

BLUEJAY - TP TESTER

Índice

Índice.....	1
Introducción.....	2
¿Para qué sirve Bluejay-TP Tester?.....	2
Puesta en marcha.....	2
Modo desarrollo.....	2
Con docker.....	2
APIs del proyecto.....	3
Glassmatrix API.....	3
Otras APIs.....	5
Estructura proyecto angular.....	6
Componentes.....	6
Páginas.....	6
Página testeo TPAs.....	6
Editar por métricas/garantías.....	6
Página métricas.....	7
Subpantalla edición de métricas.....	8
Página testeo de repositorios.....	9
Subpantalla de Ramas.....	10
Subpantalla de Pull Requests.....	11
Subpantalla de Acciones.....	11
Servicios.....	12
Bluejay Service.....	12
Files Service.....	13
Github Service.....	14
Glassmatrix Service.....	15
Métodos para Operaciones de Archivos:.....	15
Métodos para Operaciones de Tokens de GitHub:.....	15
Métodos para Acciones de GitHub:.....	16
Traducciones.....	16

Introducción

Bluejay Infrastructure es una infraestructura basada en Governify que permite auditar equipos ágiles de manera sencilla. Está compuesta por un subconjunto de microservicios de Governify que pueden ser desplegados ya sea en una sola máquina o en un clúster. Bluejay accede a múltiples fuentes para recopilar información sobre los equipos de desarrollo, como GitLab, Jira, Slack, etc., y utiliza esta información para verificar si esos equipos cumplen con un Acuerdo de Prácticas de Equipo (TPA) que incluye métricas y garantías relacionadas con la metodología ágil.

¿Para qué sirve Bluejay-TP Tester?

El objetivo de este TP-Tester es, como dice su nombre, poder probar en primer lugar las métricas que componen los TPs (Team Practices). Una vez comprobadas que estas métricas funcionan correctamente, TP-Tester te permite comprobar que todo el TPA (Team Practice Agreement) funcione correctamente. Para ello, se pueden añadir, eliminar, o modificar TPAs que se encuentran directamente en Bluejay desde este TP Tester.

Puesta en marcha

Modo desarrollo

Para levantar Bluejay-TP Tester en modo desarrollo, sigue los siguientes pasos:

1. Clona el repositorio de Bluejay-TP Tester.
2. Instala las dependencias con ``npm install``.
3. Levanta el proyecto con ``npm start``.

Con esto bastaría ya que el proyecto utiliza *concurrently* y levanta a la vez el servidor express y angular. El servidor Express (GlassMatrix API) se levanta en el puerto 6012 y la aplicación Angular en el puerto 4200.

Con docker

Para levantar Bluejay-TP Tester con docker, sigue los siguientes pasos:

1. Clona el repositorio de Bluejay-TP Tester.
2. Instala las dependencias con ``npm install``.
3. Ejecuta ``npm run docker``.


Con esto ya tendríamos el proyecto levantado en el puerto 6011 la web angular, y el servidor express en el puerto 6012.






APIS del proyecto


Glassmatrix API






Este proyecto tiene su propia API. Esta api se usa tanto para interactuar con la infraestructura de Bluejay, que permite crear TPAs, probar métricas como para realizar acciones de github en local como clonar repositorios, crear ramas, moverte de ramas, hacer commits o pushes.

Esta API se encuentra en el puerto 6012 tanto en modo desarrollo como cuando se ejecuta con docker. Esta API también utiliza Swagger, una herramienta popular para documentar APIs. Swagger ayuda a definir, construir, documentar y consumir servicios web RESTful ya que te permite incluso probar los endpoints desde una interfaz gráfica. En este caso, esta interfaz se encuentra en **localhost:6012/api** o **localhost:6012/docs** y en ella podemos ver todos los endpoints que listo y explico a continuación:

