# Bloomen
# Blockchain for Creative Content

Sergi Falcón Xarau

1 de juliol de 2019

**Resumen –** El objetivo de mi proyecto final de grado es el desarrollo de una aplicación híbrida[21] con Angular [13] para un cliente final, orientada al consumo y venta de contenido digital.
La venta de este se realizará a través de un Marketplace en el que tanto proveedores como clientes pueden gestionar la compra y venta del contenido, sin la necesidad de pasarelas de pago como Visa o PayPal debido a la tecnología Blockchain [9] y el uso de las criptomonedas.
Estos proveedores/usuarios también podrán transferirse criptomonedas entre ellos, extraer sus cuentas y reutilizarlas en otras aplicaciones del Marketplace por si desean una centralizar sus pagos en una o dividirlas.
La aplicación está dotada de una arquitectura descentralizada (Dapp [10], pronunciada Di-app) que funciona a través de Smart Contracts. Esto significa que no depende de un sistema central (sin middleware que pongan en duda la seguridad de los pagos) sí no de los usuarios que la utilizan; afectando principalmente al Back-End de la aplicación.
Este es un proyecto/producto ofrecido por la empresa Wordline S.A. bajo la supervisión técnica de un tutor de prácticas y en colaboración con mi compañero Eric Torres Perramon quien será el desarrollador del Back-End de la aplicación en su propio TFG. En el caso de mi propuesta es en su totalidad el Front-End de esta.

**Palabras clave –** Blockchain – Criptomonedas – Alastria – Angular – TypeScript – Responsive – Marketplace – SASS – NgRx – HTML – CSS – Front-End – Back-End

**Abstract –** The objective of my final degree project is the development of a hybrid application for a final customer, aimed at the consumption and sale of digital content.
The sale of this will be done through a Marketplace in which both suppliers and customers can manage the purchase and sale of content, without the need for payment gateways such as Visa or PayPal due to Blockchain[9] technology and the use of cryptocurrencies.
These providers / users may also transfer cryptocurrencies among them, extract their accounts and reuse them in other applications of the Marketplace in case you want to centralize your payments in one or divide them.
The application is equipped with a decentralized architecture (Dapp[10], pronounced Di-app) that works through Smart Contracts. This means that it does not depend on a central system (without middleware that calls into question the security of the payments) if not on the users that use it; affecting mainly the Back-End of the application.
This is a project / product offered by Wordline S.A. under the technical supervision of a tutor of practices and in collaboration with my partner Eric Torres Perramon who will be the developer of the Back-End of the application in his own TFG. In the case of my proposal is the Front-End part of this.

**Keywords –** Blockchain – Criptocoins – Alastria – Angular – TypeScript – Responsive – Marketplace – SASS – NgRx – HTML – CSS – Front-End – Back-End

## 1 INTRODUCCIÓN

• E-mail de contacte: sergifalcon93@gmail.com
• Mención realizada: Ingeniería del Software
• Trabajo tutorizado por: Lluís Gesa Bote (Área de Ciencias de la Computación e Inteligencia Artificial)
• Curs 2018/19

EL objetivo principal de este proyecto, es mostrar el aprendizaje y desarrollo de una aplicación híbrida con Angular descentralizada en la que proveedores

de contenido digital puedan tener sus propios canales/tiendas en un Marketplace, en el que los usuarios podrán pagar con criptomonedas para comprar su contenido.

El Back-End de la aplicación lo explicaré brevemente ya que no es mi parte asignada del proyecto, pero es esencial su comprensión para poder llevar a cabo toda la lógica y el desarrollo del Front-End. Este funciona con SmartContracts, tecnología de Blockchain [9] desarrollada por Ethereum [11], se trata de una arquitectura descentralizada que mantiene toda la información en las cadenas de bloques haciendo irrefutable e inalterable.

Mediante las librerías de Ethereum y TypeScript [12] se despertarán los contratos y se podrá trabajar con ellos lanzando cualquier movimiento de transacción; consultar saldo, pagar, ingresar, transferir. A demás el funcionamiento de las Dapps permite que el usuario no necesite registrarse en ningún momento, al lanzar la primera ejecución de la aplicación se le generará un hash único que le identificará y deberá guardar por si desea mantener a salvo su cuenta en un futuro, reutilizarla en cualquier otro canal o simplemente si elimina la memoria local de la aplicación de su dispositivo. Los creadores de contenido dispondrán de una pequeña herramienta de comandos para gestionar la creación de su canal o canales.

### 1.1 Motivación

Principalmente la motivación de este proyecto a surgido con el nacimiento de las nuevas tecnologías de Blockchain y las criptomonedas. No solo se pretende demostrar que estas tienen mas enfoques que el Trading o Valores de mercado, si no que además su uso nos da una nueva oportunidad a la sociedad de poder gestionar dinero, transferir, pagar o recibir sin necesidad de terceros (bancos o entidades) que enmascaren los procesos. Otra motivación es la retribución a artistas, escritores, fotógrafos, creadores de contenido etc. que hasta hoy en día se han Patreon visto afectados por la piratería o la distribución ilegal.

