29/11/2024

JULIEN BOUTREAUX

Une image contenant texte, Police, bougie, logo

Description générée automatiquement

**RT0907- Programmation Cloud**

**Rapport TP application serverless de vote**

Table des matières

[I)Introduction : 1](#_Toc183815307)

[II) : Principales Fonctionnalités : 2](#_Toc183815308)

[1)Affichage des candidats 2](#_Toc183815309)

[3) Collecte des votes 2](#_Toc183815310)

[4) Suivi de l’état des votes 2](#_Toc183815311)

[5) Gestion des utilisateurs 2](#_Toc183815312)

[III) : Services AWS utilisés 2](#_Toc183815313)

[1)Base de données DynamoDB 2](#_Toc183815314)

[2) Fonctions lambda 3](#_Toc183815315)

[3)API Gateway 3](#_Toc183815316)

[4)Amazon S3 4](#_Toc183815317)

[IV) : Echanges effectués 4](#_Toc183815318)

[1)register.html et register\_user : 4](#_Toc183815319)

[2)login.html et login-user-lambda 4](#_Toc183815320)

[3)home.html et return-listes-candidats 5](#_Toc183815321)

[4) description\_candidat.html et return-candidat 6](#_Toc183815322)

[5) Bouton voter et vote-lambda : 6](#_Toc183815323)

[6) Résultat.html et resultat-vote 7](#_Toc183815324)

[V) : Test 7](#_Toc183815325)

[VI) : Conclusion 8](#_Toc183815326)

[VII) : Annexes 8](#_Toc183815327)

# I)Introduction :

Le but de ce projet est de créer une application serverless de vote en utilisant les services d'Amazon AWS. L'objectif principal est de permettre une gestion des votes et pouvoir suivre les résultats. Ce projet repose sur plusieurs services AWS comme AWS lambda, API Gateway, DynamoDB et Amazon S3 et IAM (Identity and Access Management) chacun jouant un rôle clé dans le fonctionnement de l'application, permettant à la fois la gestion des utilisateurs, la collecte des votes et la présentation des résultats.

# II) : Principales Fonctionnalités :

## 1)Affichage des candidats

En arrivant à la page d’accueil lorsqu’ils se sont connecté les utilisateurs peuvent consulter une liste des candidats avec une image et leur nom et prénom, en cliquant sur l’image ils arrivent sur la page description de ce candidat. Chaque candidat est présenté avec son prénom, son nom et une brève description, c’est sur cette page qu’ils peuvent voter. Les images des candidats sont dynamiquement chargées depuis un stockage S3.

3) Collecte des votes  
Les utilisateurs peuvent voter pour leur candidat préféré. Le vote est enregistré dans une base de données via une fonction AWS Lambda. Chaque vote est unique et le système empêche les votes multiples pour un même candidat par le même utilisateur.

4) Suivi de l’état des votes  
les utilisateurs peuvent consulter l’état actuel du vote sur la page résultat sous forme d’un diagramme circulaire.

5) Gestion des utilisateurs  
J’ai mis en place un système d’authentification simple pour simuler les votes uniques, j’ai fait un formulaire d’inscription qui via la fonction lambda register-user peut créer un nouvel utilisateur dans la table Users de DynamoDB et le formulaire connexion qui permet d’accéder aux sites en utilisant la fonction lambda login-user-lambda qui vérifie dans la table Users si le pseudo et le mot de passe est valide. J’ai plus tard appris qu’un service AWS pouvait s’occuper de l’authentification et de l’inscription, il se nomme Amazon Cognito. Ce service aurait grandement simplifié cette partie.

# III) : Services AWS utilisés

## 1)Base de données DynamoDB

Pour la gestion des données de l'application, j'ai choisi d'utiliser DynamoDB, une base de données NoSQL entièrement managée par AWS. Au début j’ai hésité entre utiliser RDS et DynamoDB, j’ai finalement choisi DynamoDB car c’est une base de données bien plus facile à prendre en main puisque qu’elle est noSQL. De plus, la gestion des serveurs, des mises à jour ou des sauvegardes est entièrement prise en charge par AWS. DynamoDB est également conçu pour offrir une faible latence et une haute performance pour les applications. J’ai donc pensé que DynamoDB était un choix approprié pour l’architecture serverless que je veux mettre en place. J’ai trois tables importantes qui permettent le fonctionnement de mon application. La table Candidats qui stocke les candidats, leur nom, prénom, âge, un id unique et une description de leur programme. La table Users qui stocke les utilisateurs, leur nom, prénom, pseudo, âge et email. Et enfin la table Votes qui stocke les votes des utilisateurs, elle stocke l’id de l’utilisateur qui a voté, l’id du candidat pour lequel l’utilisateur a voté et le nom du vote. J’ai choisi cette approche plutôt qu’un booléen a voté sur un utilisateur pour l’évolution de l’application si jamais il y avait d’autres sondages disponibles comme cela un utilisateur ne peut voter qu’une seul fois pour un sondage avec un simple booléen ça ne serait pas possible.