Bluejay Endpoints for the Bluejay section 

POST	/glassmatrix/api/v1/tpa/save	
POST	/glassmatrix/api/v1/tpa/update	
GET	/glassmatrix/api/v1/tpa/files	
GET	/glassmatrix/api/v1/tpa/files/{fileName}	
DELETE	/glassmatrix/api/v1/tpa/files/{fileName}	

Github Endpoints for the Github section 

POST	/glassmatrix/api/v1/github/token/save	
GET	/glassmatrix/api/v1/github/token/get	
DELETE	/glassmatrix/api/v1/github/token/delete	
POST	/glassmatrix/api/v1/github/cloneRepo	
GET	/glassmatrix/api/v1/github/listRepos	

1. `POST **/glassmatrix/api/v1/tpa/save**` permite guardar un archivo. Los parámetros fileName y content se envían en el cuerpo de la solicitud. Si hay un error al guardar el archivo, se devuelve un código de estado HTTP 500. Si el archivo se guarda correctamente, se devuelve un mensaje de éxito.
2. `POST **/glassmatrix/api/v1/tpa/update**` permite actualizar un archivo existente. Los parámetros fileName y content se envían en el cuerpo de la solicitud. Si el archivo existe, se borra y se crea uno nuevo con el contenido proporcionado. Si hay un error

al guardar el archivo, se devuelve un código de estado HTTP 500. Si el archivo se actualiza correctamente, se devuelve un mensaje de éxito.

3. ``GET /glassmatrix/api/v1/tpa/files`` devuelve todos los archivos .json en la carpeta de activos.
4. ``GET /glassmatrix/api/v1/tpa/files/:fileName`` devuelve el contenido de un archivo .json específico. El nombre del archivo se pasa como parámetro en la ruta.
5. ``DELETE /glassmatrix/api/v1/tpa/files/:fileName`` permite eliminar un archivo específico. El nombre del archivo se pasa como parámetro en la ruta. Si el archivo no existe, se devuelve un código de estado HTTP 404. Si hay un error al eliminar el archivo, se devuelve un código de estado HTTP 500. Si el archivo se elimina correctamente, se devuelve un mensaje de éxito.
6. ``POST /glassmatrix/api/v1/github/token/save`` permite guardar un token. El token se envía en el cuerpo de la solicitud. Si el archivo que almacena el token existe, se borra y se crea uno nuevo con el token proporcionado. Si hay un error al guardar el token, se devuelve un código de estado HTTP 500. Si el token se guarda correctamente, se devuelve un mensaje de éxito.
7. ``GET /glassmatrix/api/v1/github/token/get`` devuelve el token guardado. Si hay un error al leer el token, se devuelve un código de estado HTTP 500.
8. ``DELETE /glassmatrix/api/v1/github/token/delete`` permite eliminar el token guardado. Si el archivo que almacena el token no existe, se devuelve un código de estado HTTP 404. Si hay un error al eliminar el archivo, se devuelve un código de estado HTTP 500. Si el archivo se elimina correctamente, se devuelve un mensaje de éxito.
9. ``POST /glassmatrix/api/v1/github/cloneRepo``: Esta ruta clona un repositorio de GitHub. Toma el propietario y el nombre del repositorio como parámetros del cuerpo de la solicitud, construye la URL del repositorio y clona el repositorio en un directorio local.
10. ``GET /glassmatrix/api/v1/github/listRepos``: Esta ruta lista todos los repositorios clonados. Lee el directorio donde se almacenan los repositorios y devuelve una lista de nombres de directorios.
11. ``GET /glassmatrix/api/v1/github/branches/:repoName``: Esta ruta lista todas las ramas de un repositorio específico. Ejecuta el comando ``git branch`` en el directorio del repositorio y devuelve los nombres de las ramas.
12. ``DELETE /glassmatrix/api/v1/github/deleteRepo/:repoName``: Esta ruta elimina un repositorio específico. Elimina el directorio del repositorio del sistema de archivos.
13. ``POST /glassmatrix/api/v1/github/createBranch/:repoName``: Esta ruta crea una nueva rama en un repositorio específico. Toma el nombre de la rama del cuerpo de

la solicitud, ejecuta el comando ``git checkout -b`` para crear la rama y luego empuja la nueva rama al repositorio remoto.

