

Dossier de projet

TITRE PROFESSIONNEL
DÉVELOPPEUR WEB ET WEB MOBILE

Développement d'un outil d'audit du code
généré par WeWeb,
un outil nocode



Kali
ezacae

Table des matières

I. PRÉSENTATION	1
II. LISTE DES COMPÉTENCES DU RÉFÉRENTIEL COUVERTES PAR LE PROJET	2
III. PRÉSENTATION DE L'ENTREPRISE	3
IV. PRÉSENTATION DU PROJET - CAHIER DES CHARGES	4
V. PRÉSENTATION DIFFÉRENTES ITÉRATIONS DU PROJET	5
Itération 1 : Développement Initial avec Coda et Automatisation	5
Itération 2 : Automatisation via CI/CD dans GitLab	6
Itération 3 : Passage à une Solution Commercialisable	6
Itération 4 : Vue.js & Supabase (Développement Personnel)	7
Itération 5 : WeWeb & Xano (Développement en cours - en Entreprise)	7
Itération 6 : Scalabilité avec Vue.js & Kubernetes (à venir)	7
VI. PRÉSENTATION DU PROJET	8
VII. RÉALISATION - CONCEPTION TECHNIQUE	10
VIII. RÉALISATION	18

I. PRÉSENTATION

Je m'appelle Jeanne, j'ai 37 ans, et je suis développeuse en reconversion, un chemin que j'ai décidé d'emprunter après une carrière dans la danse contemporaine. La pandémie de Covid-19 a bouleversé ma vie, mettant le monde de la danse en suspens, et m'a offert une opportunité inattendue de réflexion. C'est à ce moment-là que j'ai entrepris un bilan de compétences. Ce bilan a révélé une curiosité pour le développement, m'incitant à explorer cette nouvelle voie.

Ma première étape dans ce voyage a été la découverte du no-code, une approche qui m'a permis de m'initier à la création d'applications sans avoir besoin de coder. J'ai commencé par une initiation au code grâce au no-code avec les Descodeuses à Paris en 2022. Cette initiation m'a donné la confiance nécessaire pour aller plus loin et approfondir mes compétences.

C'est ainsi que j'ai rejoint Alegria.academy, où j'ai suivi une formation en programmation visuelle. Durant cette période, j'ai perfectionné mes compétences en utilisant des plateformes comme Bubble.io, Airtable, Webflow, WeWeb, et PowerApps. J'y ai exploré les possibilités du no-code tout en commençant à toucher au code traditionnel.

En parallèle de ma formation chez Alegria, j'ai eu l'opportunité de débiter ma première année d'alternance avec Ezacae, une entreprise spécialisée dans le développement d'applications en utilisant des technologies low-code et no-code. Chez Ezacae, je continue d'approfondir ma maîtrise des outils no-code tout en élargissant mes connaissances en développement web général.

Au fur et à mesure que je progresse dans cette nouvelle voie, mon désir d'aller vers le code traditionnel s'est affirmé. Je ne veux pas être simplement une "no-codeuse" qui connaît un peu le code, mais plutôt une développeuse à part entière, qui utilise les outils no-code et low-code de manière stratégique. Mon parcours continue de me pousser à intégrer de plus en plus de code dans mes projets, avec l'objectif de devenir une développeuse complète, capable de naviguer avec aisance entre le no-code, le low-code, et le développement traditionnel.

C'est pourquoi j'ai intégré le CEFIM en 2023, tout en poursuivant mon alternance chez Ezacae. Je souhaite continuer à me perfectionner en conception et développement d'applications, apprendre encore, devenir plus performante, et grandir à la fois au sein du CEFIM et d'Ezacae, qui continuent de m'accompagner dans cette passionnante aventure en alternance.

II. LISTE DES COMPÉTENCES DU RÉFÉRENTIEL COUVERTES PAR LE PROJET

Compétences	Présentation orale	Dossier de projet	Dossier professionnel
<i>Développer la partie front-end d'une application web ou web mobile sécurisée</i>			
Installer et configurer son environnement de travail en fonction du projet web ou web mobile			
Maquetter des interfaces utilisateur web ou web mobile			
Réaliser des interfaces utilisateur statiques web ou web mobile			
Développer la partie dynamique des interfaces utilisateur web ou web mobile			
<i>Développer la partie back-end d'une application web ou web mobile sécurisée</i>			
Mettre en place une base de données relationnelle			
Développer des composants d'accès aux données SQL et NoSQL			
Développer des composants métier coté serveur			
Documenter le déploiement d'une application dynamique web ou web mobile			
<i>Compétences transversales</i>			
Communiquer en français et en anglais			
Mettre en oeuvre une démarche de résolution de problème			
Apprendre en continu			

III. PRÉSENTATION DE L'ENTREPRISE

Ezacae est une société spécialisée dans le développement de logiciels, combinant les technologies low-code avec du code traditionnel. L'entreprise est structurée en deux équipes principales : l'une basée à Louviers en Normandie, composée de trois personnes, et l'autre située à Paris dans le 12ème arrondissement, avec quatre personnes, ainsi qu'un membre en télétravail.

L'approche d'Ezacae repose sur l'utilisation de plateformes low-code en complément du développement traditionnel. Cette combinaison permet une flexibilité accrue pour créer des applications sur mesure, adaptées aux besoins spécifiques de différents types de clients, qu'il s'agisse de startups, de PME, ou de grandes entreprises.

Les outils low-code utilisés par Ezacae incluent notamment :

- **WeWeb** : Un constructeur de sites web sans code.
- **Xano** : Une plateforme backend sans code.
- **Supabase** : Une alternative open-source à Firebase.
- **FlutterFlow** : Un constructeur d'applications mobiles sans code basé sur Flutter.
- **Power Apps** : Un outil de Microsoft pour créer des applications personnalisées sans code.
- **Coda** : Un outil de gestion de documents et de flux de travail collaboratif.

IV. PRÉSENTATION DU PROJET - CAHIER DES CHARGES

Cahier des charges initial :

Le projet a pour objectif de créer une application capable de traiter, gérer, et visualiser les données extraites (le code généré) de WeWeb. Les objectifs spécifiques sont les suivants :

1. **Traiter et gérer les fichiers extraits de WeWeb :**
L'application doit être capable d'analyser les fichiers extraits de WeWeb et de les intégrer dans une base de données structurée.
2. **Créer une base de données avec des tables principales :**
Une base de données doit être mise en place pour stocker les informations extraites, avec une structure claire et des tables principales bien définies pour organiser les données de manière efficace.
3. **Permettre la consultation des tables créées :**
L'application doit offrir une interface utilisateur qui permet de consulter et d'interagir avec les données stockées dans la base de données.

Équipe de Développement :

- **Client / Chef de projet** : Alexandre Husset
- **Développeur** : Gilles Biolluz

Environnement Technique / Stack :

- **WeWeb** : Utilisé pour le développement initial de l'application et l'exportation du code sur une machine locale.
- **HTML/CSS** : Pour structurer et styliser l'interface utilisateur.
- **Vue 3 & Vite** : Pour développer une interface utilisateur réactive et performante.
- **Supabase** : Pour la gestion des données et l'authentification des utilisateurs.

Fonctionnalités :

1. Connexion Utilisateur :

Les utilisateurs peuvent se connecter à l'application via une authentification sécurisée, gérée par Supabase.

2. Interface Utilisateur :

Une interface web simple et intuitive a été développée pour permettre aux utilisateurs de télécharger leurs fichiers, d'analyser le code généré, et de visualiser les résultats des audits. Cette interface rend le processus de documentation accessible, sans nécessiter de manipulation directe des scripts ou des API.

3. Accès aux Applications :

Après s'être connectés, les utilisateurs ont la possibilité d'accéder aux applications et aux différentes versions qu'ils ont téléchargées. Ils peuvent consulter les détails des applications.

4. Audit Automatisé du Code :

- Vérification du code organisé sous forme de tableau.
- Calcul du pourcentage de workflows sans gestion des cas d'erreur.
- Calcul du pourcentage de workflows sans nom.
- Calcul du pourcentage de variables sans valeur par défaut.
- Calcul du pourcentage de composants sans nom.
- **Comparaison de Versions** : Comparer deux audits de versions différentes pour identifier les différences.

V. PRÉSENTATION DU PROJET : DIFFÉRENTES ITÉRATIONS

Itération 1 : Développement Initial avec Coda et Automatisation

Objectif :

Poser les bases du projet en utilisant Coda pour la gestion des données et en automatisant les audits avec Make et Python.

Stack technique :

Coda pour le front-end et le back-end, un script Python exécuté localement, et Make pour l'automatisation des audits.

Détails :

Coda sert à la fois d'interface pour consulter les tables de la base de données et de stockage centralisé des données. Make est utilisé pour automatiser les audits, assurant la mise à jour et la conformité des données. Un script Python est développé pour analyser, transformer, et insérer les fichiers extraits dans les tables de la base de données.

Itération 2 : Automatisation via CI/CD dans GitLab

Objectif :

Automatiser l'exécution des scripts Python lors de l'extraction des fichiers depuis un dépôt GitLab, en analysant les fichiers et en hydratant les tables dans Coda.

Stack technique :

Pipeline CI/CD dans GitLab pour automatiser l'extraction et l'hydratation des données.

Détails :

Le pipeline CI/CD est configuré pour se déclencher automatiquement à chaque commit ou push, exécutant des scripts Python pour extraire et préparer les données avant de les insérer dans Coda. Les secrets sont gérés de manière sécurisée dans GitLab et injectés au moment de l'exécution. Une documentation accompagne le pipeline pour faciliter les ajustements futurs.

