La Plateforme_

Application Expense Report

—

Par Grégory

Sommaire

Présentation personnelle:

Bonjour je m'appelle Grégory Fauvel Après une formation Coding School dans le domaine du développement web à l'école la Plateforme à Marseille avec obtention du diplôme . J'ai reconduit l'aventure pour une deuxième année en concepteur d'application mobile native.

Curieux et passionné, j'ai trouvé une reconversion professionnelle qui me convient parfaitement. Maintenant j'espère en faire mon futur métier.

Présentation de l'équipe:

Le projet "Apinote "a été réalisé par Mohamed Hadjadji pour l' interface Expense Report. Olivier Crozet a travaillé sur tous les composants de l'application .

Ma partie travaillée est celle de l'interface utilisateur.

Présentation de l'Apinote:

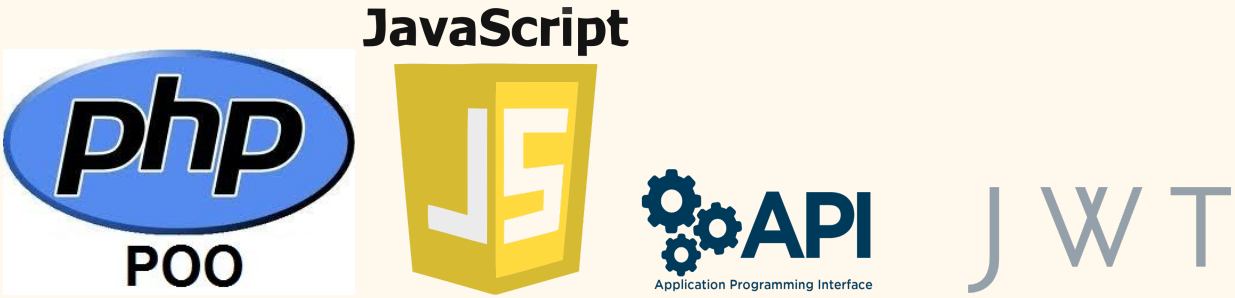
Apinote est une application de note de frais

Pour le développement de la partie présentation du projet Expense report, nous avons opté pour la recherche sur internet sur les différents projets existant pour nous inspirer de la structure et de la conception de l'application .
L'idée est de comprendre la structure et le fonctionnement d'une application de note de frais pour pouvoir évaluer une organisation de travail et surtout la répartition des tâches.
Nous nous sommes aussi inspirés du design de chaque application pour avoir une idée de la conception des pages (ex: Page des utilisateurs, ajouter une note) Il doit y avoir deux parties importantes . Deux interfaces doivent s'afficher par rapport au rang de l'utilisateur dans l'entreprise. Deux structures differentes sont proposées celle de l'employé l'autre le RH.Pour le framework front nous avons choisi React Native qui récupère via la fonction AXIOS qui récupère le chemin de la structure MVC et en décodant le JSON de l'API préalablement codé sur le framework Codeigniter.Pour soigner le design de l'application nous avons utilisé la librairie Native base qui donne desComposants essentiels de l'interface utilisateur multiplateforme pour React Native et Vue Native NativeBase est une bibliothèque de composants d'interface utilisateur gratuite et open source pour React Native permettant de créer des applications mobiles natives pour les plates-formes iOS et Android.Pour voir le rendu de de l'application et l'affichage des bugs nous avons utilisé EXPO.Expo est une plate-forme open source permettant de créer des applications natives universelles pour Android, iOS et le Web avec JavaScriptReact.

Connaissances Informatiques.

Frameworks & Outils

Langages informatiques:



Cahier des charges:

2 interfaces utilisateurs Employés et RH.

Pour la sécurité il faut une connexion avec générateur de token à la connexion.

Application codée avec une API côté Back-end.

Utiliser le Framework React Native pour récupérer en JSON le retour de l'API et l'afficher en Native.

Affichage des informations dans les interfaces faciles d'utilisation et à comprendre.

Seul le RH peut inscrire les employés et créer leur propre compte.

Gestion de l'interface complète côté responsable .

Sécurité: Les Tokens

Le Token permet l'échange sécurisé de jetons entre plusieurs parties. Cette sécurité de l’échange se traduit par la vérification de l'intégrité et de l'authenticité des données

Un JWT est une norme sécurisée et digne de confiance pour l'authentification par jeton. Les JWT vous permettent de signer numériquement des informations (appelées revendications) avec une signature et peuvent être vérifiées ultérieurement avec une clé de signature secrète.

