Rapport AWS
Groupe 8 : Projet TODO LIST

Niels Merceron
Pierre Vermeulen
Alexis Guigal
Manel Azgag

Table des matières

Chapitre 1

Introduction

Nous sommes un groupe avec peu de notions de web, donc pour nous familiariser avec les langages et autre technologie nous avons fait le choix de faire un site simple et plus particulièrement un site de to do list. Il y a beaucoup de ressource en ligne nous décrivant comment faire une to do list et donc nous aiderons tout au cours du développement de notre site pour savoir quel service implémenter de notre site ou non, si c'est à notre portée. Pour autant, faire ce site relèvera de quelques challenges de conception vu que nous ne sommes pas familiers avec beaucoup de technologie pour développer des sites web. On peut noter par exemple la gestion du back peut être vite contraignante, ou l'aspect sécurité qui doit être fait de manière la plus propre possible pour ne laisser aucune faille.

Chapitre 2

Technologie utilisée

2.1 Front end

2.1.1 Svelte

C'est un framwork simple et proche syntaxiquement du javascript standard. Il est conçu pour compiler le code au moment de la compilation plutôt qu'au moment de l'éxécution, il offre donc de très bonnes performances. Le framework est également bien documenté avec une documentation claire et précise.

2.1.2 Tailwind avec DaisyUI

C'est un framwork qui est bien plus simple que les autres framework CSS.Sa documentation , sa simplicité d'utilisation et de la modernité des design présent fait de ce framework le meilleur choix pour notre application.Il est également très facile à intégrer avec n'importe quel type de stack.

2.2 Back end

2.2.1 Nodejs/expressjs

C'est un environnement d'éxécution pour JavaScript construit sur le moteur Javascript de Chrome. Il est extrêmement populaire et il est utilisé par de très nombreuses entreprises. Il est également très simple à prendre en main.

2.2.2 JWT

Le jsonwebtoken est un standard définit dans la RFC 7519. La technologie de jsonwebtoken est publique (c'est une norme ouverte) qui a pour but d'avoir un format compact et autonome pour assurer des communications sécurisées entre deux ou plusieurs éléments. Cette technologie existe depuis 2015. Un JWT est composé de la manière suivante : - un header contenant l'algorithme de chiffrement, le type de token (JWT) - un payload (la charge utile) qui contient toutes les informations transmises à l'application - Une signature qui est générée en fonction du header ,du payload et d'une clef secrète connu que part l'application.

On peut l'utiliser dans à peut près tous les cas possibles car il est assez "fléxible" de conception. Cependant il y a trois cas où le JWT est souvent utilisé. Dans les applications de REST pour sécuriser un protocole en envoyant direct les données d'authentification, du Cros origin ressource sharing et quand il y a plusieurs frameworks utilisés. Dans notre cas, on l'utilisera pour sécuriser une connexion d'un utilisateur.

2.3 Sécurité

Pour la sécurité côter client , on utilise la bibliothèque crypto-js qui nous permet d'avoir le principaux outils de sécurité côter client (hash , encryption)

Pour la sécurité de nos données sensibles côté serveur (mot de passe, données privées) nous allons utilisé :

- **bcrypt + crypto** : bibliothèque dans le nodejs d'hachage de mot de passe + crypter des messages.

- **Helmet** : Bibliothèque Express qui permet de sécurisé son site en définissant divers en-têtes HTTP pour eviter les vulnérabilités courantes (détournement de clics, HTTP strict ...) En effet HTTP est de base ouverte et non sécurisé. Il peut notamment divulger des informations sensible à toutes les personnes ayant les compétences techniques.

- **TLS** : C'est le protocol de la sécurité de la couche de transport. Elle satisfait différents objectifs client-serveur :

- Authentification du serveur
- Authentification du client (optionnel)
- Confidentialité des données échangées (chiffrement des données)
- Intégrité des données échangées

De plus que ça, il faut prendre quelque reflexe comme sécurisé les cookies où éviter les attaques par force brute. Pour le hachage, il faut absolument hacher côté client et côté serveur pour eviter les attaques dans le canal entre

client et serveur(utilisation du hachage pour ce connecté, connaissance du clair).

2.4 SGBD

2.4.1 SQL ou NoSQL

Nous avions donc opter pour un SGBD NoSQL malgré le fait qu'on ai plus pratiqué le SGBD relationnel. Cependant le NoSQL est plus adapté et plus le projet évolue, plus le NoSQL deviennait une évidence.

2.4.2 Mongodb

MongoDB est un système de gestion de bases de données NoSQL, qui utilise un format de stockage de données basé sur JSON. Contrairement aux bases de données relationnelles, MongoDB n'utilise pas de tables et de schémas fixes, mais stocke les données dans des collections flexibles qui peuvent être modifiées sans avoir à définir un schéma préalablement. Cette approche permet une grande flexibilité pour gérer des données complexes et des schémas évolutifs, ce qui est particulièrement utile pour les applications modernes basées sur le cloud. De plus, MongoDB offre des performances élevées, une évolutivité horizontale facile et une grande disponibilité, ce qui en fait une solution de choix pour les entreprises qui cherchent à développer des applications à grande échelle. En raison de ses avantages, MongoDB est très utilisé et dispose d'une documentation de qualité, ce qui facilite son adoption et son utilisation.

Chapitre 3

Déroulement du projet

3.1 Back

3.1.1 Création des routes

- Signup - POST

Exemple de requête :

```
{
   "username": "bonjour",
   "email": "test@fafa.com",
   "password": "drfffffffff"
}
```

Exemple de réponse :

```
{
   "token": "eyJhbGciOiJIUzI1Ni5cCI6...2v0RWSOkv4"
}
```

- Login - POST

Exemple de requête :

```
{
   "email": "test@fafa.com",
   "password": "drfffffffff"
}
```

Exemple de réponse :

```
{
   "token": "eyJhbGciOiJIUzI1R5cCI6...2v0RWSOkv4"
}
```

- Createtodo - POST

Exemple de requête :

```
{
   "title": "tesvqhtret1",
   "description": "gfzghfgrfeafezrgezg"
}
```

Exemple de réponse :

```
   {"title":"tesvqhtret1",
   description":"gfzghfgrfeafezrgezg",
   "completed":false,
   "createdAt":"2023-04-28T23:57:41.194Z",
   _id":"644c5ef0c7ff2ca10c44521c","__v":0}
```

- Gettodo - GET

Exemple de requête :
Exemple de réponse :

```
[
   {
   "_id": "644c5647cd148f99d95d98a7",
   "title": "test1",
   description": "gfzghezrgezg",
   "completed": false,
   "createdAt": "2023-04-28T23:26:51.777Z",
   "__v": 0
   },
   {
   "_id": "644c5ef0c7ff2ca10c44521c",
   "title": "YES",
   "description": "mod",
   "completed": false,
   "createdAt": "2023-04-28T23:57:41.194Z",
   "__v": 0
```

```
  }
]
```

3.1.2 - Del - DEL

Exemple de requête :
Exemple de réponse :

```
{
  "message": "Todo Deleted"
}
```

3.1.3 - Modify - PUT

Exemple de requête :
Exemple de réponse :

```
{
  "_id": "644c5ef0c7ff2ca10c44521c",
  "title": "YES",
  "description": "mod",
  "completed": false,
  "createdAt": "2023-04-28T23:57:41.194Z",
  "__v": 0
}
```

Chapitre 4

Objectif accompli durant le projet

Dans cette section, on vous présente l'état final de notre projet.

Chapitre 5

Conclusion