Desarrollo de herramientas de apoyo al usuario de JASON: plataforma de procesado GNSS en el cloud

Paul Eduardo Paredes-Brito

Resumen– JASON es un servicio de posicionamiento GNSS que utiliza múltiples fuentes de información para realizar post-procesado PPK. Anteriormente JASON utilizaba protocolos en legacy code, por lo tanto, en este trabajo de fin de grado se ha refactorizado el código e implementado diferentes protocolos como FTP. A demás, se ha implementado un servicio de CORSFINDER el cual consiste en mostrar a los clientes de JASON las estaciones base disponibles e ir actualizándolas periódicamente. Finalmente se mejora la experiencia de usuario de JASON al enriquecer el aspecto visual, la presentación de información y la generación de informes automáticos de cada procesado.

Palabras clave– GNSS, PPK, PPP, SPP, AWS, serivicio cloud PPK , FastAPI, refactorización, código limpio, TDD, CORS.

Abstract– JASON is a GNSS positioning service that uses multiple data sources to perform PPK post-processing. Previously, JASON used protocols in legacy code; therefore, this final degree project has re-factorized the code to implement enhanced protocols such as FTP . Besides, a CORSFINDER service has been implemented to show JASON customers the available base stations and update them periodically. Finally, JASON's user experience is improved by enriching the visual aspect, the presentation of information, and the generation of automatic reports of each processing.

Keywords– GNSS, PPK, PPP, SPP, AWS, cloud PPK service, FastAPI, refactor, Clean code, TDD, CORS.

1 Introducción

Rokubun [1] es una compañía de telecomunicaciones de Barcelona que se especializa en la navegación GNSS (Global Navigation Satellite System) y la observación de la Tierra con el objetivo de ofrecer soluciones centradas en los campos de navegación de alta precisión, económicas y escalables. Dirigido al mercado de teléfonos móviles, vehículos aéreos no tripulados, servicios basados en ubicación, conducción autónoma, entre otros.

Tienen una plataforma de procesado cloud llamada JASON, PaaS (Positioning-as-a-Service) [2], es un sistema de procesado de datos GNSS, con el cual se consigue una geolocalización de alta precisión, utiliza diferentes estrategias según el tipo de solución como PPK (Post Processed Kinematic) PPP (Precise Point Positioning), SPP (Standard Positioning Point) [3] [4].Estas técnicas combinan mediciones GNSS de **estaciones base** de referencia cercanas a los receptores para corregir errores obteniendo una presición métrica. Soporta varios tipos de formatos, como por ejemplo: Ublox, Rinex 2/3 y Android Gnss logger, entre otros. además, cuenta con una API con la que se puede realizar el procesado a través de Bash, Python o Android.

El presente trabajo estará estructurado de la siguiente forma, en la sección 2 se tratarán los objetivos principales y secundarios, en la 3 el Estado del Arte en que se encuentra el trabajo, en el 4 la metodología que se usó, en el 5 la planificación, en el 6 se detalla el desarrollo y los componentes implicados. Por último, las conclusiones y futuro trabajo que tendrá el presente trabajo de fin de carrera.

• E-mail de contacto: pauleduardo.paredes@e-campus.uab.cat
• Mención realizada: Tecnología de la Información
• Trabajo tutorizado por: Miguel Carpio (DEiC)
• Curs: 2020/2021

2 Objetivos

2.1. Objetivo Principal

El objetivo principal es añadir valor a JASON, desarrollando nuevas funcionalidades como es la generación de reportes y la creación de un servicio de CORSFINDER.

2.2. Objetivos Secundarios

Este trabajo consta de 4 objetivos secundarios que servirán para cumplir el objetivo principal.

2.2.1. Refactorizar protocolos para obtención de estaciones

JASON utiliza múltiples fuentes de información para proporcionar post-procesado PPK de forma automática, para ello utiliza protocolos como FTP, que es un legacy code, lo que implica que no está optimizado y el funcionamiento se puede ver alterado y limitado.

2.2.2. Implementar CORSFINDER

Aprovechando el punto anterior se planea desarrollar un nuevo servicio a JASON, CORSFINDER, que servirá para visualizar las estaciones base disponibles y se irá actualizando periódicamente.

2.2.3. Generar Reportes

Se proporcionará al usuario un reporte con información técnica sobre el procesado que ha realizado, esta tarea será realizada automáticamente cada vez que se realice un procesado.

2.2.4. Mejorar la experiencia de usuario

Para mejorar la experiencia de usuario se cambiará la manera de generar gráficos estadísticos, que se utilizarán en la página web y en los reportes, que son generados automáticamente cada vez que un cliente realice un procesado.

3 Estado del Arte

El desarrollo de soluciones de geolocalización a través de navegación por satélite ha evolucionado en los últimos años con la incorporación de las nuevas tendencias tecnológicas y con ello nuevas necesidades como pueden ser el uso de UAV (vehículo aéreo no tripulado), la incorporación de sistemas de conducción autónoma, servicios basado en localización, mapeo y agrimensura, entre otras. El mercado de la navegación por satélite se podría dividir en dos grupos, el primero dirigido a un grupo masivo como pueden ser usuarios de teléfonos móviles, teniendo costes bajos y con precisión baja. El segundo grupo es el mercado profesional que tienen costes muy elevados, pero consiguiendo una precisión submétrica.

