# Amazon Web Services (AWS)

Pour commencer, nous allons sur https://aws.amazon.com, puis nous créons un compte AWS.

Une fois le compte crée, nous allons créer une application en React avec cette commande dans le terminal d’un éditeur de code: npx create-react-app [nom de l’app]

Pour vérifier si l’installation a été réussi, vous pouvez entrer dans la commande npm start et vous devriez voir une interface comme celle-ci.

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

Description générée automatiquement

Avant de commencer à coder, nous allons créer un dépôt sur GitHub.

Maintenant, nous allons commencer à travailler sur le frontend. Dans ce laboratoire, nous allons créer :

* Une page de connexion et une autre pour l’inscription.
* Une page pour la gestion d’items de votre choix (ex. recettes, jeux vidéo).
  + Elle doit comporter un moyen d’ajouter, afficher et supprimer ces items.

Puisque le but de ce laboratoire est d’intégrer les services d’AWS dans notre développement d’un projet React, l’interface de vos pages n’a pas besoin d’être soigné.

## Authentification

Pour commencer avec l’inscription, nous allons déployer notre projet React avec AWS Amplify :

1. Écrivez **Amplify** dans la barre de recherche, puis rendez-vous sur la page d’Amplify
2. Cliquez sur **Deploy an app**, choisissez **GitHub**, puis cliquez sur **Next**.
3. Sélectionnez le dépôt du projet React et sa branche, puis cliquez sur **Next**.
4. Cliquez encore sur **Next**, puis **Save and Deploy**.

Maintenant que c’est déployé, vous devriez pouvoir voir l’interface par défaut de React lorsque vous cliquez sur le lien de votre projet déployé.

**Notez que dans la prochaine partie, vous allez rencontrer des erreurs de dépendances avec les modules comme dans mon cas. Malheureusement, je n’ai pas pu prendre tout en note de ce qui a réglé le problème.**

**Pour commencer à mettre en place l’authentification, nous entrons d’abord ces commandes dans la racine de notre projet :**

1. npm install aws-amplify @aws-amplify/ui-react --legacy-peer-deps
2. amplify init
   1. Suivez les instructions pour créer un compte IAM.
3. amplify add auth
4. amplify push

Puis, faites un push sur Git pour activer un autre déploiement sur Amplify. Si vous rencontrez un problème avec le déploiement, allez dans section Hosting > Build settings, puis remplacez **npm ci --cache .npm --prefer-offline** par **npm i --cache .npm --prefer-offline.** Redéployez à nouveau.

À présent, nous pouvons enfin commencer l’authentification. Nous allons faire importation d’Amplify dans index.js et App.js.

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Une fois implémentée, nous retrouverons cette interface :

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

Description générée automatiquement

Attention : Entrez un vrai courriel que vous avez accès puisque la création d’un utilisateur nécessite une vérification par courriel.

Une fois connectée, nous retrouverons un bouton pour se déconnecter.

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

Description générée automatiquement

N’oubliez pas d’ajouter !src/aws-exports.js dans .gitignore.

## DynamoDB et Lambda

**Nous allons créer une table dans DynamoDB :**

1. Tapez **DynamoDB** dans la barre de recherche, puis cliquez.
2. Cliquez sur **Create a table**
3. Entrez le nom de la table et la clé de la partition. Nous avons mis recettes comme nom de la table et id comme clé de la partition de type string.
4. Cliquez sur **Create Table**.

