

PA DATA VIEWER

DOCUMENTATION PAR
JORDAN
CAMPILLO

Mai / Juin
2025



PA DATA VIEWER

- 1.Introduction (3)
- 2.Installation(4-5)
- 3.Utilisation / function (6-7-8-9)
- 4. Architecture(10)
- 5. API Doc.(11-12-13-14)
- 6.Dépannage / FAQ (15)
- 7. Rendu Visuel (16-17)
- 8. Conclusion (18)



1.INTRODUCTION

Cette application permet d'afficher des données médiatiques en fonction des choix de l'utilisateur. Elle repose sur une logique interactive où l'utilisateur sélectionne :

- Un **pays**, qui détermine les **enquêtes** disponibles.
- Une **enquête**, qui filtre les **cibles** correspondantes.
- Une **cible**, qui affiche dans un **tableau**, **des channels liés**.



2. INSTALLATION

Prérequis :

- Node.js (dernier stable)
- NPM ou YARN
- React (avec des librairies comme axios pour les requêtes API)

Les différentes étapes :

1. Cloner le projet avec :

`GIT CLONE "DECISIONADP/PROTOTYPES/STAGEJORDANCAMPILLO AT MAIN · KINESSO/DECISIONADP"`

Puis :

`CD (DANS LE DOSSIER DU PROJET)`

INSTALLATION (2)



Installer les instances :

“ `npm install` ”

Puis démarrer l'application :

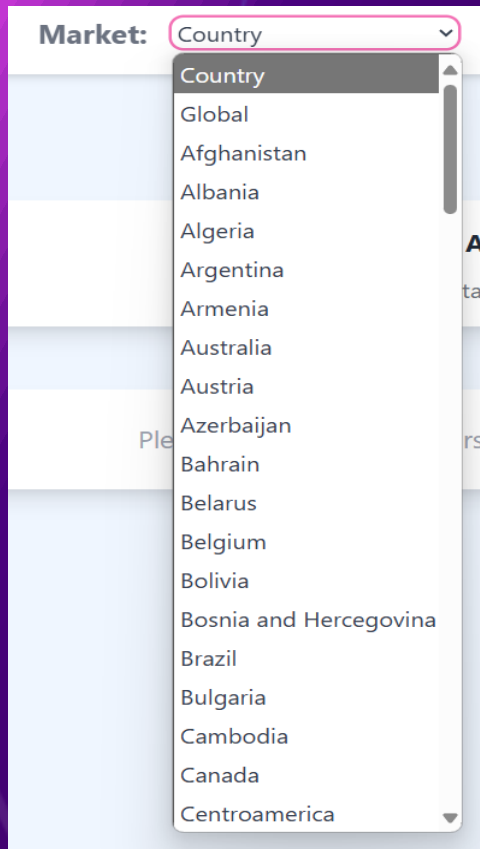
“ `npm start` ”



3.UTILISATION

Sélection du pays (Dropdown) :

Visuel :



Code :

```
<nav className="w-full bg-white shadow-md px-4 py-3 flex items-center justify-between fixed top-0 left-0 z-30 rounded-none">

  <div className="h-10 w-px bg-gray-300 mx-4" /> { /* Trait vertical */}

  <h2 className="text-2xl font-bold text-pink-500">Data Viewer</h2>
  <div className="flex items-center gap-4 flex-1 justify-center">
    <span className="text-gray-500 font-bold text-xl">Market:</span>
    <select
      className="rounded-lg focus:outline-none focus:ring-2 focus:ring-pink-400 text-gray-700"
      value={selectedCountry?.Code || ""}
      onChange={handleChange}
    >
      <option value="">Country</option>
      {countries.map((country) => (
        <option key={country.Code} value={country.Code}>
          {country.Name}
        </option>
      ))}
    </select>
  </div>
  <div style={{ width: 48 }} /> { /* Pour équilibrer l'espace à droite */}
</nav>
```

Ce code me permet d'afficher, via mes variables, la liste des pays disponible dans une API. Lors de la sélection du pays, les éléments sont récupérés via un fichier TSX (TypeScript XML) qui lors donne un type (Country.Code et Country.Name) qui va me permettre d'utiliser les endpoints suivant.

•UTILISATION

CHARGEMENT DES ENQUÊTES ASSOCIÉES AU PAYS CHOISI.

Code :

Dans ce bloc, j'utilise un Endpoint qui me permet, à partir du Country. Code récupéré lors du premier appelle et en fonction du Pays choisie, d'afficher les Survey disponibles pour le pays sélectionné. Tout comme pour le code des pays, avec TSX je viens typé les données récupérés (Survey.ID et Survey. Name).

