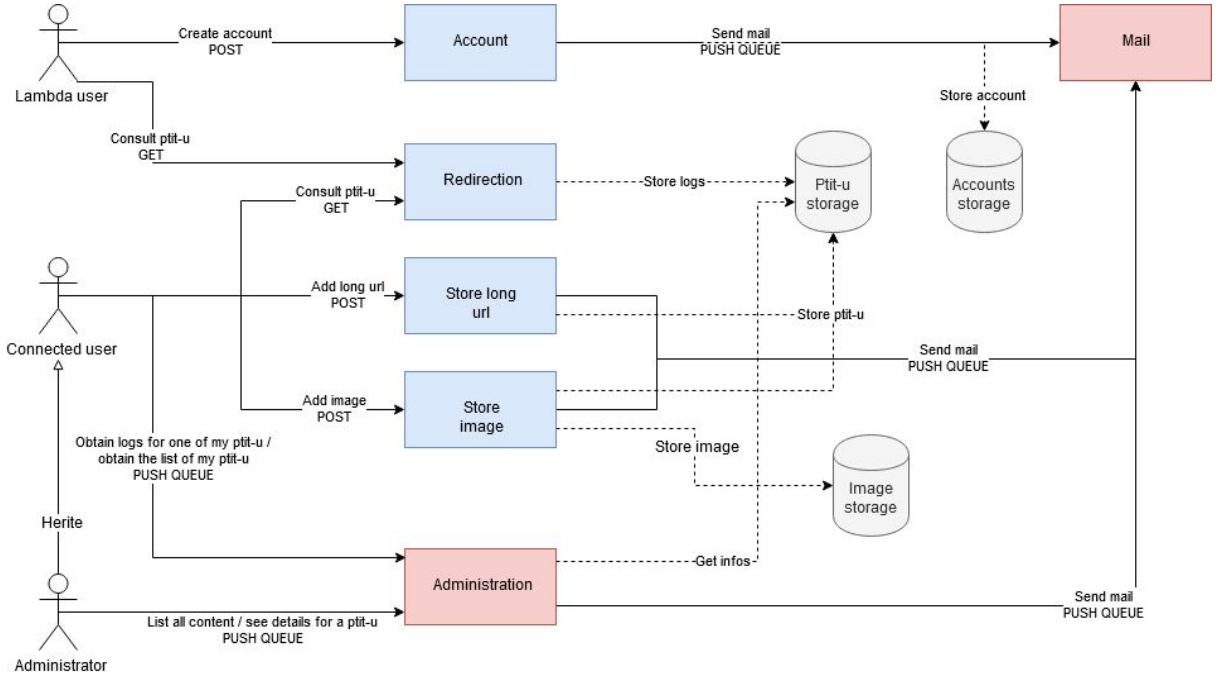
Premier rendu projet Tiny-Poly

Composition de l'équipe

Benazet Laurent
Frère Baptiste
Marilier Cyril
Tetevi Togni Fiacre

Diagramme d'architecture



Description des composants

- **Account :** gère la création des comptes utilisateurs. Une fois créé, un mail est envoyé à l'utilisateur.

- **Redirection :** redirige automatiquement l'utilisateur vers l'url correspondant a la ptit-u. Lors d'un accès à une ptit-u, on souhaite stocker les logs correspondants.

- **Store long url :** raccourcit une url en une ptit-u, puis la renvoie.

- **Store image :** stock une image et renvoie la ptit-u correspondante.

- **Administration :** fournit une interface permettant d'accéder à l'ensemble des ptit-u correspondant à un utilisateur, ainsi qu'aux détails des accès d'une ptit-u choisie.

- **Mail :** envoie des mails en fonction des différentes actions faites par l'utilisateur.

Explications de l'architecture et des communications

Nous avons considéré que les administrateurs possèdent les mêmes droits que les utilisateurs connectés, en plus de leurs droits de lister toutes les ptit-u et d'accéder aux détails de chacune d'entre elles. Les utilisateurs connectés et les administrateurs ne sont pas censés pouvoir créer un compte.

Nous indiquons sur le schéma trois systèmes de stockage. Ils sont présents uniquement à titre indicatif pour le moment car nous ne savons pas ce que nous allons utiliser pour stocker, nous savons uniquement quelles données devront être stockées et accédées.

Nous avons décidé d'utiliser le protocole REST pour l'ajout d'image et d'url longue puisque l'utilisateur veut recevoir la ptit-u générée directement : la communication est synchrone. De même, lors de la redirection, l'utilisateur veut arriver sur la page visée directement.

Ensuite, pour la création de compte, bien que le résultat soit retourné par mail, nous voulons que l'utilisateur ait accès aux fonctionnalités de l'application dès la création de son compte, avant même de recevoir le mail de confirmation de création de compte. Ainsi, nous utilisons le protocole REST pour la création de compte, afin d'être certains que le système de création de compte est disponible directement et que les droits de l'utilisateur seront actualisés dès que sa demande a été envoyée.

