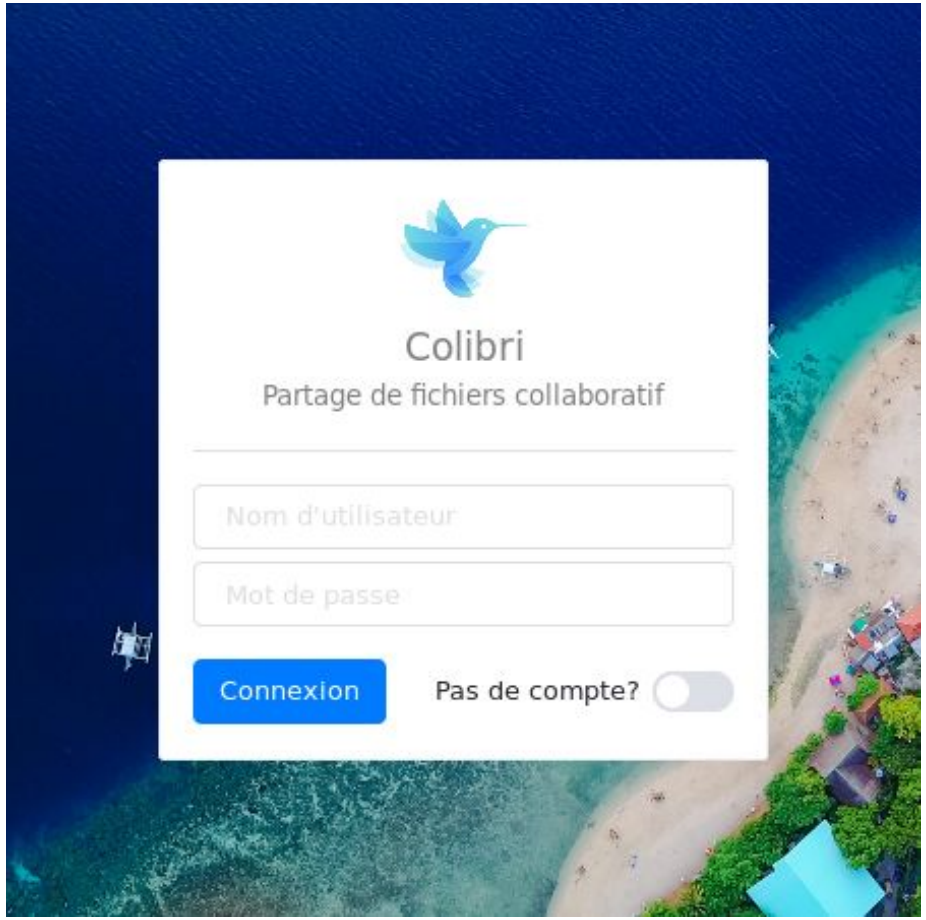
Colibri

Réseau social orienté autour du partage de fichiers

HEIG-VD 2018 - L. Delafontaine, V. Guidoux, G. Hochet & K. Pradervand

Objectifs du système

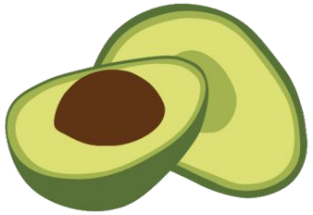
- Upload et partage de fichiers
- Visualisation des fichiers en ligne
- Classification par tags
- Fil d’actualité basé sur les tags
- Commentaires, likes
- Sauvegarde des fichiers qui nous intéressent
- Notifications en temps réel

