Evaluación Trabajo Final Career Switch 2022

# Proyecto

Título: Real State Tokenization

Alumno/s: Bertran Luciano

Corrección: Leguizamón Leandro

[Proyecto](#_65i505lxwjqa) **2**

[Presentación](#_thacqlcq75e) **2**

[Contratos](#_8i90oli4sisx) **2**

[Estructura](#_p4yj5z42f06u) 2

[Observaciones](#_wi4nsodljy2j) 3

[PriceConsumer.sol](#_wy87m9dsrff6) 3

[Comentario de constructor debería ir en el back](#_25eggoixvn) 3

[AssetFactory.sol](#_jmidleb190s) 4

[Arreglo \_divisibleAssets podría ser un mapping](#_yqaxnmcdt0fl) 4

[Formato de nombre erróneo L7 y L15](#_ounsdvk2oo9u) 4

[Funciones createDivisibleAsset L17 y allSsets L43 deberían ser external](#_qd4dws9r03zd) 4

[Funciones createDivisibleAsset L17 no tiene restricciones de acceso](#_1v5d2uxtkl52) 4

[require(transfer) L33 puede reemplazarse por safeTransfer](#_1rrlel848xc9) 4

[Función allAssets inaccesible si arreglo \_divisibleAssets crece mucho](#_4v23bodwqptu) 4

[Controller.sol](#_4carb7ou7yzl) 4

[Evento contiene palabra Event redundante en su nombre L16](#_j5yv2b58b7lt) 4

[Funciones deberían ser external](#_xhacvmzb2mxz) 4

[Función createAssetAndProtocol no aclara formato de parámetros \_price e \_initalSupply](#_t9ecpax9hmyf) 5

[Decimales poco claros L38 - 40](#_nit9xjfo6t71) 5

[Funciones L67 - L79 sin restricción de acceso](#_j8du3x85ikev) 5

[Cambios en direcciones de contratos sin eventos](#_er8cfn2engy0) 5

[Seteo de contratos en constructor ahorraría gas](#_mex3vea9wyup) 5

[Función createAssetAndProtocol nunca envía los tokens](#_t2llmgnm9d7d) 5

[No hay forma de comprar los tokens generados y el ETH inicial nunca es devuelto](#_1wvtkzlibb8) 6

[LendingBorrowingFactory.sol](#_t8eqespy2d) 6

[Variable lendingBorrowing se castea varias veces a address L26](#_hpjqovh15jto) 6

[Función allProtocols devuelve arreglo completo L43](#_b4zsvnoyxnk3) 6

[Todas las funciones pueden ser external](#_p4ryo9yxh7x7) 6

[LendingBorrowing.sol](#_de1bpoak71c7) 6

[Import interfaces/IUniswapV2Router02.sol L9](#_a7awn77z84g) 6

[Import hardhat/console.sol L11](#_hapsuqmpdzm) 6

[Se utiliza assetFactory como oráculo, pero este nunca actualiza su precio L377](#_ukludrh29vek) 6

[Varias funciones deberían ser external](#_7sxpymww5kt3) 6

[Función withdraw no permite retirar el 100%](#_ux483iiputbh) 6

[Función repay nunca utiliza el valor del parámetro L195](#_rr2m3bwjynwh) 6

[Función repay se queda con ETH sobrante en caso de saldar la deuda L205](#_pab7hrenpi4r) 7

[Función liquidate no recupera los ETH faltantes](#_1rvlru5fse9z) 7

[Funciones intermedias getForwardCollateralRatio y getCurrentColateralRatio agregan poco valor L341 y L351](#_hkyrlccomgn4) 7

[Funciones set sin eventos](#_9w0erdmweqz2) 7

[Función transfer puede drenar los fondos del protocolo](#_s290h9gj5hla) 7

[Observaciones de diseño](#_sq7y42fcjjf) 7

[Los usuarios no pueden depositar ETH](#_7r3ymr77mqzg) 7

[Podría haber un solo contrato de lending](#_2076baz3vrgp) 7

[**Tests**](#_rkgv589l0h6j) **8**

[Observaciones](#_d1s1f4h0ycxq) 8

[Deploy](#_uycdxb19ol5h) **8**

[Otros](#_ltkfywq7cx8g) **8**

[Hardhat.config contiene keys](#_yhlgcti6zi4c) 8

[El compilador no tiene activada la optimización](#_yqso5zp121g6) 8

[Conclusión](#_d8o6kbi9u5nf) **8**

# Presentación

El repositorio contiene un Readme claro y detallado sobre el funcionamiento del proyecto, breve descripción de los contratos e instrucciones para su uso.

