



SIMONIN NICOLAS & WATISSE CORENTIN
(GROUPE 4)

JAKARTA X REACT

Gestion des formation : AUDIOVISUEL



RÉSUMÉ DE LA PRÉSENTATION

NOS POINTS DE DISCUSSION

- Description de la problématique
- État de l'art rapide
- Modélisation et sa justification
- Implémentation
- Conclusion
- Démonstration



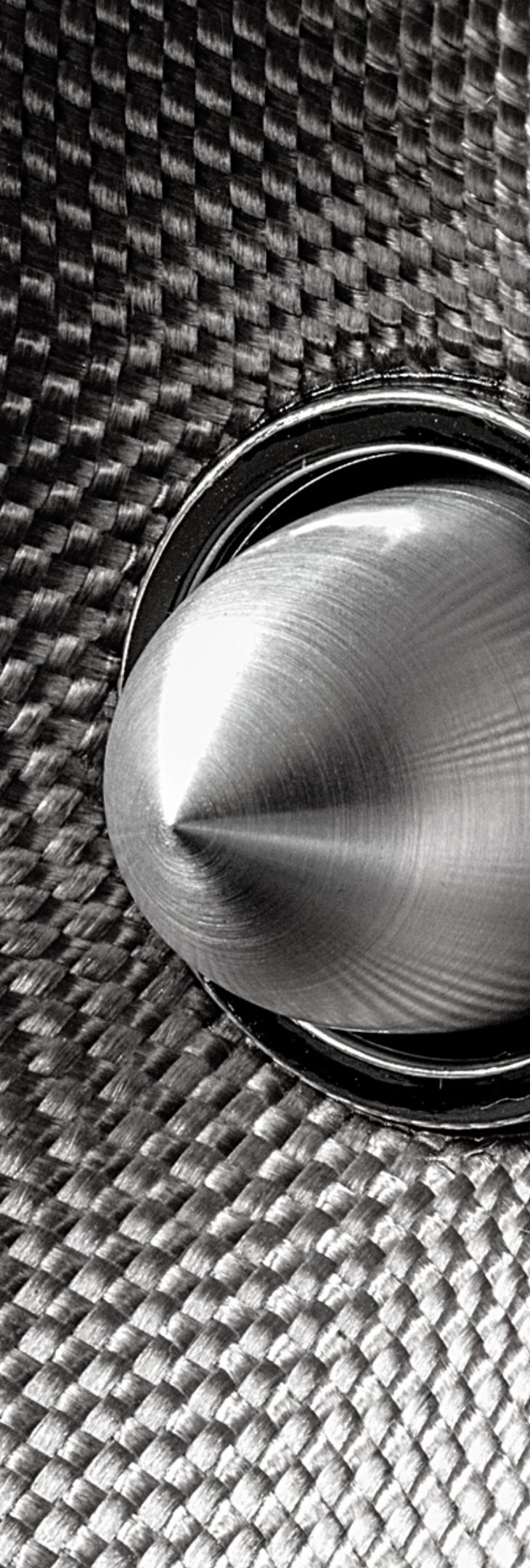
DESCRIPTION DE LA PROBLEMATIQUE

IMAGINONS

Ce projet a pour but la "Réalisation d'une application JakartaEE pour la gestion des formations", pour cela nous avons donc du commencer par choisir sur quel domaine notre système de gestion des formations portera, nous avons donc choisi le monde de l'audiovisuel.

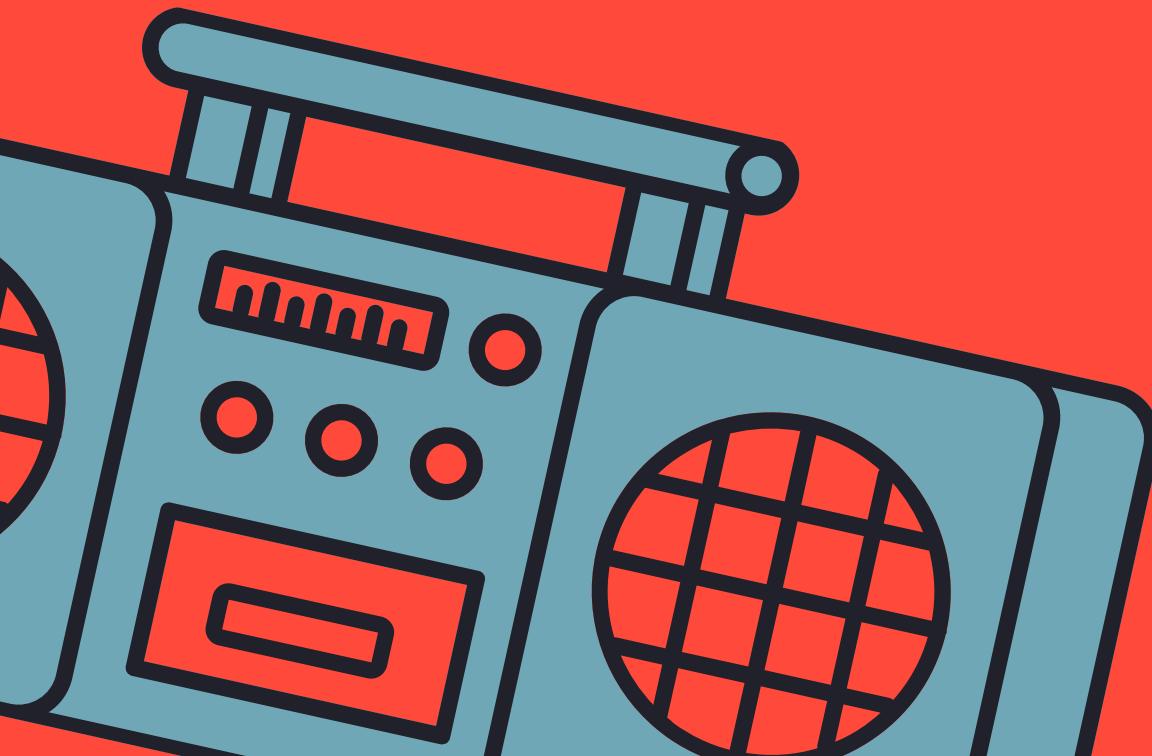
L'idée de ce projet est de permettre à différents utilisateurs d'effectuer un "test", nommé questionnaire plus tard, lui permettant de se rendre compte de quelle formation est la mieux adaptée selon les réponses qu'il aura donné durant ce questionnaire.
Ce questionnaire comprendra des questions, des QCM mais aussi des parties en réalité virtuelle, ainsi que des dépôts de projet à rendre.

