Server AREA-2022

I.Architecture du projet

Le serveur du projet est divisé en multiples dossiers présents dans ses sources (/src)

a. database

Ce dossier vous permettra d'accéder à la connexion à la base de données (index.ts / postgres.ts)
Afin de choisir une nouvelle base de donnée, il suffit de changer les valeurs données de base de donnée dans le fichier .env trouvé à la racine du projet.

ex: DB_PORT=4352 afin de lier le projet à une base de donnée sur le port 4352.

Vous retrouverez aussi une dossier model dans le quel est défini toutes les tables de la base de donnée, il suffit d'en rajouter / modifier et de les exporter dans fichier model/index.ts afin de pouvoir les instancier automatiquement au lancement du serveur.

b. middleware

Ce dossier vous permettra d'accéder aux middlewares utilisés par le projet dans les routes express, dans notre cas il n'existe qu'un middleware permettant de verifier le token JWT utilisé et de sécurisé différentes routes.

c. types

Ce dossier permet de ranger les différents types importants utilisés en typescript qui ne sont pas des classes utilisées dans le projet.

d. oauth

Ce dossier contient les différents services d'oauth utilisés dans le projet, ses oauths doivent hérités de la classe IOauthService trouvé dans le fichier Interface, et doit override deux fonction, setRefreshToken qui doit vérifier le token Bearer envoyé, et lier stocker un refresh/access_token dans la base de donnée de l'utilisateur en utilisant les queries envoyés par la route dans le parametre request.
la fonction loginCreateAccount elle, à le même comportement que la précédente, sauf qu'elle permet aussi la création et la connexion au compte via oauth.

Ces différentes classes prennent dans le constructeur, leur nom, leur lien de connexion oauth, et une booléen permettant de savoir si cet oauth permet la création de compte, ou seulement le lien à un compte déjà existant.

e. services

Ce dossier contient les différents services , actions, et réactions du projet.
Ces différents services héritent d'une classe IService et d'une fonction setup, qui permet de récupérer un access_token et de le stocker dans sa classe.
Son constructeur prend le nom du service en parametre.
Cette classe possède aussi un tableau d'action et réaction à remplir afin de le renvoyer automatiquement dans les requêtes.

Un fichier createService existe aussi, c'est une factory à service, créant le service, et lançant le setup utilisant les informations de l'utilisateur.

Il existe aussi deux autres dossiers, action et réaction.

Le dossier Réaction permet de créer différentes réactions qui héritent de la classe IRéaction

La classe réaction prend en paramètre dans son constructeur, le nom de la réaction, sa déscription, et le nom de son service.
Elle override une fonction trigger, qui quand lancé, active la fonction logique de la réaction en utilisant le client stocké dans la classe (ex: un access_token)

La classe action prend en paramètre dans son constructeur le nom de l'action, sa descrption, et le nom du service.
Elle override une fonction listen qui une fois lancé, va faire des requêtes tout les X temps afin de voir si la condition d'activation de l'action est vraie, dans ce cas, elle appelera la fonction trigger de sa réaction automatiquement stockée dans la classe Action.

Les dossiers Actions/réactions possèdent aussi deux factory permettant de générer ces actions et ces réactions.

f. config.ts

Ce fichier permet de stocker toutes les variables de l'environnement dans un object config utilisable dans tout le projet.