En L62 provee el comando para verificar el contrato, pero la dirección no será la misma para cada persona que lo use, ya que la dirección deriva, entre otras variables, de la calve pública que estemos usando.

# Contratos

## Estructura

contracts/

* interfaces/
  + IUniswapV2Factory.sol
  + IUniswapV2Pair.sol
  + IUniswapV2Router01.sol
  + IUniswapV2Router02.sol
  + IUSDZ.sol
* utils/
  + ABDKMath64x64.sol
* AssetFactory.sol
* Controller.sol
* DivisibleAsset.sol
* LendingBorrowing.sol
* LendingBorrowingFactory.sol
* PriceConsumer.sol

Los contratos en interface, exceptuando *IUSDZ.sol*, parecen ser públicos, dados por Uniswap V2, y no hechos a medida. Eso está bien, como mejora Uniswap tiene paquetes con sus contratos. En este caso se usaría *@uniswap/v2-core*, para evitar el trabajo manual y sumar archivos a */contracts*.

Sin embargo la mayoría de contratos en */interfaces* no se utilizan. De los 5 presentes solo se utilizan *IUniswapV2Router01.sol* e *IUniswapV2Router02.sol*. Y este último es importado en *LendingBorrowing.sol*, pero nunca lo utiliza. Por lo que la carpeta */interfaces* podría no estar.

## Observaciones

### PriceConsumer.sol

#### Comentario de constructor debería ir en el back

Comentarios que no agregan valor al contrato en si mismo, sino que aclaran datos para el momento del deploy, deberían estar en alguna parte del back-end. Como en el archivo de deploy o readme. Es el caso del comentario que aclara Network, Aggregator y Address.

En este caso se podría tener un json indexado por network y aggregator(par), que contenga la address. Así el deploy podría tomar la address de este archivo viendo en que red se encuentra y que par se desea, el cuál podría enviarse por parámetro en el comando o estar en un .env.

### AssetFactory.sol

#### Arreglo \_divisibleAssets podría ser un mapping

Los arreglos son costos, y este nunca es recorrido. Podría ser reemplazado por un *mapping(uint => Asset)* o por la utilidad *EnumerableMap* de openzeppelin.

#### Formato de nombre erróneo L7 y L15

*\_divisibleAssets* y *\_divisibleAssetsMap* son variables públicas, por lo que no deberían tener un guión bajo al comienzo del nombre. Este formato se utiliza para variables privadas, parámetros de funciones y/o nombre de funciones privadas o internas, ya que lo que se busca es saber su alcance con solo ver el nombre.

#### Funciones createDivisibleAsset L17 y allSsets L43 deberían ser external

Estas funciones nunca son llamadas dentro del contrato, por lo que no necesitan ser *public*, se puede limitar su alcance a *external*.

#### Funciones createDivisibleAsset L17 no tiene restricciones de acceso

La función no restringe su acceso a un owner, por lo que cualquiera podría llamarla y crear un token. Por lo que un atacante podría llamarla multiples veces para, por ejemplo, llenar el arreglo *\_divisibleAssets*. Lo que inhabilitaría la función *allAssets* y aumentaría el tamaño, y por tanto el coste de interacción, del contrato.

#### require(transfer) L33 puede reemplazarse por safeTransfer

En lugar de un require con una transfer adentro, podría utilizarse la función *safeTransfer* de *SafeERC20.sol* de openzeppelin. Sería interesante que opción es mejor en terminos de gas, por lo que esto no es un error, solo una observación.

#### Función allAssets inaccesible si arreglo \_divisibleAssets crece mucho

La función devuelve el arreglo completo, por lo que si este crece la función siempre fallará. Para esto se utiliza un sistema de punteros, enviando por parámetro el primer y último indice que se desean del arreglo, y devolviendo el rango entre estos.

### Controller.sol

#### Evento contiene palabra Event redundante en su nombre L16

No es necesario que los eventos tengan la palabra Event en su nombre. Es como a una variable agregarle “Var”. El estilo usado en eventos es comenzar por mayúsculas cada palabra y el verbo en pasado, porque es una acción que se realizó, no que se está realizando. En este caso sería *AssetAndProtocolCreated*.

#### Funciones deberían ser external

Todas las funciones de este contrato nunca son llamadas por si mismo, por lo que no necesitan ser *public*, se las puede restringir a *external*.

