicación.

Una vez realizados los pasos en el apartado 4.4, y la aplicación se muestre en el navegador que tengamos establecido por defecto en nuestra computadora, observaremos que la aplicación consta de un menú en la parte izquierda dispuesto con tres pestañas:

* Chartjs: nos redirige a los gráficos de Chartjs.
* Echarts: nos redirige a los gráficos de Echarts.
* Live: nos redirige a los gráficos de líneas utilizados para la representación de los datos del API REST sobre bitcoin.

Tanto con Chartjs como con Echarts hemos implementado los siguientes gráficos: gráficos tarta, gráfico de barras, gráfico de líneas, gráficos múltiples, gráfico de ejes-x, gráficos de barras horizontales y gráficos radar.

Adicionalmente Echarts, presenta dos tipos de gráficos más: gráficos de árbol y mapa de densidad; ya que Chartjs no los dispone.

Dentro de cada gráfico existe un botón referente a la leyenda del gráfico en cuestión, con el fin de activar o desactivar los datos que se desean mostrar.

Alguno de los gráficos, además de la leyenda posee un icono para guardar el gráfico en formato imagen.

# 6. Comparación de las dos implementaciones

## 6.1 Evaluación de los criterios en la implementación usando la tecnología Chartjs

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| Gráfico tarta | Sí |
| Gráfico de líneas | Sí |
| Gráfico múltiple eje x | Sí |
| Gráfico área stack | Sí |
| G. barras horizontales | Sí |
| Gráfico de tipo radar | Sí |
| G. Mapa de densidad | No |
| Gráfico de árbol | No |
| Gráfico de datos API | Sí |
| Líneas de Código | 618 |
| Horas de desarrollo | - |
| Velocidad de funcionamiento | Tiempo de carga web: 1.2s  Tiempo de respuesta: 0.2s |
| Almacenamiento para desarrollo \* | Gráficos ejemplo:16.5KB Gráfico API: 4.29KB |

(\*) Para el criterio de almacenamiento de desarrollo se ha excluido la parte común para ambas tecnologías que es la parte más pesada de la implementación con unos 430MB aproximadamente.

## 6.2 Evaluación de los criterios en la implementación usando la tecnología Echarts

| **CRITERIO** | **EVALUACIÓN** | |
| --- | --- | --- |
| Gráfico tarta | | Sí |
| Gráfico de líneas | | Sí |
| Gráfico múltiple eje x | | Sí |
| Gráfico área stack | | Sí |
| G. barras horizontales | | Sí |
| Gráfico de tipo radar | | Sí |
| G. Mapa de densidad | | Sí |
| Gráfico de árbol | | Sí |
| Gráfico de datos API | | Sí |
| Líneas de Código | | 1318 |
| Horas de desarrollo | | - |
| Velocidad de funcionamiento | | Tiempo de carga web: 1.4s  Tiempo de respuesta: 0.3s |
| Almacenamiento para desarrollo \* | | Gráficos ejemplo:36.8KB Gráfico API:4.66KB |

(\*) Para el criterio de almacenamiento de desarrollo se ha excluido la parte común para ambas tecnologías que es la parte más pesada de la implementación con unos 430MB aproximadamente.

# 7. Comparación de la implementación de las tecnologías

| **CRITERIOS** | **CHARTJS** | **ECHARTS** | **COMENTARIOS** |
| --- | --- | --- | --- |
| Gráfico tarta | Sí | Sí | Ambas tecnologías presentan la posibilidad de implementar el gráfico tarta/pie |
| Gráfico de líneas | Sí | Sí | Ambas tecnologías presentan la posibilidad de implementar el gráfico de líneas |
| Gráfico múltiple eje x | Sí | Sí | Ambas tecnologías presentan la posibilidad de implementar el gráfico múltiple eje-x |
| Gráfico área stack | Sí | Sí | Ambas tecnologías presentan la posibilidad de implementar el gráfico área stack |
| G. barras horizontales | Sí | Sí | Ambas tecnologías presentan la posibilidad de implementar el gráfico barras horizontales |
| Gráfico de tipo radar | Sí | Sí | Ambas tecnologías presentan la posibilidad de implementar el gráfico tipo radar |
| G. Mapa de densidad | No | Sí | Solamente Echarts presenta la posibilidad de implementar el gráfico mapa de densidad |
| Gráfico de árbol | No | Sí | Solamente Echarts presenta la posibilidad de implementar el gráfico de árbol |
| Gráfico de datos API | Sí | Sí | Ambas tecnologías presentan la posibilidad de implementar el gráfico de líneas con datos del API REST de Bitcoin |
| Líneas de Código | 618 | 1318 | Se aprecia que Echarts requiere de muchas más líneas de código que Chartjs para la creación de gráficos |
| Horas de desarrollo | - | - | Se aprecia que Echarts requiere de más horas para el desarrollo a causa de la mucha menos cantidad de documentación disponible y su más alta complejidad |
| Velocidad de funcionamiento | Tiempo de carga web: 1.3s  Tiempo de respuesta: 0.2s | Tiempo de carga web: 1.4s  Tiempo de respuesta: 0.3s | Se han medido los tiempos de carga para la página de cada modelo, así como una aproximación del tiempo de respuesta en estos gráficos. |
| Almacenamiento para desarrollo | Gráficos ejemplo:16.5KB Gráfico API: 4.29KB | Gráficos ejemplo:36.8KB Gráfico API:4.66KB | Al igual que en el caso de las líneas de código y las horas de desarrollo, el espacio de almacenamiento en disco es mayor para echarts que para chartjs. En parte porque tiene más tipos de gráficos a representar y en otra parte porque a igualdad de gráfico siempre emplea más LoC. |

# 8. Conclusiones

A partir de los desarrollos realizados para ambas tecnologías podemos concluir que la tecnología echarts, es una tecnología que permite realizar un mayor número de gráficos diferentes, y dotar de más funcionalidades a estos gráficos que la tecnología chartjs, pero a pesar de esta funcionalidad mayor, también podemos comprobar que es una tecnología mucho más complicada de implementar, ya que se requiere una mayor experiencia en el entorno de programación, ya que la documentación está menos accesible y solamente en Chino e Inglés, mientras que en chartjs sí que podemos encontrar gran cantidad de información en castellano, además echarts nos va a llevar un mayor tiempo de desarrollo a igualdad de experiencia de los programadores.

Por lo tanto, después de nuestra experiencia y a la hora de decantarnos por una tecnología u otra sería necesario estudiar bien qué tipo de proyecto queremos realizar, para saber el tipo de gráfico que necesitaremos, si este no está disponible en chartjs, sabemos que nos tendremos que decantar por echarts, pero también debemos de saber que el tiempo para realizar el proyecto no será el mismo, así como el personal que debamos contratar tampoco, y deberá ser un personal más cualificado.

Por otro lado, si tenemos impuesto el personal o el tiempo de proyecto de forma que no nos vaya a ser posible desarrollarlo con echarts por su necesidad de mayor de dedicación y experiencia, tendremos que tener claro que habrá ciertos tipos de gráficos que no tengamos permitido hacer.

Y por último en estas conclusiones, no incluimos como un factor diferenciador el almacenamiento necesario para desarrollo o la velocidad de carga de los gráficos, ya que con los valores obtenidos consideramos irrelevante los mismos, puesto que cualquier pc en la actualidad es capaz de cumplir con los requisitos de almacenamiento, así como los de velocidad de carga en función del procesador, y cualquier línea de conexión a internet también puede asumir las velocidades de carga de ambas tecnologías una vez desplegadas en la web.