14. ``GET /glassmatrix/api/v1/github/pullCurrentBranch/:repoName``: Esta ruta tira los últimos cambios de la rama actual de un repositorio específico. Ejecuta el comando ``git pull`` en el directorio del repositorio.
15. ``DELETE /glassmatrix/api/v1/github/deleteBranch/:repoName/:branchName``: Esta ruta elimina una rama específica de un repositorio específico. Ejecuta el comando ``git branch -d`` para eliminar la rama.
16. ``POST /glassmatrix/api/v1/github/changeBranch/:repoName/:branchName``: Esta ruta cambia a una rama diferente en un repositorio específico. Ejecuta el comando ``git checkout`` para cambiar a la rama especificada.
17. ``GET /glassmatrix/api/v1/github/files/:repoName``: Esta ruta lista todos los archivos en un repositorio específico. Lee el directorio del repositorio y devuelve una lista de nombres de archivos.
18. ``POST /glassmatrix/api/v1/github/commit/:repoName``: Esta ruta crea un nuevo commit en un repositorio específico. Toma el contenido del archivo y el mensaje del commit del cuerpo de la solicitud, escribe el contenido en un nuevo archivo en el directorio del repositorio y luego ejecuta el comando ``git add . && git commit -m`` para crear el commit.
19. ``POST /glassmatrix/api/v1/github/createFile/:repoName``: Esta ruta crea un nuevo archivo en un repositorio específico. Toma el nombre del archivo y el contenido del cuerpo de la solicitud y escribe el contenido en un nuevo archivo en el directorio del repositorio.
20. ``POST /glassmatrix/api/v1/github/push/:repoName``: Esta ruta empuja los cambios a un repositorio específico. Ejecuta el comando ``git push origin`` en el directorio del repositorio.

Las últimas tres rutas (``GET /api``, ``GET /docs``, y ``app.listen``) no están relacionadas con la API de GitHub. Redirigen a la documentación de la API y arrancan el servidor, respectivamente.

Otras APIs

También el proyecto interactúa con la API de github para poder realizar pushes/pulls de repositorios y comprobar así que las métricas y garantías estén funcionando correctamente. Además también interactúa con la API de bluejay para obtener los datos de computación de las métricas y obtener o crear nuevos TPAs.

Estructura proyecto angular

El proyecto utiliza la versión 13.3.11. Me decanté por usar Angular y en concreto por la versión 13 ya que esta versión Angular incluye características como el cambio de detección, la detección de zona y la renderización del lado del servidor que ayudan a mejorar el rendimiento de las aplicaciones, y es muy cómoda la interacción con distintas apis como pueden ser la api propia o la api de github.

El proyecto se encuentra dividido en 3 grupos, componentes, páginas y servicios. A continuación detallo el funcionamiento de cada uno de ellos.

Componentes
















Los componentes principales de la página son el header y footer que se muestran en todas las páginas del proyecto. Además cada subpágina de la página “home” es un componente propio.

Páginas

Página testeo TPAs

Esta página permite visualizar los TPAs que ya se encuentran cargados dentro de Bluejay.

Existings TPAs

ID	Project	Class	Options
tpa-TFG-GH-antoniosc7_tpPrueba	TFG-GH-antoniosc7_tpPrueba	TFG	  
tpa-class01-GH-cs169_fa23-chips-10.5-53	class01-GH-cs169_fa23-chips-10.5-53	class01	  
tpa-TFG-GH-antoniosc7_tpPrueba2	TFG-GH-antoniosc7_tpPrueba2	TFG	  
tpa-TFG-GH-JaviFdez7_ISPP-G1-Talent	TFG-GH-JaviFdez7_ISPP-G1-Talent	TFG	  
tpa-project01	project01	class01	  

Si seleccionamos la opción de editar, se podrá editar todo el TPA o se podrá usar un editor que permite editar métrica la métrica o añadir una nueva métrica al TPA facilitando así la edición del TPA. En esta tabla también podemos encontrar la opción de simplemente ver el TPA o borrarlo (Esto lo borraría de Bluejay directamente).