Le projet doit être réalisé en deux parties, une partie front-end en REACT et un back-end en JAKARTA EE 8.



ETAT DE L'ART RAPIDE

CE QUI EXISTE DÉJÀ



ONISEP

Éditeur public, l'**Onisep** produit et diffuse toute l'information sur les formations et les métiers



STUDYRAMA

Studyrama est un site spécialisé dans la formation et l'orientation des étudiants, l'emploi et les jobs étudiants.

Quésako ?



CONNEXION

L'utilisateur rentre ses informations (MAIL & PASSWORD), le front-end envoie les données à l'API Jakarta pour vérification, et renvoie si oui ou non les informations sont correctes

ENREGISTREMENT

L'utilisateur renseigne les informations qu'il souhaite utiliser pour la création de son compte (MAIL & PASSWORD), tout est envoyé dans l'API qui crée le compte s'il n'existe pas déjà

QUESTIONS

Les Admins (aussi appelé Formateurs), peuvent créer les questions, ils peuvent aussi les supprimer.

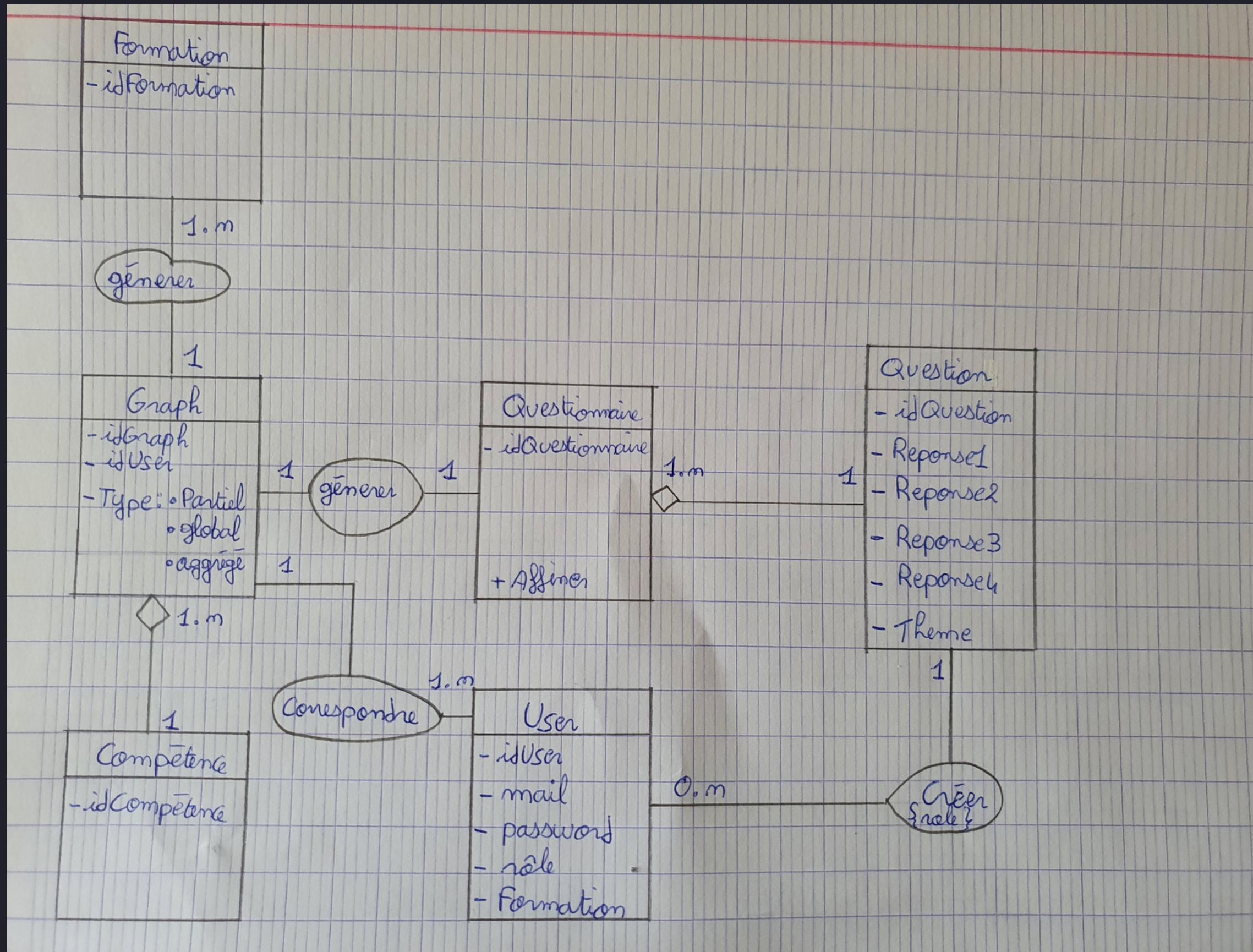
Les Admins peuvent aussi supprimer les utilisateurs existant.

