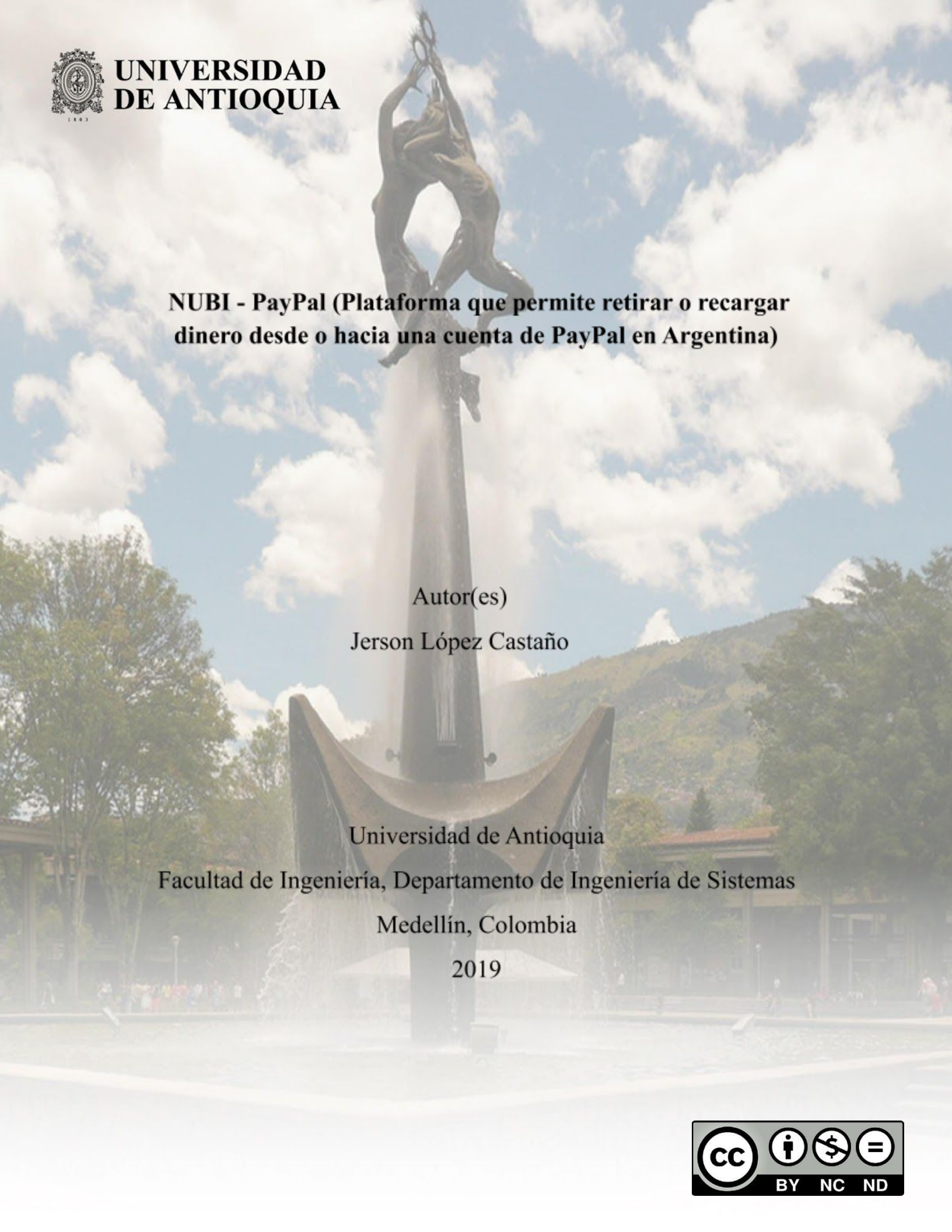


UNIVERSIDAD DE ANTIOQUIA

NUBI - PayPal (Plataforma que permite retirar o recargar dinero desde o hacia una cuenta de PayPal en Argentina)

Autor(es)

Jerson López Castaño

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Ingeniería de Sistemas

Medellín, Colombia

2019

NUBI - PayPal (Plataforma que permite retirar o recargar dinero desde o hacia una cuenta de PayPal en Argentina)

Jerson López Castaño

Informe de práctica o monografía o investigación o tesis o trabajo de grado
como requisito para optar al título de:
Ingeniería de Sistema.

Asesores (a) o Director(a) o Co- Directores(a).

Carlos Mauricio Duque Restrepo, Ingeniero de Sistemas

Universidad de Antioquia
Facultad de Ingeniería, Departamento Ingeniería de Sistemas
Medellín, Colombia
2019.

Resumen
En un mundo globalizado como el actual, toda empresa prestadora de servicios quiere lanzar su api para así estar en sintonía con el mercado que cada día pide más y más integración entre los diferentes negocios de los diferentes nichos, para muestra de esto tenemos a PayPal quien gracias a su api muchas empresas alrededor del mundo pueden hacerse partnership y crear soluciones innovadoras e integradas al core de PayPal. Este es el caso de nubi, una solución innovadora de y para Argentina en la cual su gente puede mover dólares al exterior del país o traerlos desde el exterior de ser necesario.

El problema lo encontramos cuando a medida que el tiempo avanza, cada día encontramos una mejor manera de hacer las cosas, y lo que en algún momento fue lo mejor lentamente se está deprecando y pasando a la historia, víctima de esto es nubi, que teniendo un sistema completamente funcional le tocó hacerse a la titánica tarea de reescribir su integración con PayPal gracias a que estos últimos dieron un paso adelante en su api y todo lo existente de ahí para atrás lentamente iría desapareciendo y si no se avanza con ellos entonces se está fuera del negocio.

Introducción
NUBI propone una solución tecnológica e innovadora mediante una integración entre PayPal y el banco Comafi de Argentina a través de una plataforma, permitiéndole a las personas de dicho país poder cargar o descargar sus cuentas de PayPal desde su cuenta de banco local, tomando esto como la principal característica de NUBI ya que cuando el proyecto inicio y hasta que el producto se lanzó en 2016 no existía uno similar que resolviera esta necesidad.

Con el módulo de retiros se incentiva a las PYMEs, freelancers y emprendedores a que abran sus puertas al resto del mundo, facilitando el cobro de sus ventas y retirando su saldo de PayPal directamente a una cuenta bancaria local, y con el módulo de recargas se ayuda a quienes no tienen tarjeta de crédito internacional a que puedan comprar online transfiriendo directamente desde su cuenta bancaria para recargar saldo en PayPal.

Objetivos

Objetivo General:
Desarrollar la api (Application Programming Interface) que se encargue de hacer la integración y el consumo de la nueva versión en los servicios de Paypal api (PayPal BMW2.0), velando por la calidad de servicio que actualmente caracteriza a NUBI y que el flujo del mismo no se vea afectado.

Objetivos Específicos:
- Investigar y aplicar buenas prácticas de programación en la construcción de la nueva api
- Hacer que el cambio de versión en los servicios de PayPal sea transparente al usuario.

Marco Teórico

Una interfaz de programación de aplicaciones o mejor conocida como API por sus siglas en inglés (Application Programming Interface), es un conjunto de reglas que permiten a los programas hablar entre sí mediante dos atributos denominados **request** and **response**. Request o solicitud se le llama a todos los parámetros de búsqueda del recurso que se quiere obtener, mientras que los datos que se devuelven sobre dicho recurso se llaman response o respuesta.

Es importante saber que una solicitud se compone de cuatro cosas:

1. The endpoint
2. The method
3. The headers
4. The body (or data)

También es importante hacer uso de las buenas prácticas de desarrollo para saber cómo y cuándo usar cada una de las cosas mencionadas:

El endpoint o ruta determina el recurso que se está solicitando. Se podría ver como un contestador automático que te pide que presiones 1 para un servicio, presiones 2 para otro servicio, 3 para otro servicio y así sucesivamente siendo cada número un endpoint diferente que a su vez accede a un recurso diferente.

El method o método es el tipo de solicitud que envía al servidor. Puede elegir entre estos 4 tipos principales a continuación: POST, GET, PUT, DELETE. Estos métodos proporcionan un significado para la solicitud que está realizando, ya sea que se quiera guardar, leer, actualizar o borrar un registro respectivamente.

Los headers o encabezados se utilizan para proporcionar información tanto al cliente como al servidor. Puede usarse para muchos propósitos, como la autenticación y el suministro de información sobre el contenido del cuerpo de la solicitud.

