INSTITUTO TECNOLÓGICO BUENOS AIRES

Proyecto Final

Proyecto TiX

# Desarrollo e implementación de una arquitectura horizontalmente escalable


*Autores*
Matías Gabriel
Domingues
Facundo Nahuel
Martinez Correa
Javier Pérez Cuñarro

*Tutor*
Dr. Ing. José Ignacio
Alvarez-Hamelin


18 de diciembre de 2017

# Índice

# 3. Implementación

## 3.1. Protocolo TiX

Como se explica en la sección de Arquitectura, el protocolo de paquetes de TiX fue implementado como una biblioteca común para el Cliente y el servicio tix-time-server del Subsistema de Ingesta y Procesamiento. El código está completamente en Java, contiene las clases básicas para el manejo de los paquetes tanto de parte del cliente como del servidor.

A su vez, está basado en la biblioteca Netty, la cual contiene facilidades para la implementación de clientes y servidores de alta performance tanto en TCP [35] como UDP.

### 3.1.1. Diseño general del Protocolo

El protocolo de TiX cuenta con dos tipos paquetes, uno corto y uno largo. El paquete largo es una extensión del paquete corto en el sentido que los primeros campos son similares. A su vez, el paquete largo puede ser un paquete largo simple o un paquete de datos. La diferencia entre los últimos dos, es que el segundo contiene el reporte con mediciones tomadas en el último período de tiempo.

La diferencia entre un paquete corto y uno largo es la cantidad de bytes que cada uno contiene. El paquete corto actualmente ocupa el tamaño de cuatro números de tipo long de la JVM 8, es decir, 32 bytes. En tanto que el paquete largo ocupa lo mismo que seis números de tipo long de la JVM 8 más 4.400 bytes, es decir, 4.448 bytes.

Mientras que la funcionalidad del paquete corto es exclusivamente la de crear mediciones para la calidad del enlace, el paquete largo también busca evaluar el ancho de banda. Sin embargo, esta funcionalidad actualmente está deprecada y bajo revisión debido a que un paquete de poco menos de 4,5 KB no alcanza para medir el ancho de banda de los enlaces actuales. Además, es un paquete considerablemente grande para lo que son los límites actuales de transferencia de datos en los planes para celulares. Bajo esta óptica, es necesario estudiar una mejor forma de obtener el ancho de banda real del enlace minimizando el costo para líneas de teléfonos móviles.

Los paquetes del protocolo TiX están pensados para poder ser usados con cualquier protocolo de capa de transporte, como TCP y UDP. De hecho, al no tener ningún tipo de control sobre la transmisión y el correcto arribo de los mensajes, es recomendable utilizarlo sobre TCP. Sin embargo, dados los algoritmos de reconstrucción de paquetes y optimizaciones en el protocolo TCP, como el algoritmo de Nagle [36], las mediciones pueden verse afectadas. Por este motivo, solamente el protocolo UDP es utilizado hasta que exista alguna otra forma de garantizar las correctas mediciones usando TCP. El puerto por defecto del protocolo TiX es el 4500.

### 3.1.2. Funcionamiento del Protocolo

El protocolo tiene dos formas de funcionamiento, de alto uso de ancho de banda y de bajo uso de ancho de banda. La primer forma es la que usaba el viejo Sistema TiX, mientras que el nuevo Sistema usa la segunda.

El funcionamiento general es como se describe a continuación.

1. Se define una frecuencia $f$, con la cual se enviarán los mensajes entre el cliente y el servidor.
2. Se define $N$, que será la cantidad de mediciones acumuladas en los reportes.
3. Se define $M$, que será la cantidad de reintentos de envío del paquete de datos.
4. A los $1/f$ segundos se marca un paquete con la cantidad de nanosegundos desde el comienzo del día y se envía al servidor.
5. El servidor al recibir el paquete, lo marca con la cantidad de nanosegundos desde el principio del día.
6. Antes de devolver el paquete al cliente, vuelve a marcarlo con la cantidad de nanosegundos desde el principio del día y lo envía.
7. El cliente recibe el paquete desde el servidor y lo marca con la cantidad de nanosegundos desde el principio del día.
8. Tras $N$ intercambios y $1/f$ segundos, el cliente genera un paquete de datos y lo marca con la cantidad de nanosegundos desde el principio del día. Con esto se repiten los pasos 5, 6 y 7. En caso del cliente no recibir el paquete en el paso 7, reintentará el envío $M$ cada $1/f$ segundos, tras lo cual descartará esas mediciones. En caso de recibir el paquete, lo usará como una medición más y volverá al paso 4.

Actualmente, $f$ está definida en 1, $N$ en 60 y $M$ en 5.

### 3.1.3. Taxonomía de los paquetes

Los paquetes cortos se definen de la siguiente forma:

| `t_1: long` | `t_2: long` | `t_3: long` | `t_4: long` |
|---|---|---|---|

donde cada campo son 8 bytes que contienen un número de tipo long de la JVM 8 en complemento dos a la base. Dichos números son la cantidad de nanosegundos desde el principio del día cuando el cliente envió el paquete (`t_1`), el servidor recibió el paquete (`t_2`), el servidor devuelve el paquete al cliente (`t_3`) y el cliente recibe paquete (`t_4`).

Los paquetes largos simples se definen de la siguiente manera:

| t_1: long | t_2: long | t_3: long | t_4: long |
|---|---|---|---|
| padding | | | |

donde los cuatro primeros campos son idénticos y con la misma funcionalidad que los del paquete corto. El último campo, padding, contiene un conjunto de caracteres aleatorios, con el fin de hacer que el paquete ocupe el tamaño deseado. Por último los paquetes largos con datos se definen de la siguiente forma:

| t_1: long | t_2: long | t_3: long | t_4: long |
|---|---|---|---|
| data_header: char string | | | |
| data_delimiter: char string | | | |
| user_id: long | | installation_id: long | |
| public_key: byte string | | | |
| data_delimiter: char string | | | |
| report: char string | | | |
| data_delimiter: char string | | | |
| signed_hash: byte string | | | |
| data_delimiter: char string | | | |
| padding | | | |

donde los cuatro primeros campos son idénticos y con la misma funcionalidad que los del paquete corto y largo simple.

Los campos llamados `data_delimiter` son usados para diferenciar los distintos campos de datos dentro del paquete. Estos campos con la cadena de caracteres "`;`" codificada en UTF-8 [37].

El campo data_header es un campo diferencial que contiene la cadena de caracteres "`DATA`" codificada en UTF-8. Este campo se usa para distinguir un paquete largo común de un paquete largo con datos.

El campo `user_id` es un campo de 8 bytes que contiene un número de tipo long de la JVM 8 con el ID del usuario que realizó las mediciones.

El campo `installation_id` es un campo de 8 bytes que contiene un número de tipo long de la JVM 8 con el ID de la instalación donde se realizaron las mediciones.

El campo `public_key` contiene una cadena de bytes de largo variable con la clave pública de la instalación del usuario que realizó las mediciones. La misma se realiza con el algoritmo RSA [38] con 2048 bits y está encodeada en formato X.509 [39].

El campo `report` contiene el reporte del usuario con las mediciones realizadas por esa instalación. El mismo es una cadena de bytes encodeada en base64, lo cual resulta en una cadena de caracteres de largo variable.

El campo `signed_hash` contiene el hash del reporte firmado por el usuario con la clave privada de la instalación. El mismo es una cadena de bytes que corresponde al algoritmo de firma SHA1 [40] con RSA [41].

El último campo, `padding`, contiene un conjunto de caracteres aleatorios, con el fin de hacer que el paquete ocupe el tamaño deseado.