Create TPA

Copy Default TPA

Editar por métricas/garantías

En esta página se permite añadir a un TPA alguna métrica o garantía de forma individual sin necesidad de tener que tocar en el json completo del TPA. Además se incluye un botón que introduce en el cuadro de texto un ejemplo de métrica o garantía que puede servir como guía.

⚠ Es importante que el nombre de la métrica o garantía sea exactamente igual que el id de la métrica o garantía.

Métricas

Nombre de la nueva métrica

Contenido de la nueva métrica

Crear nueva métrica

Copiar métrica de ejemplo

Página métricas

Para testear métricas, en primer lugar habrá que acceder a la página de metrics tester. En primer lugar hay una sección en donde podemos realizar acciones con las métricas ya guardadas. En la parte inferior, podemos testear una nueva métrica.

Por defecto, habrá una métrica en la zona de crear nueva metrica. Esta métrica es simplemente un ejemplo y es totalmente editable o directamente se puede borrar y usar una nueva.

Crear nueva métrica

Introduce el nombre del a

Guardar como JSON

Buscar...

Buscar

```
{
  "config": {
    "scopeManager": "http://host.docker.internal:5700/api/v1/scopes/development"
  },
  "metric": {
    "computing": "actual",
    "element": "number",
    "event": {
      "githubGQL": {
        "custom": {
          "type": "graphql",
          "title": "Get pull requests with at least one comment by member",
          "steps": {
            "0": {
              "type": "queryGetObject",
              "query": "{\r\n  repository(name: \"%PROJECT.github.repository%\", owner: \"%PROJECT.github.repoOwner%\") {\r\n    pullRequests(first: 100) {\r\n      pageInfo {\r\n        endCursor\r\n        hasNextPage\r\n      }\r\n      nodes {\r\n        bodyText\r\n        number\r\n        state\r\n        createdAt\r\n        author {\r\n          login\r\n        }\r\n      }\r\n      mergedAt\r\n      mergedBy {\r\n        login\r\n      }\r\n      comments(first: 50) {\r\n        nodes {\r\n          author {\r\n            login\r\n          }\r\n          bodyText\r\n          createdAt\r\n          reviews(first: 30) {\r\n            nodes {\r\n              state\r\n              createdAt\r\n              bodyText\r\n              author {\r\n                login\r\n              }\r\n            }\r\n          }\r\n        }\r\n      }\r\n    }\r\n  }\r\n}"
            },
            "1": {
              "type": "objectGetSubObjects",
              "location": "data.repository.pullRequests.nodes"
            }
          }
        }
      }
    }
  }
}
```

Una vez que hemos introducido la métrica que queremos testear, hay que darle a publicar. Esto interactuará con el endpoint de Bluejay y tras obtener la computación inicial, se desbloqueará el botón de obtener cálculo. Esto tardará unos segundos y devolverá los valores de las métricas que estamos testando en ese repositorio.

Post

Get Computation

Una vez que hemos comprobado la métrica (o antes si lo deseas) también habrá un botón para guardar la métrica y tenerla siempre a mano en el repositorio local con el nombre que queramos. Cuando la tengamos guardada, ya se podrá acceder a ella desde la sección superior y podremos volver ejecutarla, editarla o borrarla.

Enter file name

Save as JSON

Subpantalla edición de métricas

Si en la primera sección accedemos a editar métricas entraremos en una página como la inferior. Esta página tiene la opción de utilizar la hora actual (utilizará como fecha de inicio el comienzo de la hora actual, si son las 18:27, la hora de inicio será las 18:00 y la hora de final de computación serán las 18:59 del día en el que se realice). Si se guarda la métrica desde esta sección, se guardará con el periodo de cálculo utilizado.