**Par la suite, nous allons préparer le Lambda :**

1. Tapez **Lambda**, puis cliquez dessus.
2. Cliquez sur **Create function**.
3. Entrez le nom de la fonction. Nous avons mis fonction-recettes.
4. Choisissez la version plus récente de Node.js.
5. Dans **Permissions**, choisissez **Change default exécution rôle**
6. Cliquez sur **Create a new role from AWS policy templates**.
7. Entrez le nom du rôle. Nous avons mis rôle-fonction-recettes.
8. Pour le **Policy templates**, choisissez **Simple microservice permissions**.
9. Cliquez **Create function**

**Dans la section Code Source, collez ce code et modifiez-le au besoin :**

import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {

  DynamoDBDocumentClient,

  ScanCommand,

  PutCommand,

  GetCommand,

  DeleteCommand,

} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

const dynamo = DynamoDBDocumentClient.from(client);

const tableName = "recettes";

export const handler = async (event, context) => {

  let body;

  let statusCode = 200;

  const headers = {

    "Content-Type": "application/json",

  };

  try {

    switch (event.routeKey) {

      case "DELETE /recettes/{id}":

        await dynamo.send(

          new DeleteCommand({

            TableName: tableName,

            Key: {

              id: event.pathParameters.id,

            },

          })

        );

        body = `Deleted recette ${event.pathParameters.id}`;

        break;

      case "GET /recettes/{id}":

        body = await dynamo.send(

          new GetCommand({

            TableName: tableName,

            Key: {

              id: event.pathParameters.id,

            },

          })

        );

        body = body.Item;

        break;

      case "GET /recettes":

        body = await dynamo.send(

          new ScanCommand({ TableName: tableName })

        );

        body = body.Items;

        break;

      case "PUT /recettes":

        let requestJSON = JSON.parse(event.body);

        await dynamo.send(

          new PutCommand({

            TableName: tableName,

            Item: {

              id: requestJSON.id,

              nom: requestJSON.nom,

            },

          })

        );

        body = `Put recette ${requestJSON.id}`;

        break;

      default:

        throw new Error(`Unsupported route: "${event.routeKey}"`);

    }

  } catch (err) {

    statusCode = 400;

    body = err.message;

  } finally {

    body = JSON.stringify(body);

  }

  return {

    statusCode,

    body,

    headers,

  };

};

**Nous allons maintenant créer un API http :**

* Tapez **API Gateway**
* Cliquez sur **Create API** et dans la section **HTTP API**, cliquez sur **Build**.
* Entrez le nom de l’API. Nous avons mis api-recettes.
* Cliquez sur **Next**.
* Cliquez encore sur **Next**. Nous allons créer les routes plus tard.
* Cliquez sur **Next**, puis sur **Create**.

**Par la suite, nous allons créer les routes :**

1. Trouvez votre api que vous venez de créer.
2. Cliquez sur **Create**.
3. Pour la méthode, choisissez **GET**.
4. Pour le chemin, entrez /recettes/{id}. Vous pouvez remplace recettes par le type d’item que vous avez choisi.
5. Cliquez sur **Create**.
6. Répétez les étapes 2 à 5 pour GET /recettes, DELETE /recettes/{id}, et PUT /recettes.

**Vous devriez voir quelque chose similaire à ceci :**

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

Description générée automatiquement

**Nous allons ensuite créer une intégration :**

1. Dans **API Gateway**, cliquez sur **Integrations**.
2. Cliquez sur **Manage integrations** et puis sur **Create**.
3. Dans **Attach this integration to a route**, choisissez n’importe quelle route.
4. Pour **Integration type**, choisissez Lambda function.
5. Pour **Lambda function**, entrez le nom de votre fonction. Dans notre cas, c’est fonction-recettes.
6. Cliquez sur **Create**.

**Là, nous allons attacher notre intégration à toutes les routes :**

1. Choisissez une route, puis dans **Choose an existing integration**, choisissez le nom de votre fonction.
2. Cliquez sur **Attach integration**.
3. Répétez les étapes 1 et 2 pour toutes les routes.

