Documentation technique Arcadia

Réflexion initiale technologique

Ma réflexion initiale pour ce projet a d'abord été de prendre en compte la notion de temps. En effet une deadline étant imposée, il m'a fallu prendre en compte cette forte composante.
Deuxième point, étant dans la formation avec la spécialité Angular, je tenais à faire ce projet sous Angular pour la partie front-end.

Prenant acte des ces deux paramètres, j'ai choisi pour les autres parties du projet des librairies ou langages qui m'étaient plus ou moins familiers. Mon projet est donc sous Node.js.
Ainsi j'ai fait le choix d'utiliser Express.js pour mon API. Cela m'a également permis de fortifier mes connaissances en Javascript.
Pour ma base de données relationnelles, j'ai opté pour MySql et pour ma base de données non-relationnelles MongoDb.

Les versions sont les suivantes:

```
node.js v20.22.1
npm v10.2.4
angular/cli v17.3.0
"express": "^4.19.2",
"mongodb": "^6.6.2",
"mysql": "^2.18.1",
```

Il ne faut pas oublier bien entendu le HTML5 pour créer du contenu et le CSS3 pour lui appliquer du style.

Dans un nouveau dossier nommé arcadia j'ai fait deux sous-dossiers, un backend et l'autre frontend. J'ai construit mon application dans ces deux sous-dossiers.

J'ai ensuite construit mon architecture applicative.
J'ai donc construit mon application comme-ci dessous:
Le front-end va générer une ou des requêtes http. La requête va être envoyée dans un ServiceComponent. C'est ce service qui permet la communication avec l'API de la partie backend.
La requête http avant d'être envoyée au back-end va passer par des interceptors.
Le premier est un interceptor "error" afin de determiner si la requête ne contient pas d'éventuelles erreurs et le deuxième un interceptor d'authentification qui va rajouter dans le header de la requête un Bearer token (le token est récupéré s'il existe dans le localstorage, sinon la requête passe sans nouveau header)
Si ces étapes sont validées, le back-end reçoit la requête via le fichier index.js.
En fonction de l'URL présente dans la requête, cette dernière suit la route :
index.js → route → controller → query → base de données (Mysql ouMongodb)

Au bout du processus une réponse est fournie par la base de données et elle est récupérée par le controller.
Ce controller va faire appel à un modèle de réponse (présent dans le dossier domain/response.js) pour formater celle-ci.
La réponse est renvoyée à l'index.js, qui à son tour la transmet à la partie front-end.

Le service Angular va réceptionner la réponse avant de la refaire passer par l'interceptor error.

Si à nouveau tout est validé, la requête passe par une nouvelle étape, les guards.
Ils sont aux nombres de deux. Le premier est un guard d'authentification, il va vérifier la présence d'un token dans le localStorage. Et en fonction du component ciblé et de ses autorisations, la requête va aboutir sur une réponse différentes. Le deuxième guard va vérifier la présence d'un rôle dans le localStorage et idem que précedemment va aboutir sur une réponse adaptée.

Configuration environnement de travail

Je travaille essentiellement avec l'OS Linux Ubuntu depuis quelques années.

Pour mon IDE, j'ai fait le choix de Codium. J'ai commencé avec celui-ci il y a quelques temps et je me suis accoutumé àlui, je n'ai donc pas changé mes habitudes pour ce projet et travaillé avec des outils dont les fonctionnalités m'étaient familières.
J'ai utilisé plusieurs extensions de Codium tel que Angular Language Service, Prettier et REST Client.

Pour mettre en place un environnement de développement, j'ai tout d'abord modifié le package.json dans la partie back-end afin de définir un script de démarrage différent.
Pour lancer mon serveur, je dois écrire la commande npm run dev dans mon terminal. J'utilise ainsi nodemon qui me permet de voir en temps réél les changements.
Ainsi je peux avoir accès à mon serveur en local sur l'adresse: http://localhost:8000

Ensuite j'ai fait le choix de dockeriser mon serveur ainsi que les bases données (aussi bien Mysql que Mongodb). J'ai donc mis en place un docker.file avec les propriétés de node qui va me servir à construire l'image node.
J'ai ensuite fait un fichier docker-compose.yml pour créer un network. Ce network regroupe les services node, mysql et mongodb. Au lancement de celui-ci, les 3 imags sont chargées ainsi que les 3 containers.
Tout ceci m'a permis de travailler avec les bases de données avant qu'elles ne soient déployées.