El body o los datos contienen la información que desea enviar al servidor sobre un determinado recurso, esta opción sólo se utiliza con las solicitudes que hacen uso de los métodos POST, PUT o DELETE. Para los GET el atributo body no existe.

NUBI permite a las personas de dicho país (Argentina) poder cargar o descargar sus cuentas de PayPal desde su cuenta de banco local. Para llevar a cabo esta meta se hizo uso de una arquitectura de micro servicios ya que permite tener una lógica descentralizada, una mayor integración y escalabilidad con aplicaciones de terceros y, además que es más versátil y se pueden integrar varias tecnologías y lenguajes dependiendo de las necesidades. Para este caso solo nos centraremos en uno de estos micro servicios el cual tiene la responsabilidad de hacer la integración con PayPal (información del usuario, balance, retiros y recargas de saldo), es decir, va estar trabajando con la información que nos proveerá un tercero.

Desde los inicios del proyecto después de un respectivo estudio e investigación se tomó la decisión de que esta api se debía escribir en las tecnologías de NodeJS: gracias a su core asíncrono y su excelente manejo de concurrencia[1] ya que esto no generaría un bloqueo en el flujo de la aplicación mientras se espera la respuesta de las apis de PayPal ni por la cantidad de usuarios concurrentes, npm: es el registro de software más grande del mundo. Los desarrolladores de código abierto de todos los continentes usan npm para compartir y pedir prestados paquetes, y muchas organizaciones usan npm para administrar el desarrollo privado también[2], y postgreSQL: es un potente sistema de base de datos relacional de objetos de código abierto con más de 30 años de desarrollo activo que le ha ganado una sólida reputación de confiabilidad, solidez de características y rendimiento[3].

Metodología

El proyecto se llevó a cabo bajo una metodología scrum, la cual se define como "es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto"[8].

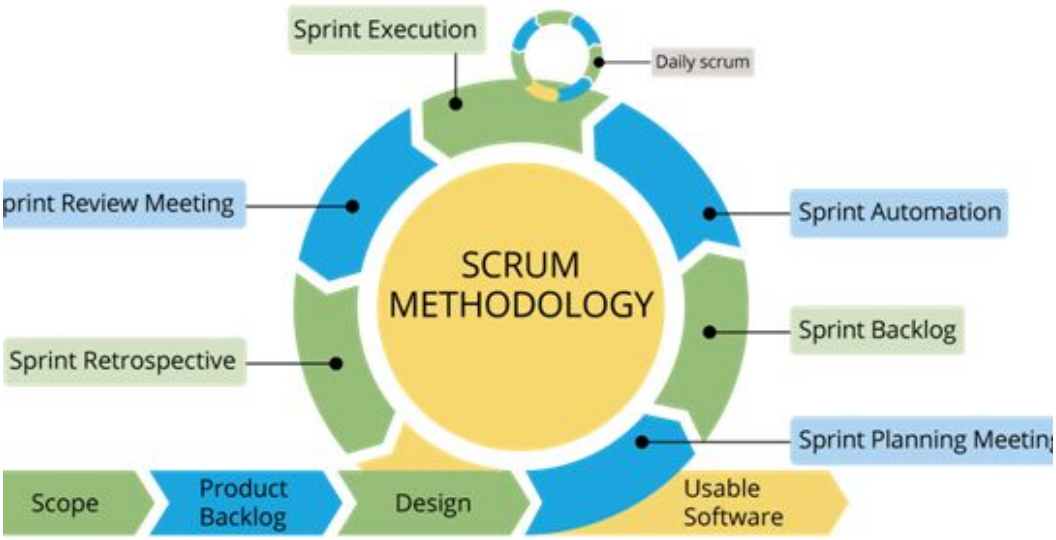


Fig 1. ¿Qué es Scrum?

En esta metodología se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Haciendo una adaptación de la metodología scrum en el proyecto y siguiendo con la metodología de trabajo manejada en la compañía, se realizó un Product Backlog de actividades para identificar las tareas a desarrollar con el fin de cumplir con el objetivo del

proyecto. Entre los tiempos estipulados en el cronograma de actividades, se tomaron algunas tareas a analizar y desarrollar en interacciones de dos semanas las cuales estuvieron plasmadas en el tablero representadas con tarjetas. Adicionalmente, se realizaron reuniones periódicas con el asesor y las personas del equipo que pudieran ayudar a alcanzar el objetivo. Se hicieron retrospectivas cada dos semanas con el asesor interno para verificar avances en los tiempos y progreso en el desarrollo, además semanalmente se llevaron a cabo reuniones con el cliente para evidenciar avances y refinar funcionalidades.

Resultados y análisis

Mis prácticas académicas las puedo dividir en 3 puntos principales, los cuales detallo a continuación:

- Aprendizaje:

Las primeras semanas me fueron asignadas historias de usuario que no contemplaban en lo absoluto la integración con la nueva api de PayPal, pero que fueron de vital importancia para entender la arquitectura actual del proyecto, pude entender los microservicios que se utilizaban en este y la responsabilidad de cada uno de ellos, tanto en el lado del backend como el del frontend, y además de esto entender los flujos de operación y cómo se iban orquestando todas las apis entre sí para llegar a trabajar como un único músculo para formar todo lo que es NUBI.

Adicional al código de backend que tuve que empezar a conocer y a entender se le sumó el paradigma de docker, el cual no fue muy complicado para mi porque en ocasiones pasadas ya había tenido la oportunidad de trabajar un poco con él, sin embargo si me fue muy útil este tiempo de aprendizaje para fortalecer mis conocimientos y aprender muchos otros que no tenía ni idea que se podían llegar a realizar con la ayuda de tan buena herramienta.

- Infraestructura:

Fue quizá personalmente el reto más importante que se logró realizar en las prácticas, y la cual tomó bastante tiempo implementar ya que la curva de aprendizaje que se requería era bastante amplia, e iba de la mano con el proceso de despliegue y de integración continua de la plataforma. Aunque este no fue el objetivo general ni uno de los específicos de las prácticas académicas, se logró dejar funcionando adecuadamente y representó gran inversión de tiempo y de mucho aprendizaje lo cual considere importante mencionar.

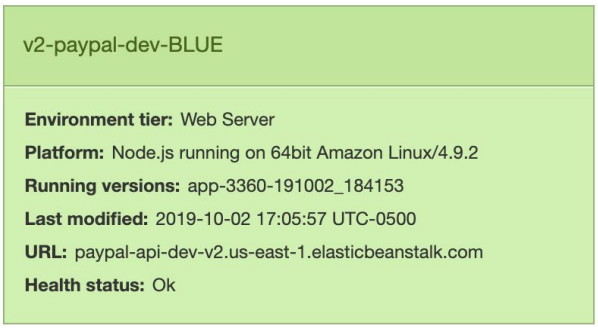
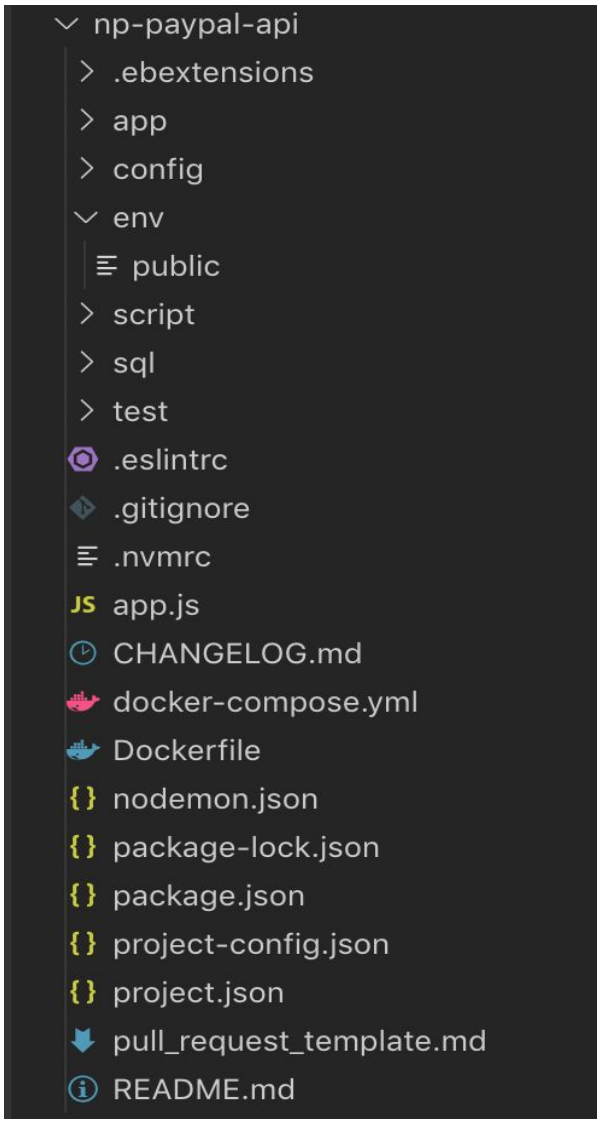