Cf : ANNEXE qui décrit l'architecture.

Itération 3 : Passage à une Solution Commercialisable

Objectif :

Faire évoluer le projet pour devenir une solution commercialisable, utilisable par des clients externes, notamment des entreprises souhaitant auditer leur propre code généré par WeWeb.

Stack technique :

WeWeb pour le front-end, et Supabase pour la gestion des bases de données, API et authentification.

Détails :

Le projet n'est plus limité à un usage interne, mais devient une application capable d'auditer le code généré par WeWeb pour des entreprises externes.

Itération 4 : Vue.js & Supabase (Développement Personnel)

Contexte

C'est le projet que je présente dans ce dossier. Afin d'approfondir mes compétences en développement web, j'ai entrepris **de reprendre intégralement le front-end** du projet en utilisant exclusivement HTML et CSS, sans recourir à WeWeb.

Objectif :

Reprendre les objectifs de l'itération 3 en utilisant Vue.js et Supabase, avec une approche plus traditionnelle du développement.

Stack technique :

Vue.js pour le front-end, Supabase pour la gestion des bases de données.

Itération 5 : WeWeb & Xano (Développement en cours - en Entreprise)

Objectif :

Passer de Supabase à Xano, une autre solution no-code pour le back-end et les API, et affiner les fonctionnalités pour une utilisation en entreprise, notamment en intégrant la capacité de gérer le code extrait des applications PowerApps.

Stack technique :

Passage de Supabase à Xano pour la gestion des bases de données et des API, avec WeWeb toujours utilisé pour le front-end.

Détails :

Cette itération représente un changement stratégique avec l'adoption de Xano, motivée par le partenariat stratégique entre EZACAE et Xano. En intégrant Xano, l'entreprise souhaite démontrer la scalabilité de cette solution, renforçant ainsi sa visibilité comme une option solide et adaptée aux besoins des entreprises.

De plus, la capacité à auditer le code extrait de PowerApps est ajoutée en réponse au partenariat avec Cagip, une entreprise qui utilise PowerApps pour ses applications métiers internes. Cette fonctionnalité permet à EZACAE de répondre aux besoins spécifiques de Cagip tout en élargissant les capacités de son application pour d'autres clients potentiels.

Itération 6 : Scalabilité avec Vue.js & Kubernetes (à venir)

Objectif :

Préparer l'application pour une utilisation à grande échelle en intégrant Kubernetes pour la gestion des déploiements.

Stack technique :

Vue.js pour le front-end, Kubernetes pour l'orchestration des conteneurs et la gestion des déploiements.

Fonctionnalités Futures :

Prévoir des options telles que la personnalisation des règles d'audit, une recherche avancée par ID et nom de composant, des rapports d'audit détaillés et exportables, le support pour d'autres outils no-code, le support multilingue, le suivi des problèmes résolus, des suggestions automatiques d'améliorations, un tableau de bord personnalisable, et un audit de performance...

VI. PRÉSENTATION DU PROJET

Contexte

Contexte de réalisation du front-end :

Le projet front-end initial a été conçu en collaboration avec l'équipe, Alexandre Husset et Gilles Biolluz, en utilisant l'outil no-code **WeWeb**. Cette approche a permis de créer rapidement une interface utilisateur fonctionnelle, en respectant les exigences du projet.

Reprise et développement personnel :

Afin de renforcer et d'utiliser mes compétences en développement web, j'ai entrepris de reprendre intégralement le front-end du projet en utilisant exclusivement HTML et CSS et vue.js, sans recourir à WeWeb et en recréant une base Supabase.

Tour d'horizon des réalisations front-end

Maquettage :

- **Maquettage de l'interface utilisateur avec Figma** : Avant de passer au développement, j'ai réalisé une maquette interactive sur Figma pour définir l'interface utilisateur.
- **Maquettage de l'organisation des composants** : Une fois la structure globale définie, j'ai conçu l'organisation des différents composants Vue.js, afin d'assurer une architecture claire et modulaire avant de commencer le codage avec Whimsical
- **Maquettage de la navigation** dans l'application avec Whimsical

Configuration Initiale :

- **Vite et Vue.js** : Le projet a été mis en place avec Vite, permettant d'optimiser le processus de développement en utilisant Vue.js, ce qui se traduit par des temps de compilation réduits et une expérience de développement plus fluide.
- **SASS** : J'ai utilisé SASS pour styliser les composants Vue.js, offrant une gestion modulaire et réutilisable des styles dans l'application.
- **Versioning avec GitHub** : Pour une gestion efficace du code, un système de versioning a été mis en place via GitHub, facilitant ainsi la collaboration et le suivi des modifications.

Fonctionnalités Front-end :

- **Authentification** : Gestion des connexions des utilisateurs via une authentification sécurisée.
- **Téléchargement et traitement des fichiers JSON** : Intégration de la fonctionnalité permettant de télécharger et de traiter les fichiers JSON générés par WeWeb.
- **Récupération et traitement des données** : Gestion de la récupération et du traitement des données stockées dans Supabase.
- **Mise en place du routeur (index.js)** : Gestion de la navigation entre les différentes vues de l'application.
- **Développement des composants Vue.js** : Création des différents composants nécessaires à l'interface utilisateur.

Tour d'horizon des réalisations Back-end

Conception de la base de données :

- **Conception initiale** : La base de données a été initialement conçue en utilisant DrawSQL pour structurer les tables et les relations entre elles.
- **Création des tables** : Les tables nécessaires ont été créées dans Supabase à l'aide de commandes SQL,

Paramétrage des appels Supabase :

- **Fichier supabase.js** : Mise en place d'un fichier supabase.js pour centraliser le paramétrage des appels vers l'API Supabase.
- **Gestion des configurations** : Les clés API, URL du projet, et autres variables sensibles sont stockées dans un fichier `.env` pour assurer la sécurité des données.

Configuration de la sécurité des données :

- **Row Level Security (RLS)** : Des politiques de sécurité au niveau des lignes ont été implémentées pour chaque table, garantissant que seuls les utilisateurs authentifiés peuvent accéder aux données.

Création d'une vue Audit :

- Une vue SQL nommée **audit** a été créée dans **Supabase**, permettant de regrouper des statistiques en utilisant des fonctions personnalisées créées directement dans Supabase.

Authentification et gestion des sessions :

- **Login/Logout** : Les fonctionnalités de connexion et de déconnexion des utilisateurs ont été implémentées en utilisant l'API Supabase. Cela inclut la gestion des sessions utilisateur, le stockage sécurisé des informations de session.
- **Sécurisation des sessions** : La gestion des sessions est assurée par le stockage sécurisé des jetons d'authentification, avec des vérifications régulières pour maintenir un niveau de sécurité élevé tout au long de l'utilisation de l'application.

Gestion des fichiers JSON :

- **Traitement des fichiers JSON** : Le backend est conçu pour gérer efficacement les fichiers JSON téléchargés par les utilisateurs. Ces fichiers, qui contiennent des données exportées de WeWeb, sont traités et insérés dans les tables appropriées de la base de données.

VII. RÉALISATION - CONCEPTION TECHNIQUE

Maquettage de la base de donnée avec DrawSQL

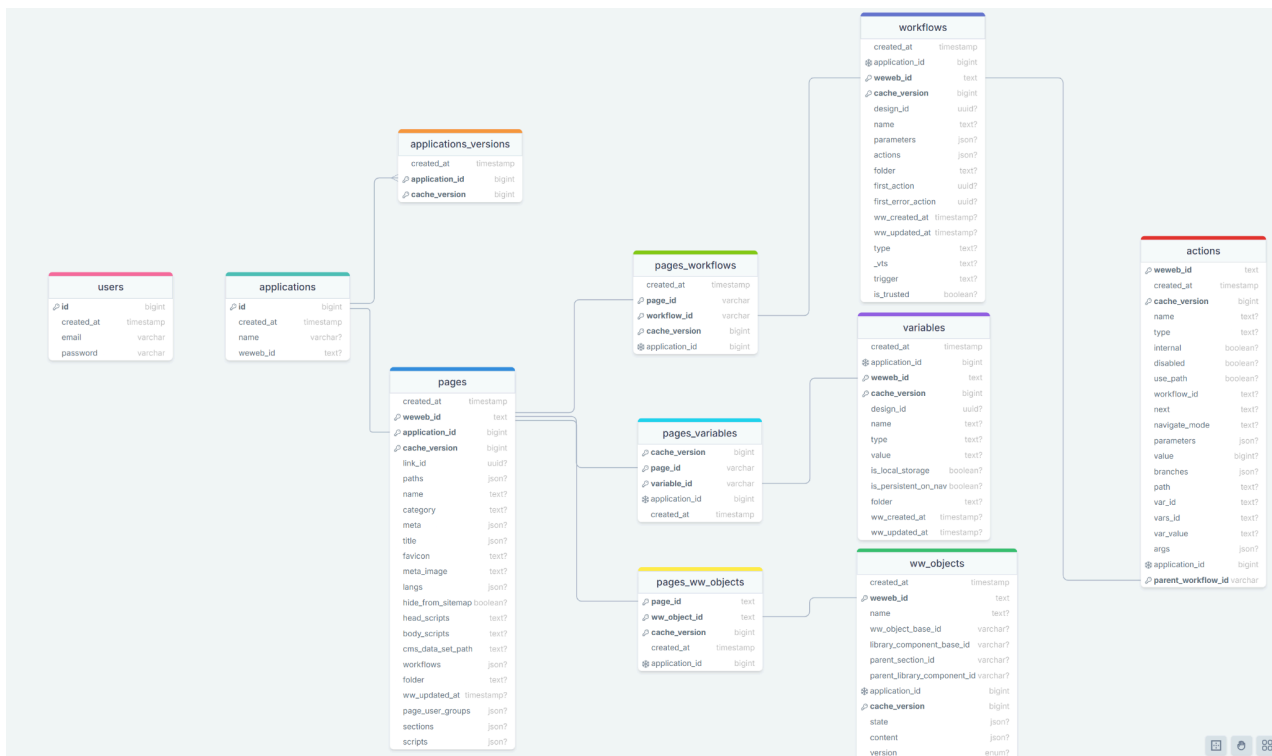


Figure : base de donnée

Entités principales :