## 2 OBJETIVOS

- FRONT-01: Recoger todos los requisitos necesarios en el Front-End de la aplicación.
- FRONT-02: Diseñar un esbozo con PowerPoint de la interfaz.
  - FRONT-02-01: Recoger todas las imágenes/iconos generados por el equipo de UX/UI.
  - FRONT-02-02: Generar dos temas tanto en claro como oscuro.
- FRONT-03: Traducción de textos. Generar un con Google Drive para todos los idiomas oficiales de la UE (alemán, danés, griego, español, portugués, sueco, checo, búlgaro y croata. + catalán).
- FRONT-04: Generar un tutorial con pasos que podrán omitir si ya conocen el funcionamiento.

### 2.1 Planificación

| Entrega | Fecha Inicio | Fecha Final |
|---|---|---|
| Informe inicial | 17/02/2019 | 10/03/2019 |
| Borrador progreso 1 | 10/03/2019 | 1/04/2019 |
| Informe progreso 1 | 1/04/2019 | 14/04/2019 |
| Borrador progreso 2 | 14/04/2019 | 10/05/2019 |
| Informe progreso 2 | 10/05/2019 | 26/05/2019 |
| Borrador propuesta final | 26/05/2019 | 5/06/2019 |
| Propuesta informe final | 5/06/2019 | 16/06/2019 |
| Propuesta presentación | 16/06/2019 | 30/06/2019 |
| Borrador Entrega final | 16/06/2019 | 22/06/2019 |
| Entrega Final | 30/06/2019 | 30/06/2019 |

- FRONT-05: Generar una semilla de Angular con todas las librerías y tecnologías necesarias.
  - FRONT-05-01: Generar una estructura de eventos con NgRx.
  - FRONT-05-02: Generar el “routing” entre componentes.
  - FRONT-05-03: Encapsular componentes hijos en los componentes padre para hacer una estructura multicapa.
  - FRONT-05-4: Generar estructura de temas en las hojas de estilos.
- FRONT-06: Implementar todo el Front-End diseñado y especificado.
  - FRONT-06-01: Generar las funcionalidades visuales con sus respectivas estructuras de datos.
  - FRONT-06-02: Introducir Mocks para las funcionalidades.
  - FRONT-06-03: Desarrollar “Loadings” para los eventos con espera.
  - FRONT-06-04: Implementar buses de eventos con NgRx.
  - FRONT-06-05: Buscar el “pixel perfect [14]” en las hojas de estilos.
  - FRONT-06-06: Asegurar en cada iteración una buena performance.
  - FRONT-06-07: Generar versiones de testing para todos los dispositivos con Cordova (12).
  - FRONT-06-08: Comprobar errores visuales y de datos en las versiones de Android y iOS.
- FRONT-07: Integrar las funcionalidades. Retirar los Mocks y usar las funciones generadas por el departamento de Back-End.
  - FRONT-07-01: Solucionar bugs generados por las funciones de los Smart-Contracts y coordinarse con el equipo de Back-End.
- FRONT-08: User testing + Performance testing.

# 3 HERRAMIENTAS Y METODOLOGÍA

La metodología utilizada en este proyecto es Kanban. Era la escogida por la empresa del proyecto, decidida por nuestros superiores junto a Integración Continua. Con el uso de la herramienta Trello podemos generar cada una de las tareas siendo lo más explícito que busquemos según su nivel de importancia, podemos añadir atajos y pistas sobre los ficheros que hay que actuar, integrantes y comentarios. Cada una de estas tareas tendrá un ciclo de vida y que deberá respetar su fecha limite de finalización. A parte cada uno de estos estarán divididos por tablones según el departamento (Front-End, Back-End, UX/UI etc...) que a su vez se irán subdividiendo por iteraciones.

El ciclo de vida de estas tareas es el siguiente:

- Lista de tareas: listado de todas las tareas que le toca al departamento en esa misma iteración.
- En proceso: las tareas que estoy realizando en ese mismo momento.
- Preparadas para validar: tareas que creo que he finalizado y necesitan una supervisión para comprobar su validez.
- Realizadas: tareas que ya se han dado por concluidas y completadas.

## 3.1 Metodología

Kanban nos limita la longitud de tareas, ayuda a la hora de que cualquiera de los dos departamentos tenga complicaciones desarrollando una funcionalidad y tenga que cambiar un método que afecte a los Mockups de Front-End o a los servicios generados por el Back-End.

Nuestros superiores pueden supervisar el flujo de tarjetas, los cambios continuos, graduales y evolutivos para sopesar si pueden tener efectos negativos.

Da la posibilidad de integrar personal de ayuda en casos de emergencia y la gran modulación del trabajo le permitiría desarrollar funcionalidades sin tener que entender todo el código fuente ni acceder a toda la documentación.

También puede ser un arma de doble filo, la motivación que lleva a cabo ver cientos de tarjetas cumplidas y validadas pueden enorgullecer al programador tanto como desmotivar cientos mas por hacer, por ello se deben generar en cada una de las fases y no de todo el proyecto en una sola vez.

## 3.2 Herramientas

A continuación, se listarán las herramientas utilizadas a lo largo del proyecto, tanto para documentación, repositorios y gestión de tareas.