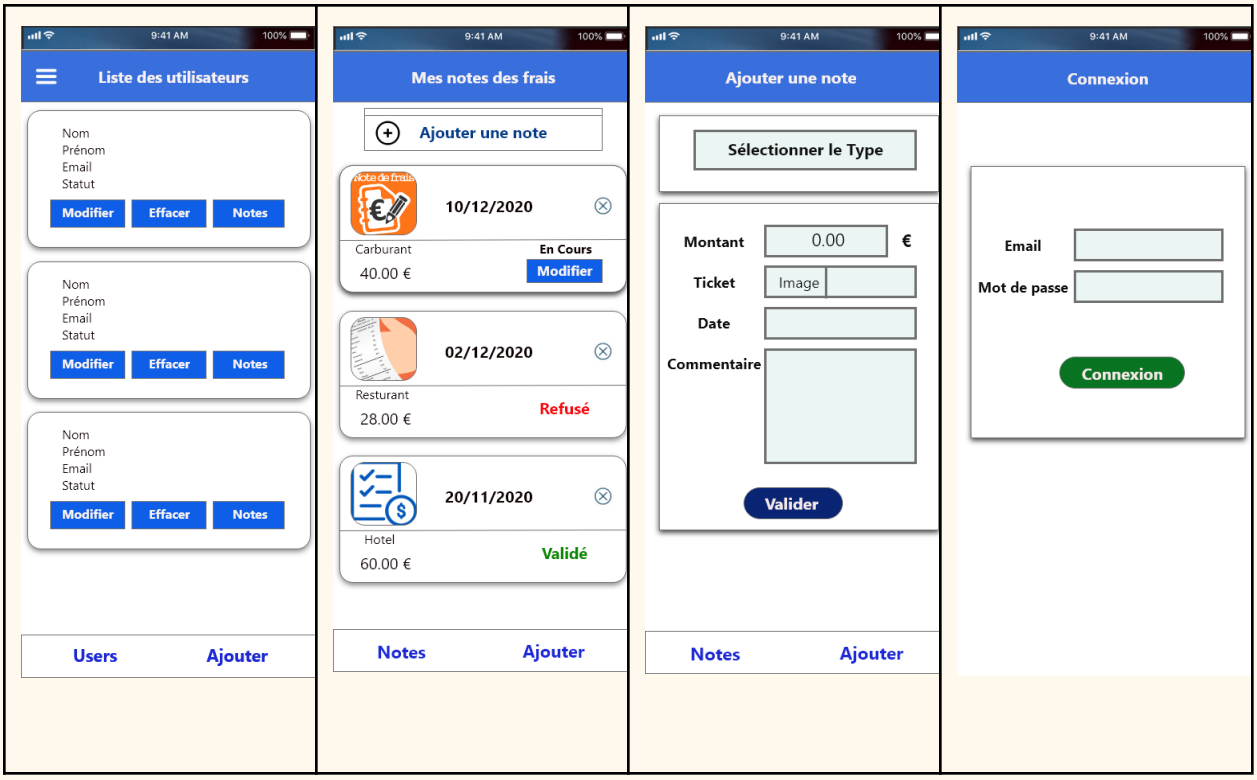
Comment?
L'application ou le client demande une autorisation au serveur d'autorisation. ...
Lorsque l'autorisation est accordée, le serveur d'autorisation renvoie un jeton d'accès à l'application.
L'application utilise le jeton d'accès pour accéder à une ressource protégée (comme une API).

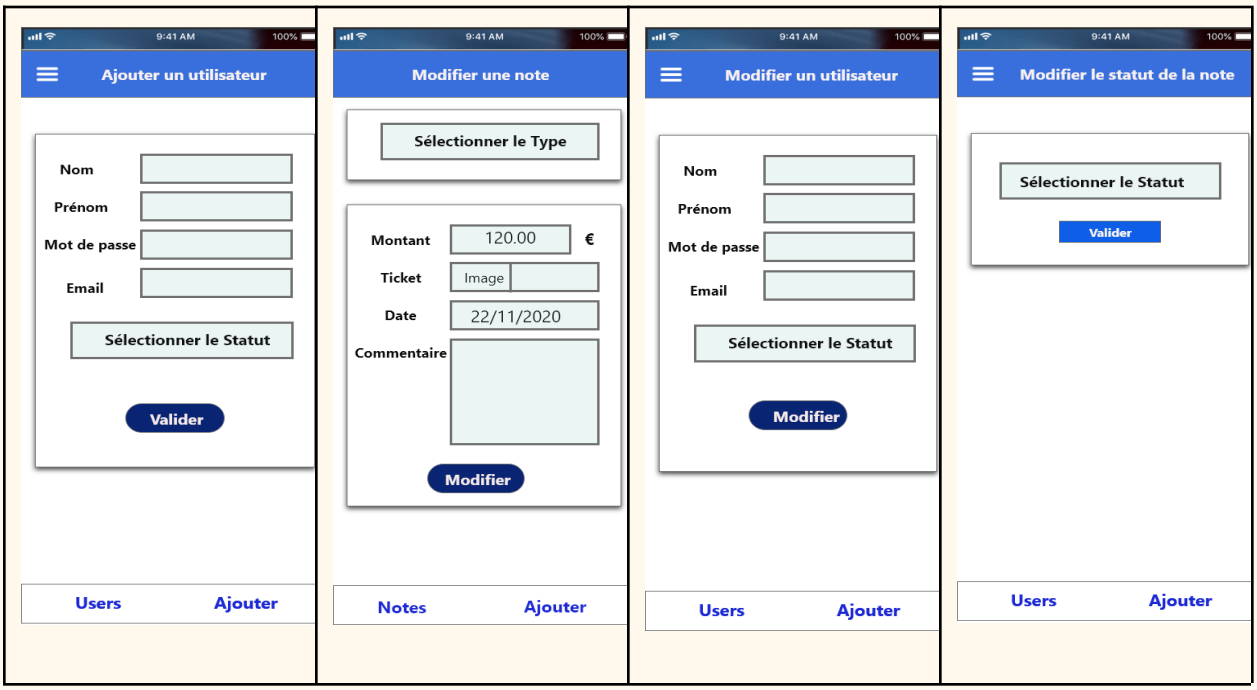
Un fichier jwtToken avec une classe avec public function __construct header('Content-Type: application/json');pour stocker le Token dans le Header
JwtToken avec les fontions suivantes LoginToken qui génèrent le Token puis GetTokenData pour Decode le token et récupérer les informations de la personne connectée.