Pour le reste des communications, nous avons décidé d'utiliser des push queue (les composants concernés sont affichés en rouge sur le diagramme), pour plusieurs raisons :

- pour les mails : nous souhaitons envoyer les mails dès que possible, car bien que la communication soit asynchrone, les utilisateurs attendent une réponse rapide. Ainsi, l'utilisation d'une push queue permet d'envoyer les mails au fur et à mesure que les requêtes arrivent sans perturber le reste du fonctionnement du système, et les requêtes seront en attente si le service de mail n'est pas disponible. Pour ce cas d'utilisation, les pull queue n'étaient pas adaptées car les mails seraient alors envoyés en batch.

- pour l'administration : la requête est asynchrone puisque les résultats sont envoyés par mail une fois les données agrégées. De plus, nous souhaitons encore une fois que les résultats soient envoyés dès que possible. C'est pour cela que nous avons décidé d'utiliser des push queue pour l'administration : le protocole REST n'est pas adapté pour une requête asynchrone, et les pull queue ne nous permettent pas d'envoyer les données dès que possible.

Pour le moment, nous avons décidé d'utiliser la même push queue pour tous les envois de mails. Cependant, il serait peut être plus intéressant de créer une push queue

pour l'envoi de mails suite à la création d'un compte, une suite à la création d'une ptit-u et une pour les réponses aux requêtes pour l'administration. Cela permettra de séparer plus clairement les responsabilités des push queue, et la montée en charge sera bien plus simple à gérer, car toutes les requêtes déclenchant un envoi de mail ne sont pas utilisées à la même fréquence. Ainsi, nous ne mettrions pas en place la même élasticité au niveau de chacune des push queue.

Pour les mêmes raisons, il pourrait être utile dans le futur de séparer la push queue à destination de l'administration en une push queue pour les utilisateurs connectés et une pour les administrateurs.

Elasticité

Nous n'avons pas utilisé d'élasticité pour le moment car dans la version actuelle de l'application cela ne nous est pas nécessaire. Cela ajouterait des instances et donc augmenterait le coût. De plus, comme il n'y a qu'un seul projet pour le moment, une grande partie de chaque nouvelle instance ne serait pas utile. Ainsi, il nous faudrait enlever les instances générées peu utilisées, ce qui se fait de façon manuelle et n'est donc pas pratique.

Cependant, nous prévoyons d'en mettre en place lorsque nous mettrons en place de vrais systèmes de stockage pour le second rendu.

Ainsi, nous pensons mettre de l'élasticité dans les systèmes de stockage car c'est une des actions qui va prendre le plus de temps et qui risque d'engorger le système s'il est surchargé.

Nous pensons également à mettre de l'élasticité dans le système de redirection et de création de ptit-u, il s'agit du coeur de métier et se doit donc d'être réactif quelle que soit la charge.

Pour résumer, nous souhaitons mettre en place de l'élasticité aux endroits critiques du système (les plus fréquemment utilisés et ceux sur lesquels la charge sera la plus grande, par exemple l'ajout d'image).

Description des interfaces

- **Create account :** Requête Post avec 3 paramètres. Name pour le nom de l'utilisateur, mail qui servira comme authentification dans le reste de l'application, et admin qui indique si l'utilisateur a le rôle d'admin ou non.

- **Store long url :** Requête Post avec 2 paramètres. Mail pour l'authentification et longurl pour l'url à raccourcir.

- **Store image :** Requête Post avec 2 paramètres. Mail pour l'authentification et image pour l'image à uploader.

- **Consult ptit-u :** Requête Get avec 1 seul paramètre, url pour la ptit-u qui va être redirigée.

- **List all content :** Requête Post avec 1 seul paramètre, mail pour savoir quelles url récupérer, qui envoie les informations dans une queue.

- **See detail :** Requête Post avec 2 paramètres. Mail pour l'authentification et purl pour la ptit-u dont on veut récupérer les informations. Elle envoie les informations dans une queue.

- **Send mail :** Requête Post avec 4 paramètres qui envoie les informations dans la push queue pour l'envoi de mail. senderMail et recipientMail pour l'expéditeur et destinataire du mail, subject pour le sujet et message pour le contenu du mail.

Estimation des coûts sans stockage

1er scénario, optimiste en coût :

Nous considérons que nous avons 1 000 utilisateurs par mois. Chacun d'entre eux ajoute une url longue (au maximum 50Ko) par jour.
Avec le simulateur de google, nous obtenons un prix de 2,60€ par mois environ. Cela représente moins d'un centime par mois par utilisateur.

2ème scénario, pessimiste en coût :

Nous considérons que nous avons 10 000 utilisateurs par mois. Chacun d'entre eux ajoute cinq fois par jour une image de 4Mo.
Avec le simulateur de google, nous obtenons un prix de 10 500€ par mois environ. Cela représente environ 1€ par mois par utilisateur.

Ces deux scénarios montrent que le prix de notre application dépendra énormément de l'usage qu'en feront nos utilisateurs. Sur une utilisation normale avec 10 000 utilisateurs mensuels on peut estimer que le prix à payer sera d'environ 1 000 euros par mois, puisque nous estimons que les utilisateurs normaux n'ajoutent en moyenne même pas une image par jour. Le reste des opérations est négligeable comme nous l'avons vu dans le scénario 1. Cela représenterait donc environ 10 centimes par mois par utilisateur.