Máster en Ingeniería del Software: Cloud, Datos y Gestión TI

NFTMARKET - WALLET SERVICE

Equipo Wallet:

- Cáceres Romero, David
- Pradas Fernández, Vanessa

URLs:

- Repositorio del BackEnd: https://github.com/NFTsMarket/Wallet-Service
- Repositorio de integración continua del BackEnd: https://github.com/NFTsMarket/Wallet-Service/blob/master/.github/workflows/okteto-deployment.yml
- Despliegue del BackEnd: https://api-fis-wallet-d-rhym.cloud.okteto.net
- Repositorio del FrontEnd: https://github.com/NFTsMarket/NFTsMarket-Frontend.git
- Enlace al despliegue de FrontEnd: https://nftsmarket.netlify.app/

Contenido

1. Nivel de acabado

El equipo desarrollador del microservicio Wallet-Service elige presentarse al nivel de acabado de hasta 9 puntos. Esto incluye todas las características del nivel de acabado de hasta 5 puntos y, como mínimo, 5 características de microservicio avanzado y 3 características de aplicación basada en microservicios avanzada.

El microservicio en cuestión, Wallet-Service, se encarga de la gestión de las carteras de los usuarios. Esto implica desde su creación hasta su correspondiente actualización al añadir fondos a la misma o al realizar una compra-venta de productos. Es por ello que debe ofrecer sus datos correspondientes a los fondos actuales de los usuarios a aquellos microservicios que se encarguen de la gestión de compras. Cabe destacar que para la inserción de nuevos fondos se ha hecho uso de una API externa, en concreto Paypal. Si desea hacer uso de sus servicios puede usar las siguientes credenciales:

Usuario: sb-u31xk8601012@personal.example.com

Contraseña: u|2$>UVp

Las características implementadas para conseguir el nivel de acabado de hasta 9 puntos, además de todas las básicas para el acabado de hasta 5 puntos, son las siguientes:

- **Microservicio avanzado**:
  - Implementar un front-end con rutas y navegación.
  - Uso del patrón materialized view para mantener internamente el estado de otros microservicios.
  - Consumo de algún API externa (distinta a la de los grupos de prácticas)
  - Tener el API REST documentado con swagger.
  - Implementación de un mecanismo de autenticación basado en JWT o equivalente.
- **Aplicación basada en microservicios avanzada**:
  - Interacción completa entre todos los microservicios de la aplicación integrando información.
  - Tener un front-end común que integre los front-ends de cada uno de los microservicios.
  - Definición de un customer agreement para la aplicación en su conjunto.
  - Hacer uso de un sistema de comunicación asíncrono mediante un sistema de cola de mensajes para todos los microservicios.
  - Implementación de un mecanismo de autenticación homogéneo para todos los microservicios.

2. Implementación de código

2.1. Nivel de acabado hasta 5 puntos

- **El backend debe ser un API REST tal como se ha visto en clase implementando al menos los métodos GET, POST, PUT y DELETE y devolviendo un conjunto de códigos de estado adecuado.**

El backend de Wallet-Service consta de un total de 6 llamadas cuyas funciones son listar carteras (GET), mostrar cartera (GET), crear cartera (POST), eliminar cartera (DELETE), modificar cartera (PUT) y añadir fondos a la cartera (PUT). Todos estos métodos están disponibles en el archivo *server.js*, en la carpeta raíz del proyecto.

```
// Crear cartera
app.post(BASE_API_PATH + "/wallet", authorizedClient, async (req, res) => {
    try {
        var wallet = createWallet(req.body.user, req.body.fund, req.body.lastTransactions, req.body.deleted, req.body.createdAt, req.body.updatedAt);

        res.setHeader('Location', '/wallet/' + wallet._id);

        return res.status(201).json(wallet);
    } catch (e) {
        return res.status(400).json(e);
    }
}
);

//Listar carteras
app.get(BASE_API_PATH + "/wallet", authorizedClient, (req, res) => {
    let limitatt = (req.query["limit"] != null && !Number.isNaN(req.query["limit"])) ? req.query["limit"] : 0;
    let offset = (req.query["offset"] != null && !Number.isNaN(req.query["offset"])) ? req.query["offset"] : 0;
    let sortatt = (req.query["sort"] != null) ? req.query["sort"] : null;
    let order = (req.query["order"] != null) ? req.query["order"] : 1;
    let filters = req.query;
    Object.keys(filters).forEach(x => {
        if (x == "sort" || x == "order" || x == "limit" || x == "offset") {
            delete filters[x];
        }
    });
    Wallet.find(filters, null, { sort: { [sortatt]: order, _id: 1 }, limit: limitatt, skip: offset * limitatt }, function (err, wallet) {
        if (err) return res.status(500).json(err);
        return res.status(200).json(wallet);
    });

});
```

```
// Obtener una cartera
app.get(BASE_API_PATH + "/wallet/:id", authorizedClient, (req, res) => {
    if (!ObjectId.isValid(req.params.id)) {
        return res.status(400).json("A wallet with that id could not be found, since it's not a valid id.");
    }

    var filter = { user: req.params.id };
    Wallet.findOne(filter, function (err, wallet) {
        if (err) {

            return res.status(500).json(err);
        } else if (wallet) {
            return res.status(200).json(wallet);
        } else {
            return res.status(404).json("A wallet with that id could not be found.");
        }
    });
});

// Borrar cartera
app.delete(BASE_API_PATH + "/wallet/:id", authorizedClient, (req, res) => {
    if(!ObjectId.isValid(req.params.id)){
        return res.status(400).json("A wallet with that id could not be found, since it's not a valid id.");
    }

    Wallet.findByIdAndDelete(req.params.id, function (err, wallet) {
        if (err) {
            return res.status(500).json(err);
        }
        else if (wallet) {
            pubsubMessage.sendMessageDeletedWallet(wallet);
            return res.status(200).json();
        } else {
            return res.status(404).json("A wallet with that id could not be found.");
        }
    });
});
```

```
// Modificar cartera
app.put(BASE_API_PATH + "/wallet/:id/:fund", authorizedClient, (req, res) => {
    if (!req.params.fund.match(/\d+\.\d+/)) {
        return res.status(400).json("Invalid fund format.");
    } else {
        var filter = { user: req.params.id };
        Wallet.findOne(filter, async function (err, wallet) {
            if (err) {
                return res.status(500).json(err);
            } else if (wallet) {
                try {
                    var temporalTransactions = wallet.lastTransactions + wallet.amount;
                    Wallet.findOneAndUpdate(filter, { fund: wallet.fund + Number(req.params.fund), lastTransactions: temporalTransactions }, function (err, doc) {
                        if (!doc) {
                            return res.status(400).json("A wallet with that id could not be found.");
                        }
                    });
                    var wallet = await Wallet.findOne(filter);
                    if (wallet) {
                        pubsubMessage.sendMessageUpdatedWallet(wallet);
                        return res.status(200).json(wallet);
                    } else {
                        return res.status(404).json("A wallet with that id could not be found.");
                    }
                } catch (e) {
                    return res.status(500).json(e);
                }
            } else {
                return res.status(404).json("A wallet with that id could not be found.");
            }
        });
    }
});
```

