À partir du projet du répertoire https://github.com/Nicolas-Chourot/API-Server-beta-1.925, ajouter la gestion de mise en cache des requêtes de verbe GET.

Étape 1 :

Ajouter un module nommé CachedRequestsManager.js qui exportera la classe CachedRequestsManager. Cette classe permettra d'ajouter dans une cache l'url de la requête, le payload de sa réponse et son ETag. Lorsque qu'une requête comportera la même url, la classe retournera dans la réponse son payload et ETag enregistrés dans la cache.

Voici les noms des méthodes à implémenter :

```
static startCachedRequestsCleaner() {/* démarre le processus de nettoyage des caches périmées */…}

static add(url, content, ETag= "") {/* mise en cache */…}

static find(url) {/* retourne la cache associée à l'url */…}

static clear(url) {/* efface la cache associée à l'url */…}

static flushExpired() {/* efface les caches expirées */…}

static get(HttpContext) {/*

        Chercher la cache correspondant à l'url de la requête. Si trouvé,

        Envoyer la réponse avec

                HttpContext.response.JSON( content, ETag, true /* from cache */)

        */…}

(Inspirez-vous de la classe RepositoryCachesManager)

Note : afficher dans la console les événements suivants :

         Ajout dans la cache avec l'url associé

         Extraction de la cache avec l'url associé

         Retrait de cache expirée avec l'url associé
```

Étape 2 :

Modifier la méthode HttpContext.response.JSON(jsonObj, ETag="", fromCache = false). Si le paramètre fromCache est faux ajouter dans la cache si la requête est de type API et que le id est non défini.

Étape 3 :

Ajouter la référence à la méthode (jouant le rôle d'un middleware) CachedRequestsManager.get dans le middlewaresPipeline. Attention, il faut choisir le bon ordre des middlewares.

Étape 4 : Me montrer le bon fonctionnement avec Postman.

Étape 5 : Remettre fichiers Html comportant les liens Glitch et GitHub.