- application
- application_version
- page
- workflow
- variable
- action
- ww_object
- page_workflow
- page_variable
- page_ww_object

Création de la base de donnée à l'aide de commande SQL

```

-- Création de la table `application`
create table
public.application (
  id bigint generated by default as identity not null,
  created_at timestamp with time zone not null default now(),
  name character varying null,
  weweb_id text null,
  constraint applications_pkey primary key (id)
) tablespace pg_default;

-- Création de la table `application_version`
create table
public.application_version (
  created_at timestamp with time zone not null default now(),
  application_id bigint not null,
  cache_version bigint not null,
  constraint application_version_pkey primary key (application_id, cache_version),
  constraint application_version_application_id_fkey foreign key (application_id) references application (id)
) tablespace pg_default;

-- Création de la table `page`
create table
public.page (
  weweb_id text not null,
  cache_version integer not null,
  created_at timestamp with time zone not null default now(),
  application_id integer null,
  link_id uuid null,
  paths jsonb null,
  name text null,
  category text null,
  meta jsonb null,
  title text null,
  favicon text null,
  meta_image text null,
  langs jsonb null,
  hide_from_sitemap boolean null,
  head_scripts jsonb null,
  body_scripts jsonb null,
  cms_data_set_path jsonb null,
  workflows jsonb null,
  folder text null,
  ww_updated_at timestamp without time zone null,
  page_user_groups jsonb null,
  sections jsonb null,
  scripts jsonb null,
  constraint page_pkey primary key (weweb_id, cache_version),
  constraint fk_application foreign key (application_id) references application (id)
) tablespace pg_default;

-- Autres tables...

-- Création de la vue `audit`
create view
public.audit as
select
  a.id as application_id,
  av.cache_version,
  get_null_error_action_percentage (a.id, av.cache_version) as null_error_action_percentage,
  get_workflow_nameless_percentage (a.id, av.cache_version) as workflow_nameless_percentage,
  get_variable_no_value_percentage (a.id, av.cache_version) as variable_no_value_percentage,
  get_ww_object_nameless_percentage (a.id, av.cache_version) as ww_object_nameless_percentage
from
  application_version av
  left join application a on a.id = av.application_id;

-- Suite du script pour la création des autres tables...

```

Figure : extraits des commandes SQL pour créer/modifier les tables de la base de donnée

Maquettage de l'interface avec Figma :

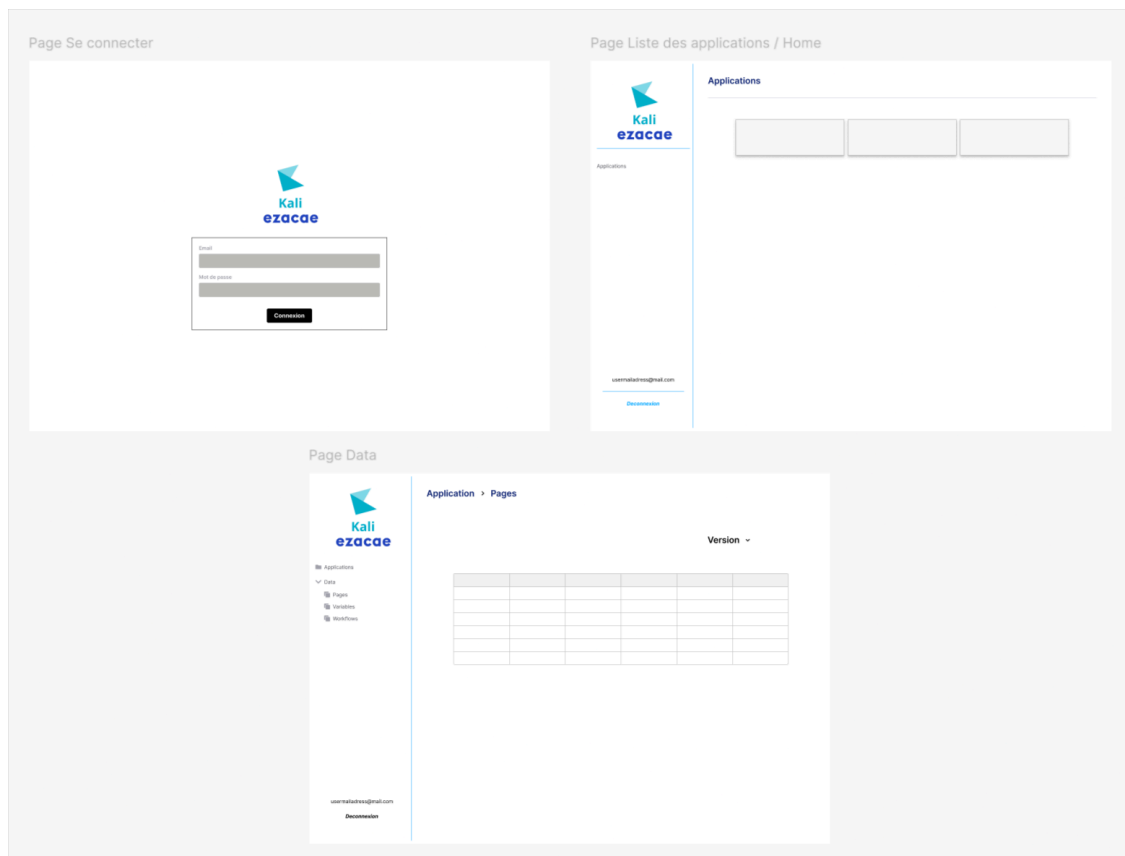


Figure : Maquette succinte de l'interface utilisateur

Maquettage du parcours utilisateur

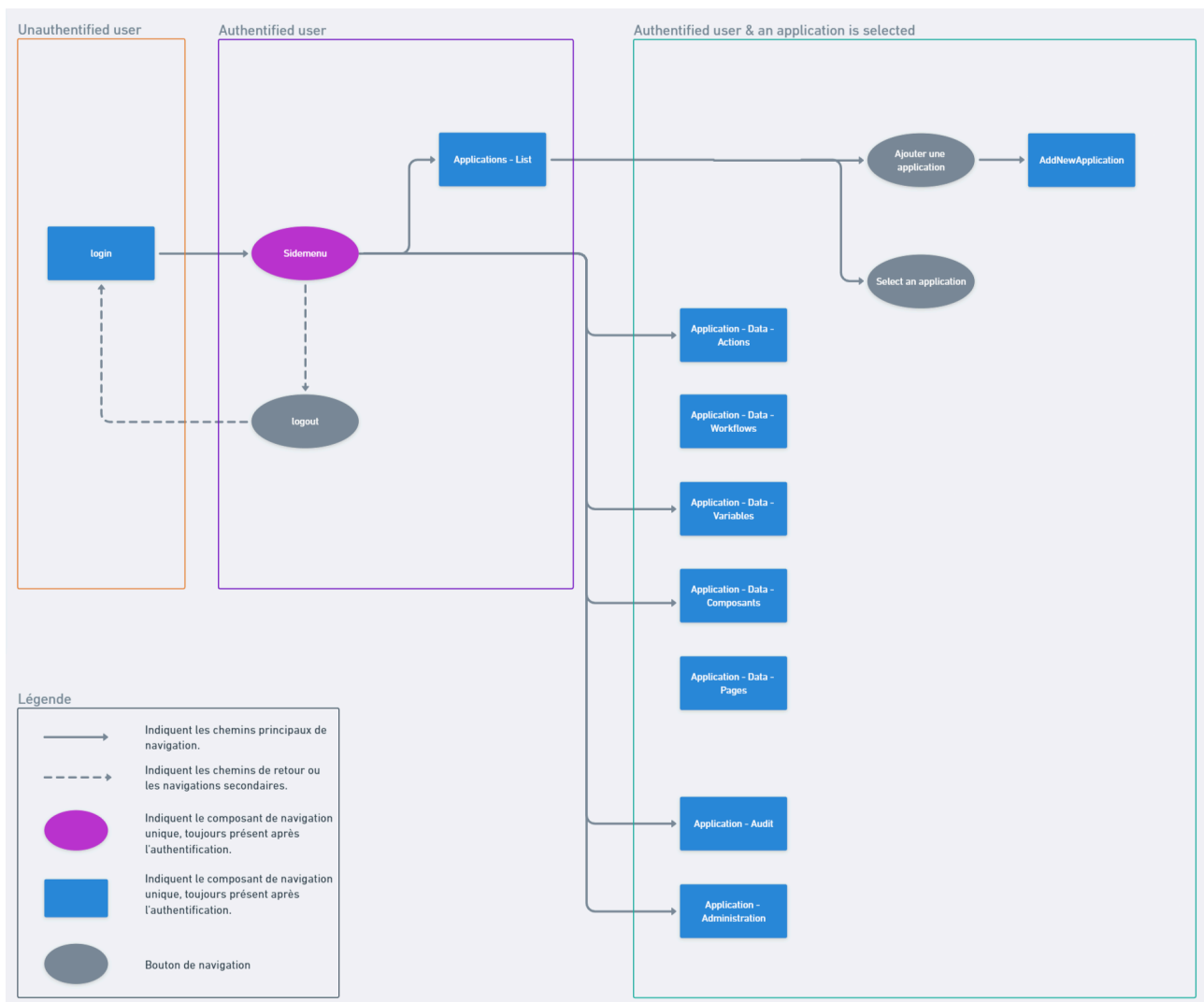


Figure : schéma du parcours utilisateur dans l'application

Développement des principaux composants Vue.js

La vue racine :

- **App.vue (Vue principale)**

- **Rôle** : Il s'agit du composant racine de l'application, le point d'entrée de toute l'application qui gère les layouts d'authentification et d'accès sécurisé. Cette vue organise l'affichage général des données récupérées pour chaque application, servant de point d'entrée pour les utilisateurs authentifiés.
- **Imbrication** : Contient les layouts **AuthenticatedLayout.vue** et **UnauthenticatedLayout.vue**, qui gèrent respectivement l'interface des utilisateurs connectés et non connectés.