- **La API debe tener un mecanismo de autenticación.**

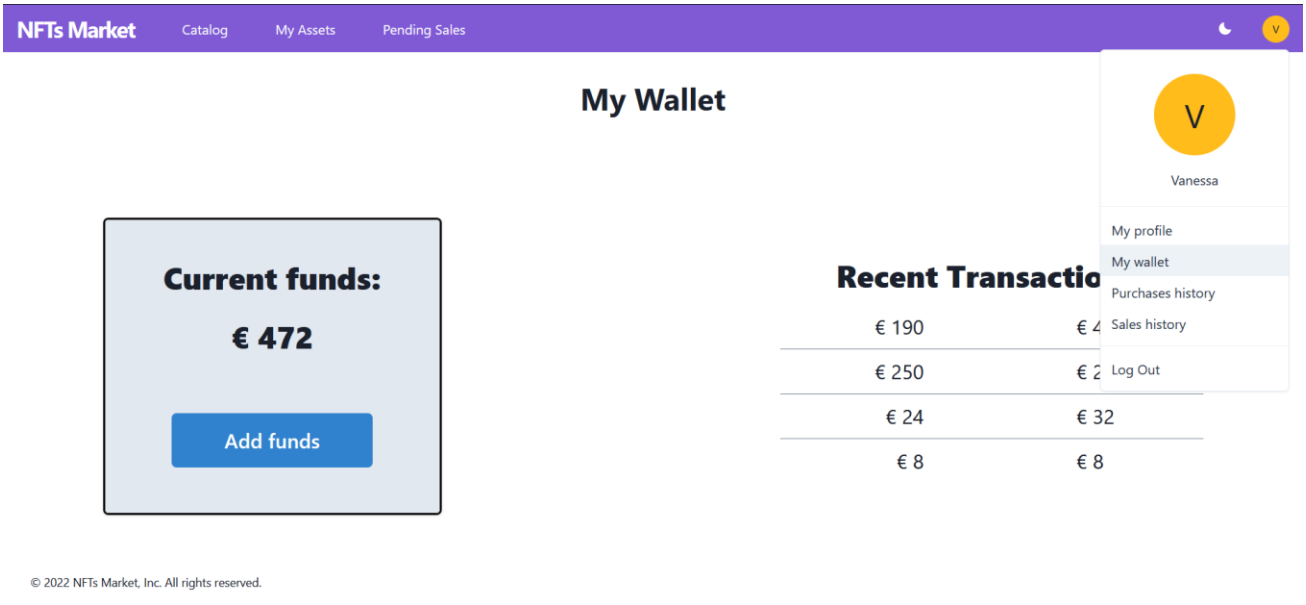
Todos los microservicios, incluido Wallet-Service, hacen uso de JWT para la autenticación del usuario. Este sistema hace uso de token asignados a un usuario en el momento de su inicio de sesión para informar al resto de micro-servicios de que el usuario en cuestión puede acceder a las funcionalidades reservadas para usuarios autenticados. Cabe destacar que, dado que Wallet-Service se constituye de las carteras personales de los usuarios, es obligatorio por tanto que para acceder a todos y cada uno de sus métodos se deba iniciar sesión previamente.