- Trello[16]: gestor de tareas por tickets distribuidos por iteraciones, que a su vez se distribuyen por departamento y luego por componentes. Muy intuitiva.
- Github[6]: repositorio en el que se localizará el proyecto de manera Open-Source.
- Sourcetree[18]: GUI con el que podremos controlar los cambios que van subiendo, no solo yo como desarrollador de Front-End si no también mi compañero como Back-End.
- Microsoft Office 365[19]: es el paquete de herramientas más utilizado para la documentación de proyectos.
- Visual Code Studio[20]: esta herramienta es un IDE que nos permite desarrollar el proyecto, desde lanzar la aplicación en local con un servidor Node, hasta editar las hojas de estilos o programar el código de los componentes. Es el IDE mas recomendado para Angular
- Google Chrome DevTools:[17] una de las herramientas más importantes son las de desarrollador que nos provee Google Chrome hacer debug en nuestro lenguaje de etiquetas, hojas de estilos y componentes de TypeScripts transpilados de AngularAngular.
- Redux [8] DevTools: extensión de navegador muy necesaria para hacer debug del flujo de eventos de las librerías NgRx en el que podemos apreciar el estado de estos en todo momento con un desplegable esquemático.
- Invision[15]: sitio web en el que los desarrolladores de UX/UI subirán los assets diseñados para ser colocados en los botones/iconos/backgrounds correspondientes.

# 4 ESTADO DEL ARTE

La tecnología de Smart Contracts es algo muy novedoso en el mundo del software y hoy en día todas las aplicaciones famosas de Criptomonedas son de Cryptocurrency (trading de criptomonedas como bitcoins). A demás de ser un proyecto innovador propuesto por Bloomen a la consultora Worldline, este se lanza a las conferencias Europeas sobre Blockchain como producto innovador.
Seguramente todo lo que podamos encontrar sea pequeñas demostraciones en los repositorios de GitHub en los que se demuestran el funcionamiento de los Smart Contracts. En mi caso toda mi 'inspiración' proviene de las vistas y Mockups propuestos por el equipo de UX/UI.

# 5 REQUISITOS

## 5.1 Requisitos Funcionales

A continuación, tenemos el recogido de los requisitos funcionales principales del Front-End de nuestra aplicación, estos están recogidos con el resto de requisitos.

- FRONT-F-01: el usuario ha de poder acceder a la cámara del terminal para poder escanear códigos QR.
- FRONT-F-04: el usuario ha de poder visualizar una lista con todos los tipos de transacciones generadas en dicha cuenta.

- FRONT-F-05: el usuario ha de poder seleccionar un contenido digital y adquirirlo en su cuenta a cambio de criptomonedas.
- FRONT-F-06: el usuario ha de poder transferir criptomonedas a otros usuarios.
- FRONT-F-08: el usuario ha de ser poder de visualizar una lista de productos
- FRONT-F-11: el usuario ha de poder visualizar en todo momento su saldo en criptomonedas.
- FRONT-F-12: el usuario ha de poder generar una cuenta nueva si entra por primera vez en una Dapp.
- FRONT-F-13: el usuario ha de poder reutilizar una cuenta existente de otra Dapp al entrar por primera vez en una.
- FRONT-F-17: el usuario ha de poder eliminar todos sus datos en referencia a una Dapp.

## 5.2 Requisitos No Funcionales

- FRONT-NF-01: Finalizar el proyecto final de grado en 300h (fecha limite 30 junio de 2019)
- FRONT-NF-02: La aplicación debe tener una apariencia visual “user friendly”.
- FRONT-NF-03: La aplicación debe ser responsiva para todos los terminales.
- FRONT-NF-04: La aplicación debe tener un tema claro y otro oscuro.
- FRONT-NF-06: La arquitectura del Front-End debe ser desarrollada con Angular, utilizando HTML, CSS con SASS, y TypeScript.
- FRONT-NF-07: El tiempo de carga de aplicación al abrirse debe ser un máximo de 2500ms en dispositivos con Android v8 en adelante e iOS v9 en adelante.
- FRONT-NF-13: La interfaz deberá tener una medía entre 5 y 10 minutos de aprendizaje según la experiencia del usuario.

# 6 Diseño

## 6.1 Actores

En esta aplicación los únicos actores posibles son los Usuarios/Clientes quienes tendrán la interacción con el todo el MarketPlace de Dapps, la compra de contenido digital, la visualización de este y las transacciones.

## 6.2 Casos de uso

Una vez hemos evaluado todas las funcionalidades de los usuarios básicos que regentaran el MarketPlace y sus Dapps, hacemos un diagrama de casos de uso con todas las opciones de las que dispone.

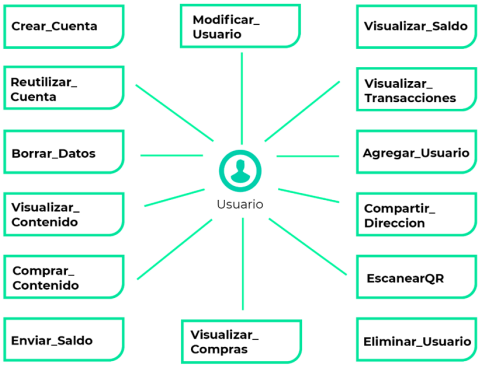


Fig. 1: Diagrama Casos de uso

## 6.3 Paginas

Decidimos basarnos en paginas y componentes ya es la arquitectura en aplicaciones híbridas más conocida y utilizada. El concepto de página en esta aplicación es básicamente el de una vista principal que puede contener diversos módulos o componentes. Por lo habitual un componente puede ser otra vista o un trozo de esta, para una mejor organización división de los datos. En cambio, un modulo por norma acostumbra a ser una herramienta (en muchos casos compartida en toda la aplicación) que tiene un uso especifico visual o a nivel interno.