Fig 2. Vista de la aplicación corriendo en aws

- Backend:

Para llevar a cabo el objetivo planteado, fue necesario modificar la api existente, responsable de la integración con PayPal, dado que por requerimientos del cliente debíamos seguir trabajando sobre el mismo repositorio de código (cabe destacar que todos los repositorios del proyecto se encuentran en BitBucket) por lo cual después de varias reuniones con el cliente y con los lideres tecnicos de la compañía se decidió de la mejor opción sería versionar la aplicación y haciendo uso del branching model empezar a hacer en una rama auxiliar a la *development* el desarrollo de la nueva api. Dado que en la compañía se cuenta con un estándar de buenas prácticas[6] y un bootstrap o proyecto base para los desarrollos de NodeJs[7], se pasó de tener la estructura de carpetas observada en la *Fig 3: Estructura de carpetas versión 1* a tener la mostrada en la *Fig 4: Estructura de carpetas versión 2.*


np-paypal-api
.ebextensions
app
config
env
public
script
sql
test
.eslintrc
.gitignore
.nvmrc
app.js
CHANGELOG.md
docker-compose.yml
Dockerfile
nodemon.json
package-lock.json
package.json
project-config.json
project.json
pull_request_template.md
README.md

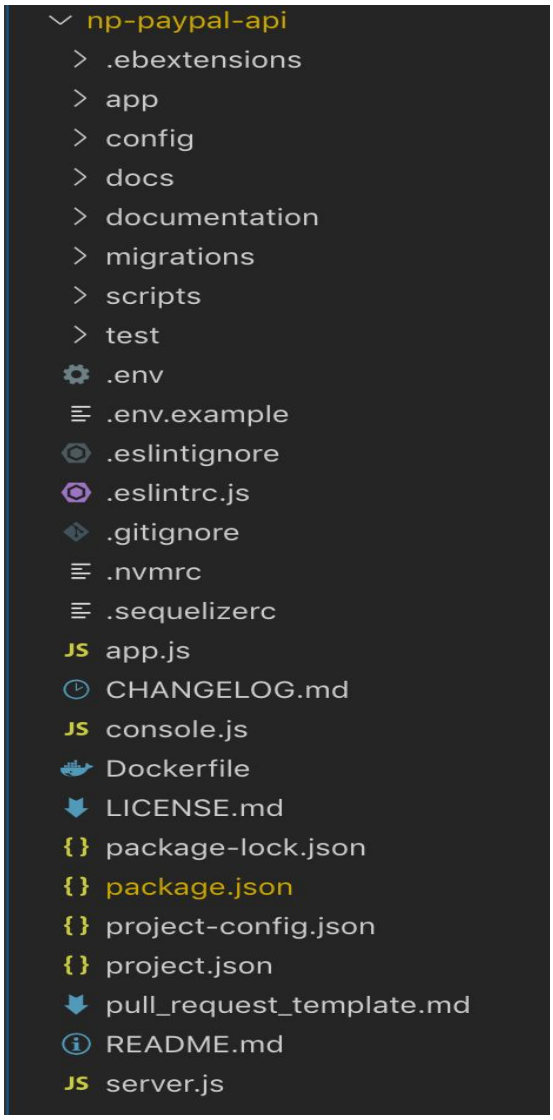


Fig 3: Estructura de carpetas versión 1

Fig 4: Estructura de carpetas versión 2

En las dos imágenes anteriores se pueden observar diferencias notables como el que las variables de entorno para desarrollo son un archivo y no una carpeta, que el levantamiento del servidor está dividido en dos archivos, uno es el *server.js* y el otro es *app.js* que se encarga de orquestar toda la configuración de rutas, migraciones, modelos, middlewares y finalmente levanta el servidor con la ayuda de *server.js.* Además se agrego la carpeta de *migrations* y las respectivas migraciones necesarias para levantar la versión 2 de la api, ya que una de las falencias presentadas en la versión 1 como nos muestra la imagen es que en la versión anterior no se contaba con estas. La ausencia de estas hace que hacer modificaciones en la base de datos sea muy rústico y complejo, ya que si ocurría un error no había forma de hacer un rollback de este. Otra mejora que podemos observar a nivel de arquitectura es el nombramiento de archivos como se puedo observar en la *Fig 5: Nombramiento de archivos,* donde en la parte izquierda se puede ver la forma en como se nombraba en la versión 1, y la cantidad de archivos

innecesarios que había ya fuera porque no se usarán o porque pertenecían al dominio de un usuario, en la parte derecha se puede notar que para el nombramiento en la versión 2 se usan únicamente palabras en singular para los archivos y en plural para las carpetas, como lo estipula la tech guide de la compañía.

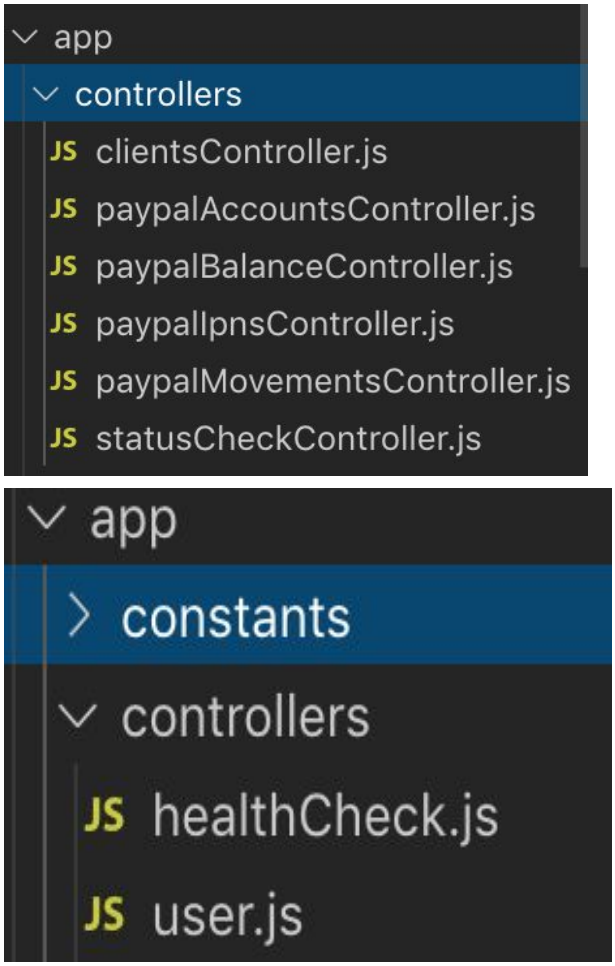


Fig 5: Nombramiento de archivos

Además se agregó la capa de *schemas* para la validación de todas las peticiones que llegan a la api. Se valida que cada petición posea los campos requeridos y el tipo de dato que se espera en cada uno de ellos. Adicionalmente agregamos la capa de *mappers,* encargada de dar formato a toda la información que llega a la api y volverla procesable para nuestro flujo (controllers, services, models…). Por último, se agregó la capa de *serializers,* en la cual le damos formato a todas las respuestas de nuestra api para que posean el estándar definido en las tech guides para todas las tecnologías backend que ofrece la compañía. Dichas capas se pueden evidenciar en la *Fig 6: Capa de mappers, serializers y schemas.*