LA MODÉLISATION !

LES DIFFÉRENTS POINTS DU BACK-END :

- Diagramme de classe
- Diagramme de droits d'accès
- Diagramme de base de données

DIAGRAMME DE CLASSE



Principe de base :

- Graph initial
- Questionnaire initial
- Réponses initial
- Graph User
- Formation User

PARTIE BACK

PARTIE FRONT

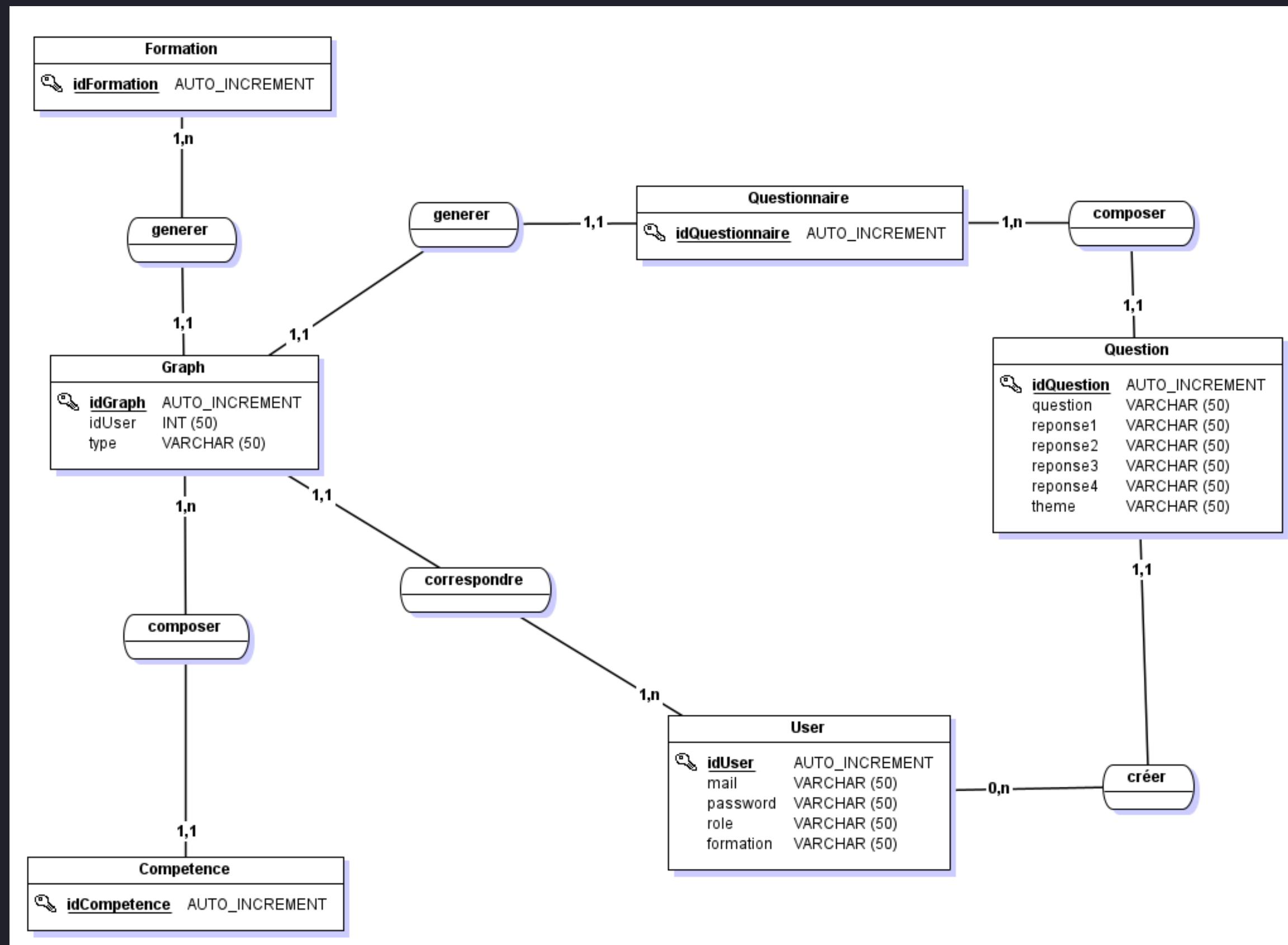
Type	URL	Action
API (JAKARTA)		
ADMIN	/admin	Renvoie un ping (pour s'assurer de la connection Front-end / Back end)
GET	/admin/list	Permet de renvoyer la liste des tous les utilisateurs ainsi que leurs informations (Role, Email).
GET	/remove/{id}	Permet de supprimer un utilisateur (après vérification de son existence) en reseignant son ID unique.
USER	/user	Renvoie un ping (pour s'assurer de la connection Front-end / Back end)
GET	/user/{id}	Permet de renvoyer les informations d'UN utilisateur en renseignant son ID unique.
GET	/login/{mail}/{password}	Permet de tester les identifiants fournit par l'utilisateur avec la BDD, renvoie une erreur si il y en a.
GET	/register/{mail}/{password}/{role}	Permet d'enregistrer un nouvelle utilisateur dans la BDD si il n'existe pas déjà
QUESTION	/questions	Renvoie un ping (pour s'assurer de la connection Front-end / Back end)
GET	/questions/{id}	Renvoie une questions est ses attribut en fonction de son ID unique renseigné.
GET	/questions/liste	Renvoie la liste des questions ainsi que l'ensemble de leurs attribut.
GET	/question/remove/{id}	Permet de supprimer une question selon l'ID unique renseigné
POST	/question/add	Permet de créer une question, les attributs seront fournis via le body du POST.