#### Función createAssetAnx|dProtocol no aclara formato de parámetros \_price e \_initalSupply

El precio dado por los oráculo tiene 8 decimales, mientras que el valor de ETH y *msg.value* está en 18 decimales, por lo que es necesario un comentario, o alguna documentación que aclare con que decimales debe enviarse el parámetro *\_price*. Además de aclarar si el precio a enviarse es en USD, ETH u otra moneda.

Algo similar ocurre con *\_initialSupply*, donde debería aclararse la cantidad de decimales. Si quiero generar 5000 tokens, debo enviar como parámetro 5000? 5000\*10^18? 5000\*10^8? 5?

#### Decimales poco claros L38 - 40

Relacionado al error anterior, se realizan varios cálculos con un comportamiento impredecible al no saber los decimales de cada valor. Por ejemplo *msg\_value\_in\_usd* tiene 26 decimales (18 de *msg.value* + 8 del precio dado por el oráculo). Luego se realiza *\_price \* \_initialSupply / ETH\_FACTOR*, que sería correcto si *\_price* tuviese 8 decimales y *\_initialSupply* 18, pero esto no se aclara por lo que el usuario, en confusión, podría enviar los valores con otro formato. Y esto suponiendo que *\_price* se envía en USD.

#### Funciones L67 - L79 sin restricción de acceso

Las funciones *setAssetFactory, setLendingBorrowingFactory y setPriceConsumer* pueden ser llamadas por cualquier dirección.

#### Cambios en direcciones de contratos sin eventos

Las funciones set que actualizan contratos que utiliza el protocolo también deberían emitir eventos. Ya que su actualización pone en riesgo el funcionamiento del protocolo.

#### Seteo de contratos en constructor ahorraría gas

Al margen de que existan las funciones set para actualizar los contratos, se podría tener un constructor que los reciba, para que así al momento del deploy ya se tengan unos contratos iniciales y evitar además del deploy, tener que enviar 3 transacciones para setear los contratos.

#### Función createAssetAndProtocol nunca envía los tokens

Esta función llama a *assetFactory.createDivisibleAsset*, por lo que este contrato recibe los tokens, pero luego no son enviados a ninguna parte. Además no hay forma de retirarlos ni comprarlos. Por lo que al tokenizar un activo mediante el Controller, solo se pierde ETH y no hay forma de coprar los tokens generados.

En los tests este error no aparece porque se llama directamente al contrato *AssetFactory*, entonces la dirección recibe los tokens que lugo utiliza para testear el lending. Sin embargo al llamar directamente a *AssetFacotry*, no es necesario enviar ETH, por lo que se pierde uno de los pilares principales del protocolo.

#### No hay forma de comprar los tokens generados y el ETH inicial nunca es devuelto

Los tokens generados mediante el *Controller* quedan bloqueado en el mismo, no hay forma de adquirirlos. Además la documentación aclara que al vender el 90% de los tokens, el usuario que lo creó recibe de regreso el ETH dado, esto nunca ocurre.

### LendingBorrowingFactory.sol

#### Variable lendingBorrowing se castea varias veces a address L26

En lugar de castear una variable varias veces es mejor hacerlo cuando se la obtiene. La variable *lendingBorrowing* se recibe con tipo *LendingBorrowing* pero nunca se lo utiliza, siempre se castea a *address*.

#### Función allProtocols devuelve arreglo completo L43

Misma observación que en contrato *AssetFactory.sol* en función *allAssets*.

#### Todas las funciones pueden ser external

Misma observación que en *Controller.sol*.

### LendingBorrowing.sol

#### Import interfaces/IUniswapV2Router02.sol L9

Importa la interfaz pero nunca la utiliza.

#### Import hardhat/console.sol L11

Librería para mostrar valores con hardhat. No debería llegar a los commits, además de que no la utiliza.

#### Se utiliza assetFactory como oráculo, pero este nunca actualiza su precio L377

Para obtener el *price* se llama a *assetFactory.\_divisibleAssetsMap* pero *AssetFactory* nunca actualiza el precio de sus tokens.

#### Varias funciones deberían ser external

Misma observación que en *Controller.sol*.

#### Función withdraw no permite retirar el 100%

Para retirar el 100% de los fondos, el usuario debe calcular cuál es el monto a solicitar. En estos casos la función podría aceptar como monto 0 para retirar la totalidad de los fondos.

#### Función repay nunca utiliza el valor del parámetro L195

El valor del parámetro *\_amount* nunca es utilizado. En L196 es reemplazado por *msg.value*.