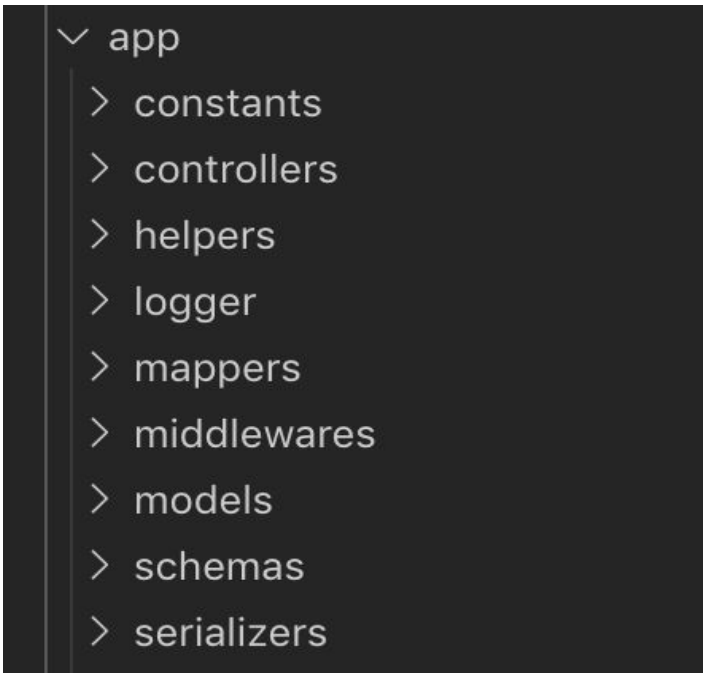


Fig 6: Capa de mappers, serializers y schemas

Algo más que se añadió y no menos importante es la capa de *helpers,* denominados como métodos que son externos a la lógica de negocio, que pueden funcionar independientemente del proyecto al que se sumen y los cuales si puedo mostrar porque no afectan el acuerdo de confidencialidad acordado con la empresa. en *Fig 7: Definición de helpers* podemos observar un par, donde por reglas del negocio de nubi se debe manejar los montos de dinero en centavos, en este archivo se ven los dos métodos que nos permiten llevar de dólares a centavos y viceversa, y como mencione anteriormente pueden ser utilizados en cualquier proyecto que requiera lo misma ya que no dependen de nada más.

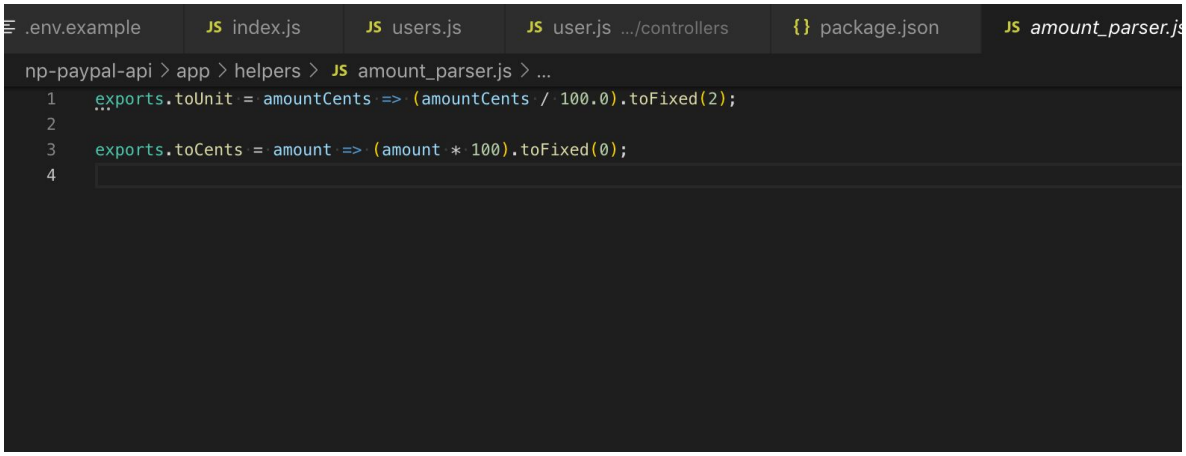
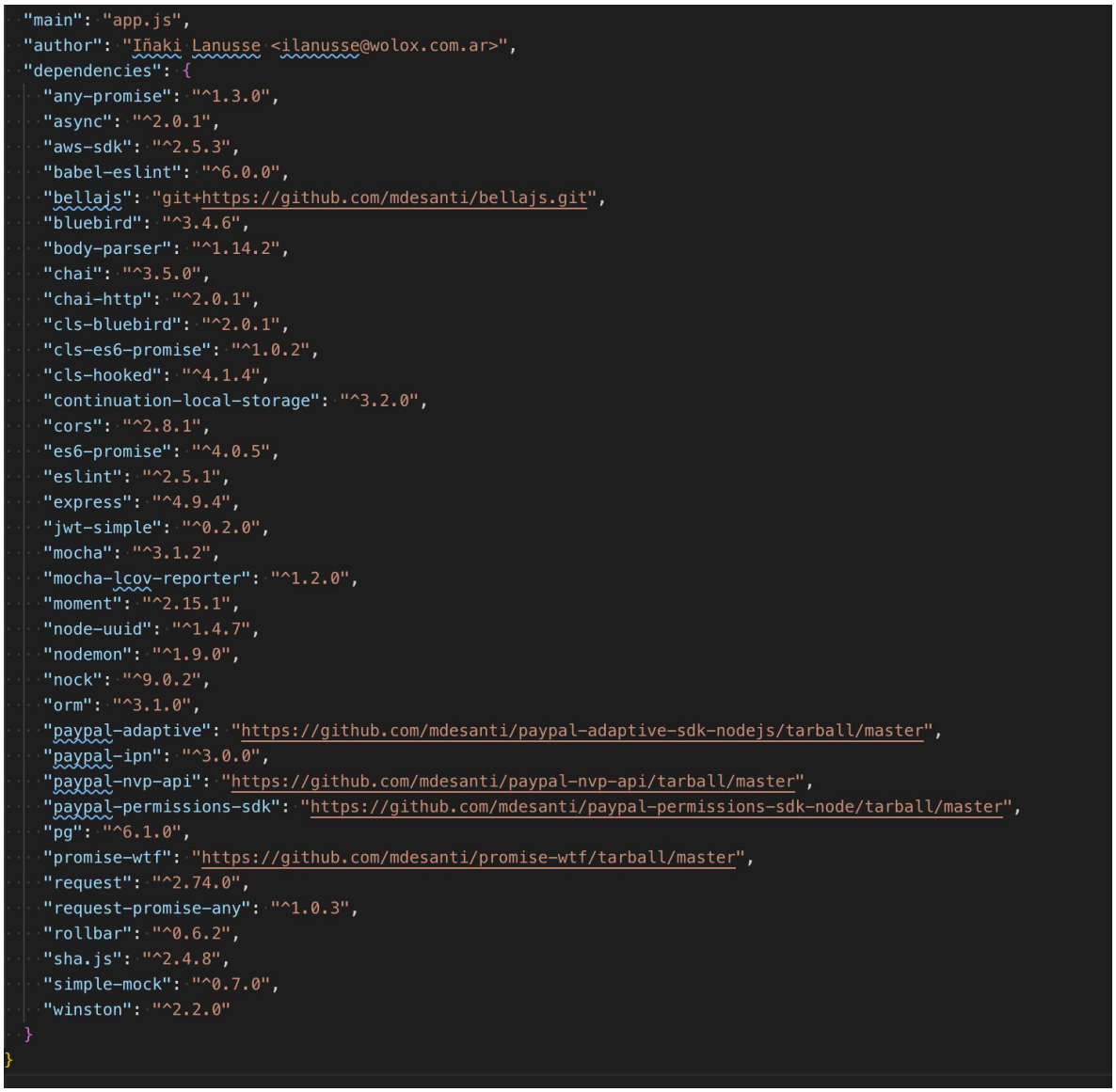


Fig 7: Definición de helpers

Dejando a un lado la estructura del proyecto pasemos a observar los cambios funcionales que se añadieron en la nueva versión de la api, empecemos por lo más importante en todo proyecto de NodeJS, el *package.json* que se podría tomar como el archivo manifiesto del proyecto, el que contiene el registro de todas la dependencias con sus versiones necesarias para el correcto funcionamiento.