Les deux vues en fonction de l'authentification :

- **AuthenticatedLayout.vue**
 - **Rôle** : Ce composant gère la mise en page pour les utilisateurs authentifiés. Il est utilisé pour toutes les routes accessibles après connexion, intégrant le menu latéral pour une navigation fluide à travers les différentes sections de l'application.
 - **Imbrication** : Intègre **TheSidemenu.vue** pour le menu de navigation latéral.
- **UnauthenticatedLayout.vue**
 - **Rôle** : Ce composant gère la mise en page pour les utilisateurs non authentifiés, incluant principalement la vue de login pour permettre aux utilisateurs de se connecter.
 - **Imbrication** : Affiche la vue de login (**Login.vue**).

Un composant de navigation :

- **TheSidemenu.vue (Composant)**
 - **Rôle** : Ce composant représente le menu latéral de l'application, accessible après authentification. Il est toujours présent dans **AuthenticatedLayout.vue**, permettant une navigation fluide entre les différentes sections de l'application.
 - **Imbrication** : Utilise **Sidemenuitem.vue** pour chaque élément du menu latéral, permettant l'accès à différentes vues de l'application.
- **Sidemenuitem.vue (Composant instancié)**
 - **Rôle** : Ce composant est utilisé dans **TheSidemenu.vue** pour représenter chaque élément du menu latéral. Chaque élément permet d'accéder à une vue spécifique de l'application.

Une vue pour visionner toutes les applications :

- **Applications.vue**
 - **Rôle** : Gère la liste des applications téléchargées. Cette vue permet aux utilisateurs de voir toutes les applications disponibles, de sélectionner celles qu'ils souhaitent consulter ou auditer.
 - **Imbrication** : Utilise **ApplicationsHeader.vue** pour l'affichage de l'en-tête

Une vue pour ajouter une application :

- **AddNewApplication (Vue)**
 - **Rôle** : Permet aux utilisateurs d'ajouter une nouvelle application en téléchargeant des fichiers JSON. Cette vue gère l'ajout de nouvelles applications.

Une vue pour visionner le détail des données d'une application :

- **Application.vue**
 - Rôle : Vue principale pour afficher les différentes sections de l'application. Elle gère les vues comme l'administration, l'audit, les pages, etc., en tant que sous-composants. Cette vue est chargée lorsque l'utilisateur sélectionne une application à consulter ou à auditer.
 - Imbrication : Contient les vues **ApplicationAdministration.vue**, **ApplicationAudit.vue**, **ApplicationData.vue**.
- **ApplicationData.vue (Composant)**
 - Rôle : Ce composant, intégré dans la vue **Application.vue**, est chargé d'afficher les données spécifiques à chaque application. Il intègre plusieurs instances de **ApplicationDataTable.vue**
- **ApplicationDataTable.vue (Composant instancié)**
 - Rôle : Affiche les données sous forme de tableaux structurés, permettant à l'utilisateur de visualiser les informations de manière organisée. Ce tableau peut être filtré grâce à **ApplicationDataTableVersionFilter.vue** pour montrer uniquement les informations pertinentes selon la version sélectionnée.
- **ApplicationDataTableVersionFilter.vue (Composant)**
 - Rôle : Ce composant est utilisé pour filtrer les données affichées dans **ApplicationDataTable.vue** en fonction de la version de l'application sélectionnée. Il permet aux utilisateurs de voir uniquement les informations pertinentes pour la version choisie.

Un composant pour visionner et gérer l'audit d'une application :

- **ApplicationAdministration.vue**
 - Rôle : Permet de gérer les données des applications, en offrant la possibilité de sélectionner les colonnes spécifiques à afficher pour chaque catégorie de données (Pages, Variables, Workflows, Composants, Actions).
 - Imbrication : Utilise **ApplicationAdministrationColumn.vue** pour fournir cette fonctionnalité de sélection dynamique des colonnes à afficher.
- **ApplicationAdministrationColumn.vue (Composant instancié)**
 - Rôle : Utilisé dans **ApplicationAdministration.vue**, ce composant permet aux utilisateurs de sélectionner quelles colonnes de données doivent être affichées pour chaque type de données (pages, variables, workflows, etc.), offrant une flexibilité accrue dans la gestion des données présentées.
- **ApplicationAudit.vue (Vue)**
 - Rôle : vue qui permet de visualiser et de comparer les résultats d'audit des applications. Les graphiques circulaires sont générés à l'aide du composant **DoughnutComponent.vue**, offrant une vue claire et concise des résultats.
 - Imbrication : Utilise **DoughnutComponent.vue** pour afficher les résultats sous forme de graphiques, facilitant la comparaison entre différentes versions d'une application.
- **DoughnutComponent.vue (Composant instancié)**
 - Rôle : Affiche les résultats d'audit sous forme de graphiques circulaires, permettant une comparaison visuelle des éléments tels que les workflows, variables, et composants manquants.

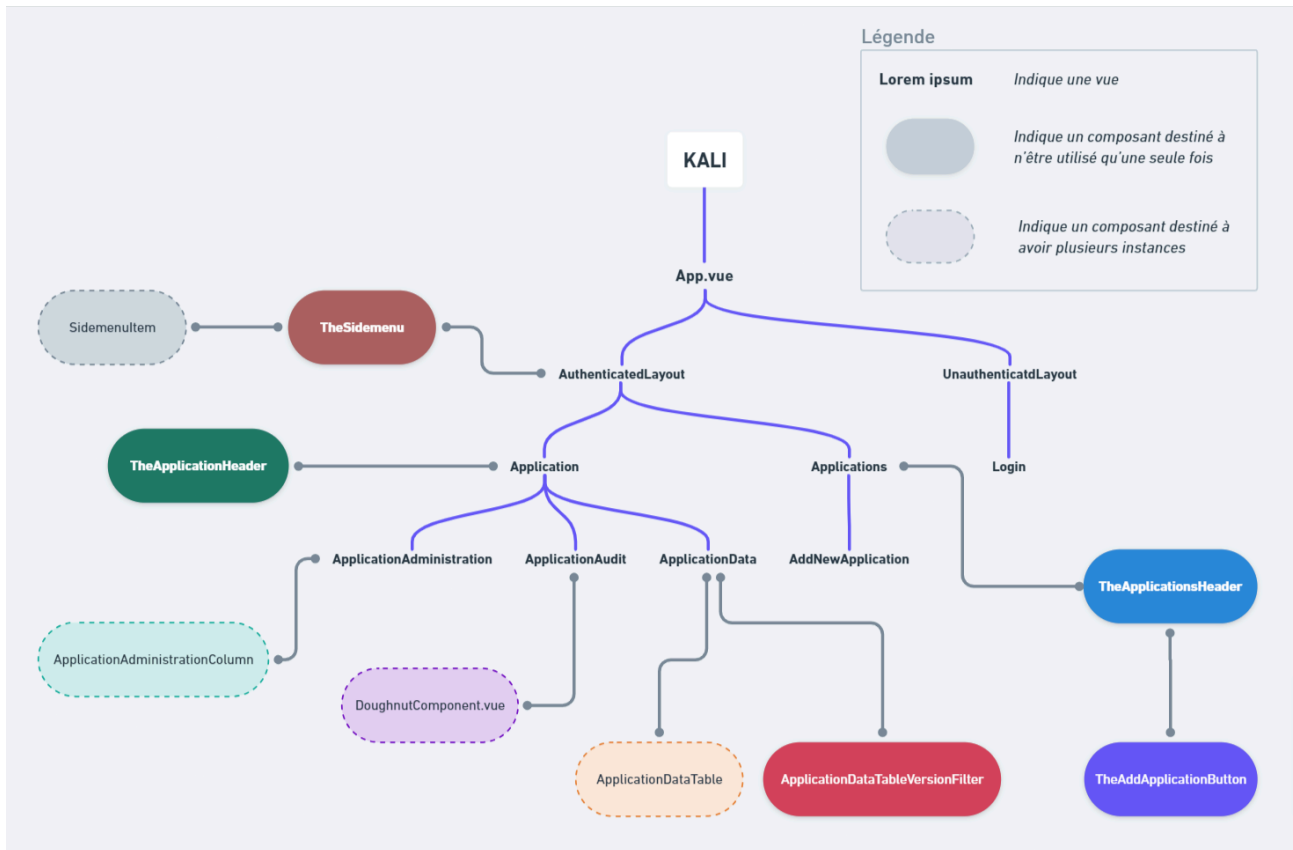


Figure : schéma de l'imbrication des composants dans l'application

VIII. RÉALISATION

Le routeur

Navigation et Routage

Le routeur Vue.js est une partie centrale de l'application, car il gère la navigation entre les différentes pages. Le fichier `index.js` contient la configuration du routeur, où chaque route est associée à un composant spécifique.