#### 3.1.4. Taxonomía de los reportes

Los reportes enviados en los paquetes largos de datos están serializados en binario, con el fin de que ocupen el menor espacio posible. Estos reportes tienen una estructura de lista o bitácora en donde cada entrada está formada por cinco campos de la siguiente forma:

| `day_timestamp: long` | `t_1: long` | `t_2: long` | `t_3: long` | `t_4: long` |
|---|---|---|---|---|

Cada uno de estos campos ocupa 8 bytes y representa un número de tipo `long` de la JVM 8 en complemento dos a la base. El campo `day_timestamp` representa la cantidad de segundos desde el 1 de Enero de 1970 a las 00:00 en que fue realizada esa medición según el reloj del Cliente. Los siguientes campos representan la cantidad nanosegundos desde el comienzo del día cuando se envió el paquete del cliente al servidor, cuando el servidor recibe el paquete, cuando el paquete es enviado desde el servidor al cliente, y cuando finalmente el cliente vuelve a recibir el paquete; respectivamente. Los nanosegundos allí registrados son en relación a los relojes de los respectivos sistemas y no están sincronizados entre sí.

### 3.2. El Subsistema de Ingesta y Procesamiento

#### 3.2.1. Servicio tix-time-server

Este es un micro-servicio que está implementado enteramente en Java.

Usa la biblioteca Netty para crear el servidor UDP del Protocolo TiX y un servidor HTTP con el fin de verificar la salud del servicio. Ambos servidores pueden configurar su puerto. El punto de entrada para verificar la salud del servicio responde a la dirección /health.

También importa la biblioteca tix-time-server para poder interpretar el Protocolo TiX y comunicarse con el Cliente.

Otra importante dependencia es la del cliente AMPQ [42] para RabbitMQ. Esta la utiliza para poder conectarse con la cola de mensajes y poner allí los paquetes de datos del Protocolo TiX.

Con el fin de garantizar una forma estándar de comunicarse con los servicios que consuman de la cola que alimenta, los utiliza la biblioteca Jackson para convertir los objetos del tix-time-core en cadenas de caracteres de formato JSON [43]. Su configuración se realiza a través del objeto ConfigurationManager, el cual tiene

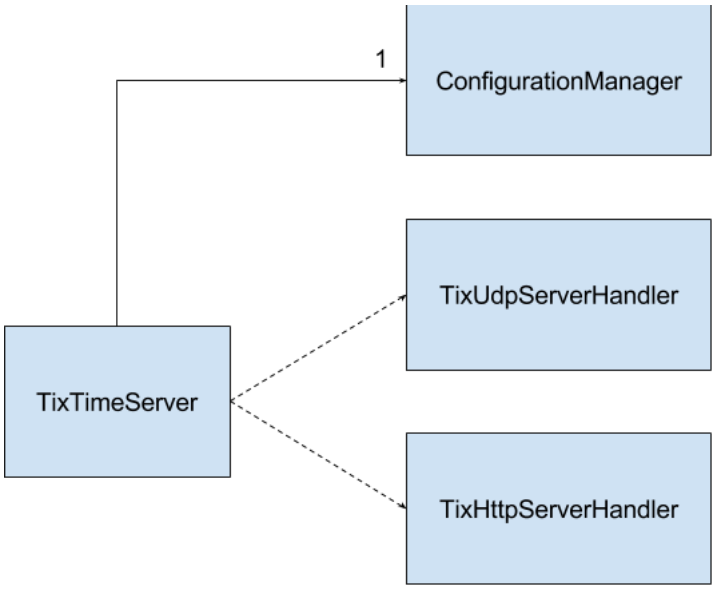


Figura 3: Diagrama UML del servicio tix-time-server

la capacidad de tomar los valores de configuración por distintas fuentes, como archivos de configuración, variables de entorno, etc. con distintas precedencias entre sí.

Además adopta el concepto de entorno, el cual define valores por defecto para las distintas variables para un entorno dado.

Esto confiere una gran flexibilidad al momento de configurar el servicio, ya que la precedencia de los valores de configuración proveniente de las distintas fuentes combinado con el uso de entornos permite tener valores por defecto globales, sumados a valores de entorno propios del servicio más valores sobrescritos por las otras fuentes para el caso de datos sensibles, como pueden ser contraseñas, o de valores dinámicos, como pueden ser niveles de reportes de eventos.

Como se puede apreciar en la Figura 3, el diseño es sencillo, de acuerdo con lo que debe ser un micro-servicio.

El servicio cuenta con pruebas de unidad que aseguran el correcto funcionamiento de cada uno de sus puntos más importantes. También posee una prueba de integración que asegura la correcta interoperabilidad con los demás servicios.

### 3.2.2. Servicio tix-time-condenser

Al igual que el servicio tix-time-server, este micro-servicio está hecho enteramente en Java. Pero en vez de ser estar hecho desde cero usando varias bibliotecas, éste usa el entorno de trabajo SpringBoot.

El entorno de trabajo SpringBoot está pensado específicamente para el prototipado y la creación de micro-servicios. Provee una gran cantidad de facilidades e importa de forma automática muchas bibliotecas que facilitan el trabajo. Además provee varias facilidades para las pruebas de unidad y de integración.

Tal es así, que el servicio cuenta con tan solo una clase con lógica de negocio. El resto de las clases son para abstraer las respuestas de la comunicación con el servicio tix-api. Esto se ve representado en el UML de la Figura 4.

Este servicio también importa la biblioteca tix-time-core. Esto lo hace para poder traducir los mensajes en JSON que consume de la cola de mensajes proveniente de tix-time-server. Además, hace uso de las facilidades de esta biblioteca para validar la integridad de dichos mensajes.

Una vez recibido y verificado el mensaje, el servicio hace llamadas a tix-api para validar el usuario y la instalación del cual provienen. En caso de éxito, procede a dejar el mensaje en el directorio configurado, bajo la ruta acordada. Dicha ruta tiene tiene la forma de /{user_id}/{installation_id}.

Cabe destacar la complejidad y cantidad de operaciones que realiza para ser un micro-servicio, pese a la gran sencillez del diseño.

Al igual que el tix-time-server, posee pruebas de unidad y de integración para asegurar el correcto funcionamiento e interoperabilidad con el resto de los sistemas.

Como detalle importante, este es el único servicio que tiene una prueba de integración en vivo, es decir, que se hace con el servicio compilado y corriendo como si fuera en su entorno normal mediante el uso de Docker.

### 3.2.3. Servicio tix-time-processor

A diferencia de los otros dos micro-servicios, este está implementado enteramente en Python3. Se basa en el entorno de trabajo Celery.

Celery es una cola de trabajos asíncrona y distribuida hecha casi en su totalidad en Python. Tiene una importante interoperabilidad con el servicio de cola de mensajes RabbitMQ, aunque puede utilizar otros similares también. Una de sus principales características es la posibilidad de programar la ejecución de tareas y distribuirla entre sus distintos esclavos. Esto es lo que hizo que Celery sea una respuesta acertada al trabajo por lotes distribuido del procesamiento de estimación de uso de red y calidad de conexión a partir de los reportes de los clientes.

Dentro de las dependencias más importantes de este servicio están las bibliotecas Numpy y SciPy. Ambas bibliotecas son de las más conocidas e importantes de la comunidad científica y matemática de Python. Se usan principalmente para hacer algunos cálculos como es la estimación del coeficiente de Hurst, regresiones lineales y simplificar el trabajo matemático.