Maquetter l’application

Adobe XD est un outil de conception d'expérience utilisateur basé sur le vecteur pour les applications Web et les applications mobiles,

Visuel de la maquette Apinote





Pour le maquettage du projet Expense Report nous nous sommes inspirés d'applications ci-dessous qui sont les meilleures sur le marché actuel et les plus simples d'utilisation elles sont déjà existantes sur internet.L'idée est de savoir de savoir comment fonctionne une application de note de frais et qui l'utilise et pourquoi? Nous avons utilisé **Adobe XD** comme logiciel de maquettage qui est simple et fait de belles maquettes .Nous avons ensuite établi les Pages aprés visualisation de la structure de l'application nous avons convenu d'établir les Pages suivantes:

Partie RH:	Partie Employés:
Listes des Employés (User.js)	Listes des note de l'employé connecté (Notes.js)
Création compte Employés (InsertionUser.js)	Création note de frais (InsertionNotes.js)
Modification compte Employés (ModifUser.js)	Modification note de frais (ModifNotes.js)
Suppression Compte Employés (User.js)	Suppression note de frais (Notes.js)
Modificatiion Statut Note de frais (Modif node.js)	Modification statut note (UserNotes.js)

Organisation et répartition du travail .

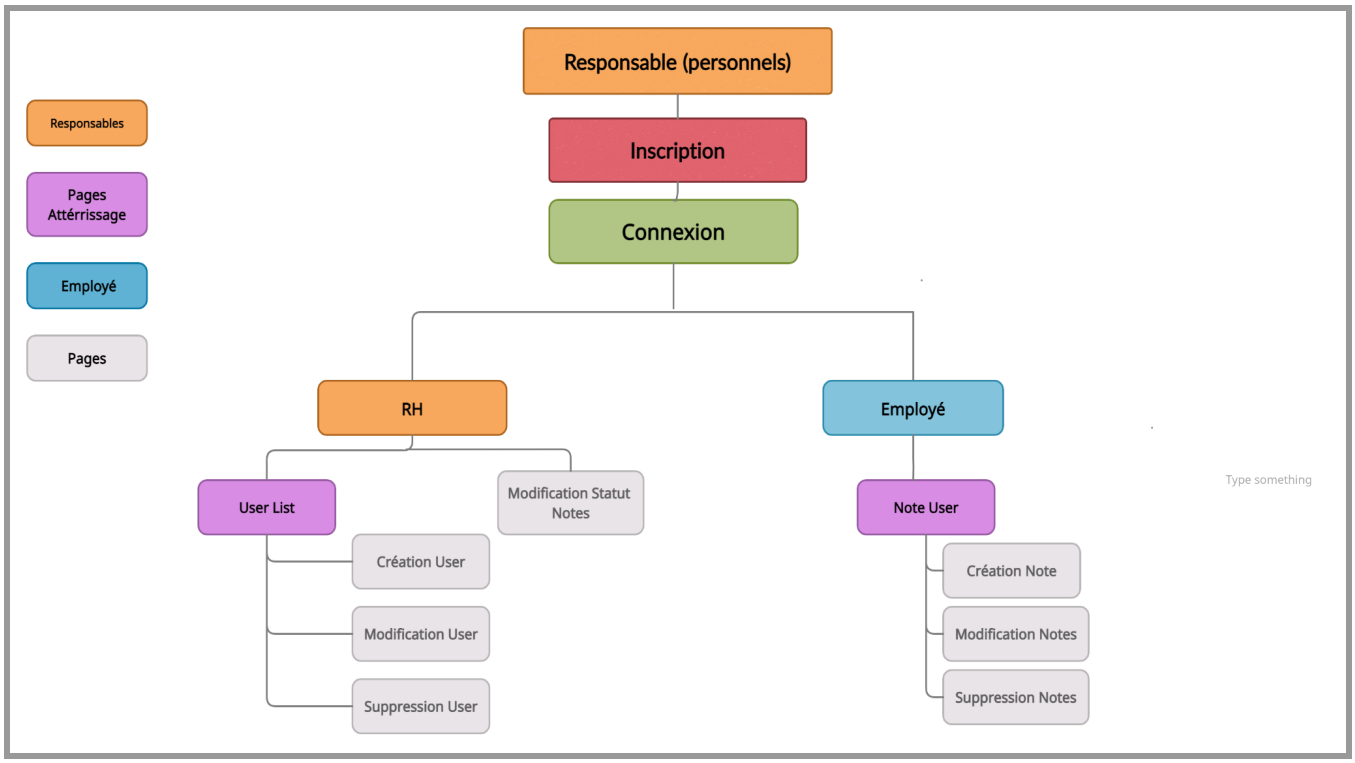
Pour l'organisation de l'environnement de développement du projet Apinote nous avons utilisé Trello qui est une application conçue pour le management des projets. Très efficace, cet outil ergonomique permet également d'être aux commandes de plusieurs projets simultanément. Par ailleurs, il s'agit aussi d'un gestionnaire de tâches qui sert à mieux classer les projets à travers une liste de tâches. Toutes les parties de l'application ont été faites sur papier avant tout code.Nous avons répartit l'application en trois: l'utilisateur,la note de frais , les composants .