Configuration des Routes

- **/** : Route principale qui charge les composants pour les utilisateurs connectés. Elle affiche la liste des applications et permet d'ajouter de nouvelles applications.
- **/login** : Route pour la connexion des utilisateurs. Elle affiche la page de login pour les utilisateurs non authentifiés.

```
const routes = [
  {
    path: '/',
    component: AuthenticatedLayout,
    children: [
      { path: 'applications', component: Applications },
      { path: 'add', component: AddNewApplication },
      {
        path: 'application/:id',
        component: Application,
        children: [
          { path: 'data', component: ApplicationData },
          { path: 'audit', component: ApplicationAudit },
          { path: 'administration', component: ApplicationAdministration }
        ]
      }
    ]
  },
  { path: '/login', component: UnauthenticatedLayout, children: [{ path: '', component: Login }] },
  { path: '/*', component: NotFound }
];
```

Figure :

Navigation et Sécurité

Pour s'assurer que seules les personnes connectées peuvent accéder aux pages sensibles, le routeur peut vérifier l'état de connexion de l'utilisateur avant de le laisser accéder aux pages. Si l'utilisateur n'est pas authentifié, il est redirigé vers la page de login.

Exemple de code pour la vérification :

```
router.beforeEach((to, from, next) => {
  const isAuthenticated = !!localStorage.getItem('authToken');
  if (to.path !== '/login' && !isAuthenticated) {
    next('/login');
  } else {
    next();
  }
});
```

Figure :

Gestion des Erreurs

Le routeur inclut une route spéciale pour gérer les erreurs, redirigeant les utilisateurs vers une page "404" si la route à laquelle ils essaient d'accéder n'existe pas.

```
{ path: '/*', component: NotFound }
```

Récupérer et afficher les données d'une application

Introduction

on a vu précédemment que préalablement à l'affichage des données l'utilisateur doit être connecté

Configuration

Fichier `supabase.js` : Configuration initiale

Le fichier `supabase.js` est central pour la gestion des interactions avec Supabase. Il configure le client Supabase avec les informations nécessaires pour communiquer avec l'API Supabase.

```
import { createClient } from '@supabase/supabase-js';

const supabaseUrl = process.env.VITE_SUPABASE_URL;
const supabaseKey = process.env.VITE_SUPABASE_KEY;

const supabase = createClient(supabaseUrl, supabaseKey);

export default supabase;
```

- **`createClient`** : Cette fonction est utilisée pour créer une instance du client Supabase, qui sera utilisée pour toutes les opérations (authentification, requêtes sur la base de données, etc.).
- **Variables d'environnement** : `supabaseUrl` et `supabaseKey` sont des variables sécurisées qui proviennent du fichier `.env`.

Fichier `.env` : Sécurité et gestion des informations sensibles

Le fichier `.env` stocke les informations sensibles comme les clés API et les URL de projets. Elles contiennent respectivement l'URL du projet Supabase et la clé API. L'utilisation des variables d'environnement permet de garder ces informations en sécurité et de les rendre facilement modifiables sans toucher au code source. Ce fichier est listé dans le fichier `.gitignore` et n'est donc pas ajouté au dépôt github de l'application.

- **VITE_SUPABASE_URL** : URL unique du projet Supabase.
- **VITE_SUPABASE_KEY** : Clé API pour accéder aux services Supabase.

Intégration dans Vue.js

L'intégration de Supabase dans l'application Vue.js se fait via l'importation du client Supabase dans les composants ou les services où il est nécessaire.

```
import supabase from './supabase';
```

Maintenant qu'on a créé le client supabase, on peut utiliser les méthodes de supabase pour l'Authentification

Authentification via Mot de Passe dans Supabase

L'authentification via mot de passe avec Supabase repose sur le service d'authentification intégré de Supabase, qui est une couche au-dessus de PostgreSQL. (cf *Annexe*). Cette fonctionnalité est utilisée dans l'application pour permettre aux utilisateurs de se connecter de manière sécurisée.

Gestion des Sessions Utilisateurs dans `store.js`

Exemple de Workflow :

1. **Login.vue** appelle `store.login(email, password)`.
2. **store.js** utilise la fonction `login` de `supabaseApi.js` pour exécuter l'authentification.
3. Si la connexion réussit, **store.js** met à jour l'état global avec les informations de l'utilisateur (`user`), sinon il enregistre le message d'erreur (`errorMessage`).
4. **Login.vue** écoute ces changements et réagit en conséquence, par exemple en redirigeant l'utilisateur ou en affichant un message d'erreur.

Le fichier `store.js` utilise l'objet `reactive` de Vue.js pour gérer l'état global de l'application, y compris les informations sur l'utilisateur connecté. Dans ce contexte, l'authentification est gérée comme suit :

- **login** : Cette fonction, créée dans `supabaseApi.js` et utilisée dans `store.js`, appelle la méthode `signInWithPassword` de Supabase pour vérifier les informations d'identification de l'utilisateur. Si les informations sont correctes, les détails de l'utilisateur sont stockés dans l'état global (`store.user`), ce qui permet à l'application de savoir que l'utilisateur est authentifié.

```
// supabaseApi.js
export const login = async (email, password) => {
  const { data, error } = await supabase.auth.signInWithPassword({
    email,
    password,
  });
  if (error) {
    throw error;
  }
  return data;
};
```

```
// store.js
import { reactive } from "vue";
import {
  login
} from "../api/supabaseApi.js";

const store = reactive({
  user: null,
  errorMessage: null,

  async login(email, password) {
    try {
      const data = await login(email, password);
      if (data && data.user) {
        this.user = data.user;
        this.errorMessage = null;
      } else {
      }
    } catch (error) {
      this.errorMessage = error.message;
    }
  }
});

export default store;
```

Connexion dans le composant Login.vue

```
// login.vue
<template>
  <div class="container-column">
    <div class="app-logo-container">
      <div class="app-logo">
        
      </div>
    </div>
    <div class="login-form-container">
      <form @submit.prevent="handleLogin" class="form-body">
        <div class="container-input">
          <label for="email">Email:</label>
          <input type="email" v-model="email" required/>
        </div>
        <div class="container-input">
          <label for="password">Mot de passe:</label>
          <input type="password" v-model="password" required/>
        </div>
        <div class="login-form-footer">
          <button type="submit" class="btn-primary">Se connecter</button>
        </div>
      </form>
      <p v-if="errorMessage" class="error">{{ errorMessage }}</p>
    </div>
  </div>
</template>
```

```
// login.vue
<script setup>
import {ref} from 'vue';
import store from "../../store.js";
import router from "../../router/index.js";

const email = ref('');
const password = ref('');
const errorMessage = ref('');

const handleLogin = async () => {
  try {
    await store.login(email.value, password.value);
    if (!store.errorMessage) {
      await router.push('/applications');
    }
  } catch (error) {
    errorMessage.value = error.message;
  }
};
</script>
```

Grâce à cette gestion centralisée dans le **store**, l'état de l'utilisateur connecté est disponible dans toute l'application, permettant d'afficher des contenus réservés ou de rediriger l'utilisateur selon son statut.

Maintenant, l'utilisateur est authentifié, il peut récupérer les données de ses applications

Sélection de l'Application et Récupération des Données

Lorsqu'un utilisateur sélectionne une application dans la vue **Applications.vue**, l'identifiant de cette application (**selectedApplicationId**) est stocké dans l'état global de l'application via le **store**. Grâce à la fonction **setSelectedApplicationId** on peut mettre à jour l'identifiant de l'application sélectionnée dans le **store**, déclenchant ainsi la récupération des données associées.

```
setSelectedApplicationId(id) {  
  this.selectedApplicationId = id;  
}
```

Récupération des pages associées à l'application

Une fois l'application sélectionnée, les données des pages associées sont récupérées depuis Supabase via une requête API. Cette récupération est gérée par la méthode **fetchPages()** dans le **store**.

Cette méthode utilise l'API **fetchPages** de **supabaseApi.js** pour envoyer une requête à la table **page** de Supabase, filtrée par l'identifiant de l'application sélectionnée. Les données récupérées sont ensuite stockées dans le **store**.

```
async fetchPages() {  
  const pages = await fetchPages(this.selectedApplicationId);  
  if (pages) {  
    this.pages = pages;  
    if (pages.length > 0) {  
      this.pageKeys = Object.keys(pages[0]);  
      if (this.selectedPageKeys.length === 0) {  
        this.selectedPageKeys = [...this.pageKeys];  
      }  
    }  
  }  
}
```


Requête API pour la récupération des pages

La requête pour récupérer les pages dans `supabaseApi.js` :

```
export const fetchPages = async (selectedApplicationId) => {
  let { data: pages, error } = await supabase
    .from('page')
    .select()
    .eq('application_id', selectedApplicationId);
  if (error) {
    console.error('Error fetching pages:', error);
  }
  return pages;
};
```

Cette fonction envoie une requête à la table `page` dans Supabase pour récupérer toutes les pages associées à l'application sélectionnée.