PARTIE BACK

PARTIE FRONT

		FRONT(React)
	/	Page d'accueil
	/Formation	Page listant les formations dans le domaine de l'audio visuel, ainsi que le résultat du questionnaire (WIP)
	/Questionnaire	Page permettant d'effectuer le questionnaire (WIP)
	/Connexion	Page permettant à l'utilisateur de se connecter en renseignant ses identifiants
	/Inscription	Page permettant à l'utilisateur de s'enregistrer en renseignant ses identifiants
	/PanelAdmin	Page permettant aux admins de voir et de supprimer les utilisateurs du site
	/QuestionAdmin	Page permettant aux adminns de voir et de supprimer les questions des formulaires
	/ListeQuestionAdmin	Page permettant aux admins de créer de nouvelles questions pour les questionnaires
	*	Page de redirection 404

MODÈLE CONCEPTUEL DE DONNÉES (MCD)



L'IMPLEMENTATION!

COMMENT ÇA MARCHE ?

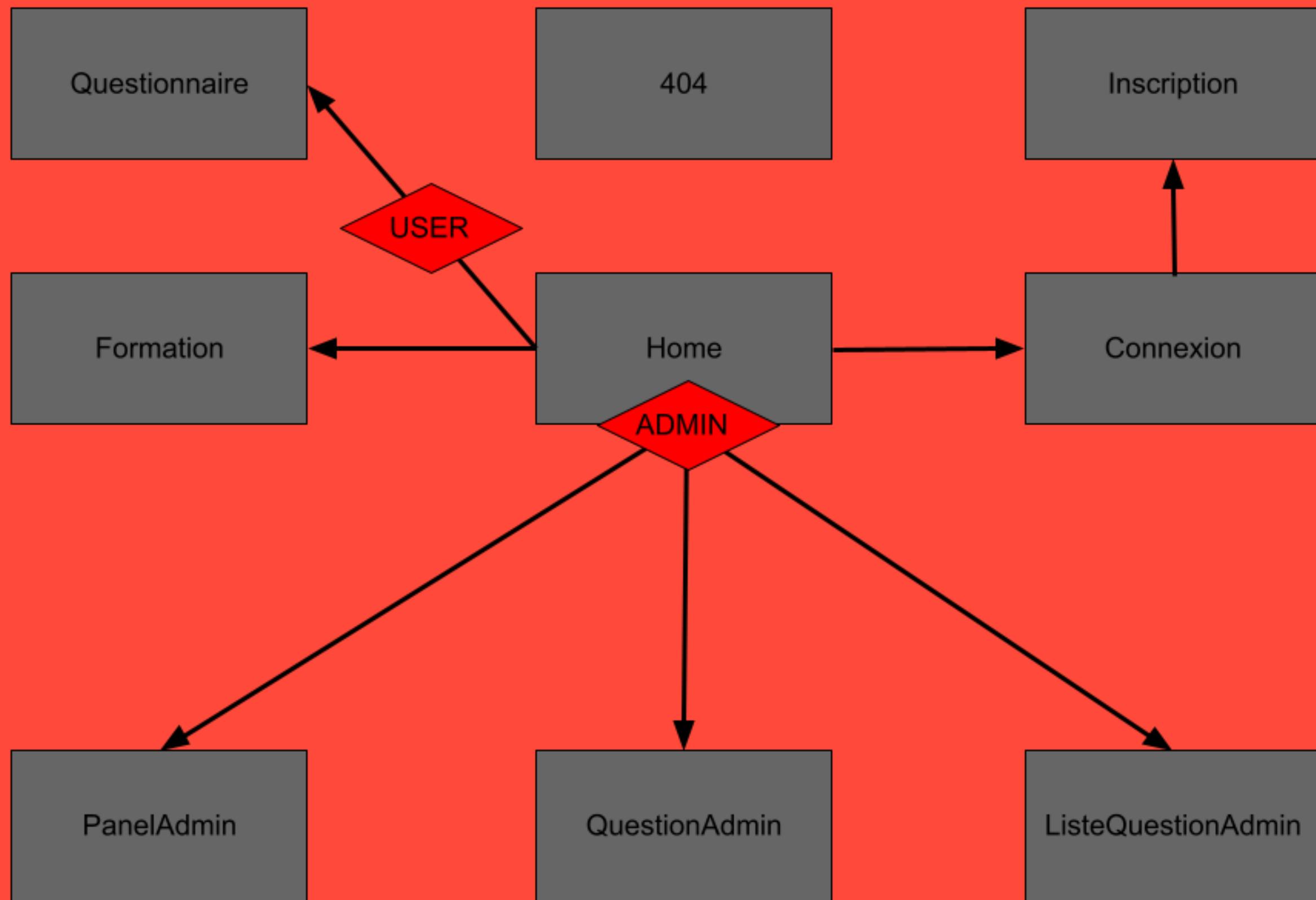


LA MODÉLISATION !

LES DIFFÉRENTS POINTS DU FRONT-END :

- Diagramme de navigation
- Structure du code

DIAGRAMME DE NAVIGATION



PAGES ACTUELLEMENT DISPONIBLE

Ici, vous pouvez retrouver les routes actuellement disponibles du front-end. En effet toutes les pages (sauf une) sont accessibles de partout via la barre de navigation se situant tout en haut de la page.

