The One :
Cahier des spécifications techniques

Maha El Kourdi
Ikram Laguir

Polytech Informatique 5A Groupe G1
Professeur :
Mr. DELIGNY Manoel

Historique des révisions :

Date	Description et justification de la modification	Auteur	Pages	Édition/Révision
13/10/2021	Création du fichier	-El Kourdi - Laguir	1-8	0A
05/11/2021	Explications ajoutées de toutes les parties	-El Kourdi - Laguir	1-11	0A
11/01/2022	Relecture	-El Kourdi - Laguir	1-11	0A
12/01/2022	Relecture et envoi	-El Kourdi - Laguir	1-11	0A

Table des matières

1 Introduction

1.1 Objectif du document

Ce document décrit les exigences techniques des trois plateformes décrites dans le cahier de charge :

— Utilisateur

— Annonceur

— Administrateur

Des IHM sont donnés à titre indicatif.

1.2 Outils utilisés

L'ensemble du code source du projet est rédigé sur WebStorm . Le Front-end est développé grâce à Angular, le Back-end est développé en NodeJS. Nous utilisons MongoDB pour le stockage de données. Les livrables sont rédigés en LaTeXen utilisant Overleaf (éditeur en ligne de LaTeX)

2 Structure statique

2.1 Analyse statique

The One sera construite via l'architecture *MEAN* (MongoDB, Express, Angular, Node.js). Nous mobiliserons ainsi :

- Une base de données MongoDB comprenant 5 collections :
  - *User* mémorisant les données de l'utilisateur.
  - *Playlist* stockant les playlists pour chaque utilisateur.
  - *Advertiser* mémorisant les données de l'annonceur.
  - *Advert* mémorisant les différentes annonces de chaque annonceur ainsi que leurs status (accepté ou refusé).
  - *Admin* mémorisant les données de l'administrateur.
- Un serveur Node.js gérant les services web échangeant des données entre la base de données MongoDB et l'application Angular.
- Une application monopage Angular modulaire gérant différents modules détaillés en 2.4.1.

2.2 Décomposition générale

La figure 1 représente la gestion des routes dans l'application Angular, ainsi que entre l'application et le serveur. Au sein de l'application, le routeur gère des routes internes pour activer temporairement des composants.
Sinon, les services Angular invoquent les services web du serveur Node.js via des requêtes HTTP spécifiant des routes RESTful. Le serveur Node.js gère également les flux de données de provenance ou en direction des bases de données du système de gestion de base de données MongoDB.