**Le résultat :**

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

Description générée automatiquement

**Nous allons maintenant vérifier si l’API et notre fonction Lambda fonctionnent :**

1. Cliquez sur **API :…**

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

Description générée automatiquement

1. Copiez le URL sous **Invoke URL**
2. Ouvrez **Postman**
3. Créez une nouvelle requête de type PUT, puis coller votre url et ajouter /recettes (ou votre item)
4. Allez dans **Body** et écrivez en suivant ce format :

{

"id": "1",

"nom": "Pizza"

}

1. Refaites l’étape 4 (et 5 au besoin) pour GET /recettes, GET /recettes/{id}, et DELETE /recettes/{id}

**Il nous reste seulement à modifier le CORS pour éviter les problèmes de permissions :**

1. Dans **API Gateway**, choisissez **CORS** dans le menu à gauche.
2. Pour **Access-Control-Allow-Origin**, entrez \* et ajouter.
3. Pour **Access-Control-Allow-Methods**, sélectionnez **GET, POST, PUT** et **DELETE**.
4. Pour **Access-Control-Allow-Headers**, tapez content-type et ajouter. Refaire ça avec authorization.
5. Cliquez sur **Save**.

## L’affichage et manipulation des items dans le frontend

Maintenant que DynamoDB et Lambda sont montés, il est temps de revenir à notre application React.

**Nous allons entrer le code suivant (faites attention à l’arborescence des fichiers) et modifiez-le au besoin :**

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

**Pour le code dans App.js :**

import { withAuthenticator } from "@aws-amplify/ui-react";

import React, { useState, useEffect } from 'react';

import { getRecettes, createRecette, deleteRecette, updateRecette } from '../services/api';

import Popup from './popup/Popup';

function App({signOut}) {

  const [recettes, setRecettes] = useState([]);

  const [error, setError] = useState(null);

  const [btnPopup, setBtnPopup] = useState(false);

  const [btnPopup2, setBtnPopup2] = useState(false);

  const [id, setId] = useState("");

  useEffect(() => {

    async function fetchRecettes() {

      try {

        const recettes = await getRecettes();

        setRecettes(recettes);

      } catch (error) {

        setError(error);

      }

    }

    fetchRecettes();

  }, []);

  if (error) {

    return <div>Error: {error.message}</div>;

  }

  async function addRecetteSubmitHandler(event) {

    event.preventDefault();

    const fd = new FormData(event.target);

    const data = Object.fromEntries(fd.entries());

    const newRecette = {

      id: data.id,

      nom: data.nom,

    };

    createRecette(newRecette);

  }

  const supprimer = (id) => {

    deleteRecette(id);

  }

  async function modifierRecetteSubmitHandler(event) {

    event.preventDefault();

    const fd = new FormData(event.target);

    const data = Object.fromEntries(fd.entries());

    const newRecette = {

      id: id,

      nom: data.nom,

    };

    updateRecette(newRecette);

  }

  return (

    <div className="App">

      <h1>Recettes</h1>

      <ul>

        {recettes.map(recette => (

          <div>

            <li key={recette.id}>{recette.nom}</li>

            <button onClick={() => supprimer(recette.id)}>Supprimer</button>

            <button onClick={() => {(btnPopup2 ? setBtnPopup2(false) : setBtnPopup2(true)); setId(recette.id)}}>Modifier</button>

          </div>

        ))}

      </ul>

      <button onClick={signOut}>Sign out</button>

      <button onClick={() => (btnPopup ? setBtnPopup(false) : setBtnPopup(true))}>Ajouter une recette</button>

      <Popup trigger={btnPopup} setTrigger={setBtnPopup} type="info">

            <form onSubmit={addRecetteSubmitHandler}>

              <label>Id</label>

              <input type="id" name="id"></input>

              <label>Nom de la recette</label>

              <input type="nom" name="nom"></input>

              <button type="submit">Ajouter</button>

            </form>

      </Popup>

      <Popup trigger={btnPopup2} setTrigger={setBtnPopup2} type="info">

            <form onSubmit={modifierRecetteSubmitHandler}>

              <label>Nom de la recette</label>

              <input type="nom" name="nom"></input>

              <button type="submit">Modifier</button>

            </form>

      </Popup>

    </div>

  );

}

export default withAuthenticator(App);

**Votre application ressemblera à ceci :**

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

Description générée automatiquement

**Maintenant, on est capable de supprimer, de modifier et d’ajouter un item. Par contre, il faut rafraîchir la page pour afficher les changements.**