En esta misma página, en la parte inferior, se incluye una nueva sección de ejecución exactamente igual que en la página anterior pero para la métrica actual.

You are executing PullRequestCerradas.json.

Save as json

Context Section

Scope Information

Project

TFG-GH-JaviFdez7_ISPP-G1-Talent

Class

TFG

Member

*

Computation Search

Type

static

Period

hourly

Initial

2024-03-14T00:00:00.000Z

From

2024-03-14T00:00:00.000Z

End

2024-03-14T23:59:59.999Z

Timezone

America/Los_Angeles

Actual hour computation

Execution Section

Search

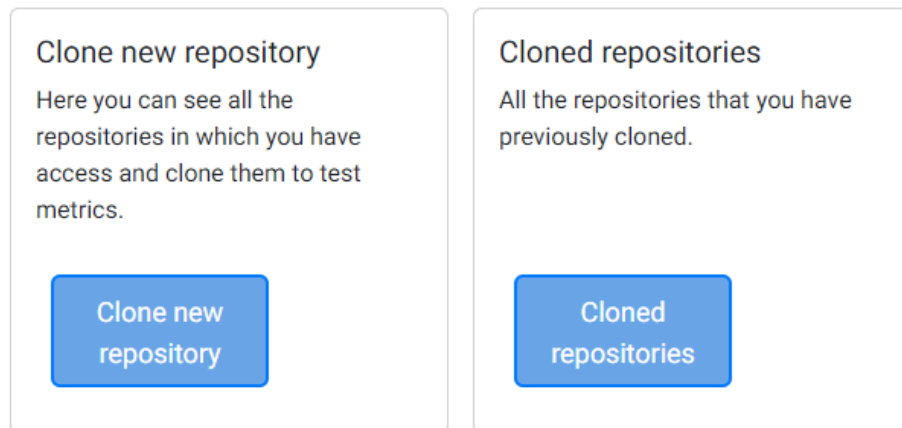
Search...

```
{
  "config": {
    "scopeManager": "http://host.docker.internal:5700/api/v1/scopes/development"
  },
  "metric": {
    "computing": "actual",
    "element": "number",
    "event": {
      "githubGraphQL": {
        "custom": {
          "type": "graphql",
          "title": "Get closed pull requests",
          "steps": {
            "name": 1
          }
        }
      }
    }
  }
}
```


Página testeo de repositorios

La primera de ella será la página para clonar un nuevo repositorio.















To clone a new repository you will need a valid token with access to that repository. For more information, visit: [Github token documentation](#). The token must be introduced in the clone new repository page.



En ella, al inicio hay una sección para añadir o editar el token de github y tras introducir un token válido aparecen todos los repositorios a los que tiene acceso con ese token. Además cada repositorio que aparezca listado tendrá dos opciones, verlo en github para asegurarnos de que es el repositorio que queremos clonar y otra pantalla opción es la de editar que permite clonar el repositorio en local.

Token found

[View Token](#)

Repository Name	Number of Branches	Last Update	View	Edit
Antoniosc7	1	2024-01-10T23:20:42Z		
Antoniosc7.github.io	1	2024-01-15T23:03:47Z		
ISSIProject	1	2022-11-25T12:57:33Z		
JerseyDetection	2	2024-01-10T23:53:57Z		
JumpMarius	5	2024-01-15T18:20:03Z		
SnakeGamePD	1	2023-09-18T11:35:27Z		
TFG-Angular	1	2023-08-29T02:14:12Z		

Por otro lado, si en la pantalla principal de Repository Tester accedemos a repositorios clonados, listará todos los repositorios que hayamos clonado ya, y para cada uno de ellos tendremos 3 opciones, una para crear / eliminar o cambiar de rama. Otra opción para realizar pull requests y otra opción que permitirá crear archivos y realizar commits y pushes.