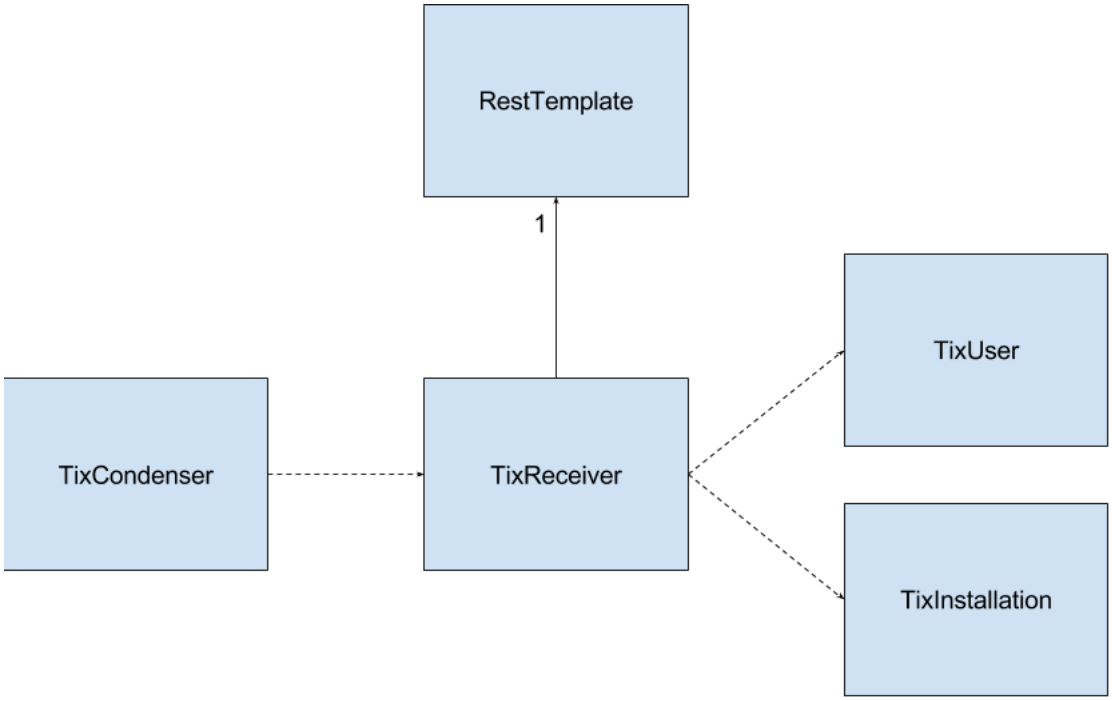


Figura 4: Diagrama UML del servicio tix-time-server

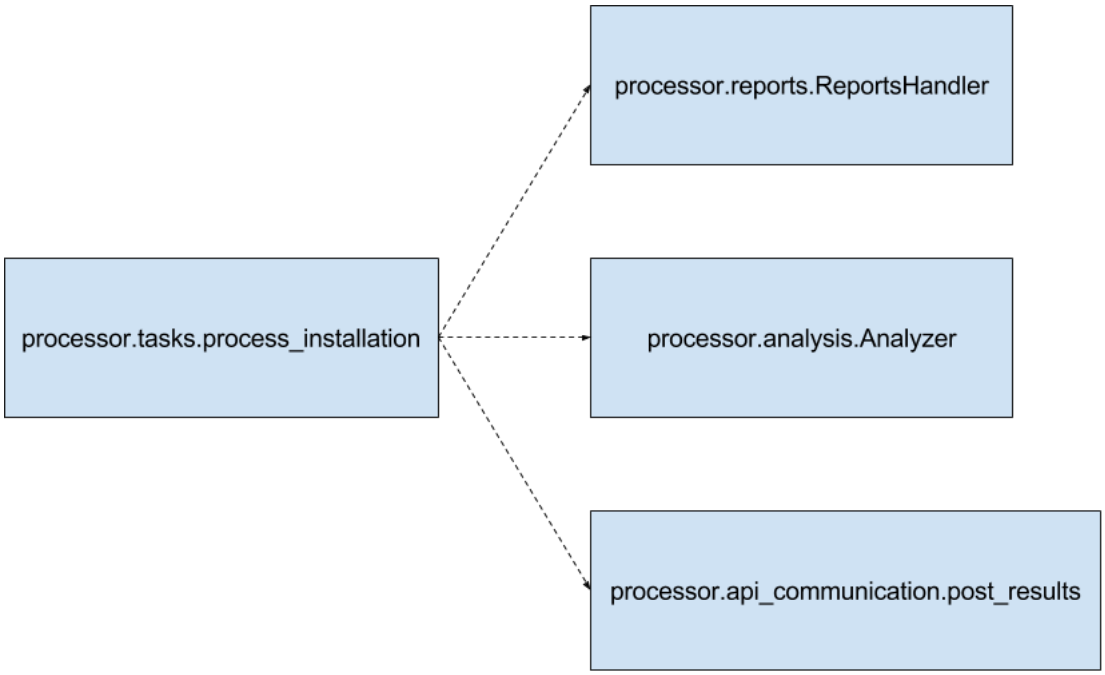


Figura 5: Diagrama UML del servicio tix-time-server

La tercer dependencia importante de este servicio es la famosísima requests. La cual usa para enviar los resultados de las estimaciones al servicio tix-api.
Por último, también usa la biblioteca PyWavelets [44], con el fin casi exclusivo de computar la transformada wavelet en la estimación del parámetro de Hurst.

Como se puede ver en las Figuras 5, 6 y 7, este servicio es más complejo que los otros dos de este subsistema. Esto se debe a que probablemente tanto el ReportsHandler como el Analyzer deberían ser bibliotecas aparte, con su propia funcionalidad.

El Analyzer es el caso más obvio para esta futura mejora. Ya que las posibilidades de hacer pruebas de unidad con esta parte del código son limitadas y precisa de un análisis exhaustivo de los resultados que arroja. Es por eso que el repositorio posee un Notebook de Python que lo acompaña. Para hacer las veces de forma de prueba exploratoria de los resultados de este paquete.

Por otro lado, el ReportsHandler es prácticamente una biblioteca de serialización y deserialización completa de los reportes del Protocolo TiX. Con lo que puede ser meritoria de su propia base de código aislada.

Por consiguiente, las pruebas de unidad en este servicio están limitadas a los paquetes que son más sencillos de probar, ReportsHandler y el paquete api_communication.