Nous avons eu 2 semaines au mois de décembre pour concevoir l'API puis 4 semaines au mois de janvier pour la conception en native.
En coupure avec l'alternance nous avons peaufiné l'application en mettant des deadlines à chaque semaine de formation.
Chaque fin de matinée et de soirée nous notions dans nos parties respectives l'avançée du projet de façon à ce que chacun de l'équipe ait accés au travail de l'autre et de déterminer des dead line pour chaque partie de l'application.
Pour la partie code de l'application nous avons utilisé GIT qui est un outil qui permet de gérer différents projets en les envoyant sur un serveur. Ce dernier est connecté à l'ordinateur d'autres développeurs qui envoient leur code et récupèrent le vôtre. Toute personne qui travaille sur un projet est connectée avec les autres, tout est synchronisé.
Nous Gitons comme on dit dans le jarguon toutes les fins de semaines pour actualiser l'avancée du projet partie code cela nous permet de débugger quand on rejoint les parties travailées chaque fin de semaines

Voici l'arborescence de l'application dans laquelle seront intégrées les différentes fonctionnalitées :

Partie RH⟶ Inscription ⟶ Connexion ⟶ Afficher la liste des employés➢ ↪Panneau d'administration ⟶ Afficher le profil utilisateur ⟶ Modifier les paramètres utilisateur ⟶Effacer l'employé -> Modification statut note (En cours , Accepté Refusé)

Partie employés⟶ Connexion ⟶ Afficher la page des note de l'utilisateur connecté➢ ⟶ Création d'une note ⟶ Modifier les paramètres de la note ⟶ Effacer la note

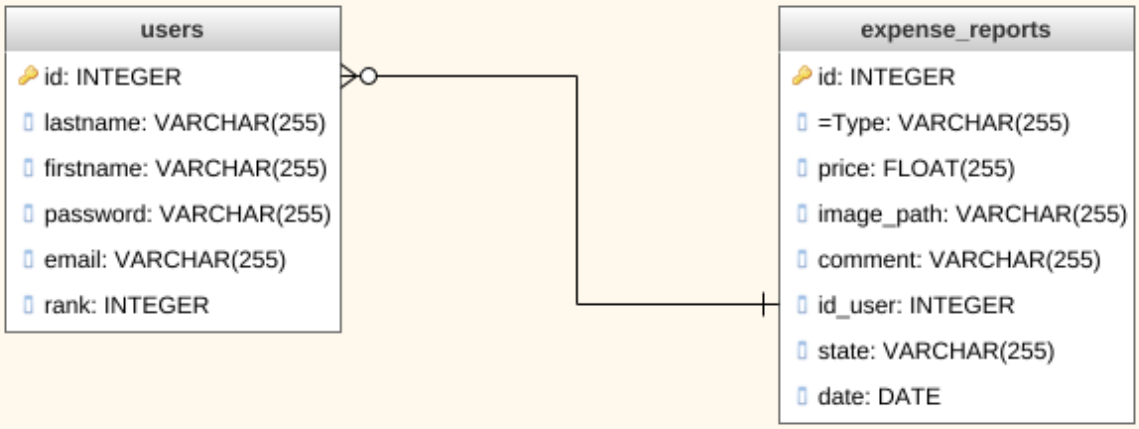
Pour pouvoir rendre une interface mobile en maquette nous avons utilisé le site Creately.