La utilización de datos de CORS [5] (Continuously Operating Reference Station) es importante para realizar diferentes técnicas de navegación o procesado. La información de estas estaciones está alojadas por organismos públicos y/o privados, que ofrecen la varias opciones para descargar esta información como pueden ser por vía FTP, FTPS o HTTP.

Por ejemplo, CDDIS (The Crustal Dynamics Data Information System) es un banco de datos inicialmente creado para proveer a un proyecto de la NASA, actualmente archiva y distribuye, principalmente, datos GNSS (incluye CORS), GPS, entre otros. Por razones de seguridad CDDIS desde el mes de octubre del 2020 deja de dar soporte al protocolo FTP anónimo sin cifrado para dar paso a protocolos más seguros como FTPS o HTTPS. Como CDDIS hay más organismos que están migrando sus servidores y para adaptarse a los nuevos requerimientos se mejorará la forma de obtener estos datos en el presente trabajo.

4 Metodología

Para el desarrollo del proyecto se siguió la metodología basada en entrega, se trata de una estrategia de desarrollo ágil la cual consiste en dividir las tareas en partes pequeñas e ir trabajando en éstas en un periodo corto de tiempo, de una a dos semanas, con el objetivo de evitar un sentimiento de baja productividad a la hora de realizar un desarrollo muy grande, con esto se pudo ver los avances más rápido, a demás de que al ser cambios pequeños el riesgo de comprometer el funcionamiento de los servicios en producción es menor.

Para ello se utilizarán diferentes herramientas como por ejemplo Git, siguiendo una política TBD (Trunk-Based Development) [6] la cual consiste en tener la rama principal(main) como única para el desarrollo, evitando ramas extras para desarrollar nuevas funcionalidades; con ello se evitan problemas de compatibilidad al hacer marges, se trabaja siempre con una versión actualizada del código, y no se pierde la trazabilidad de los commits, lo que facilita la corrección de bugs.

Junto a la política de TDB se utilizó versionamiento semántico [7], que consiste en diferenciar las versiones dependiendo del tipo de commit que se ha hecho (feat, break, fix, etc) con esto se consigue generar versiones software de forma automática. Se apuede apreciar el uso de estos componentes en la Figura 1 en el 1) la conveción del mensaje del commit. 2) Versionado del código que se genera cada vez que se realizan los push.

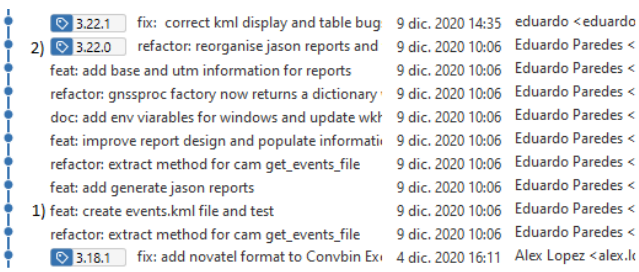

Fig. 1: Metodología para Git

También usó la metodología TDD (Test-driven development) para realizar el desarrollo de una manera segura y generar la menor cantidad de errores, en el apartado 6.3 se ampliará más esta información.

5 PLANIFICACIÓN

El proyecto se divide en tres partes como se puede observar en el Figura 13. La primera consiste en planificar y preparar las tareas a realizar, la segunda parte se divide en subsecciones, las cuales serán las características que se desarrollarán, que, a su vez, constan de tres partes: análisis, diseño y desarrollo. Y por último está la documentación que consiste en la redacción del informe, creación del poster y la defensa ante el tribunal.

Cabe resaltar que, se harán reuniones de seguimiento cada semana y al finalizar cada nueva característica desarrollada para darle continuidad al proyecto.

6 DESARROLLO

6.1. Arquitectura

En la siguiente Figura2 se puede apreciar la arquitectura y los componentes de JASON, consta de 3 partes:

- **Front-End:** Netlify se encarga de alojar y desplegar el sitio web, las tecnologías que utiliza el front-end son Angular, HTML y CSS. En este punto inicia el flujo de trabajo, es decir, el usuario sube o accede un fichero para ser procesado.
- **Cloud:** El servicio de AWS Elastic Beanstalk es utilizado para administrar y controlar los servicios web como pueden ser: Amazon SQS - sistema distribuido de colas, Amazon RDS - servicio de Base de datos relacional, Amazon S3 - servicio de almacenamiento en la nube. Aquí se almacenarán los procesados de los clientes y también se accederá a estos para generar los informes.
- **Back-End:** Está desarrollado en Python y se utiliza Docker SWARM para gestionar los contenedores Docker. Cuenta con una API responsable de responder al front-end y mantener la base de datos actualizada, el Dispatcher se encarga de preguntar a la API si hay procesos pendientes para ser procesados, y el sistema de Biling que se encarga de los cobros y facturación.

  Tiene dos módulos, Pyrok y Rokubun-core encargados de realizar las tareas de procesado y post-procesado, las cuales se refactorizaron y se adaptaron para la obtención de datos relevantes para los reportes.