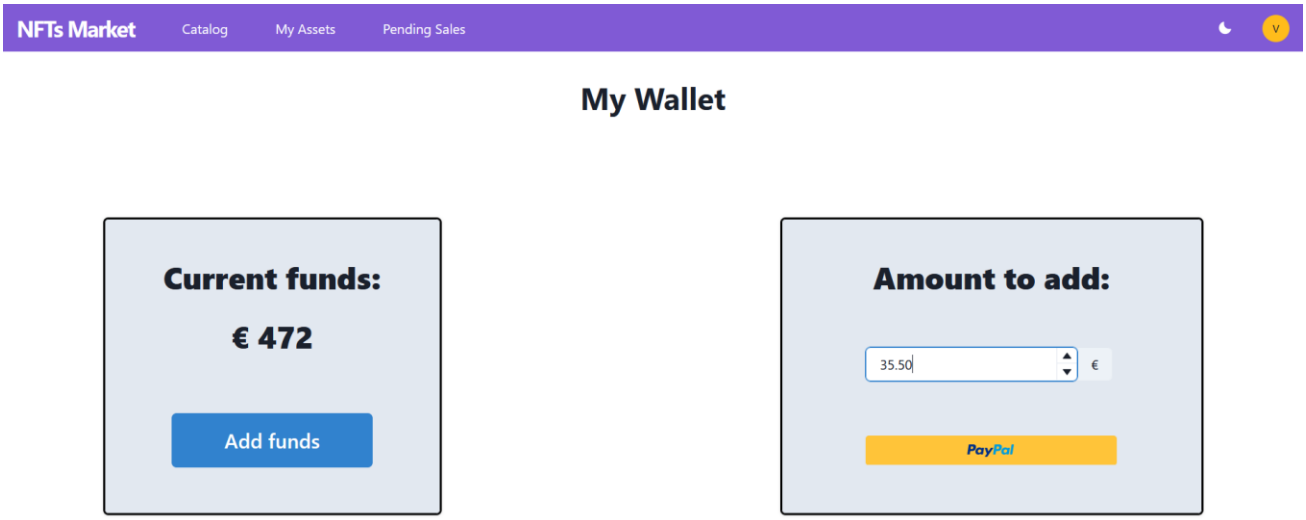
Tanto el diseño de los roles como su validación se encuentran en sus archivos correspondientes en la carpeta *middlewares* del proyecto.

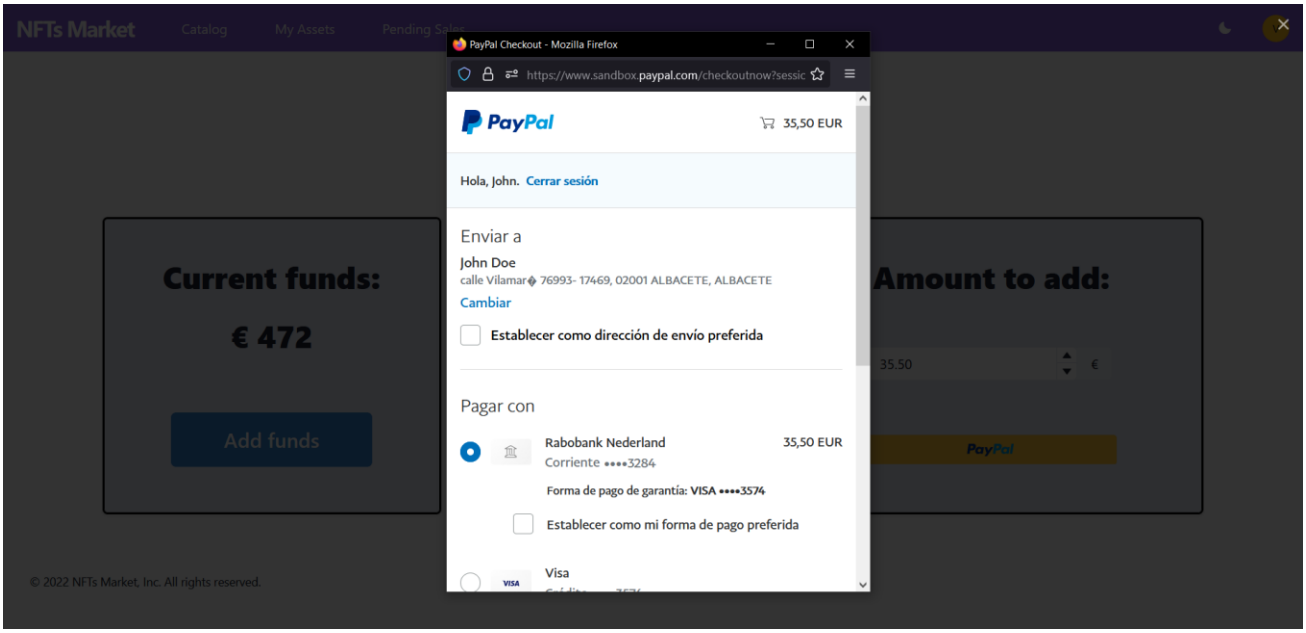
```
const authorizedClient = (req, res, next) => {
  try {
    const token = req.header("Authorization");

    if (!token) {
      return res.status(401).json({
        msg: "Token is not provided",
      });
    }
    const payload = jwt.verify(
      token.replace("Bearer ", ""),
      process.env.SECRET_KEY
    );

    if (!verifyRole(payload.role, CLIENT)) {
      return res.status(403).json({
        msg: "Unauthorized role",
      });
    }

    req.id = payload.id;

    next();
  } catch (e) {
    console.log(e);
    return res.status(401).json({
      msg: "The provided JWT is malformed",
    });
  }
};
```