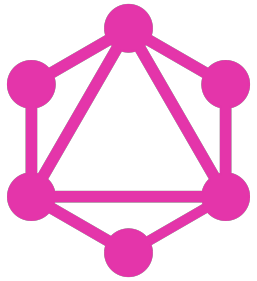



Démonstration

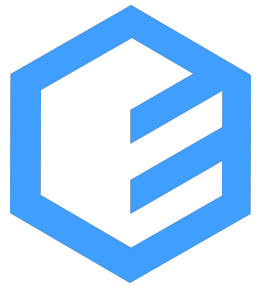
Technologies utilisées

Backend



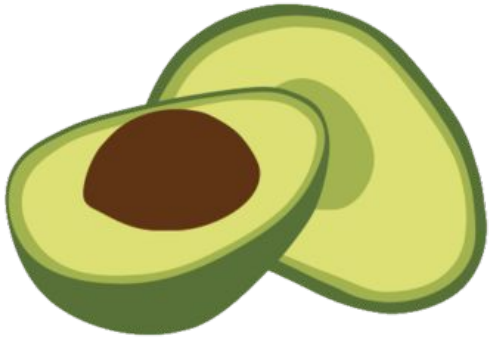


Frontend



ArangoDB

- Document + Graph + keystore
- Utilisation quasi exclusive de la documentation officielle (de très bonne qualité)
- Architecture cluster Master-Master (consistency > availability)
- Interface d’administration
- Microservices (plugins)



Arango Query Language AQL

Très simple à prendre en main

Index primaires, secondaires (_key, _id)

Requêtes peuvent (doivent) être optimisées

Beaucoup de fonctions prédéfinies (sur les array, strings…)

Subqueries, plusieurs déclarations dans la même requête

```
LET fichiers = (FOR f IN files
    FILTER LENGTH(INTERSECTION(f.tags, ${tags})) > 0
    SORT f.filename DESC
    LIMIT 10, 0
    RETURN f)
return {files: fichiers, amount: COUNT(FOR f IN files RETURN f)}
```

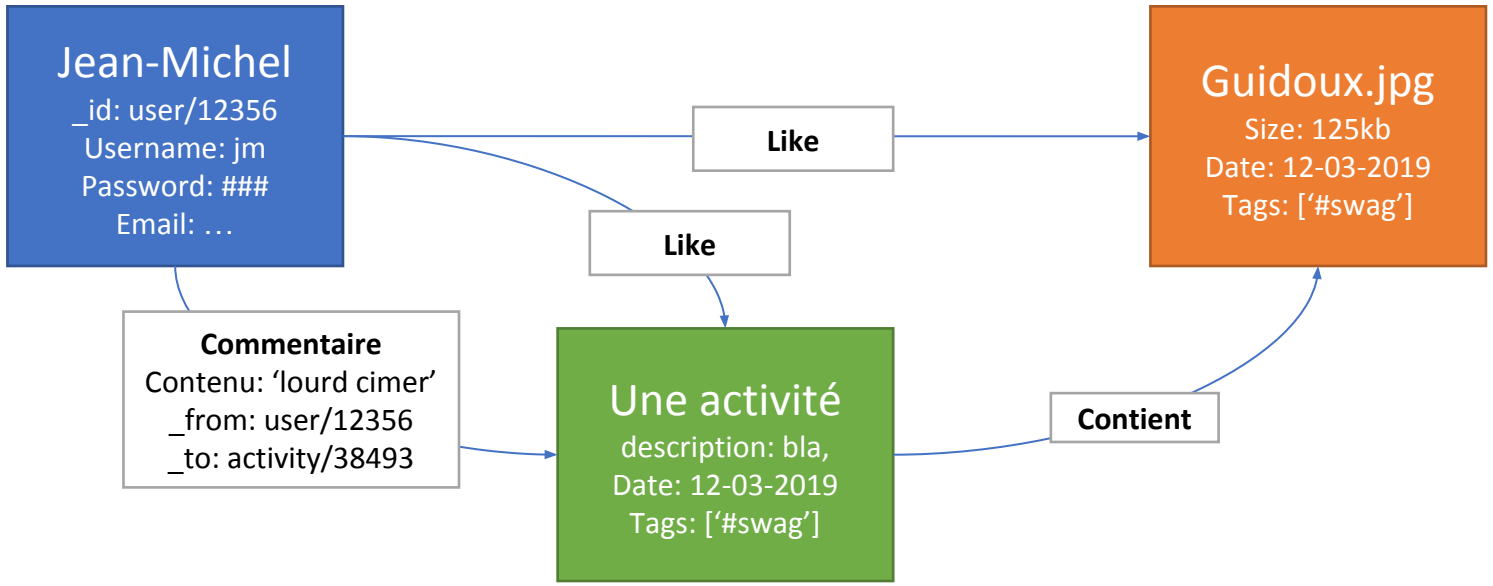
```
LET a = (FOR f IN files FILTER f.userKey == ${user._key} RETURN f)
LET b = (FOR ac IN activities FILTER ac.userKey == ${user._key} RETURN ac)
FOR c IN APPEND(a, b)
    FOR edge IN edges FILTER edge._to == c._id && edge._qualifier == 'like'
        RETURN edge
```