## 2) Fonctions lambda

Pour une application serverless sous AWS, nous devons utiliser les fonctions lambda. Avec Lambda, il n'y a aucun serveur à provisionner, gérer ou mettre à jour, ce qui simplifie considérablement l'infrastructure. Pour ce projet j’ai mis en place plusieurs fonctions lambda :

Register-user : Reçois les données du formulaire d’inscription et les écrits dans la table Users de DyanmoDB

Login-user-lambda : Reçois les données du formulaire d’inscription et vérifies dans la table Users si le pseudo et le mot de passe est valide

Return-listes-candidats : appelé par la page home.html, elle consulte la table Candidats et retourne tous les candidats disponibles.

Return\_candidat : appelé par la page description\_candidat.html, elle reçoit un id (l’id du candidat) et retrouve le candidat à partir de son id.

Vote-lambda : Reçoit les données nécessaires au vote, le nom du vote, l’id du candidat choisit et l’id de l’utilisateur qui souhaite voter ensuite elle vérifie dans la table Votes si pour l’utilisateur n’a pas déjà voté pour ce sondage grâce aux User\_Id. Si l’utilisateur n’a pas encore voté alors elle écrit le nouveau vote dans la table sinon elle envoie un message « Vous avez déjà voté pour cette élection ».

Resultat-vote : Elle retourne le résultat en pourcentage des votes pour chaque candidat.

Chacune de mes fonctions lambda utilisent DynamoDB, elles lisent et écrivent dedans donc dans l’IAM, je leurs ait accordés l’autorisation AmazonDynamoDBFullAccess

## 3)API Gateway

Amazon API Gateway est un service géré qui permet de créer, publier, sécuriser et surveiller des API RESTful. API Gateway permet de déclencher des fonctions Lambda directement via des requêtes HTTP. Cela simplifie l'interconnexion entre le client et les fonctions Lambda, sans avoir à gérer un serveur intermédiaire. Avec Gateway je peux facilement créer des endpoints pour que mon application puisse appeler la fonction lambda souhaité, par exemple pour appeler ma fonction lambda vote-lambda, j’utilise l’endpoint :

https://w6f8a3lnr6.execute-api.eu-west-3.amazonaws.com/test/vote

## 4)Amazon S3

Amazon S3 (Simple Storage Service) est un service de stockage d’objets. Il me permet d’héberger des fichiers statiques comme des images, des fichiers HTML, CSS ou JavaScript. Par exemple, les photos des candidats sont stockées dans un bucket S3, ce qui permet de les récupérer dynamiquement via leur URL dans mon application.

Par défaut, les fichiers téléversés dans un bucket S3 sont privés. Cependant, comme ces fichiers sont destinés au frontend et doivent être accessibles par les utilisateurs, j'ai configuré leur visibilité en utilisant l'option --acl public-read lors du téléversement avec l'interface en ligne de commande AWS CLI :

aws s3 cp . s3://projet-vote-s3/ --recursive --acl public-read

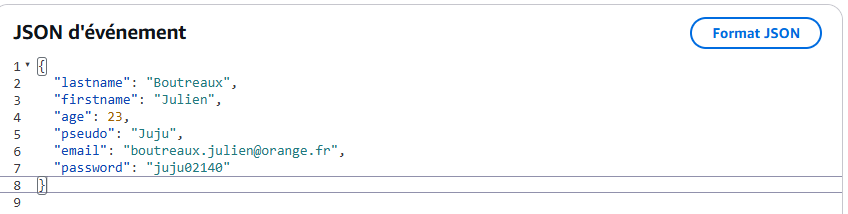
Cette configuration garantit que les fichiers, comme les images des candidats, sont directement accessibles via leur URL publique.

# IV) : Echanges effectués

## 1)register.html et register\_user :

Le formulaire de register.html envoi en POST un JSON à la fonction lambda

Structure du JSON :



Avec ces informations la fonction lambda va créer le nouvel élément dans la table Users. Si tout va bien elle renvoit un status code 200 sinon un status code 500.

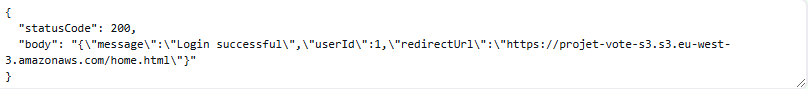
## 2)login.html et login-user-lambda

Le formulaire de login.html html envoit en POST un JSON à la fonction lambda

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquementStructure du JSON :

Avec ces informations la fonction lambda va vérifier si le pseudo et le mot de passe est valide.