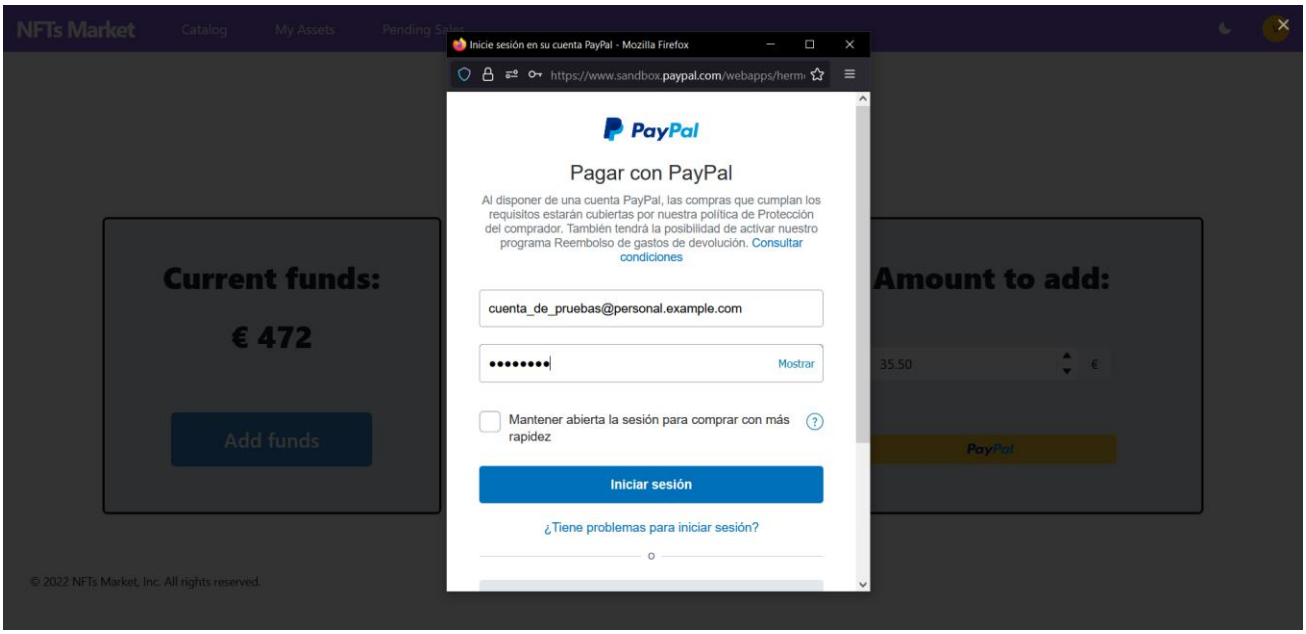
- **Debe tener un frontend que permita hacer todas las operaciones de la API (este frontend puede ser individual o estar integrado con el resto de frontends).**

Este proyecto posee un frontend integrado con el resto de microservicios. Las secciones del frontend correspondientes a Wallet-Service están compuestas de dos vistas principalmente. La primera es una vista de la cartera del usuario, donde puede comprobar sus fondos actuales y las últimas transacciones realizadas, siendo estas divididas en dos columnas representando el dinero añadido o restado y los fondos tras ese movimiento. La segunda sección es una vista para añadir fondos a la cartera, donde por un lado se permiten ver los fondos actuales y por otro se añade la cantidad que el usuario desee.









- **Debe estar desplegado en la nube y ser accesible en una URL.**

El microservicio Wallet-Service tiene su propio despliegue en Okteto. No obstante, la url base del despliegue le redirigirá a la documentación de la API en Swagger, desde la que podrá acceder a los servicios deseados. El enlace en cuestión es el siguiente enlace:

- Enlace al despliegue backend: https://api-fis-wallet-d-rhym.cloud.okteto.net

En cuanto al frontend, el despliegue común para todos los microservicios (incluido Wallet-Service) se encuentra en el siguiente enlace:

- Enlace al despliegue frontend: https://nftsmarket.netlify.app/

- **La API que gestione el recurso también debe ser accesible en una dirección bien versionada.**

El versionado de la API se realiza mediante la cadena “/api/v1” a continuación de la url base del despliegue backend de Okteto. Este versionado se realiza en el archivo *server.js* de la carpeta raíz del proyecto.