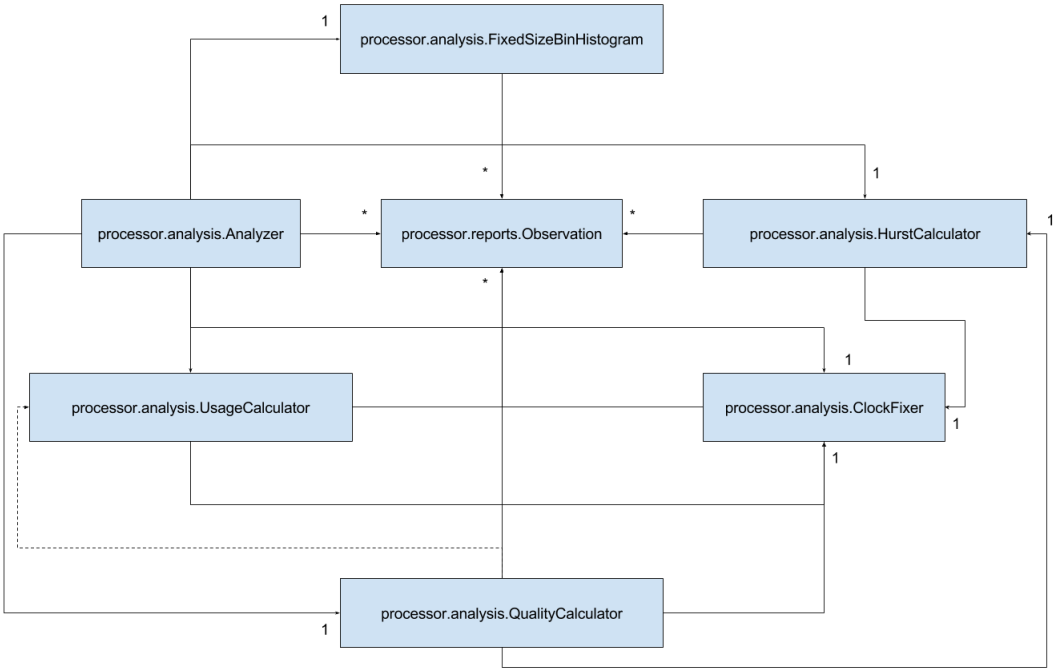


Figura 6: Diagrama UML del servicio tix-time-server

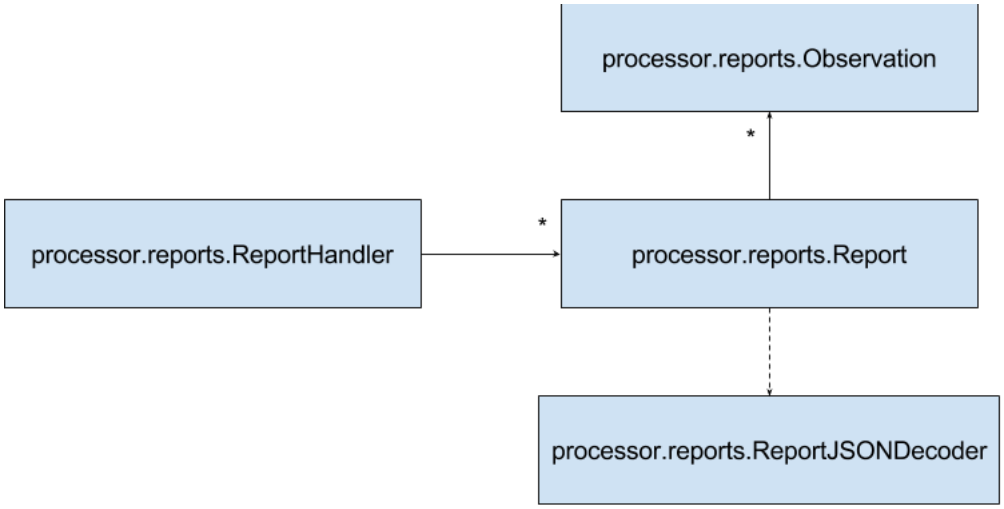


Figura 7: Diagrama UML del servicio tix-time-server

### 3.2.4. Despliegue y puesta en producción

Para automatizar la tarea de desplegar el Subsistema de Ingesta y Procesamiento se recurrió a la herramienta Docker Compose. La misma permite mediante el uso de un archivo YAML describir la configuración que debe tener el sistema, los distintos servicios que lo componen y cómo interactúan entre sí, además de sus dependencias. Además posee comandos sencillos que permiten levantar, actualizar, parar y desarmar el sistema.

Esto, junto con la posibilidad de interoperar con la herramienta Docker Swarm, permiten que el sistema en su conjunto pueda ser manejado con sencillez y robustez, además de ser escalado a lo largo de un clúster de máquinas, si es necesario.

Todo esto convierten a la herramienta en indispensable para el manejo de microservicios como los aquí explicados.

Este documento se encuentra en el repositorio tix-time-deploy y cuenta también con variables sensibles ya declaradas que se esperan existan definidas en el entorno para poder levantar el sistema completo. Estas variables son credenciales de algunos de los servicios de este Subsistema y de otros servicios de otros subsistemas.

## 3.3. El Subsistema Cliente

### 3.3.1. Aplicación Cliente

El Cliente es uno de los subsistemas del proyecto que está programado en lenguaje Java. Esto permite crear una única versión de la aplicación y despreocuparse por las diferencias de programar en código nativo para los distintos sistemas operativos. Actualmente funciona en computadoras personales, pero no sería difícil que funcione en dispositivos móviles o televisores que tengan la habilidad de ejecutar código Java. Para asegurar que el cliente no tenga las rutas del sistema definidas explícitamente en el código, se hace uso del método System.getProperty() que retorna variables propias del sistema que está ejecutando el programa. De esta manera, no es necesario definir una serie de sistemas operativos de antemano con sus respectivas rutas, sino que Java asume la responsabilidad de obtener los valores correctos ad-hoc.

Si bien el binario ejecutable es único, la aplicación se distribuye con un instalador nativo que en este caso sí varía según el sistema operativo. La secuencia esperada consiste en acceder a la página web de TiX, crear una cuenta nueva y en la misma sección descargar el instalador de Windows, el de Linux o el de OSX. En el caso de Windows, es un archivo .exe de tipo asistente y guía al usuario paso a paso. Para OSX, es un archivo .dmg que requiere que el usuario haga drag-and-drop del ícono de la aplicación en la carpeta Applications, un procedimiento familiar para estos usuarios. Para Linux, está disponible el instalador .deb, que

se puede instalar de forma sencilla usando la herramienta dpkg desde línea de comando. Para otros sistemas operativos, como última alternativa, se puede usar la aplicación sin instalador ejecutando el archivo .jar.

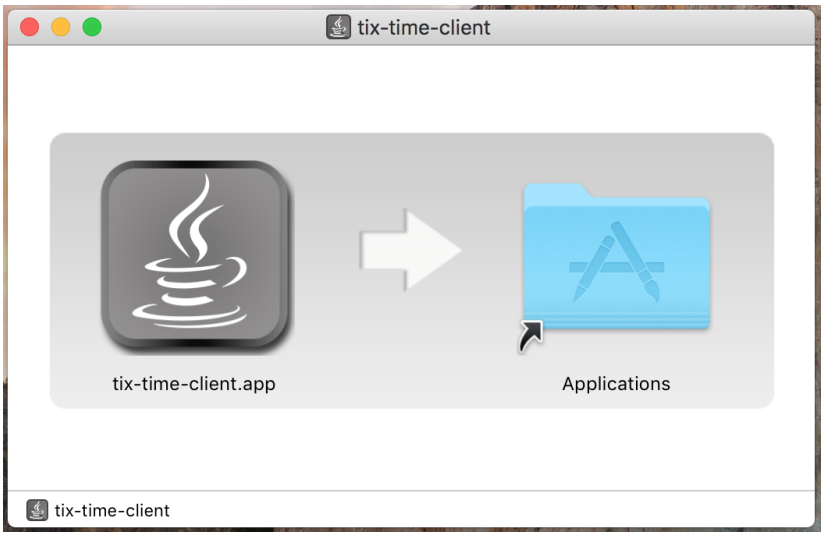


Figura 8: Instalador de la aplicación para OSX.

El cliente admite el uso de dos interfaces alternativas: una interfaz por línea de comandos (CLI) y una interfaz gráfica de usuario (GUI). La primera ha sido diseñada a modo de prueba y admite parámetros al correr el cliente - tales como nombre de usuario e instalación - para evitar el uso de un asistente que entorpezca el proceso de automatización. En cambio, la interfaz gráfica no contempla el uso de parámetros y será presentada a quienes usen el instalador de forma predeterminada.

Para la interfaz gráfica se hace uso de JavaFX [45], la plataforma gráfica de Java que ha remplazado a Swing [46] como el nuevo estándar hace varios años. El aspecto del programa resultante no equivale al de una aplicación de código nativo, pero de todas maneras es altamente customizable en apariencia. Las distintas ventanas del cliente fueron definidas mediante FXML [47], un lenguaje *markup* basado en XML pensado para interfaces de JavaFX.