Certaines pages sont limitées en accès, c'est à dire qu'il vous faut vous connecter avec un compte particulier pour accéder aux pages. Les pages "...Admin" sont réservées aux admin, c'est à dire aux personnes connectés possédant le rôle de "FORMATEUR", ainsi que la page "Questionnaire" uniquement accessible si vous êtes connecté sur site, ce qui permettra d'enregistrer le questionnaire effectué par les utilisateurs.

DOSSIER PROJET "2022-02-GROUPE-04"

Ce dossier comporte les deux parties, "audiovisuel" qui est la partie JakartaEE, et "front-end" qui est la partie React.

STRUCTURE DU PROJET (REACT)

Contient tout les fichiers src du projet, les pages en .js

COMPONENTS

Ici on retrouve tous les "components" utilisés dans le projet. Les composants sont des morceaux de code indépendants et réutilisables. Elles ont le même objectif que les fonctions JavaScript, mais fonctionnent de manière isolées et renvoient du HTML



PUBLIC

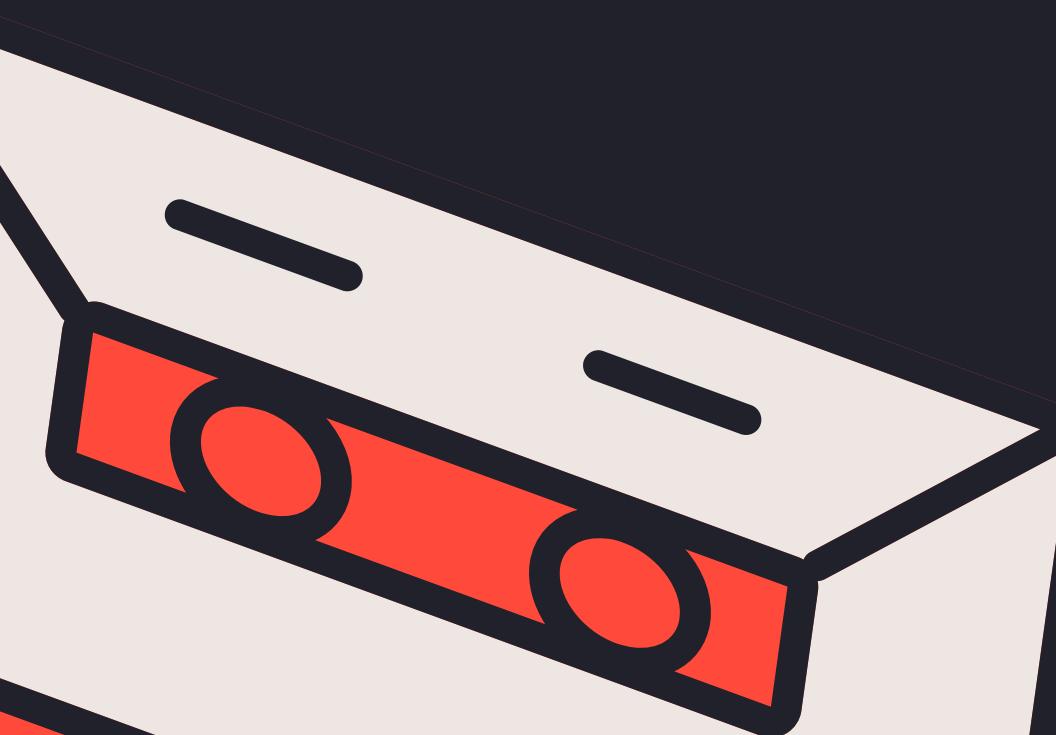
Contient tous les fichiers publiques non important tels que les CSS ou encore les images

CONTAINERS

Les "containers" peuvent être assimilés à des éléments parents d'autres composants dans une application React. Ils servent de pont entre les composants normaux, qui rendent l'interface utilisateur et la logique qui rend les composants de l'interface utilisateur interactifs et dynamiques

ORGANISATION D'UNE PAGE REACT !

**(CAS D'UN
COMPONENTS)**



1

LES PREMIERS LIGNES

Sur ces dernières vous retrouverez essentiellement les imports des fonctions react ou autre tel Bootstrap ou encore les components

2

POUR COMMENCER ...

Ouverture de la classe de l'objet en question, puis assignation de variables utilisables dans le reste de la classe ainsi que des fonctions.

3

ENSUITE ...

Après avoir ouvert la classe, avoir fait les fonctions et constantes utiles, on ouvre le return() et on commence à écrire du HTML pour le rendu du component, possibilité d'appeler les fonctions ou autre !

4

ET ENFIN ...

Pour terminer on ferme le return et la fonction, et le tour est joué !

Cas concret !

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

const ButtonStyle = {

    display:'block',
    width:'100px',
    border:'none',
    padding:'7px',
    backgroundColor:'none',
}

export default function QuestionList() {
    const [data, setData] = useState([]);
    const getData=()=>{
        fetch('/audiovisuel/resources/questions/list')
        .then(function(response){
            console.log(response)
            return response.json();
        })
        .then(function(myJson) {
            console.log(myJson);
            setData(myJson)
        });
    }
    useEffect(()=>{
        getData()
    },[])
}

function RemoveUser(Id){
    var dialog = window.confirm("Supprimer la question : "+Id);
    if (dialog) {
        alert('Question supprimé !')
        fetch('/audiovisuel/resources/questions/remove/'+Id)
        window.location.reload(false)
    }
    else {
        alert('Question non supprimé !')
    }
}
```

Cas concret ! (suite)