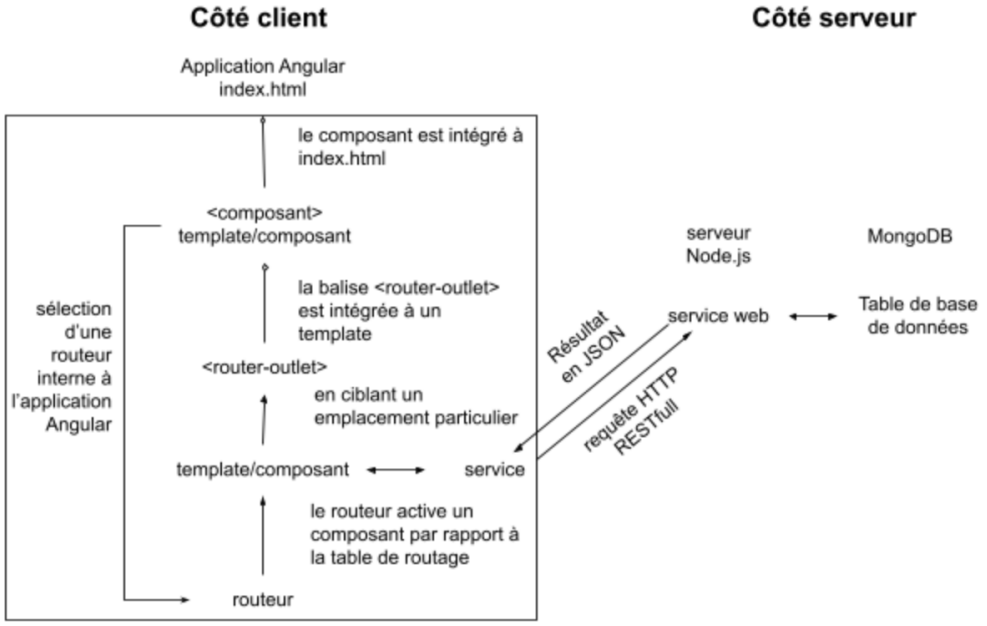

FIGURE 1 – Architecture de The One

2.3 API

Afin de rechercher les vidéos et les afficher sur nos plateformes, on utilise l'api de Youtube. En créant une clé API Youtube, on effectue des "Fetch" via l'url suivante :
https ://youtube.googleapis.com/youtube/v3/search ?part=snippet&q= + titre + keyAPI.
Pour plus d'informations, voir la documentation :
https : // developers . google . com/ youtube/ v3

2.4 Définition des classes

2.4.1 Côté client

Les différents templates des composants Angular seront imbriqués dans ce package qui contient :

- index.html
- accueil.html

On dispose d'un service "message.service" où on implémente les fonctions qu'on va utiliser ensuite dans les composants suivants d'Angular :

Utilisateur

Nom du composant	explication
accueil	Ce componsant contient les fonctions qui permettent de rechercher des vidéos qui sont envoyées au composant recherche-videos.
afficher-playlist	Ce composant contient les fonctions qui affichent une playlist et les différentes vidéos ajoutées
ajout-playlist	Ce composant permet d'ajouter une playlist, un dialog est affiché qui contient un champ pour ajouter le titre de la playlist
inscription	Ce composant permet l'inscription d'un nouvel utilisateur
lecture-video	Ce composant affiche la vidéo que l'utilisateur a ouvert
menu	Ce composant contient une navbar qui contient les routes vers les composants suivants : accueil, mes playlists et se connecter.
mes-playlists	Ce composant affiche les playlists créées par un utilisateur, il est visible uniquement quand l'utilisateur est connecté
recherche-videos	Ce composant affiche les vidéos recherchées dans le composant accueil
se-connecter	Ce composant permet à l'utilisateur de se connecter ou de se rediriger vers le composant

Administrateur

Nom du composant	explication
details	Ce composant affiche les détails d'une annonce, il peut l'accepter (Bouton valider) ou la refuser (Bouton refuser).
home	Ce composant affiche toutes les annonces, leurs états et leurs annonceurs.
login	Ce composant permet à l'administrateur de se connecter.

Annonceur

Nom du composant	explication
details	Ce composant affiche uniquement les détails d'une annonce.
home	Ce composant affiche toutes les annonces, leurs détails sous forme d'un tableau
login	Ce composant permet à l'annonceur de se connecter.
form	Ce composant permet à l'annonceur de créer une annonce en remplissant un formulaire puis valider la création.

2.4.2 Côté Serveur

Au niveau du serveur, nous retrouvons 3 fichiers principales :

- **app.js** :
  - crée le serveur web sur le port 3000
  - se connecte à la base de données Mongodb
- **config** : Contient la configuration qui permet de se connecter à la base de données depuis le serveur MongoDb
- **message.js** renvoie un message au format JSON. On a besoin de passer en paramètre *res*, la réponse que l'on envoie au client (Angular). Le paramètre *data* est un objet JavaScript.

Nous avons ensuite choisi de découper en 3 parties :

- **Models** :
  Contient le schéma des 5 collections et définit la forme des documents de chaque collection :
  - admin
  - advert
  - advertiser
  - user
  - playlist
- **Contrôleurs** :