Créer et configurer la base de données:

Pour créer ma base de données j'utilise **MySQL** est un système de gestion de bases de données relationnelle.

Diagramme de base de données nous avons créé **2 tables** avec Genmymodel. Une table users avec les attributs clé primary id ,lastname,firstname,password,email,rank & expense report clé primary id,type,price,image,comment,id_user,state,date.



Visuel BDD avec MySQL:

	Table	Action	Lignes	Type	Interclassement	Taille	Perte
☐	expense_reports	Parcourir Structure Rechercher Insérer Vider Supprimer	7	InnoDB	latin1_swedish_ci	16,0 kio	-
☐	users	Parcourir Structure Rechercher Insérer Vider Supprimer	7	InnoDB	latin1_swedish_ci	16,0 kio	-
	2 tables	Somme	14	InnoDB	latin1_swedish_ci	32,0 kio	0 o

Maintenant, vous devrez configurer votre nom de domaine. Pour ce faire, ouvrez le fichier config.php en procédant comme suit:

```
$config['base_url'] = 'http://localhost/apinote/';
```

CodeIgniter a un fichier de **configuration** qui vous permet de stocker vos valeurs de **connexion à la base de données** (nom d'utilisateur, mot de passe, nom de la base de données, etc.). Le fichier de configuration se trouve dans **apinote/application/Config/Database.php .**

Les paramètres de configuration sont stockés dans une propriété de classe qui est un tableau avec ce prototype :

```
$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'ems',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

Les composants d'accès aux données.

Les fonctions de mes **models** sont dans une classe **Users & Expenses Report** font les appels à la base de données et renvoie les informations aux contrôleurs.

```
function getUser($userId){
    $id=$this->db->where('id',$userId);

    return $user = $this->db->get('users')->row_array();//select 1 Users

}

function updateUser($userId,$formArray) {
    $this->db->where('id',$userId);
    $this->db->update('users',$formArray);//update Users where id =
}

function deleteUser($userId) {
    $this->db->where('id',$userId);
    $this->db->delete('users');//delete Users where id =
}
```

```
function all ()
{
 return $users = $this->db->get('users')->result_array();//select tous les Users

}
```

```
function create($formArray)

{
   $this->db->insert('users',$formArray);//insertion User bdd
}
```

Partie Back-end de l’application.

Pour la partie **Back End** de l’application nous avons codé une API avec le **Framework Codeigniter 3 et Visual Code** .Nous avons choisi ce framework car il est codé en PHP et il est open-source .Il offre un maximum de performance et de flexibilité dans une structure de programmation la plus simple possible.Ce MVC se nomme **Apinote.**

Le MVC est le concept central de la programmation orientée objet (POO). ... En **programmation orientée objet**, un objet est créé à partir d'un modèle appelé classe ou prototype, dont il hérite les comportements et les caractéristiques.
La **POO**, ça permet de donner un semblant de vie comportement à des entités de données, de variables, de valeurs... qu'on appellera du coup des objets.
Avantages : modularité, abstraction, productivité et réutilisabilité, sûreté, encapsulation.

La **structure MVC** permet une **conception logicielle flexible**, dans laquelle les modules de programme peuvent tous être échangés, révisés et réutilisés en un minimum d'efforts. Les modifications apportées à un composant n'ont en principe pas d'effet sur le code source des autres composants.

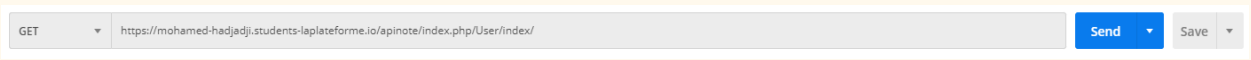
Nous avons donc **2 contrôlleurs principaux avec deux classes** qui gère la structure de l'application un pour l'**utilisateur** connecté l'autre pour la **note de frais.**
Les classes dans les contrôleurs font appellent à leurs modèles qui eux font les requêtes et appels à la base de données.
Nous avons élaboré la **méthode CRUD (create,read,update,delete)** pour les **tables User et Expense report** afin d'avoir toutes les requêtes les plus demandées.

JWT

Pour la sécurité, la création d'un Token est **généré à la connexion de l'utilisateur** par la librairie **JWT Token qui est une librairie de générateur de tokens** et se détruit en se déconnectant .
Je **Décode le Token** quand l'**utilisateur se connecte** pour récupérer **les informations du rang** de l'utilisateur et le rediriger à son interface de RH ou d'employé .