```
return (
<>
<div style={{ padding: "20px", backgroundColor:"white", margin:"20px",borderRadius:"25px" }}>
  <table className='table table-striped'>
    <thead>
      <tr>
        <th> #ID </th>
        <th> Question </th>
        <th> Reponse 1 </th>
        <th> Reponse 2 </th>
        <th> Reponse 3 </th>
        <th> Reponse 4 </th>
        <th> Theme </th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>
          {data && data.length>0 && data.map((item)=><button value={item.idQuestions} onClick={function(){RemoveUser(item.idQuestions)}} style={ButtonStyle}>{item.idQuestions}</button>)}
        </td>
        <td>{data && data.length>0 && data.map((item)=><p>{item.questions}</p>)})</td>
        <td>{data && data.length>0 && data.map((item)=><p>{item.reponse1}</p>)})</td>
        <td>{data && data.length>0 && data.map((item)=><p>{item.reponse2}</p>)})</td>
        <td>{data && data.length>0 && data.map((item)=><p>{item.reponse3}</p>)})</td>
        <td>{data && data.length>0 && data.map((item)=><p>{item.reponse4}</p>)})</td>
        <td>{data && data.length>0 && data.map((item)=><p>{item.theme}</p>)})</td>
      </tr>
    </tbody>
  </table>
</div>
</>
)
}
```

ORGANISATION D'UNE PAGE REACT !

(CAS D'UN CONTAINER)

PLUS SIMPLE !

Ici, cela est plus simple, mais surtout plus lisible. En effet ici pas question d'écrire des tonnes de choses, une fois la classe ouverte, on ne fait qu'importer les components que l'on veut utiliser, ce qui réduit drastiquement la quantité de code écrite dans le fichier !

```
import React, { useState, useEffect } from 'react';
import Navbar1 from '../../Components/Template/Navbar1/Navbar1'
import Background from '../../Components/Template/Background/Background'
import AdminList from '../../Components/Admin/UserList/UserList';
import { useCookies } from 'react-cookie'
import { Link } from 'react-router-dom'

const h1Style = {
  textAlign: "center",
  color: "white",
  zIndex: "2",
  paddingTop: "25px",
  textDecoration: "underline 2px white"
}

export default function Admin() {
  const [cookies] = useCookies(['role']);

  if (cookies.role === "formateur") {
    return (
      <>
        <Background />
        <Navbar1 />

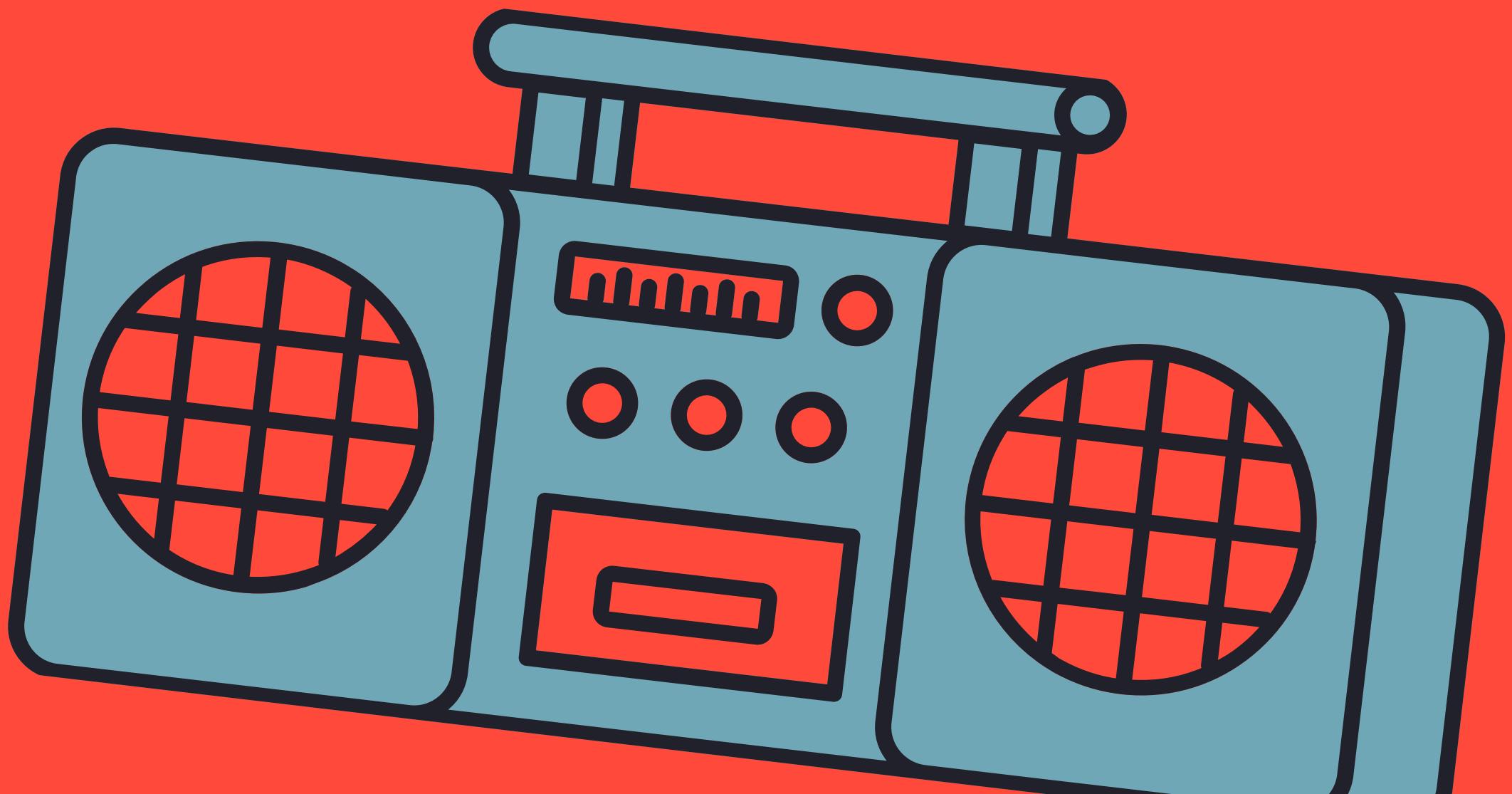
        <h1 style={h1Style}> Votre panel ADMIN </h1>