Subpantalla de Ramas

En esta pantalla se podrán gestionar las ramas del repositorio clonado localmente, se podrán crear, cambiar y eliminar ramas. Esto será útil para realizar commits y pushes en la subpantalla de acciones o para hacer pull requests.

tp-testbench

Branches Repository

* 321443211234

3231

main

New branch name

[Create Branch](#)

* 321443211234

[Delete Branch](#)

main

[Change Branch](#)

Subpantalla de Pull Requests

En esta pantalla se podrá crear una pull request, ver las pull request que estén abiertas y realizar el merge de una pull request.

Para crear una pull request se le deberá dar un título, indicar la rama que se quiere actualizar, llamado base y head será el repositorio del cual se traerán los cambios. Para cerrar la pull request se deberá introducir el número de la pull request que se quiere cerrar. Este número se podrá ver en el listado de “Open pull request” y se deberá añadir un texto para cerrarla.

tp-testbench

Create Pull Request

Base is the repository that will be updated. Changes will be added to this repository via the pull request. Head: Head is the repository containing the changes that will be added to the base.

Title:

Head:

Base:

Body:

Create Pull Request

Open Pull Requests

merge test2

Number: 3

Subpantalla de Acciones

En esta subpantalla se podrán crear archivos y realizar commits y pushes a la rama que tengamos seleccionada. Esto permitirá comprobar si alguna métrica que tenga que ver con commits o pushes funciona correctamente.

tp-testbench | Actions

[View in GitHub](#)

Available Branches

Branch Name

Current Branch:

[Change Branch](#)

Create New File

[Create File](#)

Create New Commit

[Create Commit](#)

Servicios

Bluejay Service

El servicio Bluejay Service en Angular está diseñado para interactuar con un servidor de backend que proporciona una API relacionada con acuerdos (agreements) y cálculos computacionales. Veamos cada método del servicio y lo que hace:

1. **createTpa(tpaContent: string):** Este método toma el contenido de un Acuerdo de Procesamiento de Terceros (TPA) en forma de cadena JSON, lo convierte en un objeto JavaScript utilizando `JSON.parse`, y luego lo envía al servidor a través de una solicitud HTTP POST a la URL definida en `this.url`. El servidor espera recibir un nuevo TPA y devuelve una respuesta que se emite como un Observable.
2. **getTps():** Este método realiza una solicitud HTTP GET al servidor para obtener una lista de todos los Acuerdos de Procesamiento de Terceros (TPA) disponibles. La respuesta del servidor se emite como un Observable.

3. **getTpa(id: string):** Este método realiza una solicitud GET al servidor para obtener un TPA específico identificado por su ID. La URL para la solicitud se construye concatenando el ID proporcionado al final de la URL base definida en `this.url`. La respuesta del servidor se emite como un Observable.
4. **deleteTpa(id: string):** Este método realiza una solicitud DELETE al servidor para eliminar un TPA específico identificado por su ID. La URL para la solicitud se construye de manera similar al método `getTpa()`. La respuesta del servidor se emite como un Observable.
5. **postComputation(data: any):** Este método realiza una solicitud POST al servidor para enviar datos de computación. La URL para la solicitud se define explícitamente como `http://localhost:5500/api/v2/computations`. Los datos de computación se envían en el cuerpo de la solicitud y la respuesta del servidor se emite como un Observable.
6. **getComputation(computationUrl: string):** Este método realiza una solicitud GET al servidor para obtener los resultados de una computación específica. La URL para la solicitud se pasa como parámetro y la respuesta del servidor se emite como un Observable.

En resumen, el servicio `BluejayService` proporciona métodos para interactuar con una API de backend que gestiona acuerdos y operaciones computacionales, permitiendo la creación, obtención, eliminación y consulta de datos relacionados con estos acuerdos y cálculos. Las solicitudes HTTP se realizan utilizando el servicio `HttpClient` de Angular, y las respuestas del servidor se emiten como Observables para su consumo en la aplicación Angular.