Luego de instalar el cliente, el usuario que opte por el modo GUI (comportamiento predeterminado) debe ingresar mediante una interfaz visual las credenciales creadas previamente en el sitio web y definir el nombre de su nueva instalación. Tanto la autenticación como la creación de la instalación se realizan mediante el uso de forma segura de la API REST de TiX. La instalación consiste de un asistente de tres pasos en una ventana flotante de la aplicación. Con una interfaz simple y amigable, guía a los usuarios para que en menos de un minuto cuenten con el cliente ya funcionando en sus computadoras.

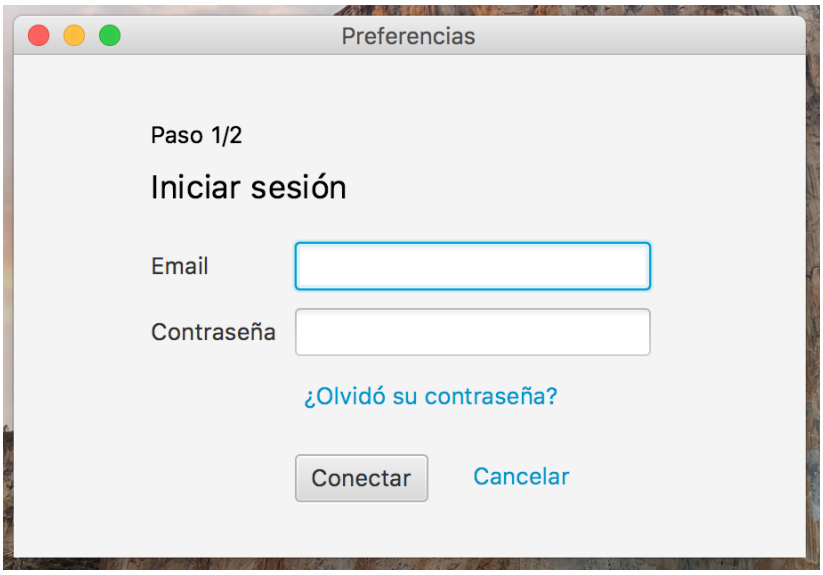


Figura 9: Ingreso de email y contraseña en cliente de OSX.

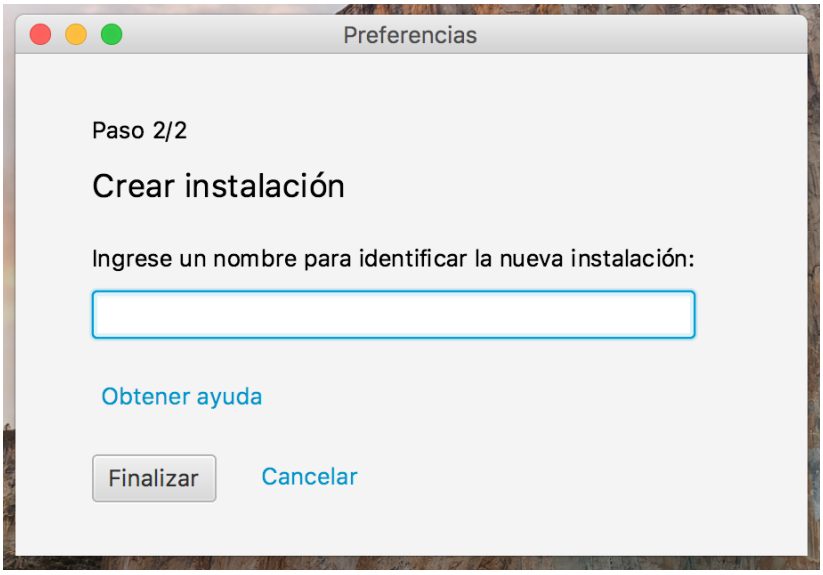


Figura 10: Ingreso de nombre de instalación en cliente de OSX.

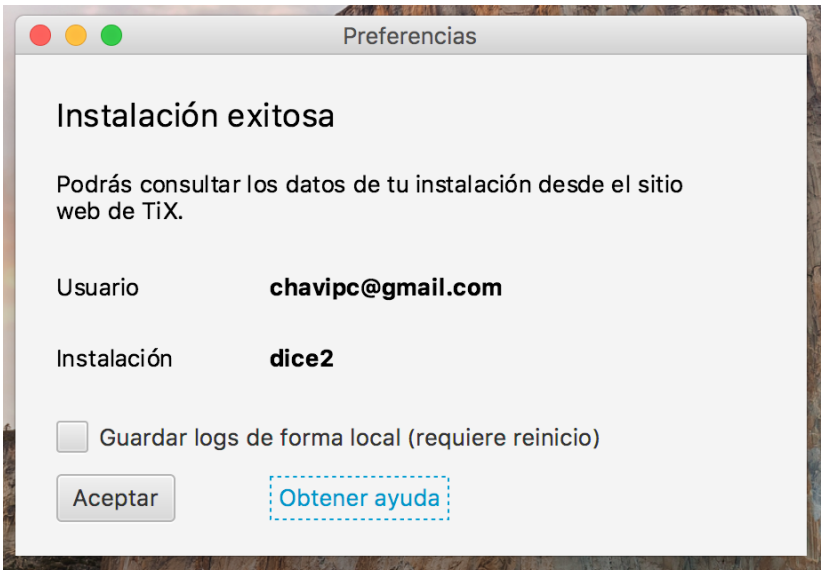


Figura 11: Fin de configuración de cliente de OSX.

Los datos de inicio de sesión y nombre de instalación son almacenados en la máquina de cada usuario. Se hace uso de la Java Preferences API [48] para almacenar las preferencias de forma eficiente según cada sistema operativo.

Una vez finalizada la configuración inicial, la aplicación queda en ejecución y comienza con las tareas de comunicarse con el servidor y realizar mediciones de tiempo. De aquí en más no es mandatoria la interacción de parte del usuario. En la barra de tareas del sistema aparece un ícono con el logo de TiX que permite acceder al menú de opciones, en caso de desearlo.

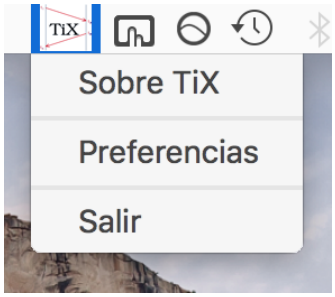


Figura 12: Menú de opciones del cliente en OSX.

La primer opción "Sobre TiX" abre una ventana que presenta información básica sobre el proyecto y provee una forma rápida de llegar a los reportes del usuario.


Figura 13: Ventana "Sobre TiX" en cliente de OSX.

La segunda opción "Preferencias" informa el nombre de usuario e instalación asociadas a la computadora, y permite almacenar registros de tiempos de forma local.

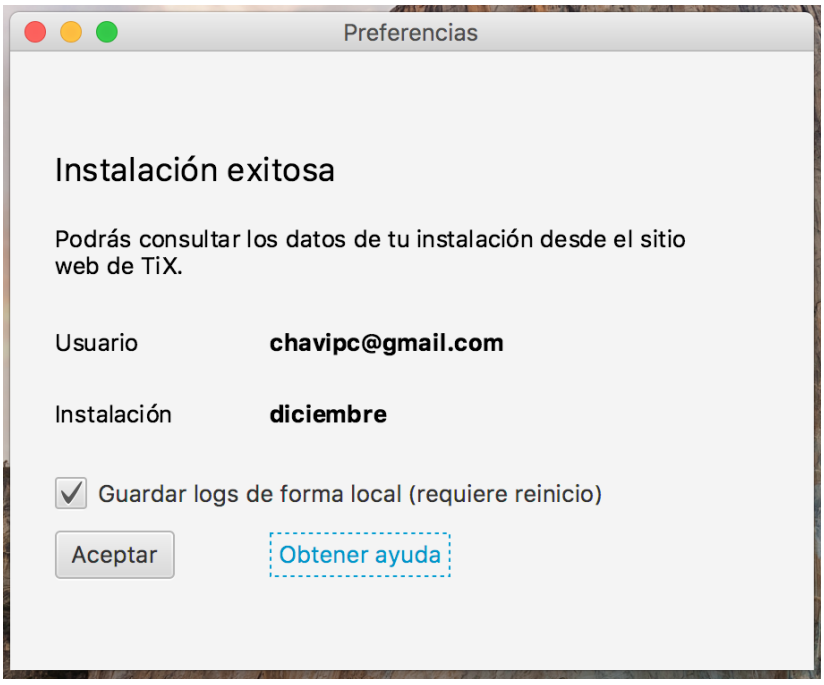


Figura 14: Ventana "Preferencias" en cliente de OSX.