### 6.3.1 Language-Selector:

En primer lugar, nada más arrancar la aplicación por primera vez lo primero que visualizaremos será la pagina para escoger el lenguaje cómo podemos ver en la figura.


Fig. 2: Selector de idiomas
Primera pantalla de la aplicació

### 6.3.2 Tutorial

Una vez se ha escogido el lenguaje y se haya almacenado en la memoria local del dispositivo se presentará la vista del tutorial que dispone de un slider informativo con los conceptos básicos, este también quedará registrado para no tener que volver a realizarlo. Ver fig. 12

### 6.3.3 Home

Quizá la segunda pagina mas importante de toda la aplicación. También hemos hablado de ella como Marketplace, lugar en el que se publicaran todas las aplicaciones registradas en nuestra cuenta. Esta cuenta únicamente de una lista de componentes que cada uno de ellos hará referencia a las Dapps pertinentes de cada autor o compañía. A demás de un botón de QR flotante que nos permite agregar nuevas Dapps con solo escanearlas.

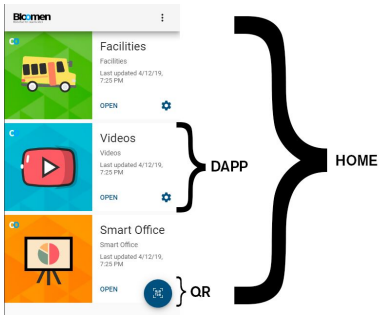


Fig. 3: Home
Pantalla principal de la aplicación

### 6.3.4 Dapp-Sections

Pantalla principal dentro de cualquier Dapp nada mas entrar registrado en ella, una vez tengamos un contrato con esa aplicación y podamos entrar lo primero que veremos será la encapsulación visual de cada una de sus secciones en una barra de navegación inferior que nos permite desplazarnos por todas sus secciones:

- Home (No es la Home del Marketplace, si no la de cualquier Dapp)
- SendCash
- ShoppingList

A su vez también disponemos de una barra en la que nos muestra nuestros créditos disponibles para esa aplicación, un botón en esa misma para añadir mas y en la barra superior otro botón de opciones extra para salir o refrescar.


Fig. 4: Dapp-Sections
Desglose de una Dapp

### 6.3.5 Send-Cash

Esta pagina se accede previamente con un componente encapsulado en dentro de la Dapp-Sections, una vez este detecta que tenemos más de 0 puntos nos permite acceder a esta pagina.
En esta disponemos de 2 campos, uno para añadir una dirección a la que queremos enviar puntos y otro para la cantidad de puntos. El campo de la dirección también trae un botón que nos permite escanear un QR para traducir la dirección a la que queremos enviar.
Como podemos apreciar la diferencia entre un componente que queda encapsulado dentro de una pagina como sería Dapp-Sections (en la sección de Send-Cash) y una pagina como Send-Cash que no dispone de la barra de navegación inferior ya que es una pagina en su totalidad.

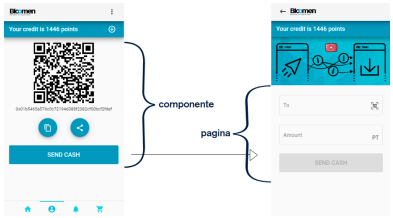


Fig. 5: Send-Cash
Envio de puntos de un usuario a otro

## 6.4 Componentes

Una vez mostradas las paginas mas relevantes y comprender el sistema de encapsulado de componentes y vista de paginas pasamos a ver los componentes mas importantes.

### 6.4.1 Dapp-Home

Es el primer componente que visualizaremos en la Dapp-Section, esta presenta las 2 opciones principales de cada Dapp que es la de comprar contenido a través de leer un QR o permitir la visualización del contenido gracias y una lista de todos los movimientos de créditos en la cuenta en formato lista. La lista tiene una estructura pensada para que únicamente haga scroll esa sección y el resto de la vista permanezca fija.

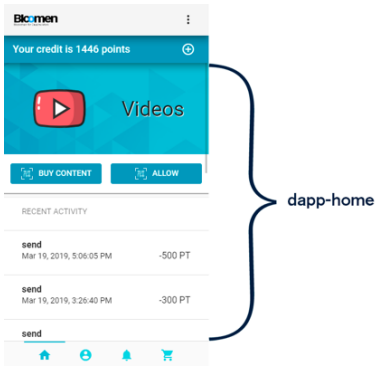


Fig. 6: Dapp-Home
Pantalla principal de una Dapp

### 6.4.2 Dapp-Credit-Header

La barra que llevamos viendo todo el rato en la zona superior de cada pagina es un componente que siempre está presente en estas y nos permite visualizar el saldo.


Fig. 7: Send-Credit-Header
Visor de puntos

### 6.4.3 Dapp-Login

Cuando accedemos por primera vez a una Dapp sin habernos registrado nos aparecerá un modal de altura y anchura completa (sabemos que es un modal por la cruceta superior a la izquierda). Este nos permite 2 opciones, crear una cuenta nueva y almacenando en la memoria local o restaurar una a través de un código de traspaso.