Files Service

Este servicio, proporciona métodos para acceder a archivos JSON almacenados localmente en la aplicación. Estos archivos JSON pueden contener datos de configuración o ejemplos.

1. **getDefaultTPA():** Este método devuelve el contenido del archivo `defaultTPA.json` ubicado en la carpeta `assets` de la aplicación Angular. Se utiliza la opción `responseType: 'text'` para indicar que el contenido del archivo debe ser tratado como texto.
2. **getBasicMetric():** Este método devuelve el contenido del archivo `basicMetric.json` ubicado en la carpeta `assets` del proyecto.
3. **getSavedMetric(fileName: string):** Este método toma el nombre de un archivo como argumento y devuelve el contenido del archivo correspondiente ubicado en la carpeta `assets/savedMetrics` del proyecto.

4. **getExampleMetric():** Este método devuelve el contenido del archivo `exampleMetric.json` ubicado en la carpeta `assets/examples` del proyecto.
5. **getExampleGuarantee():** Este método devuelve el contenido del archivo `exampleGuarantee.json` ubicado en la carpeta `assets/examples` del proyecto.

En resumen, este servicio proporciona una manera conveniente de cargar archivos JSON locales en la aplicación Angular utilizando el servicio `HttpClient`.

Github Service

En este servicio, utilizo la API pública de GitHub para realizar una serie de operaciones relacionadas con los repositorios y las solicitudes de extracción. Aquí hay un resumen de las operaciones que realizo:

1. **listBranchesForRepo(owner: string, repo: string):** Esta función obtiene la lista de ramas de un repositorio específico. Utilizo la API de GitHub haciendo una solicitud GET a la URL correspondiente del repositorio.
2. **getUserName(token: string):** Esta función obtiene el nombre de usuario asociado a un token de autenticación. Utilizo la API de GitHub haciendo una solicitud GET a la URL del usuario autenticado.
3. **listRepos(token: string):** Esta función obtiene la lista de repositorios públicos del usuario autenticado. Utilizo la API de GitHub haciendo una solicitud GET a la URL de los repositorios del usuario autenticado.
4. **getLatestCommitSha(owner: string, repo: string):** Esta función obtiene el SHA del último commit en un repositorio específico. Utilizo la API de GitHub a través de la librería `Octokit` para obtener la lista de commits y luego extraer el SHA del último commit.
5. **getRepoInfo(owner: string, repo: string):** Esta función obtiene información detallada sobre un repositorio específico. Utilizo la API de GitHub haciendo una solicitud GET a la URL del repositorio.
6. **createBranch(owner: string, repo: string, branchName: string, ref: string):** Esta función crea una nueva rama en un repositorio específico. Utilizo la API de GitHub a través de la librería `Octokit` para crear la nueva rama.
7. **createFile(owner: string, repo: string, path: string, message: string, content: string, branch: string):** Esta función crea un nuevo archivo en un repositorio específico. Utilizo la API de GitHub a través de la librería `Octokit` para crear el archivo.
8. **createPullRequest(token: string, owner: string, repo: string, prTitle: string, prHead: string, prBase: string, prBody: string):** Esta función crea una nueva solicitud de extracción (pull request) en un repositorio específico. Utilizo la API de GitHub

haciendo una solicitud POST a la URL de las solicitudes de extracción del repositorio.

9. **getOpenPullRequests**(token: string, owner: string, repo: string): Esta función obtiene la lista de solicitudes de extracción abiertas en un repositorio específico. Utilizo la API de GitHub haciendo una solicitud GET a la URL de las solicitudes de extracción abiertas del repositorio.
10. **mergePullRequest**(token: string, owner: string, repo: string, mergePrNumber: number, mergeCommitMessage: string): Esta función fusiona una solicitud de extracción específica en un repositorio específico. Utilizo la API de GitHub haciendo una solicitud PUT a la URL de la solicitud de extracción a fusionar.
11. **getBranches**(token: string, owner: string, repo: string): Esta función obtiene la lista de ramas de un repositorio específico. Utilizo la API de GitHub haciendo una solicitud GET a la URL correspondiente del repositorio.