  Está basado en un sistema de microservicios [8], es decir cada componente da la aplicación se ejecuta independientemente, frontend, API, etc. Con esto cada servicio desempeña una sola función. Esto permite que las aplicaciones sean más escalables y que se pueda desarrollar más rápido.

6.2. Entorno de trabajo

Para desarrollar las nuevas funcionalidades de JASON se trabajó bajo un entorno de pruebas basado en Docker-compose, simulando el funcionamiento real de JASON, para poder realizar el desarrollo sin comprometer el funcionamiento de producción. Para ello se desplegaron 5 contenedores: Frontend, API, Dispatcher, Mysql database, AWS S3(fake) y Pyrok.

6.3. Test

Una de las filosofías de desarrollo en Rokubun es el uso de TDD (Test driven development), primero realizar y validar los tests, una vez hechos se podrá implementar el código y deberá pasarlo. Con esto se minimizan errores, se pueden identificar problemas de funcionalidad, se evita duplicar el código, ayuda a controlar la calidad del código y crear documentación actualizada del código. En la refactorización tener test de buena calidad es fundamental para comprobar que el comportamiento no ha sido alterado una vez que el código ha sido modificado.

El desarrollo de las nuevas funcionalidades siguió estas mismas prácticas de TDD, además de adaptar y modificar con los test ya existentes.

6.4. CI/CD

JASON cuenta con un sistema de integración y entrega continua para su desarrollo, cada vez que un desarrollador realice un cambio en uno de sus componentes tendrá que pasar por unas etapas hasta que el cambio esté en producción o en el entorno de pruebas. A continuación, se muestra en la figura 3 el flujo de trabajo de CI/CD de JASON.

1. Development: Es el desarrollo que se realiza en local, aquí se realizará un clone del repositorio a desarrollar.
2. Repository: Es el repositorio alojado en un servidor propio de GitLab, en el cual se encuentran todos los componentes de JASON, como son Frontend, Dispatcher, Pyrok, API, y demás componentes.
3. CD/CI Pipeline: Consiste en varias etapas: el build desplegará cada uno de los servicios que están en contenedores Docker, se realizará un pull de las últimas imágenes para realizar los test. Hay varios tipos de tests, unitarios e integración.
4. Registry: Una vez que se construidas y probadas las imágenes de Docker serán registradas/publicadas para usarse como contenedores de servicios por separado para realizar pruebas.
5. Server: Por último, se desplegarán en producción o en el entorno de pruebas.

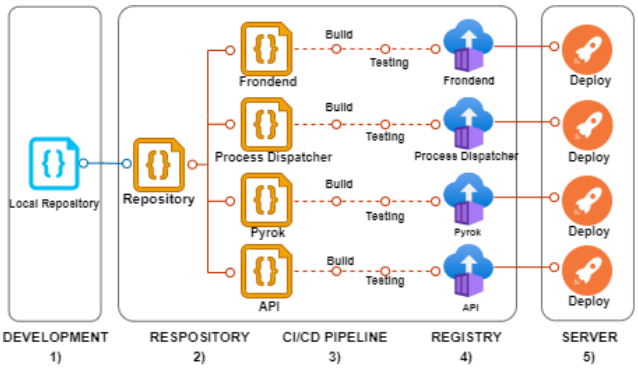

Fig. 3: Gitlab pipeline CI/CD

Con el uso de la integración y despliegue continuo ha hecho que el desarrollo de este proyecto sea más seguro y

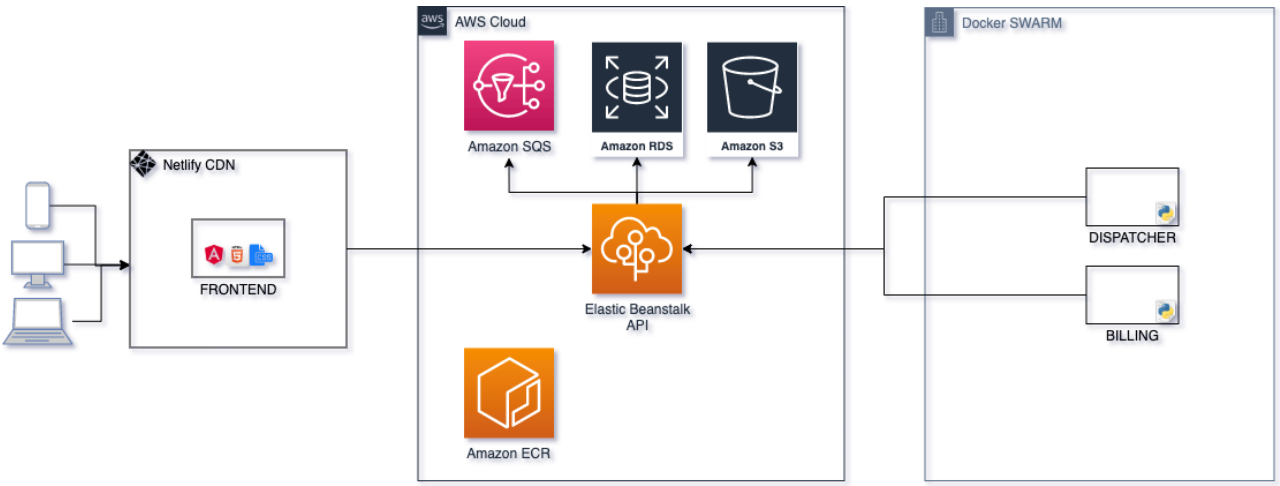

Fig. 2: Arquitectura JASON

con menos errores, a la hora de integrar o añadir una nueva funcionalidad en producción.