La última opción "Salir" interrumpe las mediciones y la comunicación con el servidor, remueve el ícono de la barra de tareas y finaliza la ejecución del programa.

Con el objetivo de asegurar un rendimiento óptimo del cliente, las clases del código que no están relacionadas con la capa de presentación forman parte de un servicio que es independiente del hilo principal de la aplicación. Esto permite que, por ejemplo, la interfaz gráfica no se congele en ningún momento de su ejecución.

Cabe destacar el uso del *plugin* FXLauncher [49] para ofrecer actualizaciones automáticas para el usuario. Cada vez que se inicia el programa (por ejemplo, al encender la computadora), se compara la versión instalada con el último lanzamiento disponible en el servidor. En caso de existir una diferencia, se descarga la versión más reciente de la aplicación principal junto con las dependencias que hayan cambiado, sin necesitar de un accionar manual. De este modo, quien quiera mantenerse al día no tiene necesidad de acceder a la página web y descargar el programa completo, y por otro lado disminuye la probabilidad de tener clientes corriendo versiones descontinuadas.

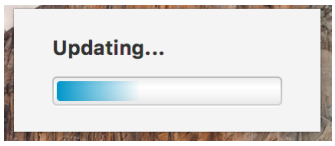


Figura 15: Búsqueda automática de actualizaciones en cliente de OSX.

### 3.3.2. Reporter

La tarea principal del cliente es intercambiar paquetes con el servidor y a partir de esto obtener mediciones de tiempo que reflejen la duración de estos intercambios.

La clase Reporter se dedica exclusivamente a conectarse con el servidor, mandar y recibir los paquetes, y persistirlos localmente según necesidad.

La primer tarea del Reporter es obtener la interfaz de red correcta y entablar un canal de comunicación con el servidor. Una vez que cuenta con la conexión, se hace uso de la clase auxiliar TixUdpClientHandler para recibir información externa y persistirla de forma temporal en un archivo local, en forma de bytes.

Los intercambios puntualmente consisten en envíos de pequeños paquetes UDP una vez por segundo, que no tienen ningún contenido definido. Se toma nota de las cuatro marcas de tiempo de envío y recepción (ver detalles en la Sección 3.1) para luego calcular la diferencia de tiempo entre ellos. Las marcas de tiempo se persisten temporalmente hasta llegar a tener un minuto completo de información.

Al finalizar el minuto, se envía un paquete "largo" al servidor con un contenido *(payload)* definido. Aquí se incluyen todas las mediciones de tiempo para luego ser procesadas. Existe un algoritmo cuya responsabilidad es confirmar que esta información llegó correctamente al servidor, y en caso contrario realiza hasta 5 reintentos de envío de mediciones. Apenas se recibe la confirmación de recepción exitosa, se detienen los reintentos.

Existe la opción de persistir los timestamps localmente, de modo permanente. En la ventana de Preferencias, al tildar la caja de "Guardar logs de forma local", se generará a partir de ese instante un archivo por minuto con la información en bytes. Se usará el directorio /tix-client-logs dentro de la carpeta *home* de cada usuario.

En cuanto a tecnologías utilizadas, la clase Reporter emplea Netty [26] para el envío y recepción de paquetes. Netty es un *framework* cliente-servidor para Java que es asincrónico y no bloqueante.

## 3.4. El Subsistema de Presentación y Administración de Datos

El subsistema de presentación de datos se puede dividir en dos servicios, los cuales funcionan de manera totalmente autónoma, el sistema encargado de guardar los datos y entregarlos a quien lo solicite (la API) y la web, cuyo fin es presentar los datos de forma concisa al usuario.

La web fue desarrollada principalmente en React.js[22], pero utiliza otras bibliotecas, tales como Material-ui[50], Redux[51] y Redux-forms[52] con el objetivo de minimizar la cantidad de código escrito y presentar la información de manera consistente.

React.js es una biblioteca de código abierto bajo la licencia MIT, la cual fue desarrollada por Facebook INC en el año 2013 para el desarrollo de aplicaciones frontend que manejan grandes volúmenes de información, de manera que no fuera necesaria la recarga del sitio. Asimismo busca generar simplicidad, escalabilidad y velocidad a la aplicación que se está desarrollando. Cabe destacar que React solo se encarga de generar la visualización del sitio.

Luego del lanzamiento de ReactJs, la comunidad comenzó a desarrollar diversas bibliotecas para compensar la falta de modelos y almacenamiento para la aplicación, es as como en 2015 un grupo de desarrolladores creó Redux, un contenedor del estado de la aplicación altamente predecible que ayuda a mantener el estado de la aplicación consistente durante el tiempo, ya que utiliza una única fuente de verdad (*source of truth*).

Para el desarrollo de la plataforma web, se utilizó un código base creado por la comunidad, React-boilerplate, el uso del mismo garantiza la utilización de las últimas tecnologías y estándares, así como también ayudar a mantener la calidad del código y bajar el el costo inicial de creación del aplicativo. Este codigo también se encuentra bajo licencia MIT, cuenta con más de 100 contribuyentes y cerca de 1000 commits.

Estos componentes conforman la web a la cual se puede acceder mediante la URL https://tix.innova-red.net.net

Por otro lado, la API se encuentra desarrollada principalmente en NodeJs[53], y diversas librerias basadas en Javascript, tales como ExpressJs[54], PassportJs[55], Knex[56], Bookshelf[57] y RamdaJs[58]. Finalmente, esta parte del proyecto tiene como dependencia IP2AS, el cual dada una IP, se encarga de resolver el AS que le corresponde.

NodeJs es un entorno de ejecución de código abierto (bajo licencia MIT), asincrónico, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google[59]. Dada su estructura, es especialmente útil para micro-arquitecturas como la propuesta para este proyecto. Su manejador de paquetes, npm[60], es considerado el ecosistema más grande de bibliotecas open source existente hoy en dia.

Una de las partes más importantes del servicio es el framework ExpressJs, este se describe como un framework web minimalista, que permite con mucha facilidad agregar nuevas rutas y modificar las existentes. Se desarrollo bajo licencia MIT y cuenta con más de 5000 commits y 200 contribuyentes.

Por otro lado, el servidor utiliza MySQL para guardar/hacer queries sobre la información, se decidió utilizar un sistema basado en SQL, ya que se busca priorizar la consistencia de los datos. Para conectar nuestro servidor a la base de datos, se utilizaron dos bibliotecas, Knex, un framework para la generación de queries de forma simple y Bookshelf.js, un ORM que corre sobre Knex, ambos de código abierto bajo licencia MIT.

Finalmente, se utilizó PassportJs para realizar la autenticación de los usuarios. Este framework utiliza el concepto de «estrategias» para cumplir su rol. Lo que permitiría, por ejemplo, implementar un método de autenticación vía Facebook simplemente agregando una nueva «estrategia».

La conjunción de estos frameworks/bibliotecas conforman la API.