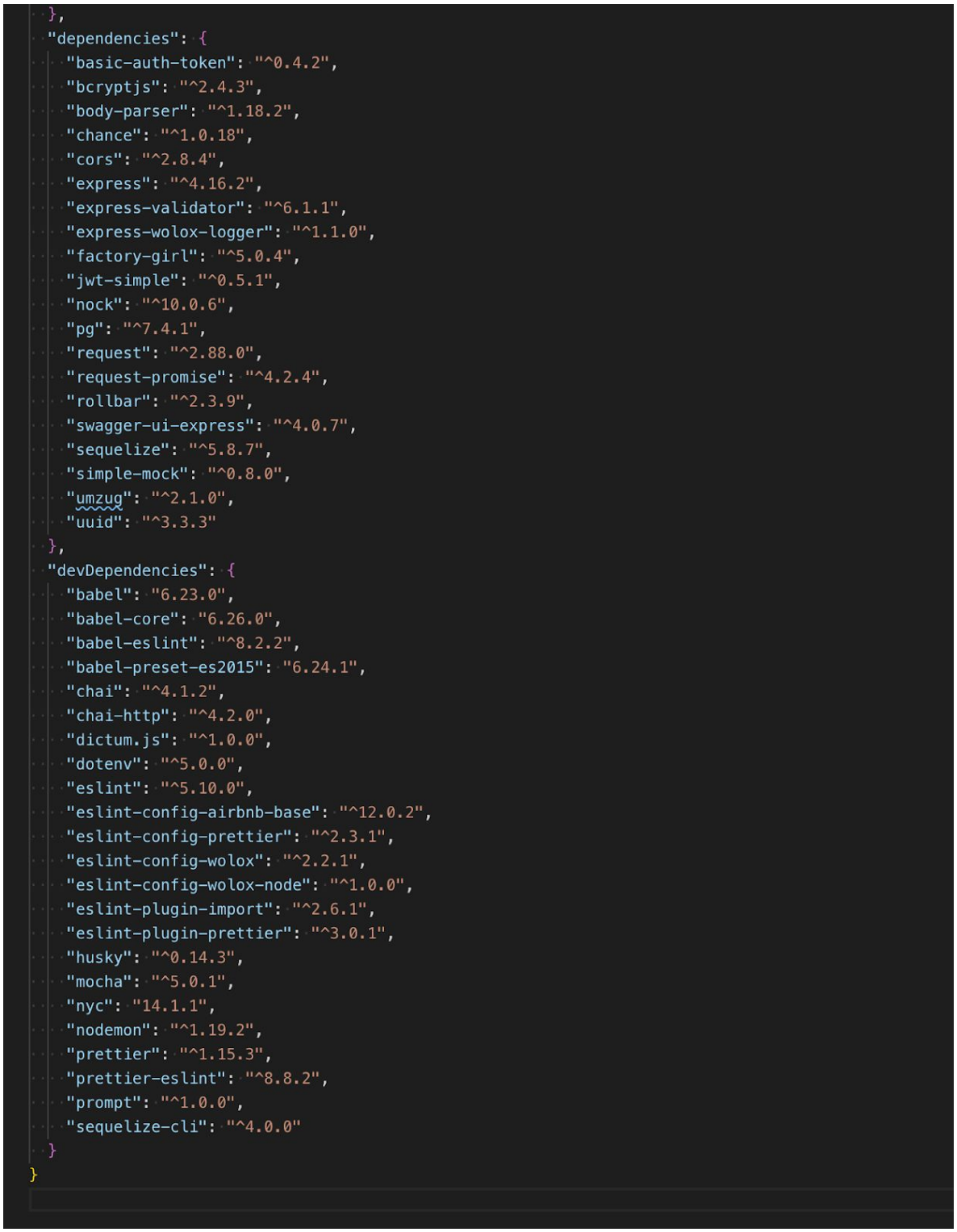
En *Fig 8: Package.json de la versión 1*. Se listan todas las dependencias que existían en el proyecto. Muchas de estas dependencias estaban deprecadas, otras no se usaban en la aplicación y algunas otras ni siquiera estaban en npm (node package manager) porque este ya no las soportaba y tenían que estar situadas en repositorios personales de alguien que si en algún momento se le ocurría borrarlos, no solo se perderían de su perfil sino que también se iría abajo todo el proyecto de NUBI ya que estas son las que conforman el sdk encargado de la integración con PayPal como lo son: *paypal-adaptive, paypal-permission-sdk, paypal-nvp-api, bellajs* y, *promise-wtf*. Además se puede observar que no se tenían separadas las dependencias que se usan para desarrollo que nada tiene que ver con el ambiente productivo ya que están estrictamente desarrolladas para agilizar y ayudar a los programadores en la etapa de desarrollo.



```
"main": "app.js",
"author": "Iñaki Lanusse <ilanusse@wolox.com.ar>",
"dependencies": {
  "any-promise": "^1.3.0",
  "async": "^2.0.1",
  "aws-sdk": "^2.5.3",
  "babel-eslint": "^6.0.0",
  "bellajs": "git+https://github.com/mdesanti/bellajs.git",
  "bluebird": "^3.4.6",
  "body-parser": "^1.14.2",
  "chai": "^3.5.0",
  "chai-http": "^2.0.1",
  "cls-bluebird": "^2.0.1",
  "cls-es6-promise": "^1.0.2",
  "cls-hooked": "^4.1.4",
  "continuation-local-storage": "^3.2.0",
  "cors": "^2.8.1",
  "es6-promise": "^4.0.5",
  "eslint": "^2.5.1",
  "express": "^4.9.4",
  "jwt-simple": "^0.2.0",
  "mocha": "^3.1.2",
  "mocha-lcov-reporter": "^1.2.0",
  "moment": "^2.15.1",
  "node-uuid": "^1.4.7",
  "nodemon": "^1.9.0",
  "nock": "^9.0.2",
  "orm": "^3.1.0",
  "paypal-adaptive": "https://github.com/mdesanti/paypal-adaptive-sdk-nodejs/tarball/master",
  "paypal-ipn": "^3.0.0",
  "paypal-nvp-api": "https://github.com/mdesanti/paypal-nvp-api/tarball/master",
  "paypal-permissions-sdk": "https://github.com/mdesanti/paypal-permissions-sdk-node/tarball/master",
  "pg": "^6.1.0",
  "promise-wtf": "https://github.com/mdesanti/promise-wtf/tarball/master",
  "request": "^2.74.0",
  "request-promise-any": "^1.0.3",
  "rollbar": "^0.6.2",
  "sha.js": "^2.4.8",
  "simple-mock": "^0.7.0",
  "winston": "^2.2.0"
}
}
```

Fig 8: Package.json de la versión 1.

En *Fig 9: Package.json de la versión 2.* se puede evidenciar la correcta separación entre las dependencias usadas en la etapa de desarrollo bajo la llave ***devDependencies*** que nos ofrece NodeJS como herramienta para dicho propósito. dentro de estas herramientas tenemos las dependencias de testing, de documentación, y los respectivos linters para asegurar la calidad y la limpieza del código, si observamos con detenimiento podemos observar que en la compañía tenemos nuestro propio linter y uno específico para NodeJS. además el número de paquetes disminuyó considerablemente ya que se dejaron las estrictamente necesarias para el correcto funcionamiento de la api y todos quedaron con versiones estables y actualizadas.



```
  },
  "dependencies": {
    "basic-auth-token": "^0.4.2",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.18.2",
    "chance": "^1.0.18",
    "cors": "^2.8.4",
    "express": "^4.16.2",
    "express-validator": "^6.1.1",
    "express-wolox-logger": "^1.1.0",
    "factory-girl": "^5.0.4",
    "jwt-simple": "^0.5.1",
    "nock": "^10.0.6",
    "pg": "^7.4.1",
    "request": "^2.88.0",
    "request-promise": "^4.2.4",
    "rollbar": "^2.3.9",
    "swagger-ui-express": "^4.0.7",
    "sequelize": "^5.8.7",
    "simple-mock": "^0.8.0",
    "umzug": "^2.1.0",
    "uuid": "^3.3.3"
  },
  "devDependencies": {
    "babel": "6.23.0",
    "babel-core": "6.26.0",
    "babel-eslint": "^8.2.2",
    "babel-preset-es2015": "6.24.1",
    "chai": "^4.1.2",
    "chai-http": "^4.2.0",
    "dictum.js": "^1.0.0",
    "dotenv": "^5.0.0",
    "eslint": "^5.10.0",
    "eslint-config-airbnb-base": "^12.0.2",
    "eslint-config-prettier": "^2.3.1",
    "eslint-config-wolox": "^2.2.1",
    "eslint-config-wolox-node": "^1.0.0",
    "eslint-plugin-import": "^2.6.1",
    "eslint-plugin-prettier": "^3.0.1",
    "husky": "^0.14.3",
    "mocha": "^5.0.1",
    "nyc": "14.1.1",
    "nodemon": "^1.19.2",
    "prettier": "^1.15.3",
    "prettier-eslint": "^8.8.2",
    "prompt": "^1.0.0",
    "sequelize-cli": "^4.0.0"
  }
}
```

Fig 9: Package.json de la versión 2.