6.5. Creación de reportes

En las Figuras [4, 5] se muestran la información común de cada reporte: La versión del software, el correo del usuario, la información de la estación base (nombre, firmware, frecuencia o satélites disponibles), el número de épocas totales y las épocas utilizables, y las constelaciones del recibidor. También información específica como el método de procesamiento, el tipo de estrategia o el reloj de alta precisión.

Engine revision:	3.23.6.post2
Processing time:	2021-01-11 14:14:20 UTC
Base station name:	base_mare_catnet.rnx
Base antenna:	MARE
Base receiver:	LEICA GR50 / 4.31/7.403
Base sampling rate:	1.0 seconds
Precise clocks:	cod21385.clk_05s
Ephemeris used:	ROKB00ESP_R_20203400000_24...
Processing method:	(PPK / PPP / SPP)
Base constellations:	Beidou, Galileo, Gps, Glonass

Fig. 4: Información común de Reportes

Account owner:	eduardo.paredes@rokubun.cat
Process Id:	6522
Rover file name:	raw_202012051328.UBX.rnx
Rover first epoch:	2020-12-05 13:28:31
Rover last epoch:	2020-12-05 14:56:23
Rover time span:	1:27:52
Rover epoch count:	52538
Rover dynamics:	Dynamic / Static
Usable Epochs	52538 (100.00%)
Rover constellations:	Gps, Glonass, Galileo, Beidou

Fig. 5: Información común de Reportes

A. **Estático**, este tipo de procesado consiste en que los receptores están fijos en durante el periodo de observación registrando épocas para calcular el posicionamiento. Este método es ideal para grandes distancias y tiene mayor precisión. Las principales aplicaciones son: Redes geodésicas, control geométrico de cartografía.

En las Figuras[6, 7, 8] se puede ver la información obtenida para las diferentes estrategias, como son las coordenadas del receptor y/o estación base, en diferentes sistemas métricos: ECEF (Earth Centered Earth Fixed), coordenadas polares, coordenadas elipsoidales o UTM (Universal Transverse Mercator) [9].

Datum:	Unknown, as established by the provided coordinates
Base position:	Earth Centered Earth Fixed
ECEF X:	4201792.291 m
ECEF Y:	177945.235 m
ECEF Z:	4779286.679 m
Latitude(φ):	48.844441746° 48° 50' 39.99028"
Longitude(λ):	2.425018075° 2.0° 25.0' 30.06507"
Ellipsoidal Height:	126.229 m
Universal Transverse Mercator	
Zone:	31
UTM X:	457813.650 m
UTM Y:	5410322.723 m
Convergence:	-0.432925°
Scale factor:	0.99962186

Fig. 6: Información Base - PPK estático

Y el cálculo de la diferencia entre las distancias entre la estación base y el recibidor.

B. **Dinámico**, es un receptor móvil que registra varias

épocas para calcular el posicionamiento de cada época. En los procesados dinámicos se añadió un mapa con el resultado del procesado y estadísticas como el número de satélites o los satélites utilizados como se muestran en las Figuras[9, 10, 11, 12].

Un problema que surgió fue al printar los procesados en el mapa se utilizaba un fichero KML el cual sería renderizado por la API de Google Maps [10] pero al tener que renderizar grandes cantidades de información el rendimiento se veía afectado al generar y posteriormente al abrir los reportes. Para solucionar este problema se optó por transformar este fichero KML a un GeoJson en el que sólo se printan los puntos necesarios para no afectar el rendimiento y que la información no se vea afectada.

El desarrollo de los reportes se realizó en cuatro partes, en la figura 14 se muestra el diagrama de secuencia del proceso de generación del reporte:

- Obtención de información del procesado: Desde **Pyrok** se refactorizó y adaptó el código con el que anteriormente se trabajaba, para facilitar el envío de información del procesado hacia el template.
- Creación de un template HTML: Se crearon varios modelos de reportes según el procesado utilizando HTML junto a **Jinja** [12], que es un herramienta de Python que sirve para crear plantillas HTML.
- Convertir HTML a PDF: Se utilizó **Wkhtmltopdf** [11] otra herramienta de Python para convertir ficheros HTML a PDF.
- Almacenar reporte: Para almacenar los reportes se utilizó AWS S3 el cual se podrá acceder a través de la web de JASON.