Affichage des pages dans l'interface utilisateur

Les pages récupérées sont ensuite affichées dans le composant `ApplicationDataTable.vue` qui est responsable de l'affichage des données sous forme de tableau.

Le composant `ApplicationDataTable`

Le composant `ApplicationDataTable.vue` est utilisé pour afficher des données sous forme de tableau dans l'application. Il prend en entrée des données et des colonnes à afficher, et gère l'affichage dynamique de ces informations.

Propriétés (Props)

Le composant reçoit deux propriétés :

- **data** : Un tableau contenant les données à afficher dans le tableau. Par défaut, c'est un tableau vide.
- **columns** : Un tableau contenant les noms des colonnes à afficher. Par défaut, c'est un tableau vide.

```
const props = defineProps({
  data: {type: Array, default: () => []},
  columns: {type: Array, default: () => []}
});
```

Filtrage des données

Les données sont filtrées (bien que dans ce code, le filtrage ne soit pas encore personnalisé) à l'aide d'une propriété calculée (`computed`). Cette propriété retourne simplement les données telles qu'elles sont reçues via les props, sans modification. Cela permet de préparer la base pour des filtres ou des transformations futures si nécessaire.

```
const filteredData = computed(() => {  
  return props.data;  
});
```

Structure du Template

Le template du composant est divisé en deux sections principales :

- **Vérification de l'existence de données** : Si les données filtrées sont vides, un message "No data available." est affiché.
- **Affichage du tableau** : Si des données sont disponibles, un tableau HTML est généré.

```
<template>  
  <div v-if="!filteredData.length">  
    No data available.  
  </div>  
  <div v-else class="container-table custom-scrollbar">  
    <table class="data-table">  
      <thead>  
        <tr>  
          <th v-for="column in columns" :key="column">{{ column }}</th>  
        </tr>  
      </thead>  
      <tbody>  
        <tr v-for="row in filteredData" :key="row.id">  
          <td v-for="column in columns" :key="column">  
            <span v-if="row[column] !== null && row[column] !== undefined">  
              {{ row[column] }}  
            </span>  
            <span v-else class="dot red-dot"></span>  
          </td>  
        </tr>  
      </tbody>  
    </table>  
  </div>  
</template>
```

Affichage des données

- **En-têtes de colonnes** : Le composant utilise les noms des colonnes passés via `props.columns` pour générer les en-têtes du tableau (`<th>`).
- **Lignes de données** : Pour chaque ligne de données (`row`) dans `filteredData`, le composant génère une ligne dans le tableau. Chaque cellule de la ligne (`<td>`) correspond à une colonne.
 - Si la donnée existe pour une cellule donnée (`row[column]` n'est pas `null` ou `undefined`), elle est affichée.
 - Si la donnée n'existe pas, un petit point rouge est affiché (indiquant une donnée manquante ou non applicable).

Envoyer et stocker les données d'une application

Après avoir vu comment gérer l'authentification via Supabase et comment récupérer les données d'une application, nous allons maintenant explorer comment envoyer et stocker les données d'une application, en prenant l'exemple des pages.

Le format des données envoyées en format json est visible dans les annexe.

Le formulaire d'envoi des fichiers :

Un formulaire dans `addApplicationData.vue` est conçu pour permettre la sélection de fichiers JSON, la lecture de leur contenu, et l'envoi de ces données à Supabase.

Sélection et lecture du fichier JSON

Lorsqu'un utilisateur sélectionne un fichier, la méthode `handleFileUpload` est déclenchée. Cette méthode utilise `FileReader` de javascript pour lire le contenu du fichier JSON et le stocker dans une variable réactive.

```

const handleFileUpload = (event) => {
  const files = event.target.files;
  for (const file of files) {
    if (file && file.type === "application/json") {
      const reader = new FileReader();
      reader.onload = (e) => {
        const content = e.target.result;
        filesContent.value.push({
          name: file.name,
          content: JSON.parse(content)
        });
        console.log("Fichier JSON téléchargé :", content);
      };
      reader.readAsText(file);
    } else {
      console.error("Veuillez télécharger un fichier JSON valide.");
    }
  }
};

```

Structure du Formulaire

Le formulaire est construit de manière simple avec un champ de sélection de fichiers, permettant uniquement les fichiers `.json`. Lors de la sélection, le fichier est automatiquement lu et prêt à être envoyé.

```



```

Envoi des Données à Supabase

Une fois le fichier JSON validé, les données sont envoyées à Supabase via une fonction API. Cette fonction, `addApplication`, utilise l'API Supabase pour stocker les données dans la base de données.

```

const addApplication = async () => {
  try {
    console.log(`Nom: ${name.value}, weweb id: ${wewebid.value}`);
    if (!name.value || !wewebid.value || filesContent.value.length === 0) {
      console.error("Veuillez remplir tous les champs et télécharger au moins un fichier JSON.");
      return;
    }
    await processFiles(filesContent.value, name.value, wewebid.value);
    resetForm();
  } catch (error) {
    console.error('Erreur lors de l\'ajout de l\'application :', error);
  }
};

```

Le script d'insertion des fichiers dans la BDD :

Le script `extractor.js` a pour objectif de traiter des fichiers JSON, d'extraire les informations pertinentes (pages, variables, workflows, composants), et de les insérer ou mettre à jour dans une base de données via Supabase.

ANNEXE



Mode Opérateur de création de dépôt d'une nouvelle app

i Ce document explique le pas à pas de création du fichier `.gitlab-ci.yml`. Il inclut un fichier de configuration CI partagé du projet « ezacae/ezacae-ci-utils » et définit une liste de variables spécifiques au projet et divers ID de colonne pour les tables Coda. Ces variables sont utilisées dans le pipeline CI pour interagir avec le document spécifique du projet, gérer les chemins de données et identifier des colonnes spécifiques dans les tables Coda pour les processus automatisés liés à la gestion des pages, etc.

► Exemple

1. Créer un fichier `.gitlab-ci.yml`

2. Inclure le fichier de configuration CI partagé

Commencez par inclure le fichier de configuration CI partagé depuis le dépôt `ezacae/ezacae-ci-utils`. Cela permet de réutiliser les configurations et d'assurer la cohérence entre tous les dépôts.

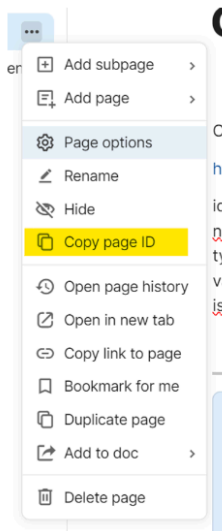
```
include:
- project: 'ezacae/ezacae-ci-utils'
  file: '/gitlab-ci-weweb.yaml'
```

3. Inclure le fichier de configuration CI partagé

Déclaration les variables pour ce projet

L'ID du document coda du projet

▼ Où le trouver ?



```
CODA_DOC_ID: "st6X2ke8SK"
// ID du document coda du projet
```

Le chemin des fichiers contenant des pages du projet

▼ Où le trouver ?

C'est l'endroit dans l'export WeWeb où se trouve les pages json de l'application. Il est toujours le même.

```
SCRIPT_DATAPATH: "public/public/data"
```

L'ID du tableau des Variables

- Où le trouver ?

```
VAR_CODA_TABLE_ID: "grid-myY7z7MsnZ"  
VAR_COLUMN_ID_VAR_ID: "c-PuyyCMeAyR"  
VAR_COLUMN_ID_NAME: "c-Neje628C2G"  
VAR_COLUMN_ID_TYPE: "c-1ZRZhJZqRF"  
VAR_COLUMN_ID_VALUE: "c-pAx7giRlfq"  
VAR_COLUMN_ID_IS_PERSISTENT: "c-zDd61yGjWP"
```

Les IDs de chaque colonne du tableau des Variables

- Où les trouver ?

```
VAR_COLUMN_ID_VAR_ID: "c-PuyyCMeAyR"  
VAR_COLUMN_ID_NAME: "c-Neje628C2G"  
VAR_COLUMN_ID_TYPE: "c-1ZRZhJZqRF"  
VAR_COLUMN_ID_VALUE: "c-pAx7giRlfq"  
VAR_COLUMN_ID_IS_PERSISTENT: "c-zDd61yGjWP"
```

L'ID du tableau des Workflows

```
WF_CODA_TABLE_ID: "grid-ZUUoAn96YY"
```

Les IDs de chaque colonne du tableau des Workflows

```
WF_COLUMN_ID_WF_ID_PAGE_ID: "c-2LxurutYLo"  
WF_COLUMN_ID_WF_ID: "c-6nTv3Z5Z55"  
WF_COLUMN_ID_WF_NAME: "c-1Yd5iIEfDn"  
WF_COLUMN_ID_PAGE_ID: "c-Nn6z4gbUAa"  
WF_COLUMN_ID_PAGE_NAME: "c-FBXazdB9Px"  
WF_COLUMN_ID_IS_PAGE_WF: "c-Xq7K5GALgE"  
WF_COLUMN_ID_TRIGGER: "c-jBNngTpk16"  
WF_COLUMN_ID_FOLDER: "c-Uy8gOf5Fb6"  
WF_COLUMN_ID_FIRST_ACTION_ID: "c-ygZEeN31No"  
WF_COLUMN_ID_FIRST_ACTION_NAME: "c-ks8K9j7_iP"  
WF_COLUMN_ID_FIRST_ACTION_TYPE: "c-K1iLqKgdRj"  
WF_COLUMN_ID_FIRST_ACTION_PARAM: "c-14YLF_8dY1"  
WF_COLUMN_ID_FIRST_ACTION_EXE_WF_ID: "c-15YYYIDwt"  
WF_COLUMN_ID_ACTION_JSON: "c-D-6I0HbaAZ"
```