- **Se debe tener una documentación de todas las operaciones de la API incluyendo las posibles peticiones y las respuestas recibidas**

Como se ha comentado anteriormente, desde la url base del despliegue de backend de Okteto se accede a la propia documentación de la API en Swagger:

- Enlace al despliegue backend: https://api-fis-wallet-d-rhym.cloud.okteto.net
- Enlace directo a la documentación: https://app.swaggerhub.com/apis-docs/D-RHYM/Wallet-Service/1.0

- **Debe tener persistencia utilizando MongoDB u otra base de datos no SQL.**

El microservicio Wallet-Service posee integración con MongoDB a través de Moongose. Puede comprobar dicha integración con la base de datos desde los archivos *index.js* y *db.js* en la carpeta raíz del proyecto.

```
const mongoose = require('mongoose');
const DB_URL = "mongodb://localhost:27017";

const dbConnect = function() {
    console.log("Trying to connect to: " + DB_URL);
    const db = mongoose.connection;
    db.on('error', console.error.bind(console, 'connection error: '));
    return mongoose.connect(DB_URL, { useNewUrlParser: true });
}

module.exports = dbConnect;
```

- **Validación de los datos antes de almacenarlos en la base de datos (por ejemplo, haciendo uso de mongoose).**

Dado que el usuario tan solo puede interactuar con las funciones de añadir fondo a la cartera, se ha implementado validación en ese método concreto para comprobar que el formato de los fondos cumpla con lo establecido en la aplicación. Adicionalmente y mediante mongoose, se han realizado otras validaciones en aquellos métodos cuyas urls exijan un Id para comprobar que dicho Id sea válido y para captar errores concretos en la transferencia de datos. Estos casos de validación son revisables en el archivo *server.js* de la carpeta raíz del proyecto. También pueden observarse en las imágenes del apartado anterior.

Por otro lado, se realiza validación mediante mongoose, como se observa en la siguiente imagen, perteneciente al modelo de la cartera.

```
var mongoose = require('mongoose');

const walletSchema = mongoose.Schema({

    user: String,

    fund: {  type: Number,
      set: v => {
        return new Number(v.toFixed(2));
      }},

    lastTransactions: Array,

    deleted: Boolean,

    createdAt: String,

    updatedAt: String

  });

module.exports = mongoose.model('Wallet', walletSchema);
```

Además, se ha realizado validación en frontend para que el input de PayPal solo pueda añadir números para introducir el dinero. Este desactiva el botón de PayPal cuando la cantidad introducida es 0, como se observa a continuación.

```
return (<>
    {(showSpinner && isPending) && <div className="spinner" />}
    <PayPalButtons
        style={style}
        disabled={amount <= 0}
        forceReRender={[amount, currency, style]}
        fundingSource={"paypal"}
```

Por último, se ha añadido una validación para que el mínimo valor numérico, y también el por defecto sea 0.

```
<NumberInput defaultValue={0} min={0} step={1.00} precision={2} bg="white" rounded='md' onChange={setAmount}>
    <NumberInputField />
```

- **Se debe utilizar gestión del código fuente y mecanismos de integración continua.**

Para la gestión del código fuente se ha hecho uso de un repositorio de Github concreto para el backend de Wallet-Service y un repositorio común a todos los servicios para el frontend:

- Enlace al repositorio backend: https://github.com/NFTsMarket/Wallet-Service.git
- Enlace al repositorio frontend: https://github.com/NFTsMarket/NFTsMarket-Frontend.git

La integración continua en el frontend se realiza mediante la propia aplicación de Netlify que permite desplegar automáticamente una rama concreta del repositorio. En cuanto a la integración continua en backend, se ha realizado mediante Github Actions, ejecutándose las pruebas del microservicio y desplegándose en Okteto a posteriori. El pipeline correspondiente al despliegue se encuentra en el archivo *okteto-deployment.yml* en la carpeta *.github/workflows* del repositorio:

- Enlace al archivo *okteto-deployment.yml* del respositorio: https://github.com/NFTsMarket/Wallet-Service/blob/master/.github/workflows/okteto-deployment.yml

- **Debe haber definida una imagen Docker del proyecto.**

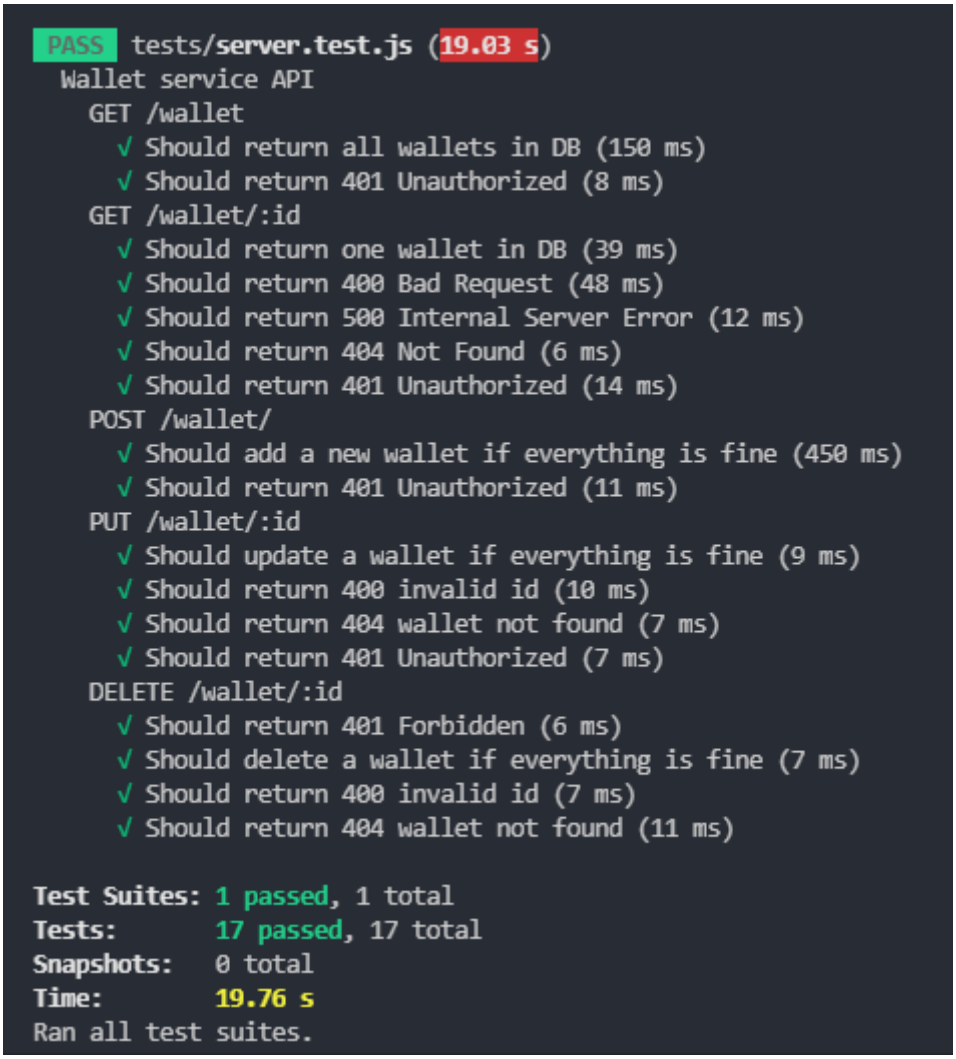
En la carpeta raíz del proyecto se encuentra el archivo *Dockerfile* encargado de crear la imagen de Docker del proyecto:

- Enlace al archivo *Dockerfile* del respositorio: https://github.com/NFTsMarket/Wallet-Service/blob/master/Dockerfile

- **Debe haber pruebas unitarias implementadas en Javascript para el código del backend utilizando Jest (el usado en los ejercicios) o Mocha y Chai o similar.**

Se han implementado un total de 17 pruebas unitarias sobre los métodos de la clase *server.js* antes mencionados, comprobando casos de éxito y error en la ejecución de llamadas a la API y las validaciones correspondientes al correcto funcionamiento de la base de datos. El archivo que contiene estos tests es *server.test.js*, dentro de la carpeta *tests*: *./tests/server.test.js*

- Enlace en github a server.test.js: https://github.com/NFTsMarket/Wallet-Service/blob/master/tests/server.test.js



- **Debe haber pruebas de integración con la base de datos.**

Se han realizado pruebas de integración sobre la base de datos. Estas pueden revisarse en el archivo *integration.spec.js*, dentro de la carpeta *tests*: *./tests/integration.spec.js*

- Enlace en github a integration.spec.js: https://github.com/NFTsMarket/Wallet-Service/blob/master/tests/integration.spec.js

2.2. Nivel de acabado hasta 9 puntos

Microservicio avanzado

- **Implementar un front end con rutas y navegación**

El frontend común a todos los microservicios hace uso de la librería next para el enrutado de la aplicación. En el caso del microservicio Wallet-Service, las vistas a las que se pueden acceder una vez iniciada sesión en la página son las siguientes:

- Enlace a vista de mi cartera: https://nftsmarket.netlify.app/wallet
- Enlace a vista de añadir fondos: https://nftsmarket.netlify.app/wallet/addFund

Para acceder a la vista de mi cartera, una vez iniciada sesión hay que pulsar sobre el botón de su usuario en la vista superior derecha y, en el menú que se desplegará, hay que pulsar en el apartado de *My wallet*.

Para acceder a la vista de añadir fondos, desde la vista de mi cartera basta con pulsar en el botón *Add funds* que se encuentra bajo sus fondos actuales. Desde esa vista, basta con añadir los fondos deseados y pulsar sobre el botón de Paypal para acceder a la ventana externa de pago con Paypal.

- **Uso del patrón materialized view para mantener internamente el estado de otros microservicios.**

El microservicio Wallet-Service hace uso del patrón materialized view gracias a la implementación de PubSub. Al crearse un usuario nuevo, Wallet-Service recibe la información de dicho usuario y su Id queda guardado en un atributo propio del modelo de Wallet, en la base de datos. Dicho atributo es usado para no realizar consultas adicionales al microservicio de autenticación, quedando asignada permanentemente dicha Wallet al usuario en cuestión. En la siguiente ilustración se puede observar cómo tiene lugar:

```
// Subscripciones a user
pubsub.subscription('wallet-created-user').on('message', message => {
    const user = JSON.parse(message.data.toString());
    console.log("MENSAJE" ,user);
    createWallet(user.id, 0, [], false, new Date().toISOString(), new Date().toISOString())
    message.ack()
});
```

Se puede observar que se emplean los datos del usuario (su id) para obtener su cartera en la siguiente ilustración:

```
// Obtener una cartera
app.get(BASE_API_PATH + "/wallet/:id", authorizedClient, (req, res) => {
    if (!ObjectId.isValid(req.params.id)) {
        return res.status(400).json("A wallet with that id could not be found, since it's not a valid id.");
    }

    var filter = { user: req.params.id };
    Wallet.findOne(filter, function (err, wallet) {
        if (err) {

            return res.status(500).json(err);
        } else if (wallet) {
            return res.status(200).json(wallet);
        } else {
            return res.status(404).json("A wallet with that id could not be found.");
        }
    });
});
```

- **Consumo de alguna API externa (distinta de las de los grupos de práctica).**

Se ha realizado la implementación de la API de Paypal en el microservicio con el objetivo de que sirva como medio para añadir fondos al usuario.