Datum:	Unknown, as established by the provided coordinates	
Rover position:	Earth Centered Earth Fixed	
ECEF X:	4201301.065 m	
ECEF Y:	184592.160 m	
ECEF Z:	4779412.212 m	
Latitude(φ):	48.846567579° 48° 50' 47.64328"	±0.064m
Longitude(λ):	2.515781113° 2° 30' 56.81201"	±0.074m
Ellipsoidal Height:	86.322 m	±0.045m
Universal Transverse Mercator		
Zone:	31	
UTM X:	464474.403 m	
UTM Y:	5410512.687 m	
Convergence:	-0.364596°	
Scale factor:	0.99961550	

Fig. 7: Información Rover - PPK/SPP/PPP estático

Base to rover baseline	
ECEF X:	-491.226 m
ECEF Y:	6646.925 m
ECEF Z:	125.533 m
Base-rover 3D dist.	6666.234 m
Vector azimuth:	-178.650°
Base-rover ellipsoid dist.:	10039.029 m
Height Δ:	-39.907 m

Fig. 8: Información Rover a Estación Base PPK estático

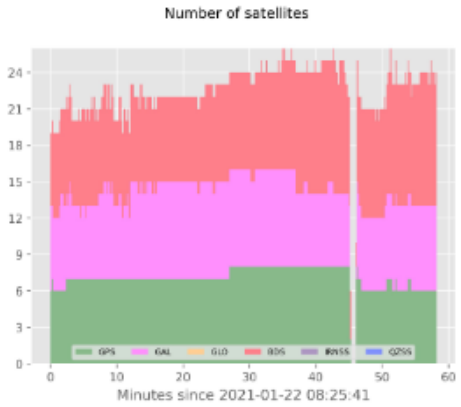

Fig. 9: Número de satélites

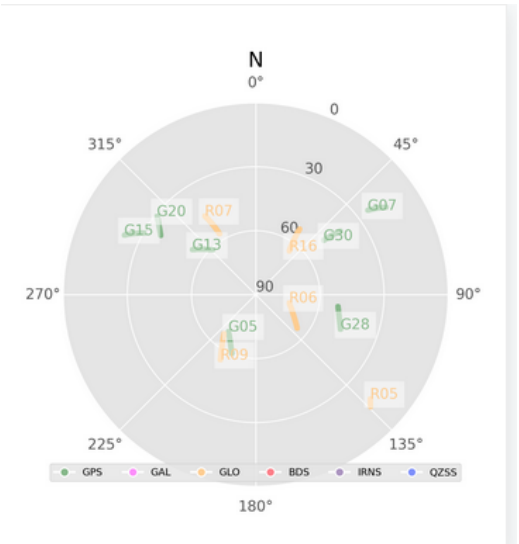

Fig. 10: Skyplot

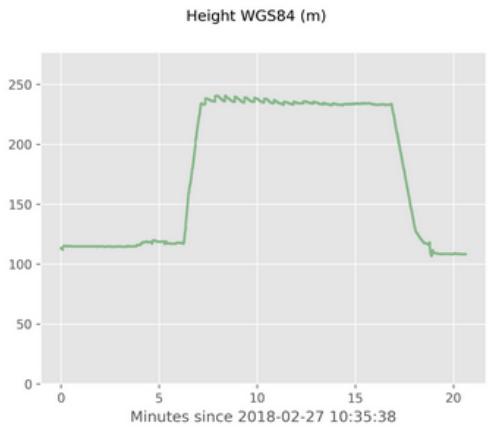

Fig. 11: Altura WGS84

Esta fue la etapa que más tiempo tomó en desarrollar debido al legacy code, se tuvo que adaptar al modelo y estructura de programar, además de utilizar librerías o código ya hecho el cual se tuvo que estudiar y revisar para poder utilizarlo de una forma correcta. Se realizaron varios code reviews los cuales fueron de gran utilidad para mejorar la calidad de código y corregir errores.

6.6. CORSFINDER

Como se ha mencionado anteriormente el uso de las estaciones es importante para realizar los post-procesados de JASON, es por eso que se creó un servicio que muestre y liste estas estaciones base y otro que actualice periódicamente. Se realizó en dos partes este servicio:

La primera parte consiste en crear un módulo de Python el cual está integrado en la librería de Pyrok para utilizarlo en los procesados. Además, siguiendo la misma arquitectura basada en microservicios se creó un contenedor Docker para el desarrollo de una API, se utilizó el framework Fast API [13] que caracteriza por alto rendimiento que tiene para realizar peticiones, además cuenta con un sistema propio para generar documentación.

Para realizar el desarrollo de la API se utilizó la arquitectura Modelo-Vista-Controlador, para separar cada tipo de lógica así facilitar el mantenimiento y la escalabilidad, esto también ayuda para la realización de test unitarios de cada componente. Se crearon tres peticiones:

- **Get stations**, que se le podrá pasar como parámetro coordenadas (latitud y longitud), o el nombre de la estación, o el nombre de la red. El resultado será un listado de GeoJSON con la información de la o las estaciones que coincidan con los parámetros ingresados. Esta información puede ser ampliada en futuras iteraciones.
- **Upload stations** en el cual acepta un fichero GeoJSON con la información de las estaciones y se actualizará en el fichero interno de estaciones.
- **Get status** Por último, una petición que devolverá la información referente a la API: la versión y el estado de la misma.

Para evitar posibles ataques o mal uso de la API se añadió un sistema de autentificación de APIkey, que podrá ser la misma que se utiliza en el servicio de JASON.

La información de estas estaciones fue almacenada en un fichero JSON debido a que su contenido es estático, variará muy poco, sólo cuando se añada una estación por una petición de la API o cuando se realice la actualización se encuentre una nueva, por este motivo no se creó una base de datos.