```
const MainPage = (props) => {

  const [doRedirect, setDoRedirect] = useState(0);

  useEffect(() => {
    const fetchData = async() => {//Récupération du Token
      if (doRedirect === 0) {
        await SecureStore.getItemAsync('token').then((val) => {
          var decoded = jwt_decode(val);  //Décoder le token
          if (decoded.rank == "RH") {     //Condition du Rank
            setDoRedirect(1)
          } else {
            setDoRedirect(2)
          }
        })
      }
    }
    fetchData();
  })


  if (doRedirect === 0) return <></>;
  if (doRedirect === 1) return <AppNav />;//Condition RanK == 1 ->RH -> Tab RH
  if (doRedirect === 2) return <AppBout />;//Condition RanK == 2 ->Salarié->Tab Salarié

}

export default MainPage;
```

Les tests unitaires ont été fait avec **POSTMAN** logiciel qui permet de tester notre code en incluant en entrant l'URL de l'API



Résultat rendu en JSON

le chemin de notre **API .** L' **API est hébergée sur plesk** qui est une interface de gestion de serveur payante.

Application Programming Interface

L' Api me sert à exposer localement ou sur le web un catalogue de fonctionnalités au service d'un programme. Le but étant de pouvoir faire communiquer des systèmes entre eux pour échanger des données

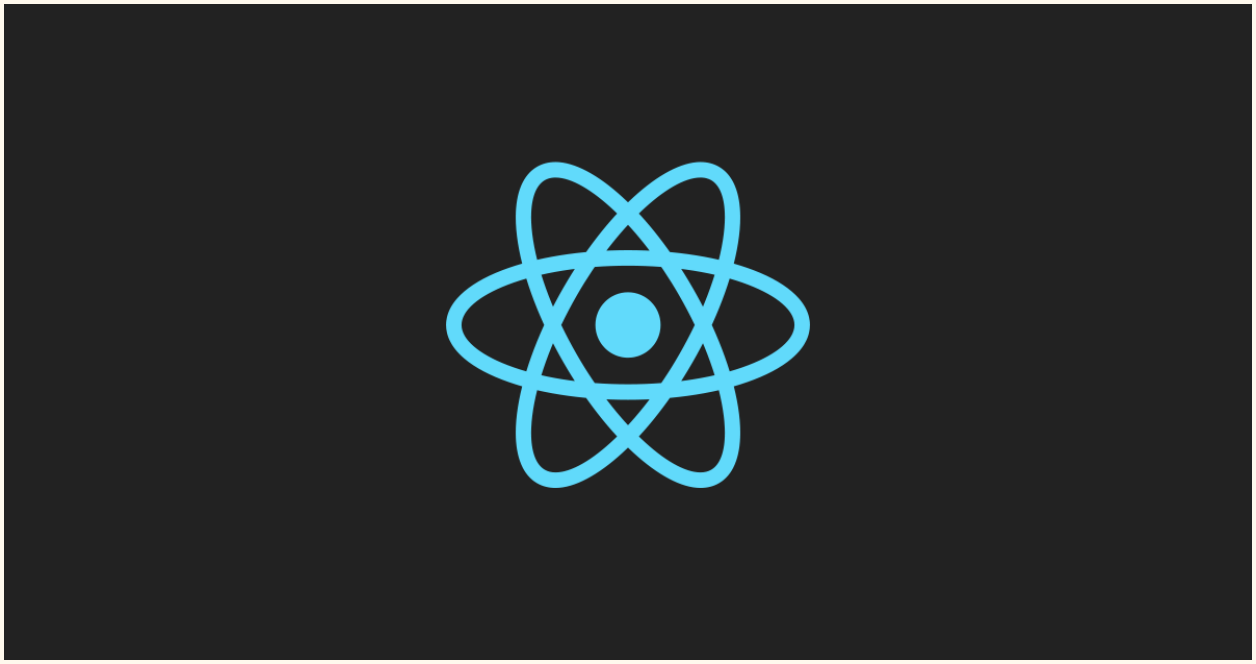
Une structure **Model Contrôleur** a été codée et **encodée en JSON,** elle possède **2 classes Users et expense report** dans les **contrôleurs** qui font appel à nos **Modèles Users et Expense Report** pour les requêtes à la bdd et enfin la récupération des données via **AXIOS** codée dans React Native . Axios est une bibliothèque JavaScript fonctionnant comme un client HTTP. Elle permet de **communiquer avec des API en utilisant des requêtes**

Tableaux CRUD représentatifs

HTTP	Fontions SQL	Descriptions
GET	Index	Liste des salariés
POST	create	Création de salarié
PUT	edit	Modifier un salarié
DELETE	Delete	Supprimer un Salarié

GET	Index_note	Lecture d'une note de frais
POST	create_note	Écrire une note de frais
PUT	edit_note	Mettre à jour une note de frais selon le statut
DELETE	Delete_note	Supprimer une note de frais

Partie front end :



Pour le framework front nous avons choisi React Native qui récupère via la fonction AXIOS qui récupère le chemin de la structure MVC et en décodant le JSON de l'API préalablement codé sur le framework Codeigniter.

Pour soigner le design de l'application nous avons utilisé la librairie Native base qui donne des design propre et soigné . Les Composants essentiels de l'interface utilisateur multiplateforme pour React Native et Vue Native NativeBase est une bibliothèque de composants d'interface utilisateur gratuite et open source pour React Native permettant de créer des applications mobiles natives pour les plates-formes iOS et Android.

NativeBase est une bibliothèque de composants qui permet aux développeurs de créer des systèmes de conception universels. Il est construit sur React Native, vous permettant de développer des applications pour Android, iOS et le Web.

Pour voir le rendu de de l'application et l'affichage des bugs nous avons utilisé EXPO.Expo est une plate-forme open source permettant de créer des applications natives universelles pour Android, iOS et le Web avec JavaScript et React.

Appel Axios API (User)

Liste des utilisateurs:
Dans cette fonction AXIOS