Contenu du dossier Controlleurs		
admin	admin.js	— **signup** : Permet à l'administrateur de s'enregistrer — **login** : Permet à l'administrateur de se conneceter — **getAdvert** : Permet à l'administrateur de récupérer tous les annonces — **updateAdvert** : Permet à l'administrateur de changer le statut de l'annonce
advertiser	advertiser.js	— **signup** : Permet à l'annonceur de s'enregistrer — **login** : Permet à l'annonceur de se conneceter — **create** : Permet à l'annonceur de créer une annonce — **getAdvert** : Permet à l'annonceur de récupérer ses annonces
user	user.js	— **signup** : Permet à l'utilisateur de s'enregistrer — **login** : Permet à l'utilisateur de se conneceter — **getAdvert** : Permet d'afficher une annonce à un utilisateur en tenant compte de son âge et son centre d'intérêt. — **getAdvertNoConnected** : Permet d'afficher une annonce aléatoire à un utilisateur.
	playlist.js	— **create** : Permet à un utilisateur de créer une playlist. — **playlists** : Permet à un utilisateur de récupérer tous ses playlists — **deletePlaylist** : Permet à un utilisateur de supprimer une de ses playlists — **addVideo** : Permet à un utilisateur d'ajouter une vidéo dans une playlist donnée. — **videos** : Permet à un utilisateur de récupérer tous les vidéos d'une playlist donnée. — **deleteVideo** : Permet à un utilisateur de supprimer une vidéo dans une playlist donnée.

generateData	generateData.js	— **generateNb** : permet de donner un nombre aléatoire entre un min et un max définis comme paramètres — **generateInterest** : permet de choisir un centre d'intérêt entre 4 préalablement définis — **generateCountry** : permet de choisir un pays parmi 3 préalablement définis — **formatDate** : permet d'écrire une date sous la forme dd/mm/yyyy — **addToDb** : permet d'ajouter un utilisateur donnée dans la base de données — **generateUser** : permet de générer des données aléatoire pour pouvoir créer un utilisateur : userName, email, password, pays, interest, dateOfBirth. — **main** : Une boucle qui fera appel à generateUser et addToDb pour ainsi insérer 10 000 utilisateur dans la base de données

— **Routes** :
Dans ce dossier, on met en place les routes qui seront servies par le serveur web : chaque route correspond à un fichier que l'on charge via un *require*. Ce fichier exporte juste une fonction, que l'on appelle quand l'utilisateur *(admin, user, annonceur)* demande à accéder à la route.

* **admin.routes.js** :
utilise la route : ***/api/admin/<la fonction souhaitée>***
* **advertiser.routes.js** :
utilise la route : ***/api/advertiser/<la fonction souhaitée>***
* **user.routes.js** :
utilise la route : ***/api/user/<la fonction souhaitée>***
* **data.route.js** :
utilise la route : ***/api/data*** pour appeler la fonction ***main***

3 Aspects dynamiques

3.1 Cas d'erreurs

Lorsque le programme détecte une erreur, il réagit en affichant une popup personnalisée selon cette dernière.
Dans les formulaires : Si l'utilisateur ne remplit pas un champ obligatoire, la sauvegarde est impossible et un message d'erreur qui s'affiche (pop up ou pop in), on retrouve cette gestion d'erreur dans les composants suivants : login, se-connecter, inscription, form.

3.2 Gestion de la mémoire

La mémoire allouée dynamiquement du programme NodeJs est **automatiquement** libérée. Cette libération automatique est réalisée par un garbage collector/ ramasse-miette.

3.3 Sécurité

Tous les mots de passes des utilisateurs, administrateurs et annonceurs sont hachés pour assurer une connexion sécurisée.

4 Déploiement

Afin de déployer l'application, on utilise Heroku. Certaines modifications sont nécessaires à apportées dans :

— le fichier package.json dans le dossier qui correspond au frontend : (Utilisateur, annonceur ou administrateur)

```
"scripts": {
    "heroku-postbuild": "ng build --prod"
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
```

— le fichier app.js dans le dossier backend, ajouter les lignes de code suivantes :

```
app.get('/', function(req,res){
res.sendFile(path.join(__dirname+'/dist/index.html'));
}
```

Ensuite, il faut se rendre sur Heroku, voici les étapes à suivre :

— Créer un compte depuis https ://id.heroku.com/login, puis se connecter

— Créer un nouveau projet the-one-project

— Depuis votre terminal, dans la racine du projet, taper la commande *Installer Heroku CLI*

— Taper la commande *Heroku login*

— Taper les commandes suivantes :

```
git init heroku
git:remote —a the—one—project
```

Maintenant on peut déployer l'application sur Heroku en utilisant Git.

```
git add .
git commit —am "Deploy"
git push heroku master
```

Pour plus d'informations, voir la documentation : `https://devcenter.heroku.com/categories/reference`