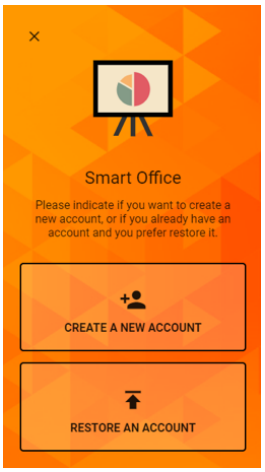


Fig. 8: Dapp-Login
Primera pantalla de una Dapp

### 6.4.4 Dapp-scanQR

Este componente se diseñó para darle más intuición a la compra de puntos, para acceder a el basta con un clic al botón + que nos encontramos en la barra de puntos. Su función es llevarnos hasta el scanner de QR's. Podemos escanear códigos de prueba para adquirir puntos en el siguiente enlace:
https://alastria1.bloomen.io/bloo-demo/#/bloomen-cards

### 6.4.5 Dapp-Shopping-List

Dentro de Dapp-Sections, podemos visualizar la lista de notificaciones, estas nos muestran productos que publica el proveedor de servicios que podemos adquirir con puntos. Si hacemos clic sobre cualquiera nos llevará a la pagina de Notification que nos permite ver un resumen del producto y comprarlo.

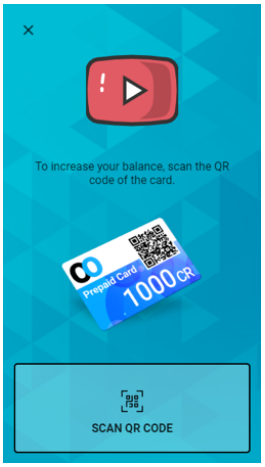


Fig. 9: Dapp-scanQR
Acceso al escaner lector de QR

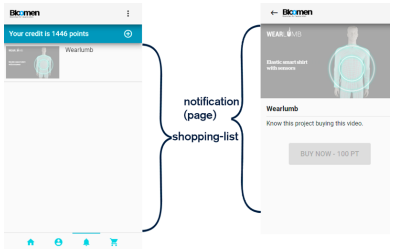


Fig. 10: Dapp-Shopping-List
Lista de elementos a comprar dentro de la Dapp

## 6.5 Estilos

Cada uno de los proveedores de servicio querrán tener su propia paleta de colores identificatorio por lo que hemos de controlar cada uno de los temas visuales de estos.

Generamos una variable global [theme] en la que indicamos que tema utilizamos. Para saber cual llamamos al almacenamiento caché (store) donde se ha indicado o se ha guardado uno por defecto.

```
this.theme$ = this.store.pipe(select(
    fromSelectors.getTheme));
```

Una vez la tenemos nuestro HTML principal que encapsula el resto de la app (sale por él router-outlet) de manera dinámica se le añadirá la clase que define el tema.

```
<div [class]="'theme-wrapper ' + (theme$ | async)
    ">
  <router-outlet></router-outlet>
</div>
```

Y ahora ya podemos definir un archivo con todos nuestros temas que contendrán una paleta identificatoria de colores y las clases de los objetos que deben cambiar. Color de la letra, color de los banners, los background image de los modales, botones etc...

## 6.6 Testing

En este apartado veremos la creación de test unitarios, de integración y de aceptación.

### 6.6.1 Test Unitarios

Para ello hemos usado el framework Jasmine[1] con el que he creado nuestros test con muchas otras funcionalidades y con el task runner Karma[2] podremos lanzarlos, crear un archivo de inicio, ver los reportes y muchas otras configuraciones todo esto visto desde el navegador.

```
1 it("deberia tener el titulo Bloomen App", async
      (()=>{
2   const fixture = TestBed.createComponent(
        AppComponent);
3   const app = fixture.debugElement.
        componentInstance;
4   expect(app.title).toEqual("Bloomen App");
5   });
```

- it(): nombre descriptivo del test.
- fixture: tipo de acción que genera la creación del componente a testear.
- app: instancia generada a testear.
- expect(): elemento que queremos comparar.
- toEqual(): elemento que debería haber para que el test salga como valido.

En la configuración de Karma[2] para lanzar los test hemos de declarado opciones como el framework que usamos, en nuestro caso Jasmine[1], todos los plugins de comporta el framework, el formato en el que queremos el reporte y en nuestro caso al quererlo por navegador lo pedimos en "html", el navegador también hay que escogerlo al igual que el puerto etc...

### 6.6.2 Test de Integración

Usé el framework Protactor[3] que nos permite generar bloques de prueba en Angular[13], con el que probar toda una selección de acciones e interacciones en el CSS para poder acabar testeando toda una serie de integración de código buscando ciertos resultados finales

```
1 element(by.id('firstName')) // Objetos con el id
2 element(by.css('.signout')) // Objetos con la
      clase
3 element(by.input('firstName')); // Campos de
      texto
4 element(by.buttonText('Close')); // Botones
```

Una vez tenemos estos objetos localizados en variables podemos aplicarles toda una serie de acciones y solicitar el resultado esperado.

```
1 element(by.id("firstName").sendKeys("Sergi")
2 //Mandamos una combinacion de teclas al elemento
      que tenga el id firstName
3 element(by.input("firstName")).clear()
4 // Limpiamos un campo de texto
5 var list = element.all(by.css(".user"))
6 // Generamos una lista con todos los elementos
      que tengan la clase user
7 expect(list.count())  // Le aplicamos la funcion
      count para saber el numero.
```

Una vez hayamos creado todos los datos y funciones a recoger de cada elemento, con la función .toBe() podremos compararlo con el resultado esperado.

```
1 expect(list.count()).toBe(3)
2 //Numero de elementos de la lista esperados: 3.
3
4 expect(list.get(0).getText()).toBe("First")
5 // Valor del primer elemento de la lista
      esperado: First.
```

## 6.7 Usabilidad y Pixel Perfect

En este apartado veremos el detalle tratado con CSS para un acabado visual estructurado y modular en el que buscamos que la aplicación sea lo mas fluida y armoniosa posible a vista e interacción humana.