```
   async componentDidMount() {

axios.get('https://mohamed-hadjadji.students-laplateforme.io/apinote/index
.php/User/index')
      .then(response => {
        this.setState(prevState => ({
            UserList: prevState.UserList = response.data
        }));
      });
  }
```

Création des Utilisateurs:

```
 axios

.post("https://mohamed-hadjadji.students-laplateforme.io/apinote/index.php
/User/create", bodyFormData)
        .then((response) => {
        })
        .catch((err) => {
          throw err;
        });
    };
```

Modification des Utilisateurs:

```
 axios

.post("https://mohamed-hadjadji.students-laplateforme.io/apinote/index.php
/User/edit/"+this.props.route.params.id,bodyFormData)
        .then(() => {

        })
        .catch((err) => {
          throw err;
        });
    };
```

Suppression des Utilisateurs:

```
  Delete = (id) => {
   axios

.delete("https://mohamed-hadjadji.students-laplateforme.io/apinote/index.p
hp/User/delete/"+id )
      .then(() => {

      })
      .catch((err) => {
        throw err;
      });
  };
```

Navigation Tab et Stack

Les applications mobiles sont rarement constituées d'un seul écran. La gestion de la présentation et de la transition entre plusieurs écrans est généralement gérée par ce que l'on appelle un navigateur.

Nous avons donc créé un tab et stack pour le RH & expense report dans un fichier que j'ai appelé **navi.js** c'est ce fichier qui gère la navigation complète de l'utilisateur.

2 Tab pour Ajout User et Liste des Users.(Tabulation visible sur l'application)
4 Stack (Appel des pages RH : Connexion,ModificationUser,Note de l'user,Modification statut Notes)

```
const Tab = createBottomTabNavigator();

function Navi() {//Tab et Stack-> RH
  return (
    <Tab.Navigator>
      <Tab.Screen name="Users" component={UserList}  />
      <Tab.Screen name="Ajouter User" component={Inputs} />
    </Tab.Navigator>
  );
}

const Stack = createStackNavigator();
export function AppNav() {
  return (
    <NavigationContainer independent={true}>
      <Stack.Navigator >
      <Stack.Screen name="Application Notes de Frais" component={Navi} />
      <Stack.Screen name="HomePage" component={HomePage} />
      <Stack.Screen name="Modif" component={Modif} />
      <Stack.Screen name="UserNotes" component={UserNotes} />
      <Stack.Screen name="EditRHnote" component={EditRHnote} />

      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

2 Tab pour Ajout Notes et Liste des note de l'utilisateur connecté(Tabulation visible sur l'application)
4 Stack (Appel des pages RH : Connexion,ModificationUser,Note de l'user,Modification statut Notes).

```
const TabSalarie = createBottomTabNavigator();

export function Navi_salarie() {//Tab et Stack Salarié

  return (

      <TabSalarie.Navigator>
        <TabSalarie.Screen name="Notes" component={Notes}/>
        <TabSalarie.Screen name="InsertionNote" component={InsertionNote}/>
      </TabSalarie.Navigator>

  );
}

  const StackSalarie = createStackNavigator();

 export function AppBout() {
    return (
      <NavigationContainer independent={true}>
        <StackSalarie.Navigator >
        <StackSalarie.Screen name="Mes Notes de Frais" component={Navi_salarie} />
        <StackSalarie.Screen name="Modifier La Note" component={ModifNote}/>
        </StackSalarie.Navigator>
      </NavigationContainer>
    );
  }
```

Grâce à ces deux Tab et Stack j'ai screen les pages des parties concernées pour chaque rank d'utilisateur puis en décodant le Token je récupère les informations dont le rank de l'employé pour pouvoir enfin rediriger l'utilisateur à son interface personnelle .