Ahora vayamos a lo realmente importante, la integración con PayPal, para ello en la versión 2 tenemos un servicio que se encarga de esta tarea del cual se muestra una

porción del código en la *Fig 10: servicio para la comunicación con paypal de la versión 2.*

```
const request = require('request-promise'),
  tokenGeneration = require('basic-auth-token'),
  constants = require('../constants'),
  { paypalErrorBuilder } = require('../utils/paypal_error_builder'),
  logger = require('../logger'),
  errors = require('../errors'),
  uuidv4 = require('uuid/v4');

Complexity is 4 Everything is cool!
exports.accessToken = (clientId, secret) => {
  const token = tokenGeneration(clientId, secret);

  const options = {
    method: 'POST',
    url: `${constants.PAYPAL_URL}/v1/oauth2/token`,
    headers: {
      Authorization: `Basic ${token}`,
      Accept: 'application/json',
      'Content-Type': 'application/json'
    },
    form: {
      grant_type: 'client_credentials',
      content_type: 'application/x-www-form-urlencoded'
    },
    json: true
  };

  return request(options).catch(err => {
    const paypalError = paypalErrorBuilder(err);
    logger.error(`Error while trying to get an app access token: ${paypalError.message}`);
    throw errors.paypalError(paypalError);
  });
};

Complexity is 4 Everything is cool!
exports.tokenService = (appAccessToken, code) => {
  const options = {
    method: 'POST',
    url: `${constants.PAYPAL_URL}/v1/oauth2/token`,
    headers: {
      Authorization: `Bearer ${appAccessToken}`,
      Accept: 'application/json',
      'Content-Type': 'application/json'
    },
    form: {
      grant_type: 'authorization_code',
      content_type: 'application/x-www-form-urlencoded',
      code
    },
    json: true
  };
```

Fig 10: servicio para la comunicación con paypal de la versión 2.

Como notamos en la imagen anterior, la nueva integración con PayPal esta basada en tokens, está centralizado en un solo archivo y esta vez no se requiere el uso de un

sdk como se observa en *Fig 11: sdk para la comunicación con paypal de la versión 1.* Que se necesitaba en la versión 1 de la api y que además se encuentra fragmentado en varios archivos más como se puede ver en *Fig 12: servicios para la comunicación con paypal de la versión 1.* En dicha versión, la comunicación misma era compleja e inmantenible.

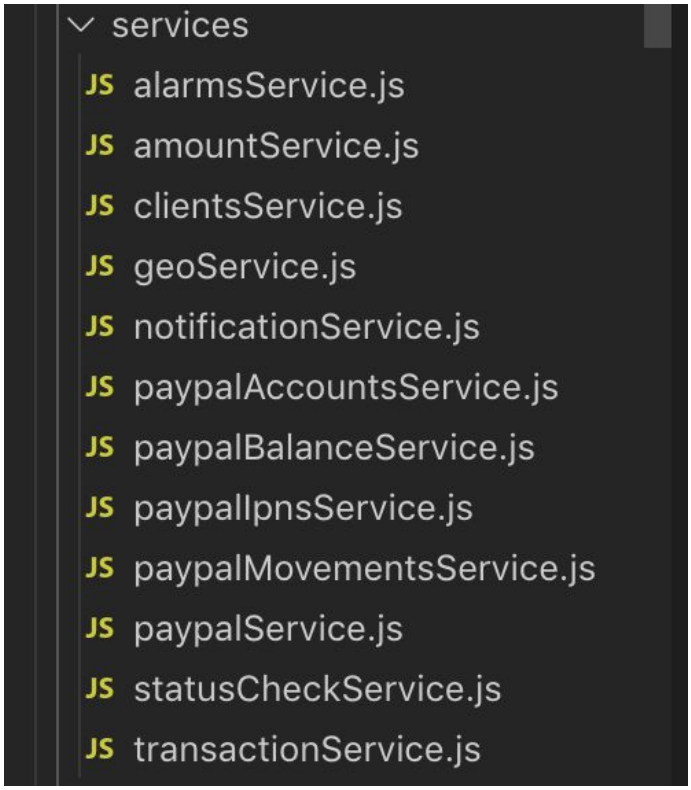
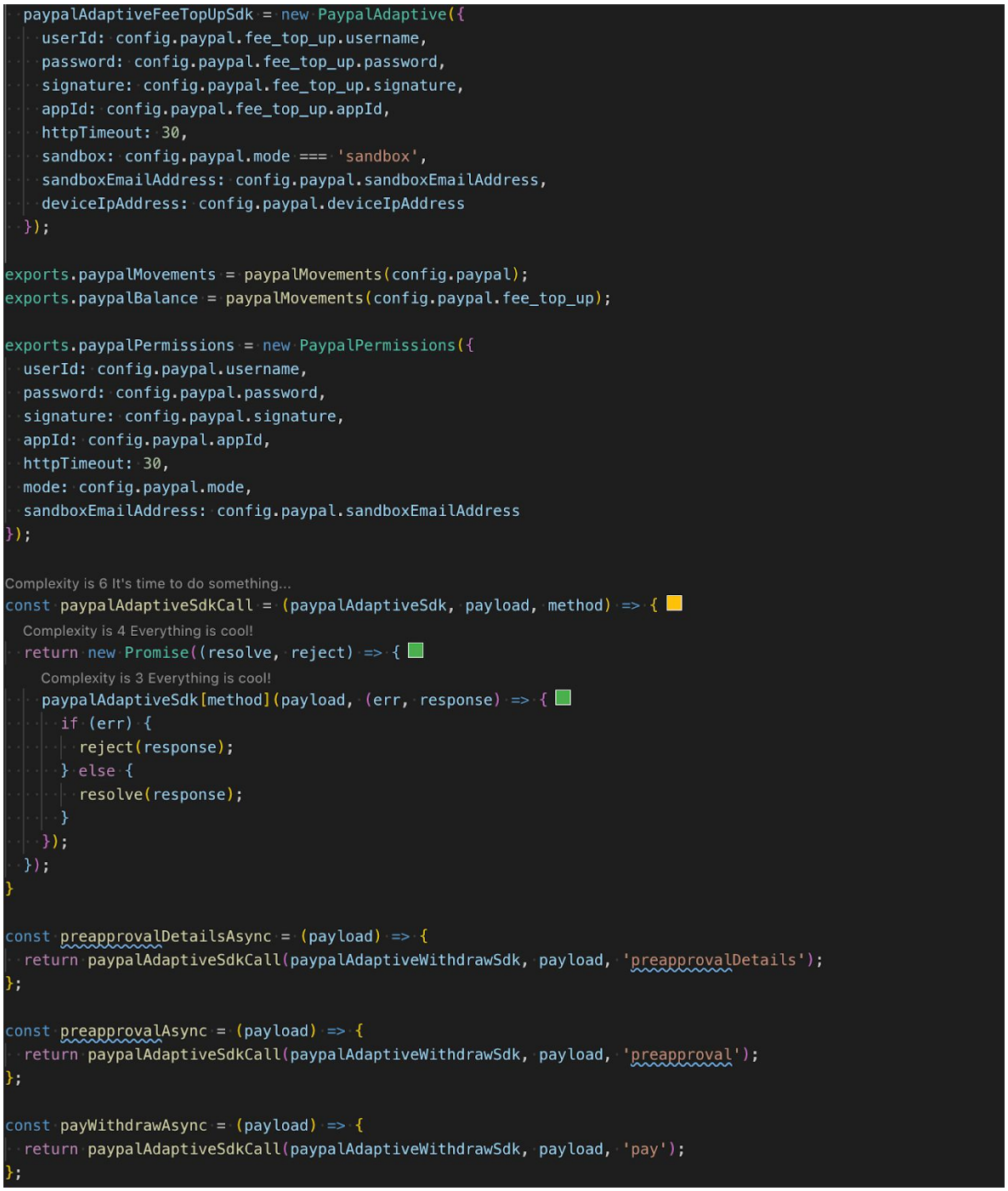


Fig 12: servicios para la comunicación con paypal de la versión 1.