- Paletas de colores
- Proporciones
- Alineaciones
- Márgenes
- Iconos
- Tamaños de fuente
- Etc.

Podemos medir la perfección de este testeando esta en cientos de usuarios de diversas edades buscando un uso inmediato y fácil sin necesidad de instrucciones por medio de la intuición.

### 6.7.1 Librería de CSS

Hemos escogido la librería Materials [4] ya que esta nos provee con lo último en diseño de UI con la UX más conocida en dispositivos Android ya mundialmente conocido por todo el entorno de Google.

- Campos de texto
- Formularios
- Botones
- Parrillas
- Sombras de deshabilitado
- Iconos
- Colores

La lista es tan extensa como elementos de vista podemos llegar a proveer a una aplicación pero estos son los mas habituales y los que siguen un patrón visual muy reconocido.

### 6.7.2 Personalización

A veces no es suficiente con utilizar solo la librería de Material si no que también queremos generar o modificar nuestro propio CSS como hemos visto anteriormente en personalización de temas. El problema sucede cuando estas librerías actúan una vez han leído todo el DOM y aplican sus propios estilos, estos se superponen a nuestros cambios previos por lo que debemos avisarles en las hojas de estilos que actuaremos en un futuro, es decir, cuando la librería ya ha efectuado.

Únicamente añadiendo un pequeño tag ::ng-deep y afectando a la clase que queramos (ya sea de Material o propia) le estamos diciendo a la hoja de estilos que se aplique una vez esté todo el DOM y las librerías activas.

### 6.7.3 Parrilla (Grid)

El orden y colocación de los elementos es uno de los conceptos más básicos en las hojas de estilos y uno de los mayores problemas a los que deben enfrentarse los programadores. En muchas de las ocasiones deben hacerse cálculos para prever cambios en la anchura de la vista y mover dichos contenedores para una vista igual de intuitiva. En esta situación he decidido utilizar unas librerías que permiten generar un grid con solo añadir unos componentes en los propios tags HTML y esta provee con código CSS nativo, las librerías son únicas para Angular y como he dicho solo nos traduce a código nativo.

Las propiedades en el CSS son de la familia Flex y sus principales son:

- Display: flex; convierte el tag tipo flex.
- Justify-content: row — column ; indica la dirección en la que deben ir los hijos del componente.
- Align-items: start — center — end — etc. . . ; indica las posiciones y separaciones que debe haber entre hijos del componente.
- Flex-wrap: warp — none ; indica si los hijos deben bajar a la siguiente fila si no caben en el padre o sobreponerse.

Dicha librería FxFlex[5] la podemos encontrar en los repositorios de NPM[7] o Github.

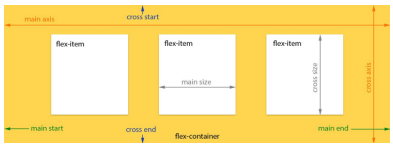


Fig. 11: FxFlex
Estructura del funcionamiento de FxFlex

## 7 Mockups

Como hemos nombrado anteriormente en los modelos de las estructuras de datos (Objetos), muchos de estos nos deben llegar de diversas maneras a través de servicios recibiendo datos en formato JSON o recogiéndolos de la memoria caché. En muchos de los casos los programadores de Back-End pueden no tener creadas las funcionalidades o que la base de datos ni siquiera esté preparada, por lo tanto la mejor manera de empezar a testear nuestro código contra estructuras de datos es a través de Mockups.

El Mockup es la estructura de datos que queremos recibir en caso de solicitarla para tratarla, las estructuras son en formato JSON.

```
"user": {
    0: {
        "name": "Joe Douglas",
        "id": "0x0912391"
    },
    1: {
        "name": "Mike Ourglass",
        "id": "0x1829731"
    }
}
```

Para ello decidimos alojarlos en la carpeta de assets y crear una ruta para acortar en el fichero tsconfig.json.

```
    "baseUrl": "src",
    "paths": {
      "@mocks/*": ["app/assets/mocks/*"],
```

Ahora con importar desde @mocks con un sobrenombre podremos utilizarlo.

```
import * as Mockup-user from '@mocks';
```

## 8 Loading

Los tiempos de carga en esta aplicación son bastante visibles, al ser operaciones complejas y con cierto riesgo hemos de asegurarnos de que el usuario no pueda generar una combinación de botones con la que se puedan hacer mas operaciones de las debidas, o haya un uso inapropiado de los contratos.

Por lo tanto generaremos un sistema de pantallas de carga que se superpongan al estado actual una vez se inicie una acción para que el usuario no pueda clicar en mas opciones. Cada uno tendrá su propia animación que identificará qué clase de movimiento realiza.