La segunda parte se creó otro contenedor, **CORS updater**, el cual utilizará el servicio de la API creada anteriormente. La función de éste es realizar actualizaciones de las estaciones base periódicamente utilizando ECS [14] de AWS que permite gestionar contenedores, y una de sus funcionalidades es programar tareas, a la cual se le asignó un evento con el lapso de tiempo en el que lanzará el proceso de actualizar las estaciones.

6.7. Refactorizar servicio FTP

Para obtener la información de las estaciones base es necesario descargar ficheros Rinex desde varios servidores; anteriormente tenían sistema basado en plantillas, un fichero XML, el que contaba con la información de las redes, protocolo, hostname, URLs, nombre del fichero, versión del fichero, entre otra información. Con la que se construían las URIs para de cara red descargar de forma dinámica cada una de las estaciones. Con el transcurso del tiempo la información de estas estaciones fue cambiando como, por ejemplo, el tipo de protocolo (HTTPS o FTPS) o las rutas o los nombres de los ficheros o dejaron de publicar información de las estaciones. Esto ha generado perdida de información, alrededor mil estaciones de diferencia con respecto a versiones anteriores.

Para solucionar este problema se refactorizó el proceso de descarga, primero se cambió el template XML por varios ficheros JSON para facilitar el mantenimiento de las redes, eliminar información innecesaria y mostrar la información más clara. Segundo se refactorizó el código para que utilice la información de estos ficheros JSON y pueda descargar los ficheros de las estaciones base. Por último, se actualizó la información de cada red accediendo a cada una de ellas, el cambio más habitual fue el cambio de nombre de la ruta y el tipo de compresión, de Z a gzip.

El principal problema a la hora de refactorizar el código fue intentar a provechar el código antiguo, que era muy difícil de seguir y se repetía en varias clases, por lo cual se rehízo desde cero el sistema de descargar para tiendo cuenta los posibles escenarios futuros para incorporar nuevas funcionalidades. El punto crítico fue en que el comportamiento de los procesados no se vea afectado por este cambio.

7 Resultados

Después de realizar este trabajo de fin de grado se ha conseguido el objetivo principal que fue, añadir valor a la plataforma de JASON, se desarrollaron nuevas funcionalidades, la creación de reportes. A día 07 de febrero de 2021 se han generado 150 reportes en producción. Se ha reportado un bug desde la puesta en producción en un procesado dinámico en el que el mapa no tenía información suficiente para centrarlo y hacía que el todo el procesado fallace. Después

Fig. 12: Mapa de procesado dinámico

de corregir este problema no se han reportado más incidentes.

El servicio de CORSFINDER está en el entorno de pruebas para que en los próximos días pueda ser puesto en producción.

8 Conclusiones

Rokobun es una empresa especializada en la navegación GNSS que ofrece soluciones centradas en la navegación de alta precisión, económica y escalable; tiene JASON un servicio de post-procesado en el cual se centró este trabajo de fin de grado. Teniendo como objetivo principal el agregar valor a JASON desarrollando nuevas funcionalidades. En el desarrollo de estas funcionalidades se detalló la arquitectura de JASON, que está basado en un sistema de microservicios, el entorno de trabajo con el que cuenta para su desarrollo, y el flujo de trabajo que tiene cada vez que se añada una nueva funcionalidad con la integración y entrega continua.

El trabajar con legacy code fue una complicación en las primeras etapas del desarrollo ya que había mucho código por revisar y hasta no estar familiarizado el progreso fue lento, además de adaptarse a las políticas de desarrollo.

La primera funcionalidad desarrollada fue la **generación de reportes** automáticamente de cada procesado, clasificando los reportes en estáticos y dinámicos. El desarrollo constó de cuatro partes: obtención de información, creación de template HTML, conversión de HTML a PDF y el almacenamiento en S3 y la base de datos. También se puede acceder a través de la página web de JASON.

También se desarrolló un servicio para listar, actualizar o añadir una nueva estación base, CORSFINDER, tiene dos componentes un módulo en Python que utiliza Pyrok, el segundo componente una API hecha en FastAPI la cual se puede realizar las peticiones mencionadas anteriormente. A demás de ir actualizándolas periódicamente mediante un servicio de AWS.

Por último, se refactorizó el sistema FTP de descargas, cambiando el template XML por varios ficheros JSON para facilitar su mantenimiento, también se corrigió y actualizó la información de cada una de las redes para evitar la pérdida de información.

9 Futuro Trabajo

9.1. Reportes

A los reportes se podrá añadir o modificar nueva información de acuerdo a las especificaciones requeridas; A demás que se podrá utilizar la información generada para mostrarla en la página web de JASON.

9.2. Corsfinder

En Corsfinder se podría implementar una aplicación web que muestre en un mapa, con un marcador, el posicionamiento de las estaciones base, coordenadas, ficheros para descargar, y más información. Otra opción que se puede añadir es que el usuario pueda buscar las estaciones por coordenadas, nombre, o red. Como también pueda añadir, editar, o eliminar esta información, En el caso en que la información varíe con mayor frecuencia, se tendría que implementar una base de datos para agilizar todos los procesos.