Dans le fichier Navi.js j'ai créé une redirection Mainpages/ Homepage qui est la connexion de l'application.

```
//Stack redirection homepage/mainpage
export function NavHome() {
  return (
    <NavigationContainer independent={true}>
      <Stack.Navigator headerMode={false}>
        <Stack.Screen name="HomePage" component={HomePage} />
        <Stack.Screen name="MainPage" component={MainPage} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

J'ai créé une page Mainpage qui fait la liaison avec mon fichier App.js pour la redirection des interfaces c'est elle qui décode le Token récupère les infos et redirige avec les conditions suivantes:

```
const MainPage = (props) => {

  const [doRedirect, setDoRedirect] = useState(0);

  useEffect(() => {
    const fetchData = async() => {//Récupération du Token
      if (doRedirect === 0) {
        await SecureStore.getItemAsync('token').then((val) => {
          var decoded = jwt_decode(val);  //Décoder le token
          if (decoded.rank == "RH") {     //Condition du Rank
            setDoRedirect(1)
          } else {
            setDoRedirect(2)
          }
        })
      }
    }
    fetchData();
  })


  if (doRedirect === 0) return <></>;
  if (doRedirect === 1) return <AppNav />;//Condition RanK == 1 ->RH -> Tab RH
  if (doRedirect === 2) return <AppBout />;//Condition RanK == 2 ->Salarié->Tab Salarié

}

export default MainPage;
```

Les Components:

Un composant React fonctionnel est tout simplement une **fonction qui va retourner un bloc de JSX**. Cette manière de créer un composant est la plus efficace et la plus simple.

Les components React Native, plus couramment appelés components, *tout court*, correspondent aux éléments graphiques simples que l'on retrouve sur les applications mobiles natives : **Text, Button, Image, ScrollView, View, WebView.**

Exemple: PickerInput.

Dans cet exemple j'utilise le Pickerinput pour avoir le choisx du rank.

```
<Item picker >
  <Picker
    selectedValue={this.props.selectedValue}
    prompt="Rank"
    onValueChange={this.props.onValueChange}
    mode="dropdown"
    onChange= {this.props.onValueChange}
  >
    <Picker.Item label="RH" value="RH" />
    <Picker.Item label="Salarié" value="Salarié" />
  </Picker>
</Item>
```

Ces **components** existent déjà et sont mis à disposition par React Native. La liste est disponible sur la documentation des components React Native. Nous allons construire nos vues avec ces components.
Pour notre application il nous faut une liste de User pour cela il me faut une Flatlist.

Dans cette Flatlist j'ai mis un bloc **Card** pour séparer les utilisateurs et dans les card j'ai mis des **Button** pour récupérer les notes créer, modifier ou supprimer le compte qui récupère le props.navigation pour naviguer de page en page .

```
<View>
Card>
 <CardItem>
    <View style={{margin: 5, flex: 1, flexDirection: 'column', justifyContent: 'center'}}>
     <View>
        <Text>Nom: {item.lastname}</Text>
        <Text>Prénom: {item.firstname}</Text>
        <Text>Email: {item.email}</Text>
        <Text>Statut:{item.rank}</Text>
     </View>
      <View style={{margin: 5, flex: 1, flexDirection: 'row', justifyContent: 'center'}}>
        <View style={{margin: 2, flex: 1, flexDirection: 'row', justifyContent: 'center'}}>
        <Button onPress={createTwoButtonAlert}>
        <Text>Effacer</Text>
        </Button>
        </View>
        <View style={{margin: 2, flex: 1, flexDirection: 'row', justifyContent: 'center'}}>
        <Button onPress={() =>this.props.navigation.navigate('Modif',{id:item.id})} bordered>
        <Text>Modifier</Text>
        </Button>
        </View>
        <View style={{margin: 2, flex: 1, flexDirection: 'row', justifyContent: 'center'}}>
        <Button onPress={() =>this.props.navigation.navigate('UserNotes',{id:item.id})} bordered>
        <Text>Notes</Text>
        </Button>
        </View>
      </View>
     </View>
  </CardItem>
</Card>
iew>
```

Une **Flatlist** est un composant d'affichage de liste suivants ne restituent que les éléments actuellement affichés à l'écran. Cela en fait un choix performant pour afficher de longues listes de données.
Item Separator Component -> View .
data -> le JSON de l'API
keyExtractor -> Utilisé pour extraire une clé unique pour un élément donné à l'index spécifié. La clé est utilisée pour la mise en cache et comme clé de réaction pour suivre la réorganisation des articles. L'extracteur par défaut vérifie item.key, puis item.id, puis revient à l'utilisation de l'index, comme le fait React.

```
<FlatList
ItemSeparatorComponent={() =>< View/>}
data={jsonList}
keyExtractor={(item) =>item.id.toString()}
renderItem= {({ item }) => {
```