### 8.1 Servicio Loading

Para ello debemos crear un servicio capaz de ser llamado desde cualquier parte de la aplicación que requiera de un tiempo de carga, por lo que crearemos un módulo en la carpeta de módulos compartidos llamado Loader.

Este se debe instanciar desde la raíz del proyecto.

```
 <blo-loader
  [isLoading]="isLoading"
  [isLoadingCamera]="(isLoadingCamera$ | async)">
</blo-loader>
```

Y de esta manera el servicio se encarga de generar una promesa con el siguiente código que finalizará una vez obtenga la respuesta de que la cámara o la transacción ha finalizado.

```
this.loader.next(true);
```

En nuestra aplicación por el momento se encuentran 2 tipos de pantallas de carga que podemos llamar a través de los inputs del componente.

- Loading básico: cualquier tipo de transferencia o afectación a los contratos.
- Loading cámara: abrir el scanner de códigos QR.

## 9 Entorno de traducción de textos

Nuestra aplicación constará de 5 idiomas a los que traducir absolutamente todos los textos. Al ser una aplicación a nivel europeo se han implementado los 5 idiomas más utilizados en las diferentes comunidades: inglés, español, alemán, griego y catalán (por cortesía de la casa).

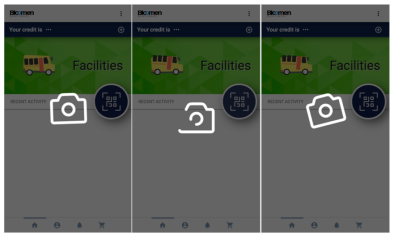


Fig. 12: Loading
Animación en pantallas de carga

Al utilizar Angular 7.0 trabajamos con el entorno “npm” de Node.js y de este extraemos el modulo de i18n que nos brinda un servicio simple y liviano en el que a través de un archivo .json podemos llamar a un solo ítem y que este venga traducido el idioma escogido por el usuario.

Nada más entrar en la aplicación, previo al tutorial y cualquier otro evento esta nos pedirá que escojamos un idioma y lo almacenará en la memoria caché del móvil, para no tener que volver a pedírnoslo. Ver figura 1.

Una vez tenemos en caché el idioma que desea el usuario las librerías escogerán el archivo json según su preferencia, todos siguen la misma estructura, pero cambian los textos.

El interior del .json mostrara textos tal que así:

```
"tutorial": {
   "0": {
      "title": "Can a complex technology make our life simpler?",
      "text": "This is a blockchain-based wallet to access Bloomen decentralized applications (Dapps)"
   },
   "1": {
      "title": "Discovery & Enrollment",
      "text": "Search for Bloomen Dapps and aggregate them easily with a simple, anonymous sign in"
   }
```

En el caso de que quisiéramos usar el titulo de la primera pestaña del tutorial deberíamos llamarlo en el HTML de la siguiente manera.

```
<div>{ tutorial.0.title | translate }</div>
```

## 10 Cambios y rectificaciones

A lo largo de todo este proyecto he descartado objetivos con un sentido muy genérico que por su contenido se han ido realizando con el paso y su explicación está fuera los márgenes de la documentación de un trabajo de fin de carrera.

A continuación los enumero y detallo las razones:

- ~~FRONT-06-01~~: Generar las funcionales visuales con sus respectivas estructuras de datos:
  - Su propio título indica lo general que puede ser esta tarea y un error el hecho de plantear siquiera, cuando mas adelante redacto la manera en la que genero estructuras de datos y como se trabaja con ellas
- ~~FRONT-06-07~~: Generar versiones de testing para todos los dispositivos con Cordova:
  - El testing de la aplicación como código Typescript se traduce al mismo que el transpilado a Java nativo o Swift. Eso quiere decir que las versiones generadas son idénticas a la del Javascript nativo que hemos programado y por lo tanto no es necesario testear algo que ya fue previamente testeado en su versión original.
- ~~FRONT-06-06~~: Asegurar la perfomance de la aplicación:
  - Al estudiar como ingeniero informático se deben emplear siempre las complejidades temporales más adecuadas en todo momento. La perfomance de la aplicación será tan pesada o liviana como el tamaño de proyecto y recursos a utilizar.
- ~~FRONT-06-08~~: Comprobar errores visuales y de datos en las versiones de Android y iOS:
  - Como ya he mencionado en el apartado de testing, las pruebas de aceptación no forman parte de mi proyecto.
- ~~FRONT-07~~: Integrar las funcionalidades. Retirar los Mocks y usar las funciones generadas por el departamento de Back-End:
  - Son partes del proyecto que no deben ser mencionadas ya que son tareas mas irrelevantes a nivel de documentación, incluso algunas partes son mencionadas en el apartado de Mockups o servicios.
- ~~FRONT-07-01~~: Solucionar bugs generados por las funciones de los Smart-Contracts y coordinarse con el equipo de Back-End:
  - Solucionada con el apartado de testing.
- ~~FRONT-08~~: User testing + perfomance testing:
  - Solucionada y resumida con el apartado de testing.

## 11 Conclusiones

En primer lugar quiero exponer que el resultado ha sido mucho mejor de lo esperado, la aplicación ha cogido desde el principio un acabado minimalista, limpio y fluido que hizo que toda la construcción de esta fuese motivante y continua.