Concernant la partie front-end, Angular dispose de la commange ng serve qui permet de tester en local son application sur l'adresse: http://localhost:4200/
Il m'a simplement fallu créer un fichier environments afin de séléctionner l'URL locale du serveur.

Pour mettre en place l'environnement de production, j'ai également fait un nouveau script dans le fichier package.json. La commande pour lancer le serveur en mode production, je dois écrire npm run start. Tout changement effectué sur le code n'impacte pas le serveur.
Côté front-end, la commande native ng build pemet de construire le projet afin qu'il se retrouve dans un environnement de production.
Avec mon dossier environment, ng build va utiliser l'URL deployée du serveur.

Afin de tester mon serveur avant de le relier à mon application Angular, je testais les requêtes grâce à l'extension Codium REST Client. J'ai crée un fichier test/test.api.http dans lequel j'écris les requêtes et les envoi pour test. Un nouvel onglet s'ouvre avec la réponse.

J'ai également utilisé Workbench pour faire des tests sur ma base de données relationnelle et Atlas pour ma base de données non-relationelle.

Modèle conceptuel de données

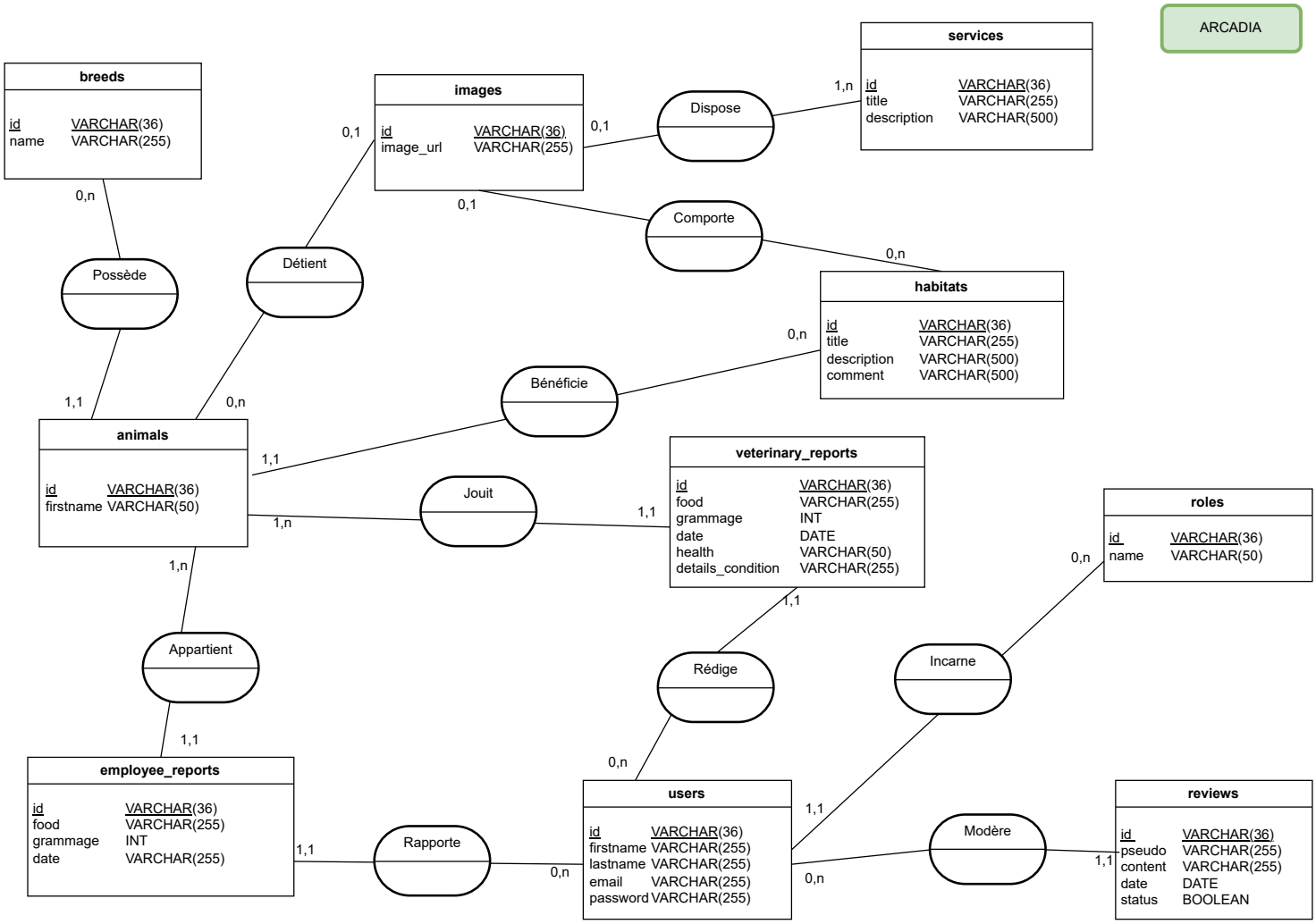


Diagramme de cas d'utilisation

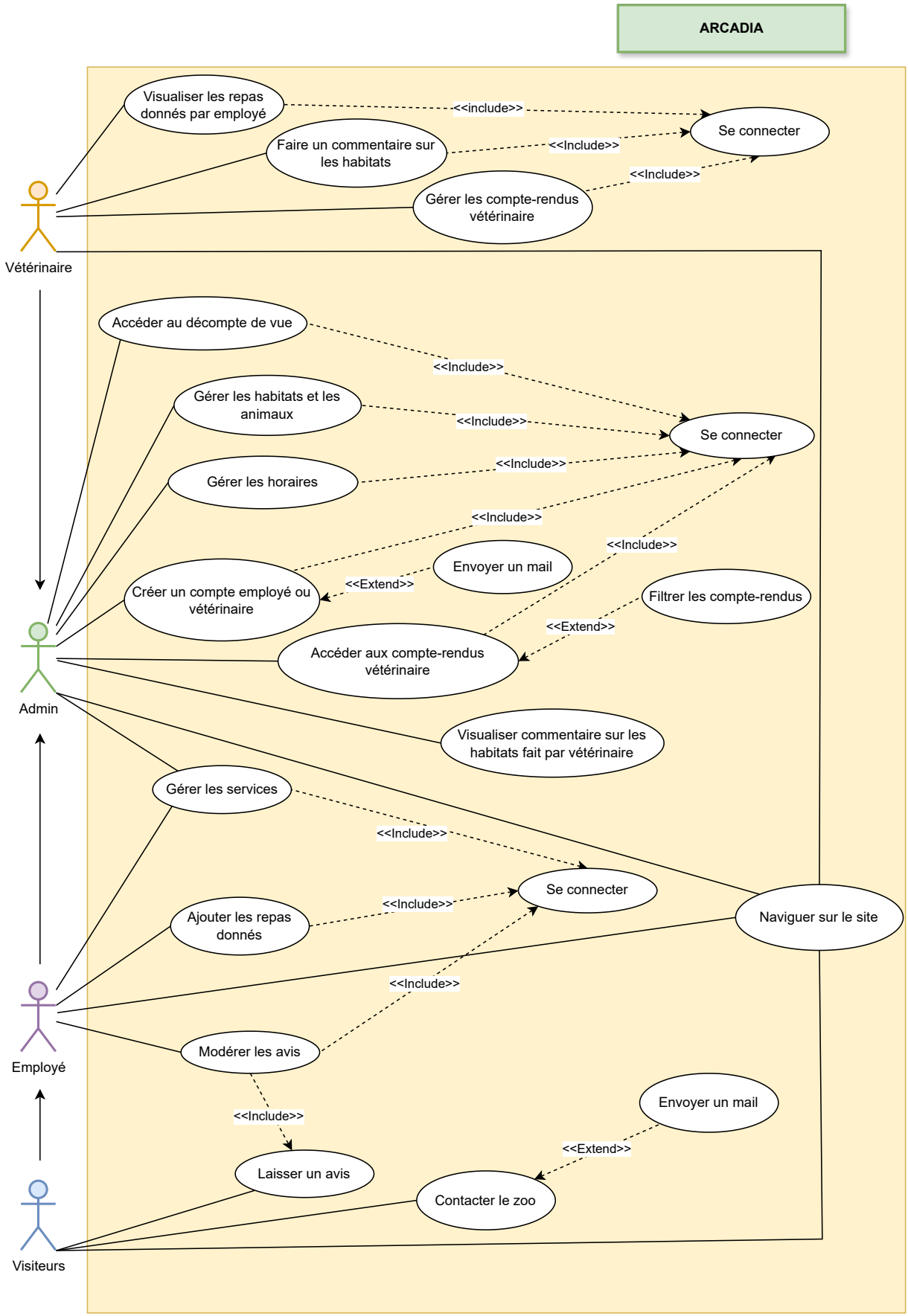
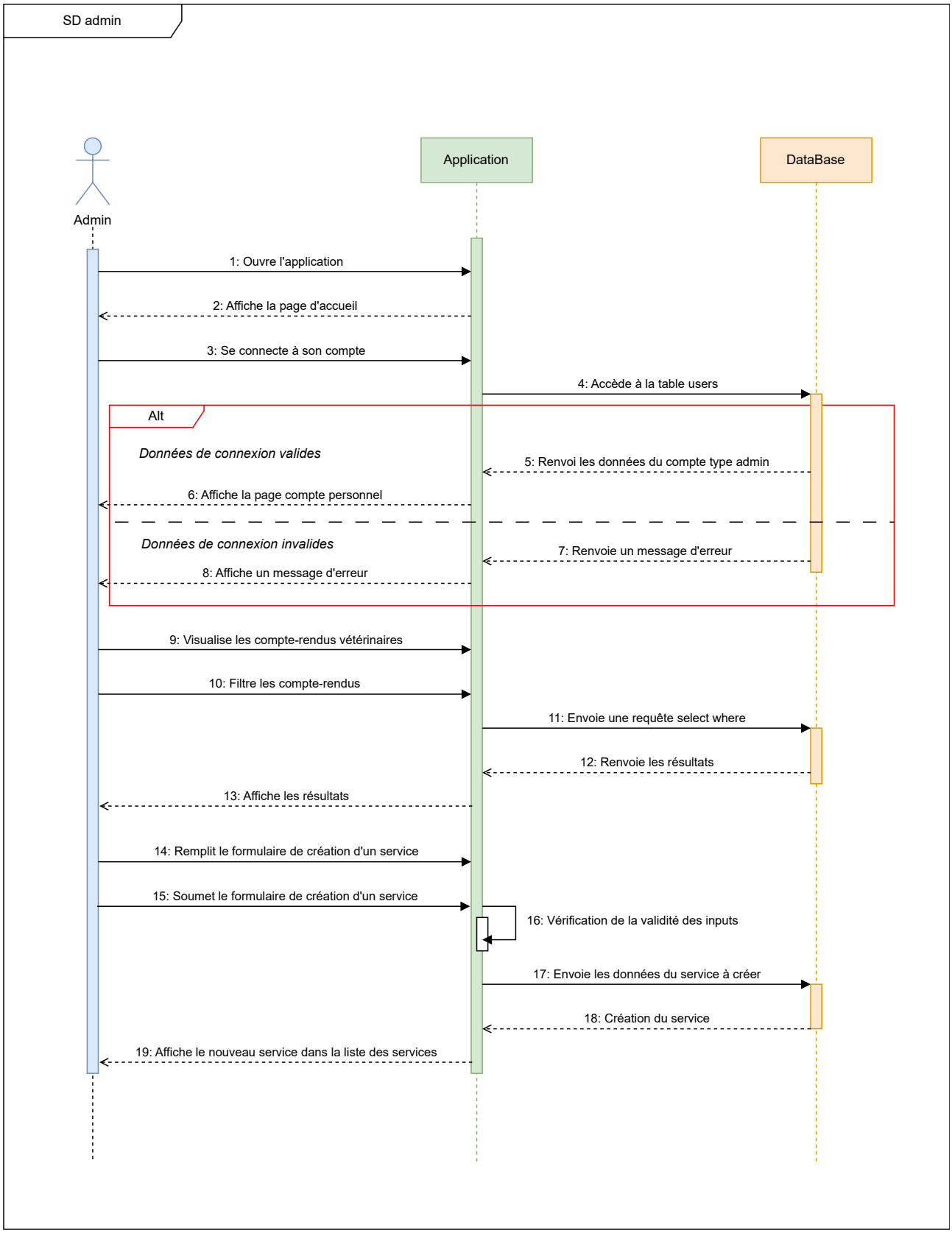