L'ID du tableau des Pages

```
PAGE_CODA_TABLE_ID: "grid-91Ae51Z14N"
```

Les IDs de chaque colonne du tableau des Pages

```
PAGE_COLUMN_ID_PAGE_ID: "c-h0-qebK_0K"  
PAGE_COLUMN_ID_PAGE_NAME: "c-0UN9_5BNvL"  
PAGE_COLUMN_ID_FAVICON: "c-8Fj0QVU-p1"  
PAGE_COLUMN_ID_TITLEFR: "c-Sa9rKLHhCy"  
PAGE_COLUMN_ID_META_DESC: "c-qgDHBfurwR"  
PAGE_COLUMN_ID_META_KEYWORDS: "c-Glhh-NHa0f"  
PAGE_COLUMN_ID_META_SOCIALDESC: "c-PytMzgcJfE"  
PAGE_COLUMN_ID_META_SOCIALTITLE: "c-FeaP_TNlP6"  
PAGE_COLUMN_ID_META_STRUCTUREDDATA: "c-ZJ60S9qwDh"
```

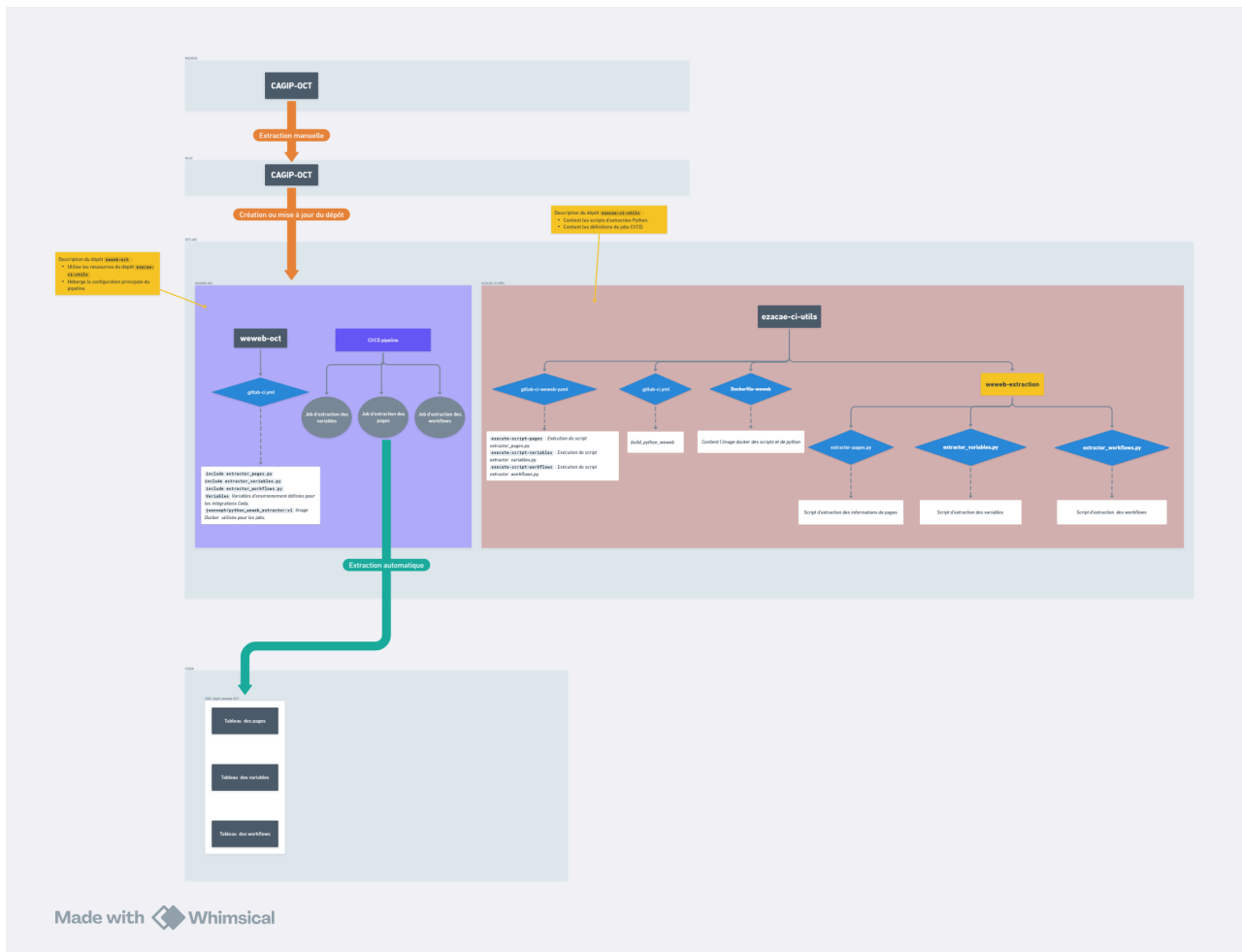
L'ID du tableau des Collections

```
COLLECTION_CODA_TABLE_ID: "grid--yTYWLRcVv"
```

Les IDs de chaque colonne du tableau des Collections


```
COLLECTION_COLUMN_ID_COLLECTION_ID: "c-B0u5kYZSId"  
COLLECTION_COLUMN_ID_TYPE: "c-C3bf8zXZnw"  
COLLECTION_COLUMN_ID_COLLECTION_NAME: "c-Z9eGfW0wZF"  
COLLECTION_COLUMN_ID_LASTSVNCDATE: "c-6S0HEu9HSG"  
COLLECTION_COLUMN_ID_ERROR: "c--o1yft7Q10"
```

ANNEXE



ANNEXE

Page Se connecter




Email

Mot de passe

Connexion

Page Liste des applications / Home

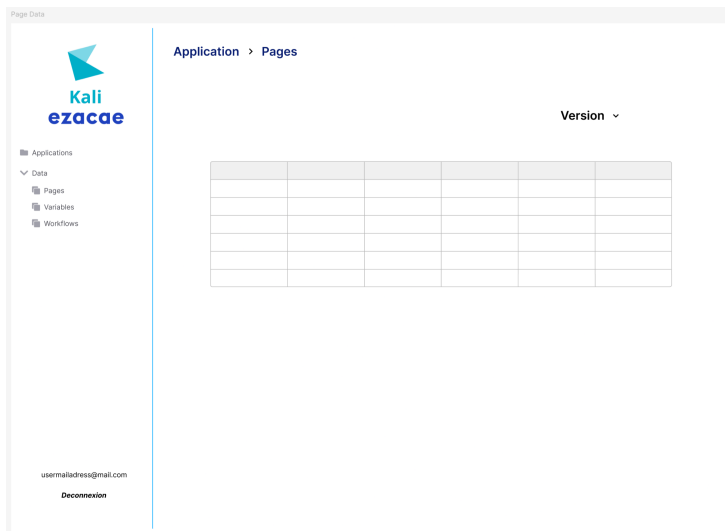


Applications

usermailadress@mail.com

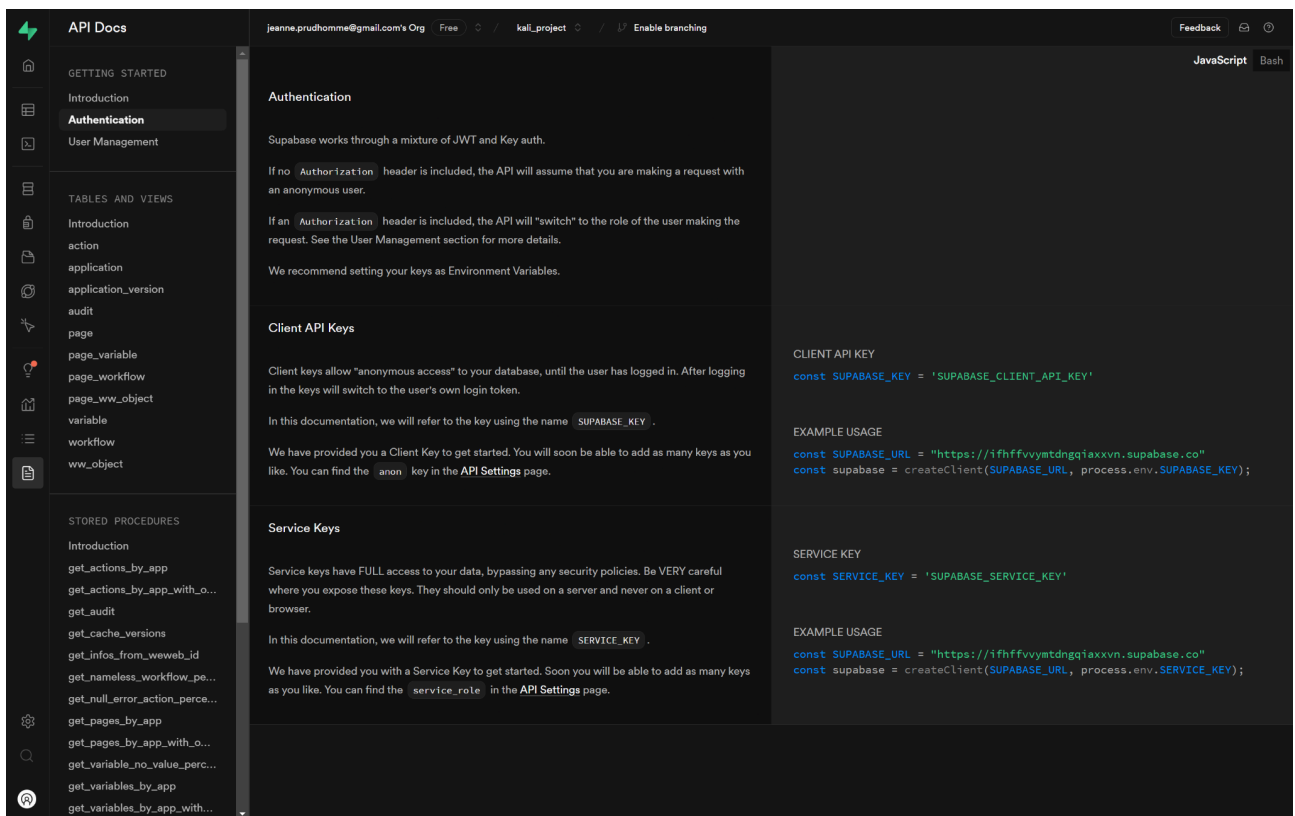
Deconnexion

Applications



ANNEXE

La documentation e Supabase concernant l'intégration de Supabase



ANNEXE

Les méthodes intégrés de Supabase pour géré l'authentification des utilisateurs :