Muchas de las decisiones de diseño como la división de departamentos, la metodología, las entregas fueron tomadas por parte de la empresa por una clara experiencia previa en proyectos. Nuestras preferencias en este caso tampoco se vieron maltratadas ya que el camino fue una constante lluvia de conocimientos y aprendizaje.

La ayuda de modular la aplicación en FrontEnd y BackEnd ha sido una idea mas que exitosa por toda la dedicación emprada en cada uno de los ámbitos, esta ha hecho que ninguna de las dos partes tambalee.

Una de mis conclusiones principales es que en aplicaciones de cierta envergadura es crucial la separación de las dos partes del desarrollo.

Las fases mas problemáticas fueron a la hora de estructurar los componentes, ya que no existe una forma idónea de seguir ni una que premie un escalado tanto vertical como lateral en la aplicación. Los tests E2E también dieron problemas a la hora de lanzarse en multiples versiones compiladas de la aplicación.

Algo que también ayuda pero a la vez puede ser problemático son las librerías de terceros, en este caso dependemos completamente de los servicios de Blockchain de Alastria ya que es nuestro principal motor. Pero en el caso de Material, que viene a ser una de las librerías que nos proporciona muchos elementos de HTML y CSS con ciertos estilos y funciones predefinidos, nos arriesgamos a que una actualización pueda generar grandes costes de trabajo en un futuro si esta no sigue acoplandose bien.

## 11.1 Línea de continuación

Este es un proyecto de Worldline Iberia S.A y como tal seguirá su curso con nuevas actualizaciones e implementaciones ya de momento no hay fecha de Release. Pero por el momento se podría considerar que hay un 95 por ciento del trabajo estimado realizado.

## AGRADECIMIENTOS

Deseo dar las gracias a ciertas personas que sin ellas esto podría haber cojeado y no hubiera cogido la misma velocidad y ganas con las que he estado trabajando hasta la fecha. A Jordi Escudero, lead del proyecto en la empresa de Bloomen que ha sido la batuta y diccionario en todo momento. A Eric Torres Perramon, desarrollador de la parte de BackEnd en su proyecto y gran amigo al que acudir en momentos de sosiego. Hemos comentado y construido este proyecto codo con codo. A mi tutor de proyecto y exprofesor Lluís Gesa que siempre ha sido una inspiración y gratificación poder contar con una persona tan cercana con la que compartir y solucionar mis temores en el proyecto. Y por último a mis padres por dármelo todo.

## REFERÈNCIES

[1] Jasmine
Framework for testing on FronEnd.
https://jasmine.github.io/

[2] Karma
Herramienta de ejecución de test de FrontEnd.
https://karma-runner.github.io/

[3] Protactor
Protractor is an end-to-end test framework for Angular.
http://www.protractortest.org/

[4] Material
Library for Angular to help with visual UI tools.
https://material.angular.io/

[5] FxFlex
A Library for Grid and Responsive Angular DOM.
https://github.com/angular/flex-layout/wiki/fxFlex-API

[6] Github
Biggest open code repository on internet
https://github.com

[7] NPM
Manejador de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript
https://www.npmjs.com/package/i18n

[8] Redux
Biblioteca de código abierto de JavaScript para administrar el estado de la aplicación. Se usa más comúnmente con bibliotecas como React o Angular para crear interfaces de usuario.
https://es.redux.js.org/.

[9] Blockchain
Estructura de datos en la que la información contenida se agrupa en conjuntos
https://es.wikipedia.org/wiki/Cadenadebloques

[10] Dapp
Una app que no depende de un sistema central, sino que depende de la comunidad de usuarios que la utilizan.
https://miethereum.com/smart-contracts/dapps/

[11] Ethereum
Plataforma open source, descentralizada que permite la creación de acuerdos de contratos inteligentes entre pares, basada en el modelo blockchain.
https://www.ethereum.org/.

[12] TypeScript
Lenguaje de transpilado para JavaScript.
https://www.typescriptlang.org/.

[13] Angular
Framework de Google diseñado para crear aplicaciones tanto móviles como de escritorio.
https://angular.io/.

[14] Pixel Perfect
Buenas practicas enfocadas al retoque estético digital.
https://www.nirmal.com.au/exactly-pixel-perfect-web-design/

[15] Invision
Herramienta para exportar pantallas.
https://www.invisionapp.com/

[16] Trello
Software de administración de proyectos con interfaz web.
https://trello.com/

[17] Google Chrome DevTools
Conjunto de herramientas de creación web y depuración integrado en Google Chrome.
https://developers.google.com/web/tools/chrome-devtools/?hl=es

[18] Sourcetree
Cliente para gestionar proyectos Git.
https://www.sourcetreeapp.com/

[19] Microsoft Office 365
Paquete de herramientas para gestión de documentos.
https://www.office.com/

[20] Visual Studio Code
Editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS
https://code.visualstudio.com/

[21] Aplicación híbrida
Las aplicaciones móviles híbridas son una combinación de tecnologías web como HTML, CSS y JavaScript, que no son ni aplicaciones móviles verdaderamente nativas, porque consisten en un WebView ejecutado dentro de un contenedor nativo
https://solidgeargroup.com/desarrollo-de-apps-hibridas-con-ionic?lang=es

# A Anexo

## A.1 Tutorial


Fig. 13: Tutorial
Recorrido de pantallas del tutorial inicial