g. interface.ts

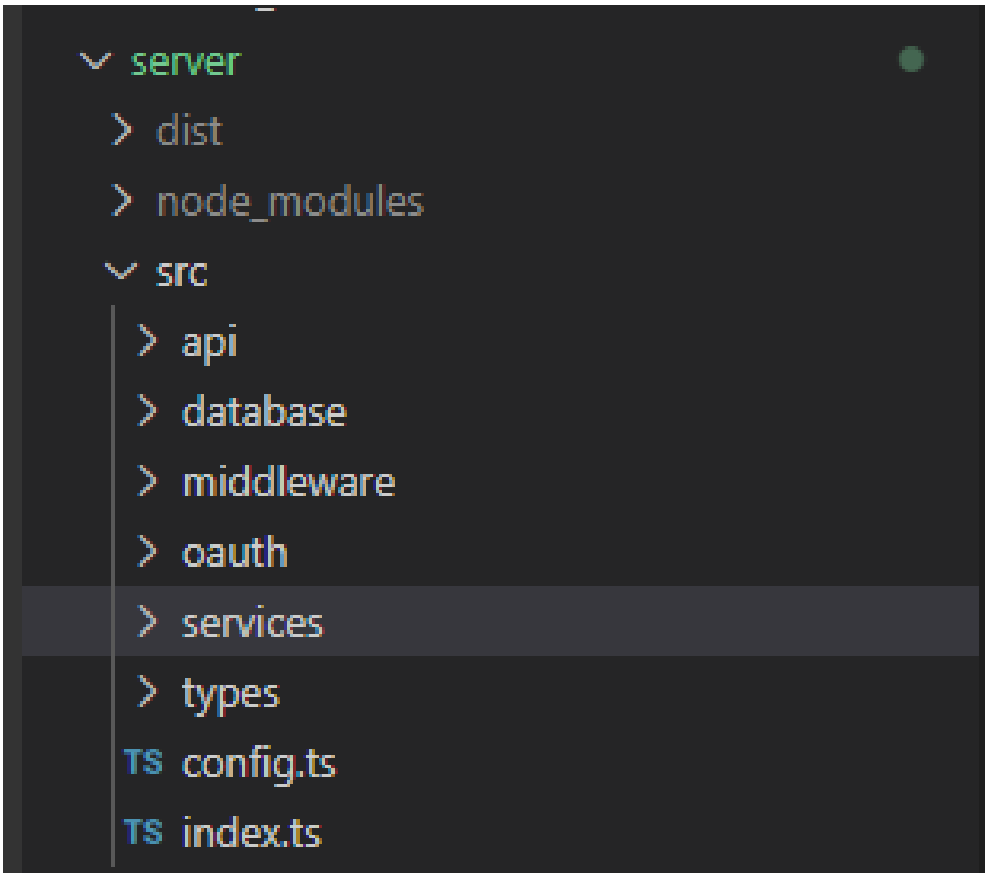
Ce fichier possède les différentes interfaces utilisées dans le projet IService, IAction, IRéaction.

h. index.ts

Ce fichier lance toute la logique du serveur, en appelant les fonctions de connexion à la base de donnée, et lançant les différentes routes.
Elle possède aussi de multiples fonctions comme stopService qui prend le nom d'un service et l'id d'un utilisateur en parametre et permet d'éteindre le service chez l'utilisateur donné.
removeComponent qui prend un id User et un Id de widget afin de le supprimer.
et setupServices qui va lancer tout les services des utilisateurs et allumer leur différents widgets.

i. api

Ce dossier contient les différentes routes utilisées dans le projet, triées par domaine d'utilisation dans différents fichiers



II.Comment rajouter une Action/Reaction ?

Afin de rajouter une action / réaction, il suffit d'hériter de créer une nouvelle classe héritant de soit IAction, IRéaction et d'override sa fonction avec la logique voulue comme présenté précédemment, une fois votre classe faite, construisez une instance de cette classe avec les bons parametres dans le bas de son fichier, et passez l'instance dans le export default afin qu'elle soit injectée dans la logique du serveur ! Et voila, votre action/réaction est générée mais pas encore utilisable !
Afin de la rendre utilisable par l'utilisateur, il suffit de la rajouter dans sa factory (create.ts d'action ou create.ts de réaction) une fois cette étape finie, votre action / réaction est totalement utilisable et fonctionelle !
Si jamais une requêtes de vos actions réaction échoue avec un code 403, n'oubliez pas d'appeler dans son cas d'erreur la fonction stopService afin de demander à l'utilisateur de se reconnecter au service et de générer un token valide !

```
export class chatReaction extends IReactions {
    async trigger() {
        try {
            if (this.state === false)
                return;
            this.params.channel = this.client.currentNick;
            if (this.params.channel && this.params.channel[0] == '#')
                await this.client.say(this.params.channel, this.params.response);
            else
                await this.client.say('#' + this.params.channel, this.params.response);
        } catch (error) {
            console.log(error)
            stopService(this.serviceName, this._userId);
        }
    }
}

const chatAnswer = new chatReaction("Chat", "Says a message on a specific channel", "twitch");

export default [
    chatAnswer,
]
```

III.Comment rajouter un oauth ?