- **Tener el API REST documentado con swagger.**

Como se ha comentado anteriormente, la API está documentada en Swagger y es accesible desde la url base del despliegue en Okteto o desde su enlace concreto:

- Enlace al despliegue backend: https://api-fis-wallet-d-rhym.cloud.okteto.net
- Enlace directo a la documentación: https://app.swaggerhub.com/apis-docs/D-RHYM/Wallet-Service/1.0

- **Implementación de un mecanismo de autenticación basado en JWT o equivalente.**

Como se ha comentado anteriormente, todos los servicios (incluido Wallet-Service) hacen uso de JWT para la autenticación. Puede acceder a los archivos en cuestión en la carpeta *Middlewares* del proyecto o desde el siguiente enlace en el repositorio:

- Enlace a la carpeta *Middlewares* del repositorio: https://github.com/NFTsMarket/Wallet-Service/tree/master/middlewares

Aplicación basada en microservicio avanzado

- **Interacción completa entre todos los microservicios de la aplicación integrando información.**

Se han integrado todos los microservicios de la aplicación mediante un mecanismo PubSub.

- **Tener un front end común que integre los front ends de cada uno de los microservicios.**

Se ha desarrollado un front end común al que puede acceder desde las siguientes direcciones:

- Enlace al repositorio frontend: https://github.com/NFTsMarket/NFTsMarket-Frontend.git
- Enlace al despliegue de frontend: https://nftsmarket.netlify.app/

- **Definición de un customer agreement para la aplicación en su conjunto.**

Se ha desarrollado un documento en relación a este punto:

https://drive.google.com/file/d/1sHw3xc1Q5fgdCtxRmxt03c6535xWgW7u/view?usp=drivesdk

- **Hacer uso de un sistema de comunicación asíncrono mediante un sistema de cola de mensajes para todos los microservicios. Si no es para todos, debe justificarse de forma razonada.**

Se ha implementado un sistema PubSub de envío y suscripción de datos. La información recibida proviene de los microservicios de autenticación, de quienes se recibe la información del usuario que se crea al hacer sign-up para proceder con la creación de su correspondiente cartera, y el microservicio de compra, con quienes se intercambia información relativa a los fondos de la cartera a la hora de proceder con la compra-venta de productos. La definición del sistema PubSub se realiza

en los archivos *pubsub.js* y *pubsubMessages.js* en la carpeta raíz del proyecto, mientras que su implementación se realiza en el archivo *server.js*:

- Enlace al archivo *server.js* del respositorio: https://github.com/NFTsMarket/Wallet-Service/blob/master/server.js
- Enlace al archivo *pubsub.js* del respositorio: https://github.com/NFTsMarket/Wallet-Service/blob/master/pubsub.js
- Enlace al archivo *pubsubMessages.js* del respositorio: https://github.com/NFTsMarket/Wallet-Service/blob/master/pubsubMessages.js

```
// Subscripciones a user
pubsub.subscription('wallet-created-user').on('message', message => {
    const user = JSON.parse(message.data.toString());
    console.log("MENSAJE" ,user);
    createWallet(user.id, 0, [], false, new Date().toISOString(), new Date().toISOString())
    message.ack()
});

pubsub.subscription('wallet-deleted-user').on('message', message => {
    const user = JSON.parse(message.data.toString());
    console.log("MENSAJE DELETE" ,user);
    Wallet.findOneAndDelete({ user: user.id }, function (err, wallet) {
        if(wallet){
            sendMessageDeletedWallet(wallet);
        }
    });

    message.ack()
});

// Subscripciones a compra
pubsub.subscription('wallet-created-purchase').on('message', message => {
    const user = JSON.parse(message.data.toString());
    let buyerId = user.buyerId;
    let cost = user.amount;
    addAmountToUserWallet(buyerId, -cost);
    message.ack()
});

pubsub.subscription('wallet-updated-purchase').on('message', message => {
    const user = JSON.parse(message.data.toString());
    let sellerId = user.sellerId;
    let cost = user.amount;
    addAmountToUserWallet(sellerId, cost);
    message.ack()
});

pubsub.subscription('wallet-deleted-purchase').on('message', message => {
    const user = JSON.parse(message.data.toString());
    let buyerId = user.buyerId;
    let cost = user.amount;
    addAmountToUserWallet(buyerId, cost);
    message.ack()
});
```

- **Implementación de un mecanismo de autenticación homogéneo para todos los microservicios.**

Se ha implementado un sistema de autenticación mediante JWT en todos los servicios del sistema como se ha ejemplificado en varios de los puntos anteriores.