En la imagen anterior se puede notar que tenemos 5 servicios para poder hacer la conexión con paypal, en parte esto era necesario debido a que la información de PayPal tampoco estaba alojada en un solo lugar y había que hacer llamado a múltiples recursos para poder recolectar la información de un usuario y la necesaria para sus operaciones. La mejora en nuestra api no fue solo una iniciativa propia de NUBI sino que también venía acompañada de una decisión tomada por parte de PayPal de centralizar su información en un solo lugar, esto condujo a que en el momento en que ellos hicieran el cambio, la manera en como NUBI operaba hasta hoy no serviria más y esto tendría un impacto muy fuerte para la compañía (Banco Comafi).



```
paypalAdaptiveFeeTopUpSdk = new PaypalAdaptive({
    userId: config.paypal.fee_top_up.username,
    password: config.paypal.fee_top_up.password,
    signature: config.paypal.fee_top_up.signature,
    appId: config.paypal.fee_top_up.appId,
    httpTimeout: 30,
    sandbox: config.paypal.mode === 'sandbox',
    sandboxEmailAddress: config.paypal.sandboxEmailAddress,
    deviceIpAddress: config.paypal.deviceIpAddress
  });

exports.paypalMovements = paypalMovements(config.paypal);
exports.paypalBalance = paypalMovements(config.paypal.fee_top_up);

exports.paypalPermissions = new PaypalPermissions({
  userId: config.paypal.username,
  password: config.paypal.password,
  signature: config.paypal.signature,
  appId: config.paypal.appId,
  httpTimeout: 30,
  mode: config.paypal.mode,
  sandboxEmailAddress: config.paypal.sandboxEmailAddress
});

Complexity is 6 It's time to do something...
const paypalAdaptiveSdkCall = (paypalAdaptiveSdk, payload, method) => {
  Complexity is 4 Everything is cool!
  return new Promise((resolve, reject) => {
    Complexity is 3 Everything is cool!
    paypalAdaptiveSdk[method](payload, (err, response) => {
      if (err) {
        reject(response);
      } else {
        resolve(response);
      }
    });
  });
}

const preapprovalDetailsAsync = (payload) => {
  return paypalAdaptiveSdkCall(paypalAdaptiveWithdrawSdk, payload, 'preapprovalDetails');
};

const preapprovalAsync = (payload) => {
  return paypalAdaptiveSdkCall(paypalAdaptiveWithdrawSdk, payload, 'preapproval');
};

const payWithdrawAsync = (payload) => {
  return paypalAdaptiveSdkCall(paypalAdaptiveWithdrawSdk, payload, 'pay');
};
```

Fig 11: sdk para la comunicación con paypal de la versión 1.

En la *Fig 13: tests de la versión 2.* se puede observar que otro valor agregado en la nueva versión son los tests, pues los existentes en la versión 1 estaban desactualizados y fallando, por ende no aportaban nada al proyecto. Ahora tenemos todas las funcionalidades cubiertas y debidamente testeadas.

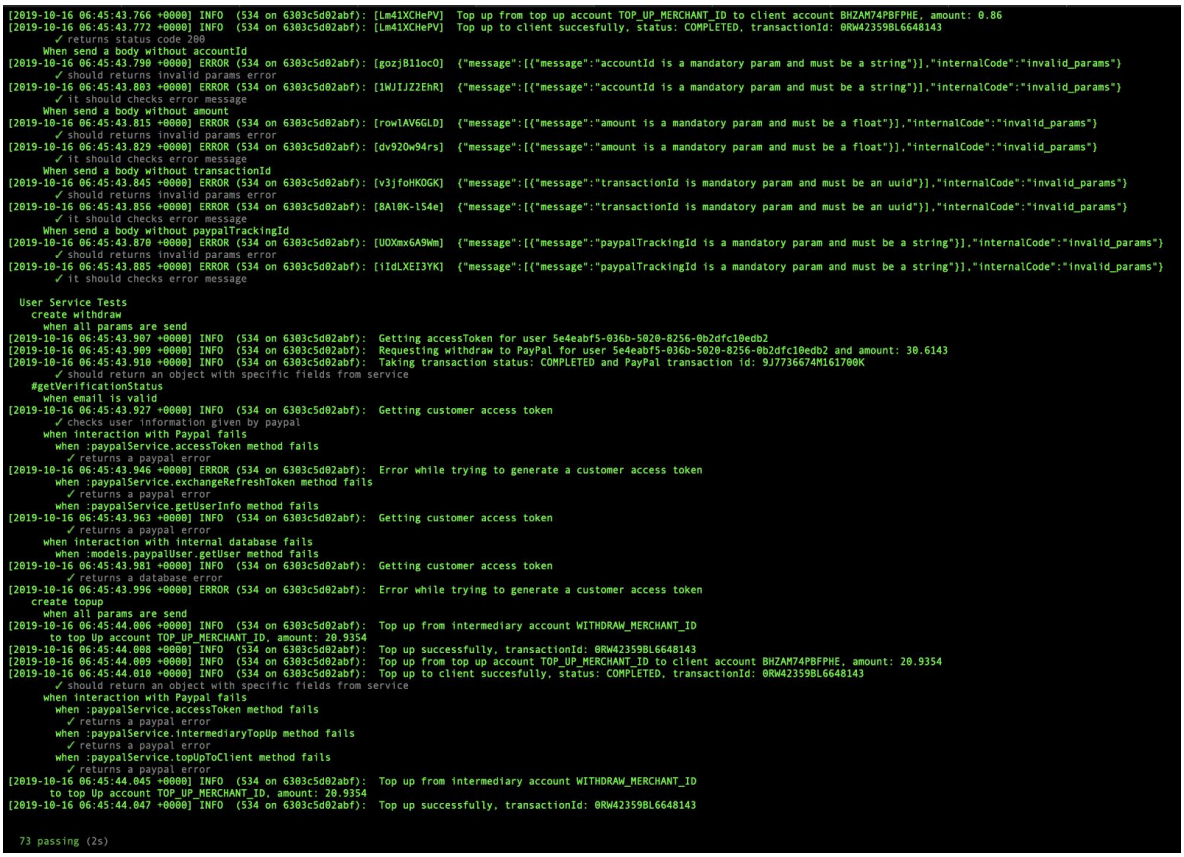


Fig 13: tests de la versión 2.

Por último, en la siguientes imágenes se adjuntará evidencia del correcto funcionamiento de la api. Como se puede apreciar en la url, ya está desplegado en un ambiente de prueba y se está haciendo la última fase de testing por parte del banco para su futuro paso a producción. Cabe resaltar que otro motivo por el que no se ha pasado a producción es por retrasos en el desarrollo del frontend, pero el backend ya está todo terminado. Se adjuntará la evidencia de los servicios más significativos ya que son muchos y por temas de confidencialidad no se pueden mostrar todos.

En la siguiente imagen, se muestra la respuesta cuando un usuario NUBI decide sincronizar su cuenta PayPal. Dicha información es recuperada de PayPal y persistida en la base de datos para futuras operaciones del usuario.

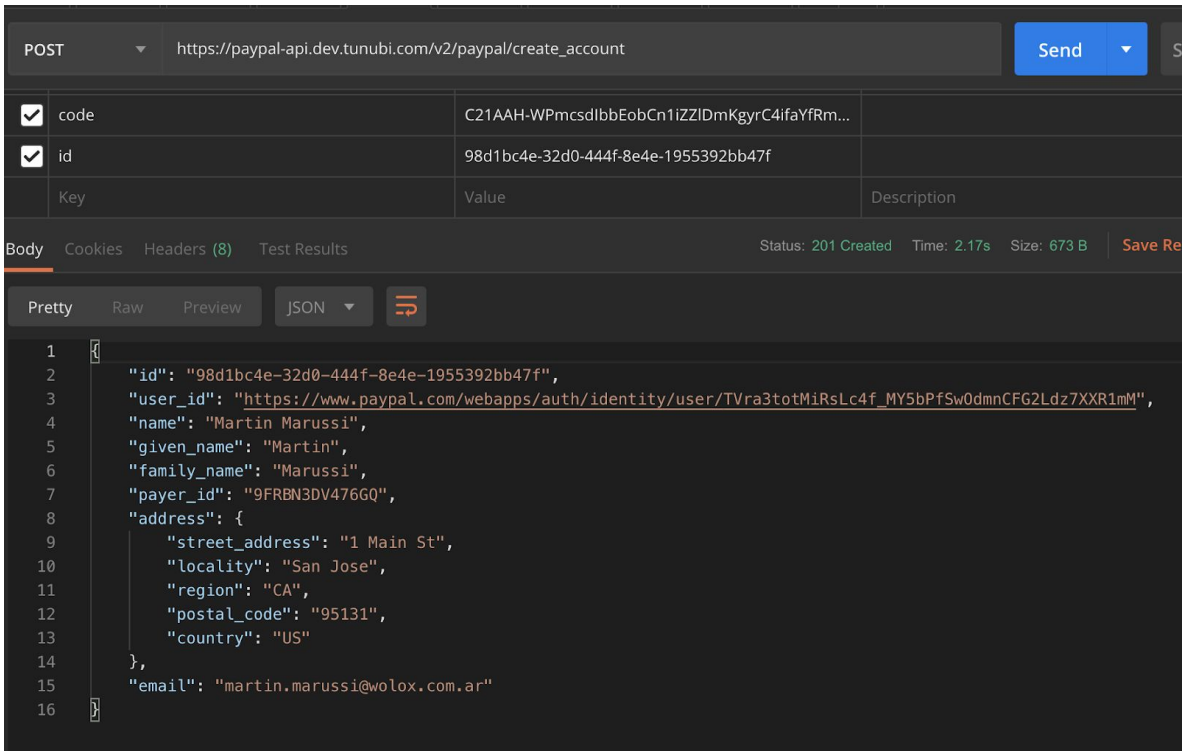


Fig 14: Sincronización de un usuario versión 2.