Diagramme de séquence



Documentation du déploiement et démarches suivies

Déploiement de l'application Angular en statique:

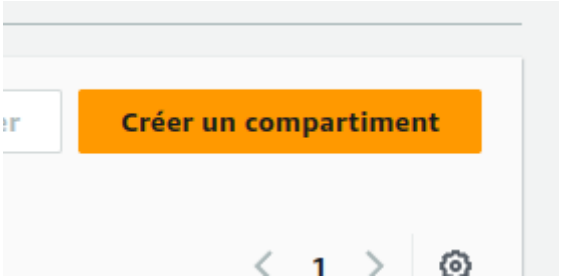
J'ai fait le choix de déployer la partie front-end de mon projet sur AWS.
Ce choix s'explique par sa notoriété, sa facilité de mise en oeuvre et son plan gratuit.

Il faut commencer par se créer un compte sur Amazon Web Service avec un mail et mot de passe.

Une fois connecté, chercher dans la barre de recherche S3.

Maintenant, il faut créer un bucket/compartiment.

Cliquer sur:



Pour bénéficier du tier gratuit il faut laisser les paramètres par défault.
Il suffit de renseigner le nom du compartiment, qui doit être unique dans la zone géographique sélectionnée, de désactiver la checkbox "Bloquer tous les accès publics".
Et enfin cliquer sur "créer un compartiment".

Maintenant il faut sélectionner le compartiment fraichement crée, cliquer sur "charger". Une nouvelle fenetre s'ouvre et ici il faut télécharger ou glisser tous les fichiers présents dans le dossier dist de l'application (dossier automatiquement crée à la commande ng build faite préalablement).
Une fois les fichiers chargeés, il suffit de cliquer sur charger pour revenir à la page récapitulative des compartiments.
A nouveau sélectionner le compartiment, dans l'onglet Autorisations, un policy bucket est nécessaire. Il faut copier-coller :