- Enlace a la carpeta *Middlewares* del repositorio: https://github.com/NFTsMarket/Wallet-Service/tree/master/middlewares

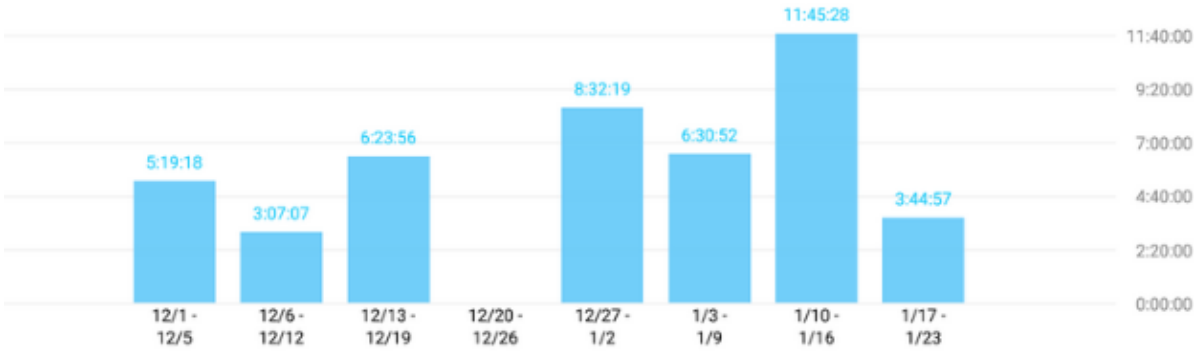
3. Análisis de esfuerzo

Summary Report

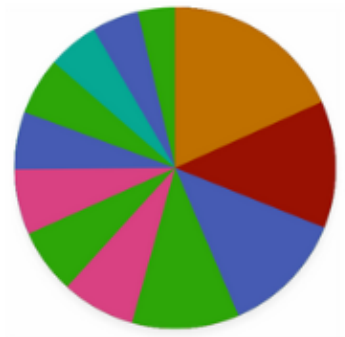
toggl track

12/01/2021 – 01/23/2022

TOTAL HOURS: 45:23:57



PROJECT	DURATION
FIS	45:23:57

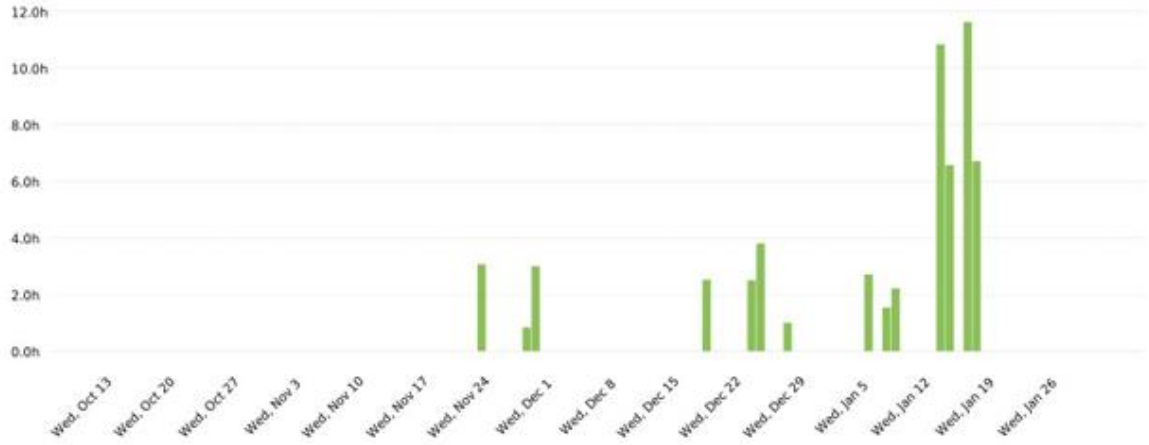


TIME ENTRY	DURATION
Tarea-022: Pequeñas correcciones	8:16:08
Tarea-013: Implementación de Paypal	5:48:29
Tarea-18: Integración con pubsub	5:39:36
Tarea-013: Implementación de Paypal y Tarea-011: Crear vista de añadir crédito	4:56:12
Tarea-20: Integración de FrontEnd y BackEnd	3:21:45
Tarea-001: Creación de los CRUD	2:57:25
Tarea-011: Crear vista de añadir crédito	2:57:16
Tarea-006: Realizar Mock-Ups	2:39:42
Tarea-19: Configuración de JSON Web Tokens	2:35:58
Configuración el entorno	2:21:53
Tarea-010: Añadir pruebas de integración	2:07:50
Definición y reparto de tareas	1:41:43

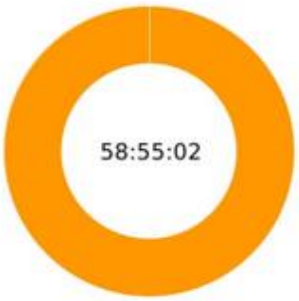
Summary report

13/10/2021 - 29/01/2022

Total: 58:55:02 Billable: 58:55:02 Amount: 0.00 USD

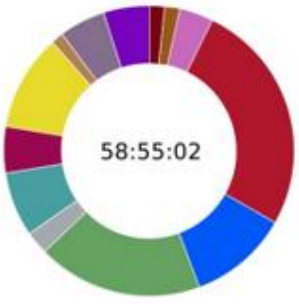


Project



FIS	58:55:02	100.00%

Description



Tarea-003	03:00:00	5.09%
Tarea-023	03:00:14	5.09%
Mock-up design	00:50:00	1.41%
Tarea-004	06:18:00	10.69%
Set-up workspace	03:03:50	5.19%