Message du lambda en cas de réussite :

Message du lambda en cas d’échec :

En cas de réussite la fonction lambda envoi l’url ou doit être rediriger l’utilisateur ici la page home.html. L’application sauvegarde aussi l’id de l’utilisateur en faisant localStorage.setItem('userId', body.userId). Ce n’est pas très sécurisé mais j’ai choisi d’opter pour ce choix car ce n’est pas vraiment l’objectif premier de ce projet.

## 3)home.html et return-listes-candidats

Dès que l’utilisateur arrive sur home.html, la page appelle la fonction lambda return-listes-candidats en GET.

Une image contenant texte, Police, capture d’écran, information

Description générée automatiquementRéponse de return-listes-candidats :

Quand l’utilisateur clique sur l’un des candidats, cela va le rediriger vers la page descriptive du candidat, la page description\_candidat.html, voici la redirection :

https://projet-vote-s3.s3.eu-west-3.amazonaws.com/description\_candidat.html?id=${candidat.Candidat\_Id}

description\_candidat.html va récupérer l’id du candidat dans l’url et va ensuite l’envoyer à la fonction lambda return-candidat.

## 4) description\_candidat.html et return-candidat

La page envoi l’id su cnadidat en POST à la fonction lambda.

Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquementStructure JSON envoyée à return-candidat :

Une image contenant texte, capture d’écran, Police, document

Description générée automatiquementRéponse de return-candidat :

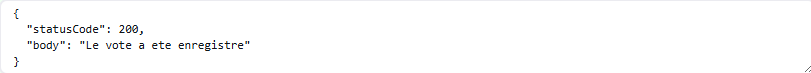
## 5) Bouton voter et vote-lambda :

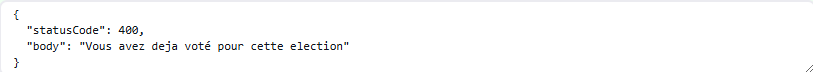
Si l’utilisateur clique sur le bouton voter pour le candidat sur la page de description alors le bouton appellera la fonction vote-lambda en POST.

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquementStructure JSON envoyée à la fonction lambda :

Avec ces informations la fonction lambda vérifie dans la table Votes si l’utilisateur qui a l’id 123 a déjà voté pour le sondage « éléction\_2024 », s’il n’a pas voté auparavant alors la fonction crée le nouveau vote dans la table sinon elle envoie un message à l’utilisateur « vous avez déjà voter pour cette élection ».

Réponse positive de la fonction lambda :

Réponse négative de la fonction lambda :

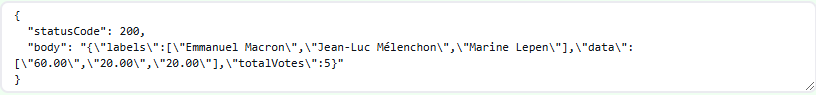
## 6) Résultat.html et resultat-vote

En arrivant sur la page resultat.html, elle va directement appeler la fonction lambda en POST en lui envoyant le nom du vote.

Une image contenant texte, ligne, Police, capture d’écran

Description générée automatiquementStructure JSON envoyée à la fonction lambda :

Elle va consulter la table Votes pour retrouver tous les votes associés à election\_2024 et elle calcule le nombre total de vote pour chaque candidat et le pourcentage de votes obtenus par rapport au total des votes.

Réponse de la fonction lambda :

La page resultat.html va pouvoir ensuite créer le diagramme en fonction de la réponse de la fonction lambda.

# V) : Test

URL de l’index :

https://projet-vote-s3.s3.eu-west-3.amazonaws.com/index.html

Connexion avec un utilisateur ayant déjà voté :

Pseudo : Juju

Mot de passe : juju02140

Connexion avec un utilisateur n’ayant pas voté :

Pseudo : Toto

Mot de passe : toto02140

# VI) : Conclusion

Ce projet a permis de mettre en œuvre une solution complète de gestion de votes en ligne en utilisant des services cloud d'AWS, notamment AWS Lambda, DynamoDB, API Gateway et S3. En choisissant ces technologies, j'ai pu bénéficier d'une architecture serverless, garantissant à la fois la scalabilité, la simplicité de gestion et une réduction des coûts d'infrastructure. J’ai appris à travailler avec DynamoDB et les fonctions lambda afin d’avoir une application capable de gérer des sondages et d’afficher les résultats en temps réel.

# VII) : Annexes

1. Une image contenant Visage humain, capture d’écran, habits, homme

   Description générée automatiquementPage d’acceuil
2. Une image contenant Visage humain, texte, capture d’écran, Site web

   Description générée automatiquementPage description d’un candidat

Une image contenant texte, Visage humain, capture d’écran, logiciel

Description générée automatiquement3)Cas ou le vote est enregistré

Une image contenant capture d’écran, Visage humain, Site web, personne

Description générée automatiquement4)Cas ou l’utilisateur a déjà voté

Une image contenant capture d’écran, texte

Description générée automatiquement5) Page des résultats : resultat.html