Otra funcionalidad que se podría agregar es un sistema de notificación de versiones, cada vez que se realice el proceso genere un mensaje y se envía a un canal de Slack la información de la actualización, por ejemplo: las estaciones nuevas, estaciones no encontradas, duplicadas, etc.

9.3. Protocolos de descarga

El sistema de descarga se podrá a incorporar nuevos protocolos de descarga como pueden ser HTTP, HTTPS o S3. Para agilizar el proceso de descarga se podría utilizar diferentes métodos como puede ser el uso de threads para la descarga, aunque los servidores FTP suelen estar limitados a un número de descargas simultaneas. Otra opción es montar varios contenedores en AWS con lo no habría el problema mencionado anteriormente.

10 Agradeciemientos

Quiero agradecer a Rokubun por la confianza que pusieron en mi para realizar este trabajo de fin de carrera, a Alex López, que fue tutor y mentor por parte de Rokubun.

Y finalmente a Miguel Carpio, tutor por parte de la universidad por ayudarme en todo el proceso de este trabajo.

REFERENCIAS

[1] Rokubun. [Online]. Available: https://rokubun.cat/. [Ac-cessed: 21-Sep-2020].

[2] JASON PaaS. [Online]. Available: https://jason.rokubun.cat/. [Accessed: 21-Sep-2020].

[3] "Precise Point Positioning," Precise Point Positioning - Navipedia. [Online]. Available: https://gssc.esa.int/navipedia/index.php/Precise_Point_Positioning. [Accessed: 29-Sep-2020]

[4] "How PPK works," How PPK works - Reach RS/RS docs. [Online]. Available: https://docs.emlid.com/reachrs/common/tutorials/ppk-introduction/. [Accessed: 29-Sep-2020].

[5] N. O. A. A. US Department of Commerce, "CORS Homepage," National Geodetic Survey, 17-Aug-2009. [Online]. Available: https://www.ngs.noaa.gov/CORS/. [Accessed: 07-Nov-2020].

[6] Paul-Hammant, "Trunk Based Development," Introduction. [Online]. Available: https://trunkbaseddevelopment.com/. [Accessed: 05-Feb-2021].

[7] Conventional Commits. [Online]. Available: https://www.conventionalcommits.org/en/v1.0.0/. [Accessed: 05-Feb-2021].

[8] Microservicios Amazon, [Online]. Available: https://aws.amazon.com/es/microservices/.

[9] "Cartesian and ellipsoidal coordinates," Cartesian and ellipsoidal coordinates - Navipedia. [Online]. Available: https://gssc.esa.int/navipedia/index.php/Cartesian_and_ellipsoidal_coordinates. [Accessed: 05-Feb-2021].

[10] Google. [Online]. Available: https://developers.google.com/maps/documentation/javascript/kml. [Accessed: 05-Feb-2021].

[11] wkhtmltopdf. [Online]. Available: https://wkhtmltopdf.org/. [Accessed: 03-Nov-2020].

[12] "Jinja," Jinja. [Online]. Available: https://jinja.palletsprojects.com/en/2.11.x/. [Accessed: 03-Nov-2020].

[13] FastAPI. [Online]. Available: https://fastapi.tiangolo.com/. [Accessed: 10-Jan-2021].

[14] .ªmazon Elastic Container Service". [Online]. Available: https://aws.amazon.com/es/ecs. [Accessed: 15-Jan-2021].

[Accessed: 25-Jan-2021].

APÉNDICE

A continuación se muestra los apéndices utilizados.