        <AdminList />
      </>
    )
  } else {
    return (
      <>
        <Background />
        <Navbar1 />

        <div style={{ textAlign: "center", marginTop: "15%" }}>
          <h1 style={{ color: "red", fontSize: 100 }}>Accès interdit !</h1>
          <h3 style={{ color: 'white' }}>Pour accéder à cette page, vous devez vous connecter avec un compte formateur!</h3>
          <p><Link to="/Connexion" style={{ color: 'red', textDecoration: "none" }}>Connexion...</Link></p>
        </div>
      </>
    )
  }
}
```

RETOUR SUR LE BACK !

CE QUI EST DÉJÀ IMPLÉMENTÉ !



USER RESOURCE

IMPLÉMENTATION DU REGISTER ET DU LOGIN

```
@GET  
@Consumes("application/x-www-form-urlencoded")  
@Path("login/{mail}/{password}")  
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
public String find(@PathParam("mail") String mail, @PathParam("password") String password) {  
    Query q = em.createNamedQuery("User.findByMailUser");  
  
    User user;  
    user = (User) q.setParameter("mailUser", mail).getResultList().stream().findFirst().orElse(null);  
  
    if (user==null) { // cas mail non trouvé  
        JsonObject value = Json.createObjectBuilder()  
.add("1", "mail not found")  
.build();  
  
        return value.toString();  
    } else if(!user.getPasswordUser().equals(password)) {  
        JsonObject value = Json.createObjectBuilder()  
.add("2", "mail or password incorrect")  
.build();  
  
        return value.toString();  
    } else {  
        JsonObject value = Json.createObjectBuilder()  
.add("3", user.getRoleUser())  
.build();  
  
        return value.toString();  
    }  
}
```

L'implémentation de ces deux fonctionnalités ont été possibles grâce aux différentes méthodes présentes dans l'entité User.

REGISTER :

Nous avons utilisé la méthode `User.findByMailUser` afin de créer un `query`. En ajoutant le mail au `setParameter`, et en ajoutant un `ResultList`, cela a permis de rechercher un User dans la liste d'utilisateurs de la BDD grâce à son mail.

Nous effectuons ensuite une vérification de l'existence du compte dans la BDD, et utilisons `em.persist` pour créer un nouveau User dans la BDD. Sinon s'il n'existe pas, nous renvoyons un message d'erreur.

LOGIN :

Nous utilisons le même procédé que pour le `register`, sauf que cette fois au lieu de créer un User, nous allons simplement le rechercher dans la BDD et vérifier si les données reçues correspondent bien à un User existant. Sinon, nous renvoyons un message d'erreur suivant l'erreur (mail/password incorrect ou mail not found).

ADMIN RESOURCE

IMPLÉMENTATION DE LA LIST USER ET DU REMOVE

```
@GET  
@Consumes("application/x-www-form-urlencoded")  
@Path("remove/{id}")  
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
public String find(@PathParam("id") Integer id) {  
    Query q = em.createNamedQuery("User.findByIdUser");  
    User u = (User) q.setParameter("idUser", id).getResultList().stream().findFirst().orElse(null);  
  
    if (u==null) {  
        JsonObject value = Json.createObjectBuilder()  
.add("1", "user not found")  
.build();  
  
        return value.toString();  
    }  
    else {  
        em.remove(u);  
        JsonObject value = Json.createObjectBuilder()  
.add("2", "user deleted")  
.build();  
  
        return value.toString();  
    }  
}
```

Nous utilisons toujours les méthodes présentes dans l'entité User.

LIST USER:

Dans un public List, nous allons utiliser la méthode **User.findAll** afin de rechercher tous les User présents dans la BDD.

Puis nous utilisons ce **query** et ajoutons un **getResultSet** qui nous permettra d'obtenir la liste complète de ces User.

REMOVE :

Afin de rechercher un utilisateur par son Id dans la BDD, nous utilisons le même procédé que le **register** de **UserResource**, cependant nous remplaçons le **findByIdUser** par **findByIdUser**. Ensuite s'en suit la même structure de code que le **register**, mais en remplaçant le **em.persist** par un **em.remove**.

Nous renvoyons également un message d'erreur si l'utilisateur n'existe pas, sinon un message "user deleted".

QUESTION RESOURCE

IMPLÉMENTATION DE LA LISTE, DE L'AJOUT, ET DE LA SUPPRESSION D'UNE QUESTION

Cette fois-ci, nous utiliserons les méthodes de l'entité Questions.

Le principe des trois fonctionnalités restent la même que les fonctionnalités de User & Admin Resource, cependant la réception des données n'est pas la même.

En effet, là où avant nous utilisions l'url pour transmettre les données, nous utilisons désormais le body.

Quelle différence ?

La grande différence est que les pathParam du public String deviennent des FormParam.