The screenshot displays the Supabase API documentation for User Management. The interface is divided into three main sections: a sidebar on the left, a central text area, and a right-hand code area.

Sidebar: The left sidebar contains a navigation menu with the following categories:

- API Docs
 - GETTING STARTED
 - Introduction
 - Authentication
 - User Management
 - TABLES AND VIEWS
 - Introduction
 - action
 - application
 - application_version
 - audit
 - page
 - page_variable
 - page_workflow
 - page_wvw_object
 - variable
 - workflow
 - ww_object
 - STORED PROCEDURES
 - Introduction
 - get_actions_by_app
 - get_actions_by_app_with_o...
 - get_audit
 - get_cache_versions
 - get_info_from_weweb_id
 - get_nameless_workflow_pe...
 - get_null_error_action_perce...
 - get_pages_by_app
 - get_pages_by_app_with_o...
 - get_variable_no_value_perc...
 - get_variables_by_app
 - get_variables_by_app_with...

Central Text Area: This section provides detailed information about the User Management API, including a general overview and specific endpoints for various authentication methods.

Right-hand Code Area: This section displays the JavaScript code snippets for each authentication method, using the Supabase JS client.

User Management Overview: Supabase makes it easy to manage your users. Supabase assigns each user a unique ID. You can reference this ID anywhere in your database. For example, you might create a `profiles` table references the user using a `user_id` field. Supabase already has built in the routes to sign up, login, and log out for managing users in your apps and websites.

Sign Up: Allow your users to sign up and create a new account. After they have signed up, all interactions using the Supabase JS client will be performed as "that user".

Log In With Email/Password: If an account is created, users can login to your app. After they have logged in, all interactions using the Supabase JS client will be performed as "that user".

Log In With Magic Link Via Email: Send a user a passwordless link which they can use to redeem an access token. After they have clicked the link, all interactions using the Supabase JS client will be performed as "that user".

Sign Up With Phone/Password: A phone number can be used instead of an email as a primary account confirmation mechanism. The user will receive a mobile OTP via sms with which they can verify that they control the phone number. You must enter your own twilio credentials on the auth settings page to enable sms confirmations.

Login Via SMS OTP: SMS OTPs work like magic links, except you have to provide an interface for the user to verify the 6 digit number they receive. You must enter your own twilio credentials on the auth settings page to enable SMS-based Logins.

Verify An SMS OTP: Once the user has received the OTP, have them enter it in a form and send it for verification. You must enter your own twilio credentials on the auth settings page to enable SMS-based OTP verification.

Log In With Third Party OAuth: Users can log in with Third Party OAuth like Google, Facebook, GitHub, and more. You must first enable each of these in the Auth Providers settings here. View all the available Third Party OAuth providers. After they have logged in, all interactions using the Supabase JS client will be performed as "that user". Generate your Client ID and secret from: [Google](#), [GitHub](#), [Gitter](#), [Facebook](#), [Bitbucket](#).

User: Get the JSON object for the logged in user.

Forgotten Password Email: Sends the user a log in link via email. Once logged in you should direct the user to a new password form. And use "Update User" below to save the new password.

Update User: Update the user with a new email or password. Each key (email, password, and data) is optional.

Log Out: After calling log out, all interactions using the Supabase JS client will be "anonymous".

Send A User An Invite Over Email: Send a user a passwordless link which they can use to sign up and log in. After they have clicked the link, all interactions using the Supabase JS client will be performed as "that user". This endpoint requires you use the `service_role_key` when initializing the client, and should only be invoked from the server, never from the client.

JavaScript Code Snippets:

USER SIGNUP

```
let { data, error } = await supabase.auth.signUp({
  email: 'someone@email.com',
  password: 'pyNIRYfQouguMTcObx1'
})
```

USER LOGIN

```
let { data, error } = await supabase.auth.signInWithPassword({
  email: 'someone@email.com',
  password: 'pyNIRYfQouguMTcObx1'
})
```

USER LOGIN

```
let { data, error } = await supabase.auth.signInWithToken({
  email: 'someone@email.com'
})
```

PHONE SIGNUP

```
let { data, error } = await supabase.auth.signUp({
  phone: '+13334443333',
  password: 'some-password'
})
```

PHONE LOGIN

```
let { data, error } = await supabase.auth.signInWithOtp({
  phone: '+13334443333'
})
```

VERIFY PIN

```
let { data, error } = await supabase.auth.verifyOtp({
  phone: '+13334443333',
  token: '123456',
  type: 'sms'
})
```

THIRD PARTY LOGIN

```
let { data, error } = await supabase.auth.signInWithOAuth({
  provider: 'github'
})
```

GET USER

```
const { data: { user } } = await supabase.auth.getUser()
```

PASSWORD RECOVERY

```
let { data, error } = await supabase.auth.resetPasswordForEmail(email)
```

UPDATE USER

```
const { data, error } = await supabase.auth.updateUser({
  email: 'new@email.com',
  password: 'new-password',
  data: { hello: 'world' }
})
```

USER LOGOUT

```
let { error } = await supabase.auth.signOut()
```

INVITE USER

```
let { data, error } = await
supabase.auth.admin.inviteUserByEmail('someone@email.com')
```

ANNEXE

Doc sur la lecture des données de la table page

Read rows

To read rows in `page`, use the `select` method.

[Learn more](#)

READ ALL ROWS

```
let { data: page, error } = await
supabase
  .from('page')
  .select('*')
```

READ SPECIFIC COLUMNS

```
let { data: page, error } = await
supabase
  .from('page')

  .select('some_column,other_column')
```

READ REFERENCED TABLES

```
let { data: page, error } = await
supabase
  .from('page')
  .select(`
    some_column,
    other_table (
      foreign_key
    )
  `)
```

WITH PAGINATION

```
let { data: page, error } = await
supabase
  .from('page')
  .select('*')
  .range(0, 9)
```

Filtering

Supabase provides a wide range of filters.

[Learn more](#)

WITH FILTERING

```
let { data: page, error } = await
supabase
  .from('page')
  .select("*")
  // Filters
  .eq('column', 'Equal to')
  .gt('column', 'Greater than')
  .lt('column', 'Less than')
  .gte('column', 'Greater than or
equal to')
  .lte('column', 'Less than or equal
to')
  .like('column', '%CaseSensitive%')
  .ilike('column',
'%CaseInsensitive%')
  .is('column', null)
  .in('column', ['Array', 'Values'])
  .neq('column', 'Not equal to')
  // Arrays
  .contains('array_column',
['array', 'contains'])
  .containedBy('array_column',
['contained', 'by'])
```

ANNEXE

```
text tree table
cacheVersion : 10
page : {
  id : 6ccb4814-c671-4f83-a320-b49a37f24761
  linkId : 6ccb4814-c671-4f83-a320-b49a37f24761
  paths : { 2 props }
  name : userDetails
  category : value
  meta : { 5 props }
  title : { 0 props }
  favicon : null
  metaImage : null
  langs : [ 1 item ]
  hideFromSitemap : false
  headScripts : null
  bodyScripts : null
  cmsDataSetPath : null
  workflows : [ 1 item ]
  0 : {
    id : 69e1bc2c-375a-4f6b-bd3c-67360614cf7a
    _vts : 1698767340034
    name : Secure page
    actions : { 1 prop }
    trigger : before-collection-fetch
    isTrusted : true
    firstAction : c0dda0f9-5567-46c6-a3f3-5c7678bdf6fb
  }
}
folder : admin/
updatedAt : 2023-10-31T16:02:42.485Z
pageUserGroups : [ 0 items ]
sections : [ 3 items ]
  0 : {
    uid : 0876a1c4-b6a3-4429-882d-bc64b714c799
    linkId : 07eca35e-fd4c-4f1d-94e7-8770f8913cd6
    sectionTitle : Sidemenu
  }
  1 : { 3 props }
  2 : { 3 props }
}
scripts : { 3 props }
}
sections : { 3 props }
wwObjects : { 126 props }
collections : [ 3 items ]
variables : [ 20 items ]
formulas : [ 0 items ]
workflows : [ 6 items ]
libraryComponents : [ 0 items ]
}
```