A.1. Diagrama de Gantt

B.2. Diagrama de flujo de Reportes

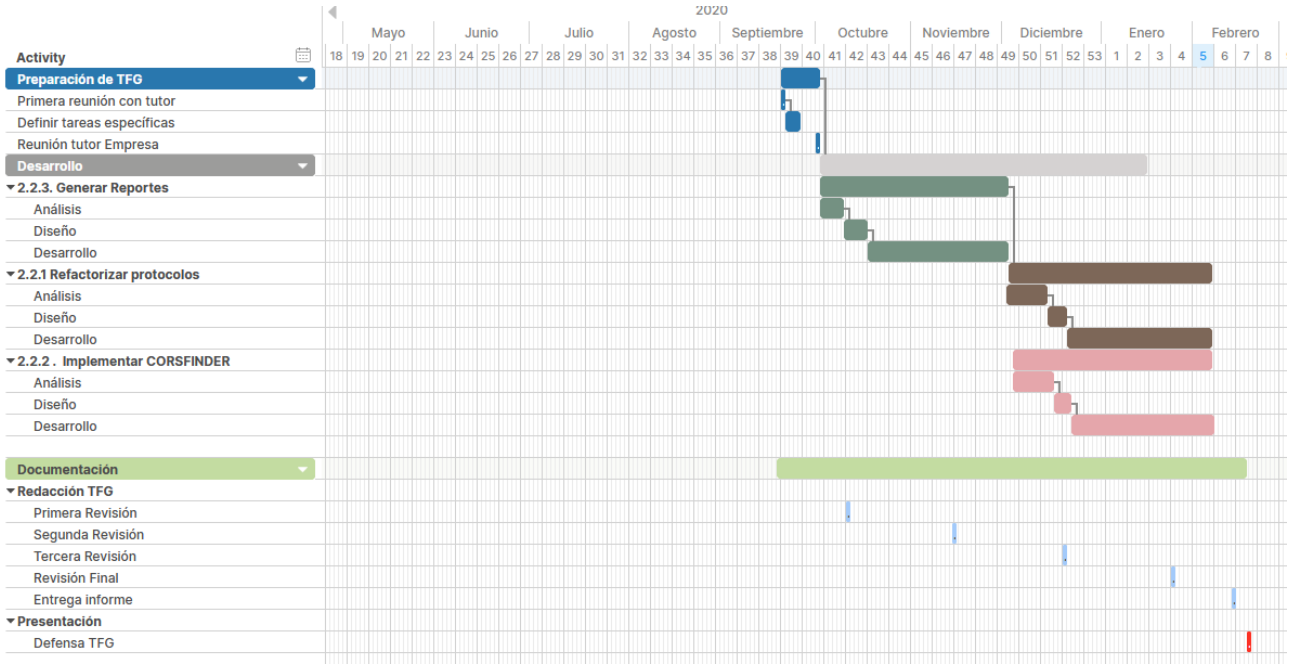

Fig. 13: Diagrama de Gantt

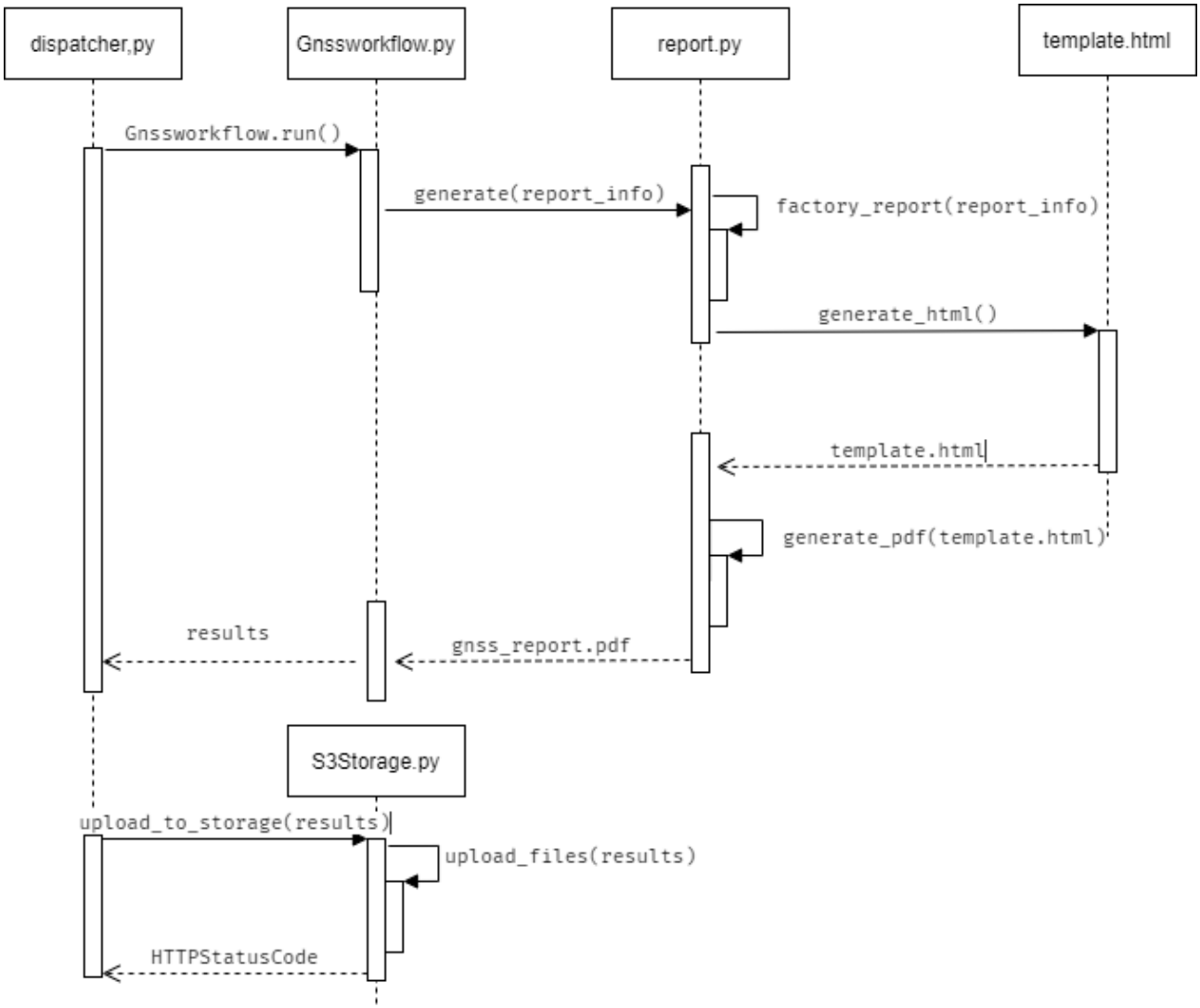

Fig. 14: Diagrama de flujo Reportes