En resumen, estoy utilizando la API pública de GitHub realizar diversas operaciones relacionadas con repositorios y solicitudes de extracción, aprovechando las funcionalidades proporcionadas por la API para interactuar con los recursos de GitHub.

Glassmatrix Service

El servicio GlassmatrixService en Angular está diseñado para interactuar con un servidor de backend propio que he explicado anteriormente en el apartado de Glassmatrix API. En resumen realiza operaciones relacionadas con archivos, tokens de GitHub y acciones específicas de GitHub, como clonar repositorios, manipular ramas, crear y editar archivos, y realizar operaciones de commit y push.

Métodos para Operaciones de Archivos:

1. **saveToJson**(fileName: string, data: any): Envía una solicitud POST al servidor para guardar datos en un archivo JSON especificado.
2. **deleteFile**(fileName: string): Envía una solicitud DELETE al servidor para eliminar un archivo especificado.
3. **loadFiles**(): Envía una solicitud GET al servidor para obtener una lista de nombres de archivos disponibles.
4. **updateFile**(fileName: string, data: any): Envía una solicitud POST al servidor para actualizar un archivo JSON existente.

Métodos para Operaciones de Tokens de GitHub:

1. **getToken**(): Envía una solicitud GET al servidor para obtener un token de GitHub.
2. **saveToken**(token: string): Envía una solicitud POST al servidor para guardar un token de GitHub.
3. **deleteToken**(): Envía una solicitud DELETE al servidor para eliminar un token de GitHub.

Métodos para Acciones de GitHub:

1. **cloneRepo**(owner: string, repoName: string): Envía una solicitud POST al servidor para clonar un repositorio de GitHub.
2. **listRepos**(): Envía una solicitud GET al servidor para obtener una lista de repositorios de GitHub disponibles.
3. **deleteRepo**(repo: string): Envía una solicitud DELETE al servidor para eliminar un repositorio de GitHub.
4. Métodos relacionados con operaciones de ramas, commits y manipulación de archivos en un repositorio de GitHub.

En resumen, el servicio GlassmatrixService proporciona una interfaz para interactuar con diferentes aspectos de un servidor de backend relacionado con operaciones de archivos y acciones específicas de GitHub. Esto permite a la aplicación Angular realizar diversas operaciones, como guardar y eliminar archivos, manipular repositorios de GitHub y gestionar tokens de autenticación de GitHub. Las solicitudes HTTP se realizan utilizando el servicio HttpClient de Angular, y las respuestas del servidor se emiten como Observables para su consumo en la aplicación Angular.

Traducciones

Gracias a la librería ngx-translate <https://github.com/ngx-translate/core>. Por defecto, TP Tester, estará en el idioma del navegador. Esta librería permite además añadir traducciones de una forma sencilla ya que bastaría con traducir el JSON de un idioma a otro y no habría que modificar ningún otro tipo de fichero.

Este es un fragmento de cómo funciona el .json de la traducción a español:

```
"METRICS_LOADER": {
  "TITULO1": "Métricas guardadas",
  "TITULO2": "Crear nueva métrica",
  "MESSAGE_TEXT": "No hay métricas guardadas",
  "FILE_NAME": "Nombre del archivo",
  "VIEW_FILE": "Ver archivo",
  "EXECUTE_EDIT_FILE": "Ejecutar / Editar archivo",
  "DELETE_FILE": "Eliminar archivo",
  "VIEWER": {
    "VIEWING": "fileName."
  }
}
```


Ejemplos de uso

¿Cómo pruebo una métrica?