En la siguiente imagen, se muestra la respuesta cuando un usuario NUBI decide desvincular su cuenta paypal.

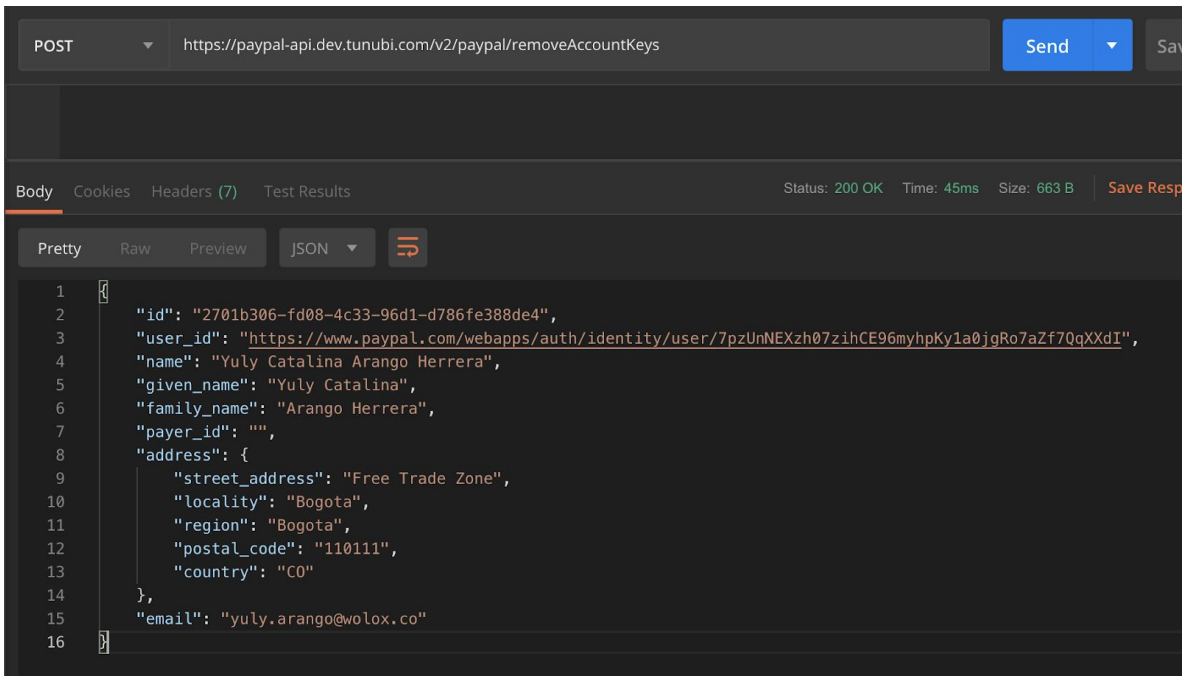


Fig 15: Desvinculación de un usuario versión 2.

En la siguiente imagen, se muestra la respuesta cuando se pide el balance de un usuario con una cuenta sincronizada, dicho monto es recuperado de PayPal y se le muestra al usuario en centavos por reglas del negocio como se mencionó anteriormente.

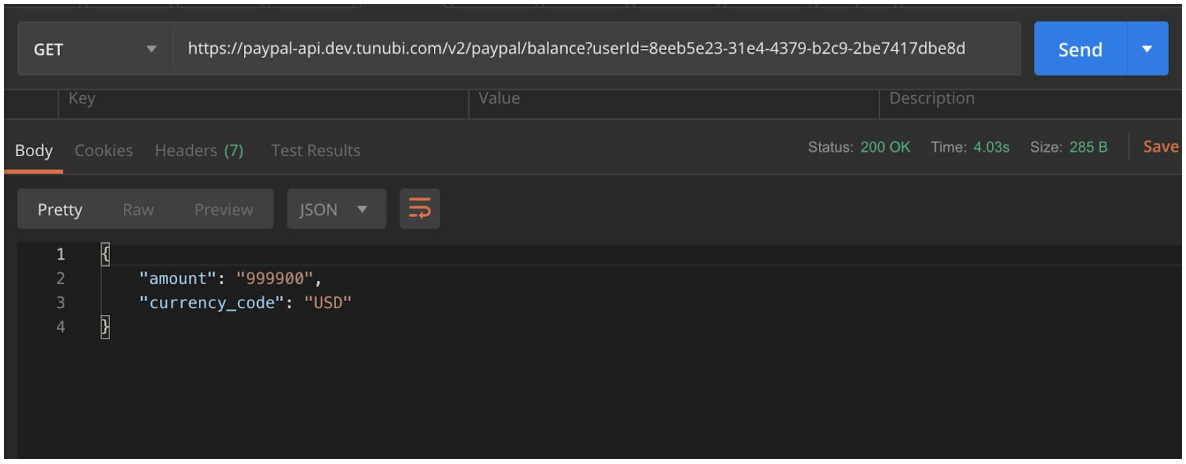


Fig 16: Balance de un usuario versión 2.

Conclusiones:

Mirando los objetivos planteados inicialmente y el producto final, se puede decir que se cumplieron los objetivos de la práctica académica satisfactoriamente.

Adicionalmente, otras conclusiones que nos permite expresar el proyecto son:

- Las metodologías ágiles permiten un mayor dinamismo en el diseño de soluciones de software, ya que hacen que la corrección de errores y el refinamiento de la idea inicial hagan parte continua del flujo de la metodología. Así estos procesos (entre otros) se mantienen en constante mejora hasta el final del proyecto. Estas metodologías permiten enfocarse de principio a fin en la solución más óptima, agregando valor al negocio fácilmente.
- Se pudo notar que al usar mejores prácticas de desarrollo los tiempos de implementación se reducen y a sí mismo la escalabilidad y la mantenibilidad de la aplicación.
- El uso de un orm soportado y aceptado por la comunidad como lo es *sequelize* ayuda considerablemente tanto en la etapa de desarrollo como en la de producción, ya que cuenta con una amplia documentación y soporte para cual tipo de error que se pueda presentar.
- Los tests son una parte fundamental de todo proyecto ya que ayudan a solventar fallos en la fabricación de software, ayudan a prevenir errores y aseguran una parte

de la calidad del código, además que ayudar a probar la correcta funcionalidad del proyecto en la etapa de desarrollo.

- Las prácticas académicas son el primer acercamiento al campo laboral, donde se puede aplicar los conceptos y definiciones adquiridas en el campo académico.

Referencias Bibliográficas

[1] NODE.JS, FUNDACIÓN, 2019, Acerca | Node.js. *Node.js* [online]. 2019. [Accessed 18 May 2019]. Available from: https://nodejs.org/es/about/

[2] About npm | npm Documentation, 2019. *Docs.npmjs.com* [online]. 2019. [Accessed 20 May 2019]. Available from: https://docs.npmjs.com/about-npm

[3] PostgreSQL: The world's most advanced open source database, 2019. *Postgresql.org*[online]. 2019. [Accessed 20 May 2019]. Available from: https://www.postgresql.org/

[4] "Understanding And Using REST APIs — Smashing Magazine", Smashing Magazine, 2019. [Online]. Available: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/ [Accessed: 29- May- 2019].

[5] M. Masse, REST API design rulebook. Beijing: O'Reilly, 2012.

[6]"Wolox/tech-guides", GitHub, 2019. [Online]. Available: https://github.com/Wolox/tech-guides/blob/master/nodejs/docs/node-standard-and-best-practices.md [Accessed: 14- Oct- 2019].

[7]"Wolox/express-js-bootstrap", GitHub, 2019. [Online]. Available: https://github.com/Wolox/express-js-bootstrap [Accessed: 14- Oct- 2019].

[8]"Qué es SCRUM", Proyectos Ágiles, 2019. [Online]. Available: https://proyectosagiles.org/que-es-scrum/ [Accessed: 16- Oct- 2019].