```
@POST
@Consumes("application/x-www-form-urlencoded")
@Path("add")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public String find(@FormParam("Questions") String Questions, @FormParam("Reponse1") String Reponse1,
    @FormParam("Reponse2") String Reponse2, @FormParam("Reponse3") String Reponse3,
    @FormParam("Reponse4") String Reponse4, @FormParam("Theme") String Theme) {
    Query q = em.createNamedQuery("Questions.findByQuestions");

    Questions questions;
    questions = (Questions) q.setParameter("questions", Questions).getResultList().stream().findFirst().orElse(null);

    em.persist(new Questions(1,Questions,Reponse1,Reponse2,Reponse3,Reponse4,Theme));

    JSONObject value = Json.createObjectBuilder()
        .add("1", "question successfully created")
        .build();

    return value.toString();
}
```

LES LIMITES ET PERSPECTIVES !

1

LIMITES

Malheureusement par manque de temps nous n'avons pas réussi à finir le projet à temps. Cela signifie donc qu'il manque pas mal de fonctionnalités comme les questionnaires et les graphes...

2

PERSPECTIVES

Nos perspectives seraient de pouvoir terminer le projet un jour, malheureusement le stage se profile et ne nous laissera pas le temps de s'occuper du projet de sitôt, mais peut-être quand nous serons en vacances !



Conclusion !

REACT

Ce projet fut une bonne occasion de découvrir quelque chose dont j'ai toujours entendu parlé dans le monde du web : le React. React est un outil puissant bien que sous utilisé dans ce projet par sa découverte récente !

Corentin.W

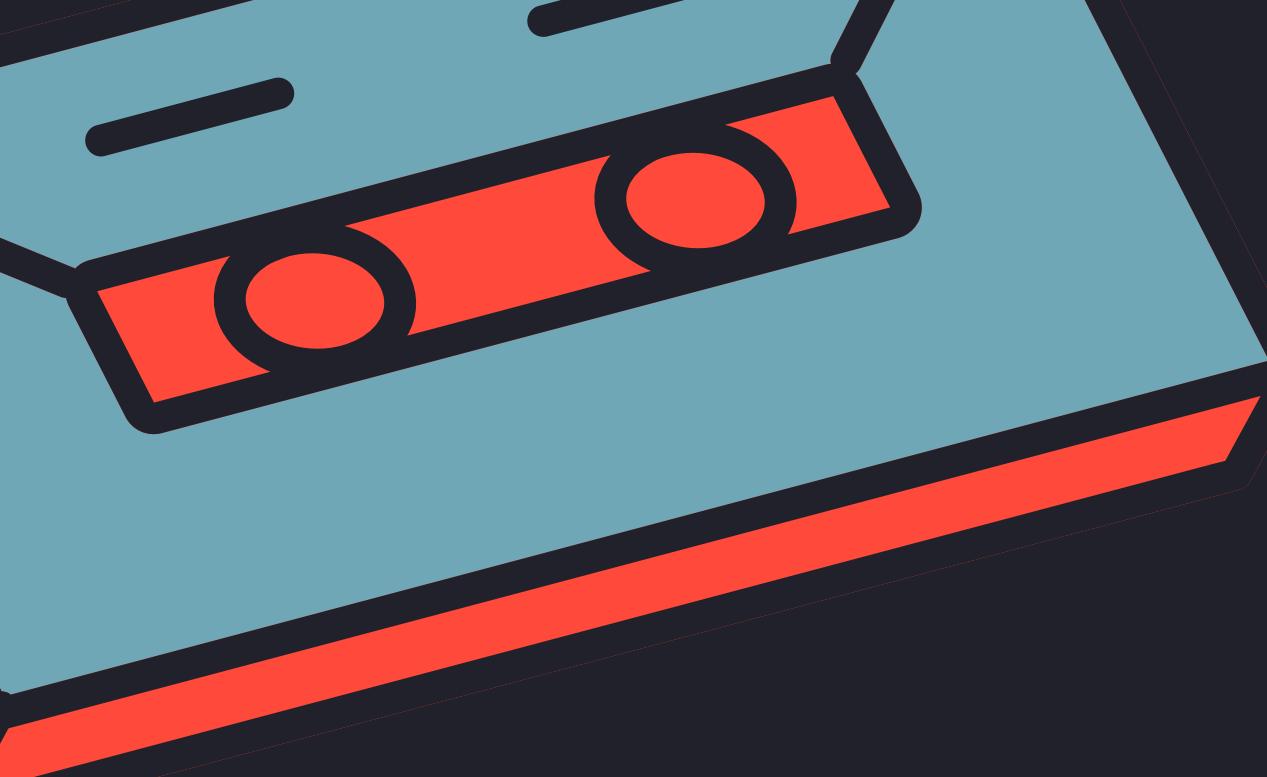
JAKARTA

Ce fût une expérience non triviale. En effet, le fait de rencontrer des erreurs à de nombreuses occasions, parfois même pour rien, m'a un peu rebuter de Jakarta. Cependant, cela reste une découverte malgré tout, et je pense qu'avec plus de temps nous aurions pu créer plus de fonctionnalités, et j'aurais pu améliorer mon affinité à l'égard de ce langage.

Nicolas.S

AUTRE

Ce projet fut aussi l'occasion de travailler en équipe, chose que nous avions peu fait jusque là, ce n'est pas toujours simple, mais on s'arrange toujours. Durant ce projet nous avons aussi utilisé davantage GitHub pour simplifier le versionnage du projet et le partage des fichiers.



Passons à la démonstration !



GITHUB

<https://github.com/insset-l3mn/2022-02-groupe-04>



TWITTER

@WatisseCorentin
@Luffy_s_Gamer