### 3.4.1. Despliegue y puesta en producción

De la misma manera que el Subsistema de Ingesta y Procesamiento, la API utiliza el sistema de docker-compose para su proceso de deployment, bajo este compose, se encuentra también la base de datos (MySQL) e IP2AS. De esta forma, una vez que se corre el comando de ejecución, la API ya cuenta con todas las dependencias que necesita para correr correctamente.

Por otro lado, el despliegue de la web se realiza mediante TravisCI, una vez finalizada la etapa do construcción del paquete, en donde se compila y minifica el código, este los copia a la carpeta correspondiente para su ejecución.

### 3.4.2 Definición de la API

Como fue mencionado anteriormente, la API expone una interfaz en la cual cualquier cliente se puede conectar con facilidad y acceder a los datos del sistema, esta se encuentra bajo la URL: https://tix.innova-red.net/api y expone 4 contratos:

Contrato de Usuario:

- username: Nombre de usuario
- role: Rol que cumple el usuario, puede ser usuario o administrador
- id: ID del usuario
- enabled: Boolean que indica si el usuario está activo o no.

Contrato de Instalación:

- id: ID de la instalación
- name: Nombre de la instalación
- publickey: Clave pública correspondiente a la instalación
- providers: (opcional) Lista de proveedores asociados a la instalación

Contrato de Proveedores:

- id: ID del proveedor
- name: Nombre del proveedor

Contrato de Métrica:

- upUsage: Métrica de uso de la coneccion de subida
- downUsage: Métrica de uso de la coneccion de bajada
- upQuality: Métrica de calidad de la coneccion de subida
- downQuality: Métrica de calidad de la coneccion de bajada
- timestamp: Hora del reporte
- location_id: ID de la ubicación donde fue generado el reporte
- provider_id: ID del proveedor donde fue generado el reporte
- user_id: ID del usuario que genero el reporte

También expone los siguientes métodos:
Método: GET /api

- Uso: Conocer si el servicio está corriendo actualmente o no
- Respuesta esperada: 200 OK

Método: POST /api/register

- Uso: Permite al cliente generar un nuevo usuario dentro del sistema.
- Cuerpo requerido:
  - captcharesponse: string que devuelve el reto de Re-Captcha
  - username: Nombre de usuario que se quiere crear

    - password1: Contraseña del usuario
    - password2: verificación de la contraseña
- Respuesta esperada: 200 OK con el contrato del usuario recién creado.

Método: POST /api/login

- Uso: Permite al cliente generar un token para autentificarse con la plataforma
- Cuerpo requerido:
    - username: Nombre de usuario
    - password: Contraseña del usuario
- Respuesta esperada:
    - token: JWT token utilizado para autenticar al usuario.
    - El resto del contrato de usuario.

Método: POST /api/recover

- Uso: Permite al cliente generar el token de recupero de contraseña el cual es enviado al email del cliente.
- Cuerpo requerido:
    - email: Email del usuario a recuperar
- Respuesta esperada: 200 OK

Método: POST /api/recover/code

- Uso: Permite al usuario modificar su contraseña junto al token enviado anteriormente.
- Cuerpo requerido:
    - email: Email del usuario a modificar
    - code: Código de verificación enviado en el email anteriormente
    - password: Contraseña a setear en el usuario seleccionado
- Respuesta esperada: 200 OK

Método: GET /api/user/:id

- Uso: Devuelve la información básica del usuario.

- Parámetros:
  - id: ID del usuario
- Respuesta esperada: 200 OK con el contrato del usuario

Método: PUT /api/user/:id

- Uso: Editar la información del usuario
- Parámetros:
  - id: ID del usuario
- Respuesta esperada: 200 OK con el contrato del usuario

Método: GET /api/user/:id/installation

- Uso: Devuelve la lista de instalaciones pertenecientes al usuario
- Parámetros:
  - id: ID del usuario
- Respuesta esperada: 200 OK con una lista de contratos de instalaciones.

Método: GET /api/user/:id/installation/:installationId

- Uso: Devuelve una instalación en particular
- Parámetros:
  - id: ID del usuario
  - installationId: ID de la instalación deseada
- Respuesta esperada: 200 OK con el contrato de la instalación

Método: PUT /api/user/:id/installation/:installationId

- Uso: Editar una instalación
- Parámetros:
  - id: ID del usuario
  - installationId: ID de la instalación deseada
- Cuerpo requerido:
  - name: Nombre de la instalación

- Respuesta: 200 OK con el contrato de la nueva instalación

Método: DELETE /api/user/:id/installation/:installationId

- Uso: Borrar la instalación deseada
- Parámetros:
  - id: ID del usuario
  - installationId: ID de la instalación deseada
- Respuesta esperada: 200 OK

Método: GET /api/user/:id/provider

- Uso: Devuelve la lista de proveedores asociados al usuario
- Parámetros:
  - id: ID del usuario
- Respuesta esperada: 200 OK con una lista de contratos de proveedores

Método: GET /api/user/:id/provider/:providerId

- Uso: Devuelve el proveedor deseado
- Parámetros:
  - id: ID del usuario
  - providerId: ID del proovedor
- Respuesta esperada: 200 OK con el contrato de proveedor

Método: GET /api/users/:id/reports

- Uso: Devuelve los reportes asociados al usuario
- Parámetros:
  - id: ID del usuario
- Respuesta esperada: 200 OK con una lista de contratos de reportes

Método: POST /api/user/:id/installation/:installationId/reports

- Uso: Crear un nuevo reporte
- Parámetros:

        - id: ID del usuario
        - installationId: ID de la instalación
    - Cuerpo requerido:
        - ip: IP donde se generó el reporte
        - upUsage: Metrica de utilizacion de subida
        - downUsage: Metrica de utilizacion de bajada
        - upQuality: Metrica de calidad de subida
        - downQuality: Metrica de calidad de bajada
        - timestamp: Momento en el que se genero el reporte
    - Respuesta esperada: 200 OK con el contrato de reporte

Método: GET /api/admin/users

- Uso: Devuelve todos los usuarios actuales en el sistema
- Respuesta esperada: 200 OK con una lista de contratos de usuario

Método: GET /api/admin/reports

- Uso: Devuelve todos los reportes según el criterio establecido
- Parámetros opcionales:
    - startDate: Fecha de inicio
    - endDate: Fecha de fin
    - providerId: ID del proveedor
- Respuesta esperada: 200 OK con una lista de contratos de reportes.

# Referencias


[1] J. Postel, "User datagram protocol," Internet Engineering Task Force, Tech. Rep., aug 1980. [Online]. Available: https://www.ietf.org/rfc/rfc768.txt

[2] D. Mills, "Network time protocol (NTP)," Internet Engineering Task Force, Tech. Rep., sep 1985. [Online]. Available: https://tools.ietf.org/html/rfc958

[3] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on networking*, vol. 5, no. 6, pp. 835–846, 1997.

[4] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on networking*, vol. 2, no. 1, pp. 1–15, 1994.