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "PublicReadGetObject",
         "Effect": "Allow",
         "Principal": "*",
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::myangularbucket31/*"
      }
   ]
}
```

Dans l’onglet Propriétés, il faut se rendre tout en bas de la page au niveau Hébergement de site Web statique et cliquer sur modifier.

Dans cette nouvelle page, il faut cocher activer hébergement de site Web statique et Héberger un site Web statique dans types d’hébergement.

Dans les champs Document d’index et Document d’erreur, on doit spécifier la page d’accueil.
J’ecris donc pour mon projet index.html.

Et enfin je clique sur Enregistrer les modifications.

De retour, dans l’onglet Propriétés, le lien de dépoiement se trouve dans la partie Hébergement de site Web statique.

<u>Deploiment de ma base de données relationnelle MySql:</u>

Pour les mêmes raisons que pré-citées, j’ai fait le choix de AWS.

Le service AWS pour les bases de données relationnelles se nomme RDS.
Une fois dans le service, je clique sur Créer une base de données.

Une nouvelle page s’ouvre, je choisis:
- la méthode de création standard,
- MySQL,
- Modèles offre gratuite,

- dans Paramètres:
    - je donne un nom à mon cluster,
    - un utilisateur principal,
    - coche Autogéré pour définir mon propre mot de passe,
- dans Connectivité:
    - Accès public oui

Tous les autres paramètres sont laissés par défaut.
Je finalise la création de ma bdd en cliquant sur Créer une base de données.

En sélectionnant ma base de donnée, sur l’onglet Connectivité et Sécurité je récupère l’endpoint afin de me connecter à Workbench (contrairement à Docker où j’initiais ma base de données au démarrage des containers sur la base du fichier dbinit/init.sql, RDS ne le permet pas et m’oblige à passer par un client).
L’endpoint va également me servir à définir la variable d’environnement DB_HOST sur mon serveur.

Déploiement de ma base de données non-relationnelle Mongodb:

Ce service étant bien plus complexe sur AWS, je me suis orientée vers Atlas MongoDB.

Une fois le compte crée et connecté, je clique sur créer un cluster.
Je sélectionne le type de cluster “shared”, le plan gratuit tier et clique sur créer cluster.

Je vais dans l’onglet Collections et je crée ma database et mes collections.

Pour récupérer l’endpoint, je clique sur connect qui ouvre une fenêtre. En fonction de ce que je veux récupérer je choisis driver (pour définir ma variable d’environnement MONGODB_URI), compass ou shell (pour gérer ma base de données depuis le client compass ou le terminal).

Déploiement de mon serveur:

Je persiste à utiliser AWS pour déployer mon serveur.
Cette fois-ci le service est EC2.

La première étape consite à créer une instance.

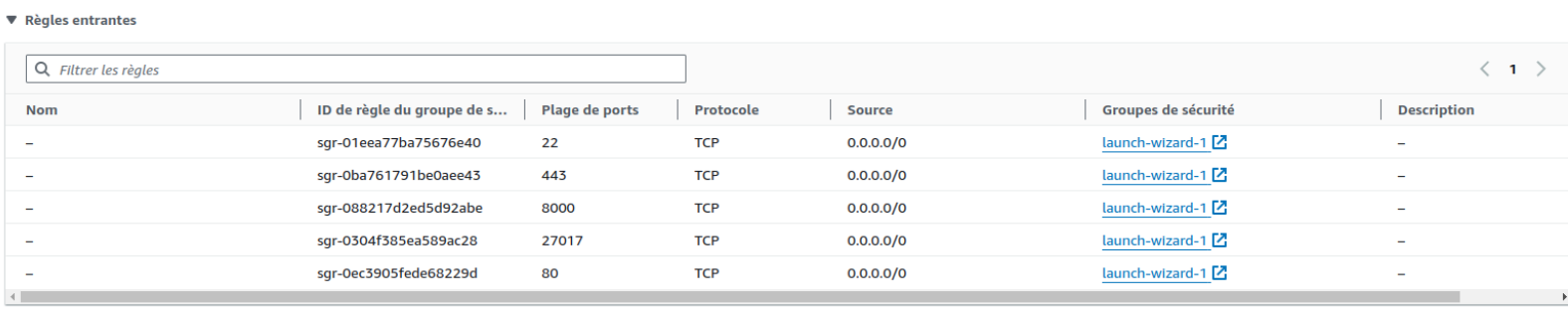
Je clique sur Lancer une instance, je lui donne un nom, je choisis l’image de départ (ici ubuntu).
Dans la partie Paire de clé, je créé une paire de clé. Le téléchargment du fichier .pem se fait automatiquement. Je vais le stocker dans un fichier intitulé ssh.
Dans la partie Paramètres réseau, je coche Autoriser le traffic https et http depuis l’internet.

Je laisse tous les autres paramètres par défaut pour bénéficier de l’offre tier gratuite.

Je clique sur Lancer l’instance.

De retour sur le tableau de bord, quand l’état de l’instance est en cours d’éxécution, je la sélectionne.

Je vais dans Sécurité afin de paramétrer mon groupe de sécurité tel que ci-dessous :



Ensuite je clique sur Se connecter, ce qui m’ouvre une nouvelle page. Je vais dans l’onglet SSH. J’ouvre un nouveau terminal et je suis les commandes de la documentation :

chmod 400 "nodeinstance.pem", commande qui vérifie les droits.

ssh -i "desktop/ssh/nodeinstance.pem" ubuntu@ec2-13-39-80-204.eu-west-3.compute.amazonaws.com, commande qui me permet d’être dans le terminal AWS.

Je fais un apt sudo update et upgrade.
J’installe git avec install git.
Maintenant je peux faire un clone de mon repository github.

J’ai ainsi accès à mon dossier arcadia.
Je navigue dans mon dossier backend et j’installe pm2 qui va me permettre de maintenir mon serveur actif 24h/24.

Je lance la commande pm2 start src/index.js.

Je récupère l’endpoint de mon instance EC2 pour paramétrer mon url de production dans ma partie front-end.

Le déploiement de mon application est désormais complet.