Afin de rajouter un service d'oauth , il suffit de créer un fichier dans le dossier d'oauth, de créer une nouvelle classe overridant les deux fonctions dites précedemment, et de suivre la logique demandée (loginCreateAccount => Récupérer les queries du lien, récupérer un token et créer / se connecter sur l'area | setRefreshToken => Récupérer un access_token et le stocker dans la db de l'utilisateur)

une fois l'oauth généré, il suffit de l'export, et de l'export default depuis l'index.ts du dossier oauth afin qu'il soit directement injecté avec les autres services d'oauth.

```typescript
export default class IOAuthService {
    serviceName: string;
    oauthLink: string;
    canCreate: boolean;
    constructor(_serviceName: string, _oauthLink: string, _canCreate: boolean) {
        this.serviceName = _serviceName;
        this.oauthLink = _oauthLink;
        this.canCreate = _canCreate
    }

    async setRefreshToken(req: Request, res: Response, next: NextFunction, verifier: any, tokenSecret: any, token: any): Promise<any> {
        throw new Error("not implemented, please extends this interface and set this function according to your service oauth");
    }

    async loginCreateAccount(req: Request, res: Response, next: NextFunction, verifier: any, tokenSecret: any, token: any): Promise<any>{
        throw new Error("not implemented, please extends this interface and set this function according to your service oauth");
    }
}
```

III.Comment rajouter un service utilisant l'oauth ?

Afin de rajouter un service utilisant les accès précedemmen récupérés grâce à l'oauth, il faut créer un nouveau fichier dans le dossier service, créer une nouvelle classe héritant de IService, override la fonction setup qui dans sa logique dans stocker soit un client dans this.client, soit un access_token (dépend de l'utilisation depuis les action/réaction).

Une fois votre classe générée, n'oubliez pas de l'instancier, de rajouter ses différentes actions/réactions dans son instatnce grâce à sa fonction addAction et addRéaction et de l'export.
Une fois export, il suffit de la rajouter dans la factory de service, et dans le tableau d'export global dans le fichier index.ts !

Voila votre service, votre oauth, et vos actions réactions son générées !

IV.Les différentes routes

GET /about.json

Arguments: aucun
Nécessite Bearer token: non

Cette route renvoie les différentes informations du serveur comme ses services, les actions réactions disponibles ...

GET /oauths

Arguments: aucun
Nécessite Bearer token: non

Cette route renvoie sous forme de tableau les diffférentes noms d'oauths disponibles.

ex { oauths: ['twitch', 'reddit'] }

GET /oauths/create

Arguments: aucun
Nécessite Bearer token: non

Cette route renvoie sous forme de tableau les diffférentes noms d'oauths disponibles permettant la création de compte et la connexion sur notre plateforme.

ex { oauths: ['twitch', 'reddit'] }

POST /login

Arguments: username password => body JSON
Nécessite Bearer token: non

Cette route permet la connexion à la plateforme sur les informations du body sont correctes, nom d'utilisateur / mot de passe, et renvoie un token de connexion.

ex : { token: "caca' }

POST /register

Arguments: username password => body JSON
Nécessite Bearer token: non

Cette route permet la création de compte utilisant les informations récupérées du body (nom d'utilisateur et mot de passe).

Son code de retour permet de savoir si l'utilisateur à été crée
ex :

200 => Crée
400 => Bad request

GET /auth/link/:id

Arguments: id (nom d'oauth)
Nécessite Bearer token: non

Cette route permet de récupérér le lien permettant la connexion à un oauth selon l'id donné.

GET /auth/callback/:id

Arguments: id (nom d'oauth)
Nécessite Bearer token: non

Cette route permet la connexion ou la création de compte grâce aux données récupérées dans le l'url de redirect_uri, elle va appeler les fonctions nécessaire selon l'oauth choisi, elle peut recevoir un nombre différent de queries dépendant du service demandé.

GET /api/service

Arguments: aucun
Nécessite Bearer token: non

Cette route renvoie tout les services disponibles et toutes les actions réactions de ses services sous forme de tableau
ex:

{ services: [{serviceName: 'twitch', actions: []] }

GET /me

Arguments: aucun
Nécessite Bearer token: oui

Cette route renvoie les informations de l'utilisateur, dont son nom, et les services auxquels il est connecté.

ex : {
oauths: ['twitch'],
username: 'caca'
}

PUT /password

Arguments: password dans le body
Nécessite Bearer token: oui

Cette route permet de changer le mot de passe de l'utilisateur par le nouveau mot de passe donné dans le body.