[5] Oracle, "Java Language Documentation." [Online]. Available: https://docs.oracle.com/en/java/

[6] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, A. Arendsen, T. Risberg, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B. Hale, A. Colyer, J. Lewis, C. Leau, M. Fisher, S. Brannen, R. Laddad, A. Poutsma, C. Beams, T. Abedrabbo, A. Clement, D. Syer, O. Gierke, and R. Stoyanchev, "Spring Framework Documentation." [Online]. Available: https://docs.spring.io/spring/docs/3.1.1.RELEASE/spring-framework-reference/html/

[7] Red Hat JBoss Middleware, "Hibernate ORM Documentation." [Online]. Available: http://hibernate.org/orm/documentation/

[8] Apache Software Foundation, "Apache Wicket Framework Documentation." [Online]. Available: https://ci.apache.org/projects/wicket/apidocs/1.5.x/index.html

[9] R. Mordani, Oracle, N. Abramson, Apache Software Foundation Art Technology Group Inc.(ATG), K. Avedal, BEA Systems, H. Bergsten, Boeing, Borland Software Corporation, Developmentor, J. Hunter, IBM, InterX PLC, R. Johnson, Lutris Technologies, New Atlanta Communications, LLC, Novell, Inc., Persistence Software, Inc., Pramati Technologies Progress Software, SAS Institute, Inc., Sun Microsystems, Inc., Sybase, and G. Wilkins, "JavaTM Servlet 2.4 Specification," Java Community Process, techreport 154, Nov. 2003. [Online]. Available: https://jcp.org/en/jsr/detail?id=154

[10] Python Software Foundation, "Python 2 Language Documentation." [Online]. Available: https://docs.python.org/2/

[11] R Development Core Team, "The R Language Manuals." [Online]. Available: https://cran.r-project.org/manuals.html

[12] J. Buck and L. Hambley, “Capistrano Documentation.” [Online]. Available: http://capistranorb.com/

[13] Y. Matsumoto, “Ruby Language Documentation.” [Online]. Available: https://www.ruby-lang.org/en/documentation/

[14] The Open Group and IEEE Std, “crontab,” The Open Group, techreport, 2001. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html

[15] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” phdthesis, University of California, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[16] Travis-CI Community, “Travis-CI.” [Online]. Available: http://travis-ci.org/

[17] Docker, Inc., “Docker.” [Online]. Available: https://www.docker.com/

[18] ——, “Dockerhub.” [Online]. Available: https://hub.docker.com/

[19] ——, “Docker Compose Documentation.” [Online]. Available: https://docs.docker.com/compose/

[20] Apache Software Foundation, “Apache Maven.” [Online]. Available: https://maven.apache.org/

[21] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java Virtual Machine Specification, Java SE 8 Edition*, A.-W. Professional, Ed. Addison-Wesley Professional, 2014. [Online]. Available: https://docs.oracle.com/javase/specs/jvms/se8/html/index.html

[22] Facebook, Instagram, and Community, “React framework (reactjs).” [Online]. Available: https://reactjs.org/

[23] T. Koppers, S. Larkin, J. Ewald, J. Vepsäläinen, K. Kluskens, and W. contributors, “Webpack.” [Online]. Available: https://webpack.js.org/

[24] M. Jones, J. Bradley, and N. Sakimura, “JSON web token (JWT),” IETF, Tech. Rep., may 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

[25] E. R. Fielding and E. J. Reschke, “Hypertext transfer protocol (HTTP/1.1): Authentication,” IETF, Tech. Rep., jun 2014. [Online]. Available: https://tools.ietf.org/html/rfc7235

[26] Netty Project Community, “Netty.” [Online]. Available: https://netty.io/

[27] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, and M. Simons, “Spring Boot Documentation,” 2017. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[28] The Scipy Community, "NumPy Reference Guide." [Online]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/

[29] ——, "Scipy reference guide." [Online]. Available: https://docs.scipy.org/doc/scipy-1.0.0/reference/

[30] J. D. Hunter, M. Droettboom, T. A. Caswell, and The Matplotlib Community, "Matplotlib user's guide." [Online]. Available: http://matplotlib.org/users/index.html

[31] The scikit-learn Community, "scikit-learn documentation." [Online]. Available: http://scikit-learn.org/stable/documentation.html

[32] Jupyter Team, "Jupyter documentation." [Online]. Available: https://jupyter.readthedocs.io/en/latest/contents.html

[33] A. Solem, "Celery user manual." [Online]. Available: http://docs.celeryproject.org/en/latest/

[34] Pivotal, "Rabbitmq documentation." [Online]. Available: https://www.rabbitmq.com/documentation.html

[35] J. Postel, "Transmission control protocol," DARPA Internet Program, Tech. Rep., sep 1981. [Online]. Available: https://tools.ietf.org/html/rfc793

[36] J. Nagle, "Congestion control in IP/TCP internetworks," Internet Engineering Task Force, Tech. Rep., jan 1984. [Online]. Available: https://tools.ietf.org/html/rfc896

[37] F. Yergeau, "UTF-8, a transformation format of ISO 10646," Internet Engineering Task Force, Tech. Rep., nov 2003. [Online]. Available: https://tools.ietf.org/html/rfc3629

[38] R. L. Rivest, A. Shamir, and L. M. Adleman, "Cryptographic communications system and method," United States Patent Patent 4,405,829, 1983. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=4405829.PN.&OS=PN/4405829&RS=PN/4405829

[39] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, and R. Nicholas, "Internet x.509 public key infrastructure: Certification path building," Internet Engineering Task Force, Tech. Rep., sep 2005. [Online]. Available: https://tools.ietf.org/html/rfc4158

[40] Q. H. Dang, "Secure hash standard," National Institute Of Standards and Technology, Tech. Rep., jul 2015. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[41] B. Kaliski, "PKCS #1: RSA encryption version 1.5," Internet Engineering Task Force, Tech. Rep., mar 1998. [Online]. Available: https://tools.ietf.org/html/rfc2437

[42] S. Aiyagari, A. Richardson, M. Arrott, M. Ritchie, M. Atwell, S. Sadjadi, J. Brome, R. Schloming, A. Conway, S. Shaw, R. Godfrey, M. Sustrik, R. Greig, C. Trieloff, P. Hintjens, K. van der Riet, J. O'Hara, S. Vinoski, and M. Radestock, "Advanced message queuing protocol," , techreport, 2008. [Online]. Available: https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

[43] T. Bray, "The JavaScript object notation (JSON) data interchange format," Internet Engineering Task Force, Tech. Rep., mar 2014. [Online]. Available: https://tools.ietf.org/html/rfc7159

[44] F. Wasilewski, "Pywavelets documentation." [Online]. Available: https://pywavelets.readthedocs.io/en/latest/#license

[45] Oracle, "Javafx." [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html

[46] ——, "Swing." [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

[47] ——, "Fxml." [Online]. Available: https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html

[48] ——, "Java preferences api." [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/preferences/index.html

[49] E. Syse and F. contributors, "Fxlauncher." [Online]. Available: https://github.com/edvin/fxlauncher

[50] Call-Em-All, "Material-ui." [Online]. Available: http://www.material-ui.com

[51] E. Rasmussen, "Redux." [Online]. Available: https://redux.js.org/

[52] ——, "Redux-form." [Online]. Available: https://redux-form.com

[53] J. y. c. Linux Foundation, "Node.js." [Online]. Available: https://nodejs.org

[54] I. y. c. StrongLoop, "Expressjs." [Online]. Available: https://expressjs.com/

[55] A. y colaboradores, "Passportjs." [Online]. Available: http://www.passportjs.org/

[56] T. G. y colaboradores, "Knexjs." [Online]. Available: http://knexjs.org/

[57] B. y colaboradores, "Bookshelfjs." [Online]. Available: http://bookshelfjs.org/

[58] R. y colaboradores, “Ramdajs.” [Online]. Available: http://ramdajs.com/

[59] Google, “Chrome v8.” [Online]. Available: https://developers.google.com/v8/

[60] I. Z. S. y colaboradores, “Npm.” [Online]. Available: https://www.npmjs.com/

[61] Azureus Software, Inc., “Vuze bittorrent client web page.” [Online]. Available: https://www.vuze.com/

[62] H. Hope, “inxi documentation.” [Online]. Available: https://smxi.org/docs/inxi-man.htm

[63] Apache Software Foundation, “Apache JMeter Documentation.” [Online]. Available: http://jmeter.apache.org/usermanual/index.html