#### Función repay se queda con ETH sobrante en caso de saldar la deuda L205

Si el monto enviado es mayor a la deuda más los interéses, entonces la deuda se vuelve 0, pero el monto sobrante no es enviado de regreso al usuario. Por lo que no hay forma de enviar la cantidad exacta para saldar la deuda.

#### Función liquidate no recupera los ETH faltantes

Al liquidar una posición el objetivo es que el liquidador provea los tokens adeudados, a cambios de los tokens del deudor. En este caso el liquidador debería entregar los ETH faltantes y a cambios el protocolo entregar los ERC20 que pertenecían al deudor. La función no exige este ETH, por lo que al ser ejecutada se queda sin los ETH (los tiene el deudor) y sin los ERC20 (se los envía al liquidador).

#### Funciones intermedias getForwardCollateralRatio y getCurrentColateralRatio agregan poco valor L341 y L351

Ambas funciones llaman a *\_getCollateralRatio*. El cambio entre estas es tan sutil, y *getCurrentCollateralRatio* se utiliza solo una vez, que es mejor que la llamada inicial aclare los parámetros. Estas funciones agregan poco valor y aumentan el coste de gas.

#### Funciones set sin eventos

Las funciones set que actualizan contratos, comsiiones u otro valor que utiliza el protocolo también deberían emitir eventos. Ya que su actualización pone en riesgo el funcionamiento del protocolo.

#### Función transfer puede drenar los fondos del protocolo

La función solo está restringida al owner, pero permite enviar cualquier cantidad arbitraria de ETH a donde desee. Esto es un riesgo para los fondos de los usuarios.

## Observaciones de diseño

### Los usuarios no pueden depositar ETH

El contrato no acepta el depósito de ETH registrando a los usuarios. Por lo que no hay incentivo para ser lender. Solo recibe ETH mediante la creación del token en Controller, pero el ETH dado inicialmente (10%) tampoco es recuperado nunca y no se utiliza tampoco para añadir liquidez a un DEX. Por lo que si alguien desea tokenizar un activo, los ETH dados al comienzo se pierden y los tokens que recibiría tampoco tienen valor en el mercado, por lo que no pueden ser venidos.

### Podría haber un solo contrato de lending

En principio la cantidad de datos a almacenar por cada lending permite tener un solo contrato que maneje todos los mercados de lending mediante mappings o arreglos. Tener un contrato distinto por cada lending, además de consumir gas para su creación, se vuelve un problema para su interacción. Porque se debe tener alguna lista con la dirección de cada token y su contrato de lending. Sea en otros contratos, en un front-end, bots, etc.

# Tests

## Observaciones

Los tests solo prueban cuestiones básicas como deploy, reverts sobre fondos insuficientes y algunas funciones. Pero no prueba cuestiones críticas como borrow y repay de deudas, que al crear un token mediante el Controller este devuelva los tokens correspondientes, liquidaciones, retirar el 100% del collateral, etc.

# Deploy

En general está bien. Solamente las direcciones de chainlink podrían estar en un json. Misma observación que en *PriceConsumer.sol* sobre el constructor.

# Otros

### Hardhat.config contiene keys

Todo tipo de keys, como claves publicas, privadas, claves de api, etc, deben ser manejadas mediante un .env que se deba configurar para utilizar el proyecto. Las claves no deben estar hardcodeadas en el código ni expuestas al publico.

### El compilador no tiene activada la optimización

En *hardhat.config.ts* la propiedad *solidity* debería también tener un bloque tipo:

settings: {

optimizer: {

enabled: true,

runs: 200

}

},

Esto permite reducir el tamaño de los contratos en la compilación.

# Conclusión

La idea me gustó mucho pero quizás fue muy compleja para el nivel. A lo largo de los contratos hay errores críticos que rompen el correcto funcionamiento del protocolo. También existen varios errores que podrían considerarse código basura, o archivos que ocupan lugar en el repositorio pero nunca son utilizados.

Hay que repasar visibilidad de funciones. El error más grave es sin dudas el drenaje de fondos. Los protocolos también hay que pensarlos desde la perspectiva de seguridad del usuario. En este caso se debería haber tenido otra variable que diferencie los ETH ganados por el protocolo que pueden retirarse, en lugar de permitir retirar cualquier cantidad.

Sin embargo es un esquema complejo, pero deben solventarse los errores antes de utilizarse como parte de un portfolio.

Tests más detallados sobre el flujo de operaciones y los valores modificados por las mismas hubiesen permitido detectar estos errores y corregirlos.

**Nota final: 6/10, aprobado.**