PUT /disconnect/:id

Arguments: le nom du service dans l'url
Nécessite Bearer token: oui

Cette route permet de déconnecter l'utilisateur du service donné dans l'url.

GET /api/component

Arguments: aucun
Nécessite Bearer token: oui

Cette route permet de renvoyer tout les widgets crées par un utilisateur avec leur état actuel.

ex : { data : [
{ id: 1
serviceReactionName: 'twitch'
actionName: 'caca',
serviceActionName: 'caca',
reactionName: 'caca',
state: false
}
] }

POST /api/component

Arguments:
- serviceReactionName: string
- serviceActionName: string
- actionName: string
- reactionName: string

Nécessite Bearer token: oui

Cette route permet de générer un nouveau widget pour un utilisateur en utilisant les arguments du body.

DELETE /api/component/:id

Arguments: l'id du widget dans l'url

Nécessite Bearer token: oui

Cette route permet d'arrêter et supprimer un widget en particulier si il est possédé par l'utilisateur du token.

V. Tout ça c'est bien ? mais comment je lance le projet !

Afin de lancer le projet, verifiez de bien avoir docker installé sur votre machine.
Une fois docker installé, il suffit de créer deux fichier .env, un à la racine du projet, et un à la racine du dossier server et de remplir les bonnes variables (voir Readme.md à la racine).

Une fois ces fichiers crées et remplis correctement, il ne vous reste plus qu'à lancer la commande sudo docker-compose build et ensuite sudo docker-compose up à la racine du projet !

Une fois tout les conteneurs générés, il ne vous reste plus qu'à profiter de ce super projet de connecter tout votre petit monde :) !!!!!!!

VI. Les outils utilisés

- VueJS
- Flutter
- NodeJS / Typescript
- Docker
- PostgreSQL
- JSONWebToken
- Express

VII. Les services utilisés

- Twitch
- Reddit
- Twitter
- Github
- Trello

VIII. Page 20 et toujours pas parlé de comment télécharger l'apk ?????????

Si tu es arrivé jusqu'ici vaillant chevalier, il est l'heure pour toi d'apprendre à télécharger la SUPER giga bonne APK du projet dévéloppé par notre maitre à tous J-B roesch.

Afin de la télécharger, une fois tout les conteneurs docker lancés grâce à sudo docker-compose up.

Il suffit d'attendre que l'apk soit instancié par le conteneur, et vous pouvez le télécharger depuis le navigateur en cherchant le lien :
http://localhost:8081/client.apk
Evidemment, le début du lient peut changer selon son domaine d'hébergement mais on est trop pauvre pour héberger de toute manière LOL !

XXX. Un jour j'ai commencé LOL

insert piano music here
Un jour j'ai commencé lol
un pote m'a dit
joues-y c'est trop génial t'y passera ta vie
Un jour j'ai commencé lol

insert more music here

C'était un jeu trop cool
Meme si encore peu connu
c'était nouveau y'a peu d'champ
Et une super commu !

Je jouais qu'avec mes potes
C'était juste pour le délire
Jamais j'aurai deviné ce que ça allait devenir

Un jour j'ai joué en ranked
Un pote m'a dit
allez viens jouer tqt je vais te carry
Un jour j'ai joué en ranked

Les drafts c'était bizzare mais je m'y suis vite fait
Pick mes mains et ban tout ces putains de champion pété
Je jouais juste pour le fun
je faisais que de troller

Malgré ça j'ai carry mes games et je suis monté

Un jour je me suis fait ban
j'étais vraiment triste
mais enfaite c'était une erreur de riot
Un jour je me suis fait bien

Je n'avais plus qu'un but
c'était de devenir le meilleur
Pour ca j'ai décidé de devenir le plus grand streamer
j'avais plein de nouveaux potes
avec qui je pouvais jouer
sans leur soutien je n'y serai surement pas arrivé

Aujourd'hui je suis diamant
J'ai tout plein de fan
et peu enfin vivre de cette passion
aujourd'hui je suis diamant

Bientot je passerai master
J'aurai reussi et ça sera grace a tout votre soutien

Bientot je passerai master..
insert outro here