Structure en graphe

Graphe orienté, tout est document (_from, _to)

Arcs tous dans la même collection (API graphs)

Suppression facilitée



Typescript & NodeJS

- Javascript pas typé, aucune garantie
- Typescript = interfaces, classes, types, généricité…
- Driver AQL bas niveau, aucune vérification
- Structure de type entité + Manager
- Vérification avec Joi.JS

Exemple de code

```
interface IActivity extends IBase {
     content?: string;
     userKey: string;
     tags: Array<string>;
}

class Activity extends Base implements IActivity {
     content?: string;
     userKey: string;
     tags: Array<string>;

     constructor() {
          super();
          this.tags = [];
     }

     buildSchema() : object {
          return {
               content: Joi.string(),
               userKey: Joi.string().required(),
               tags: Joi.array().required(),
          };
     }
}
```

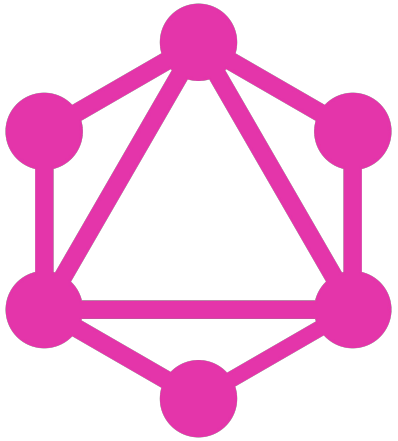
```
find<T extends IBase>(key: string) : Promise<T> {
     return this
          .collection
          .document(key)
          .catch(() => null);
}

update<T extends IBase>(key: string, item: T): Promise<T> {
     Joi.assert(item, item._getSchema());

     return this
          .collection
          .update(key, item)
          .then(() => item);
}
```

GraphQL

- Apollo server avec websockets
- GraphQL très adapté pour des graphes
- Apollo supporte l’upload de fichiers
- Directives personnalisées
  - `@aql(query: "FOR u IN users FILTER u._key == @current.userKey RETURN u", single: true)`
  - `name: String! @capitalize`
- Express pour le streaming



Vue.js

- Router et vuex (cache)
- ElementUI: bibliothèque de composants
- Bootstrap: quelques éléments (grille, layout…)
- Sass (css++) pour le style

Apollo et vue-apollo

- Vue-apollo même niveau que react
- `fetchMore` pour le mur
- Force des fragments, et une union
- Subscriptions pour les notifications

Cache

- Deux caches différents
- Cache vuex pour les données enregistrées en localstorage
- Cache Apollo peine à MàJ les listes
- InMemoryCache query par ID

Vuex

Données persistées en localstorage
(userId, jwt, détails d'UI…)

Apollo cache

Toutes les données des requêtes

Bilan

- Déploiement
- Nouvelles technologies
- Peu de temps
- Communication


Merci pour votre attention !