Visuel :

▼ Survey

ID : 8229 Name : USA - 2024 (For PA App)

Dans notre cas , les Etats-Unis sélectionné.

```
<h2 className="text-lg font-bold text-gray-800 select-none mr-4">Survey</h2>
</div>
<div className={`transition-opacity duration-300 ${showSurvey ? "opacity-100" : "opacity-0 pointer-events-none h-0"}}`>
  {survey && survey.ID ? (
    <ul>
      <li
        key={survey.ID}
        className="bg-blue-50 rounded-lg px-3 py-2 mb-2 flex items-center"
      >
        <span className="text-gray-700 mr-2">
          <strong className="text-pink-500">ID :</strong> {survey.ID}
        </span>
        <span className="text-gray-700">
          <strong className="text-pink-500">Name :</strong> {survey.Name}
        </span>
      </li>
    </ul>
  ) : (
    <p className="text-gray-500">No country selected.</p>
  )
}
</div>
</div>
```

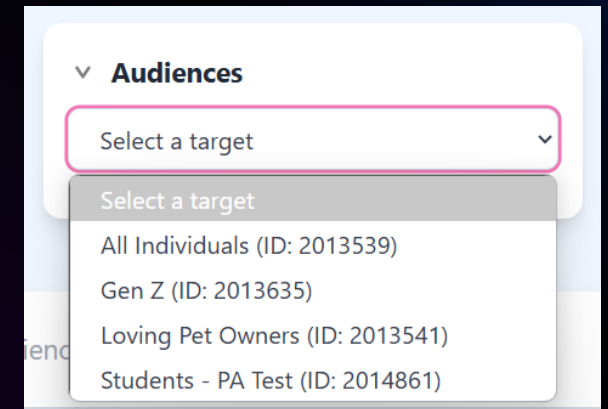

UTILISATION

CHARGEMENT DES CIBLES EN FONCTION DE L'ENQUÊTE CHOISIE :

Code :

```
<h2 className="text-lg font-bold text-gray-800 select-none mr-4">Audiences</h2>
</div>
<div className={`transition-opacity duration-300 ${showTarget ? "opacity-100" : "opacity-0 pointer-events-none h-0"}}>
  {targetList && targetList.length > 0 ? (
    <select
      className="border border-gray-300 px-4 py-2 rounded-lg w-full focus:outline-none focus:ring-2 focus:ring-pink-400 text-gray-700 mb-4"
      value={selectedTarget ?? ""}
      onChange={e => {
        const id = Number(e.target.value);
        setSelectedTarget(id);
        onTargetSelect(id);
      }}
    >
      <option value="" disabled>Select a target</option>
      {targetList.map(target => (
        <option key={target.ID} value={target.ID}>
          {target.Name} (ID: {target.ID})
        </option>
      ))}
    </select>
  ) : (
    <p className="text-gray-500">No target available.</p>
  )
}
```

Ensuite nous avons une liste déroulante avec les différentes cibles disponible pour chaque enquête sélectionnée.



Cibles pour USA-2024

UTILISATION

SÉLECTION DE LA CIBLE → AFFICHAGE DU TABLEAU DES CHANNELS LIÉS.

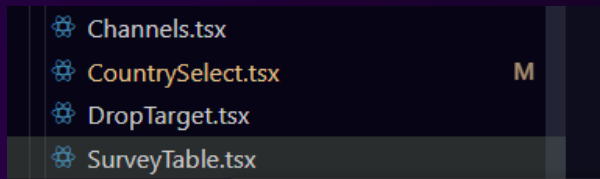
Et pour terminer, je créer un tableau afin d'accueillir les informations du dernier endpoint afin de pouvoir présenter les pourcentages , les noms et autres... toujours en TSX.

Review Channels			
STANDARD CHANNEL ID	NAME	MAX REACH	DECAY
111	Audio-Not specified	95.13%	10.00%
229	Cinema - Not specified	52.86%	10.00%
120	Direct-Not specified	98.56%	10.00%
126	Display-Not specified	99.24%	10.00%
232	Gaming-Not specified	48.34%	10.00%
217	Online Video-Not specified	88.38%	10.00%
170	OOH-Not specified	76.44%	10.00%
180	Print-Not specified	55.90%	10.00%
201	Retail-Not specified	96.29%	10.00%
204	Search-Not Specified	97.02%	10.00%
205	Social	88.81%	10.00%
225	TV-Linear	55.40%	10.00%
220	TV-Streaming	95.34%	10.00%

```
<div className="w-full max-w-5xl mx-auto bg-white rounded-xl shadow-lg p-6">
  <h2 className="text-xl font-bold mb-4 text-gray-800">Review Channels</h2>
  <table className="min-w-full divide-y divide-gray-200">
    <thead className="bg-pink-500">
      <tr>
        <th className="rounded-tl-lg px-4 py-3 text-left text-xs font-semibold text-white uppercase tracking-wider">
        <th className="px-4 py-3 text-left text-xs font-semibold text-white uppercase tracking-wider">Name</th>
        <th className="px-4 py-3 text-left text-xs font-semibold text-white uppercase tracking-wider">Max Reach</th>
        <th className="rounded-tr-lg px-4 py-3 text-left text-xs font-semibold text-white uppercase tracking-wider">
      </tr>
    </thead>
    <tbody className="bg-white divide-y divide-gray-100">
      {channels.map((channel, idx) => (
        <tr key={channel.StandardChannelID} className={idx % 2 === 0 ? "bg-white" : "bg-gray-200"}>
          <td className="px-4 py-2 rounded-l-lg">{channel.StandardChannelID}</td>
          <td className="px-4 py-2">{channel.Name}</td>
          <td className="px-4 py-2">
            {channel.ReachCurve?.MaxReach !== undefined
              ? formatPercentage(channel.ReachCurve.MaxReach)
              : "N/A"}
          </td>
          <td className="px-4 py-2 rounded-r-lg">
            {channel.Decay !== undefined
              ? formatPercentage(channel.Decay)
              : "N/A"}
          </td>
        </tr>
      ))}
    </tbody>
  </table>
</div>
```

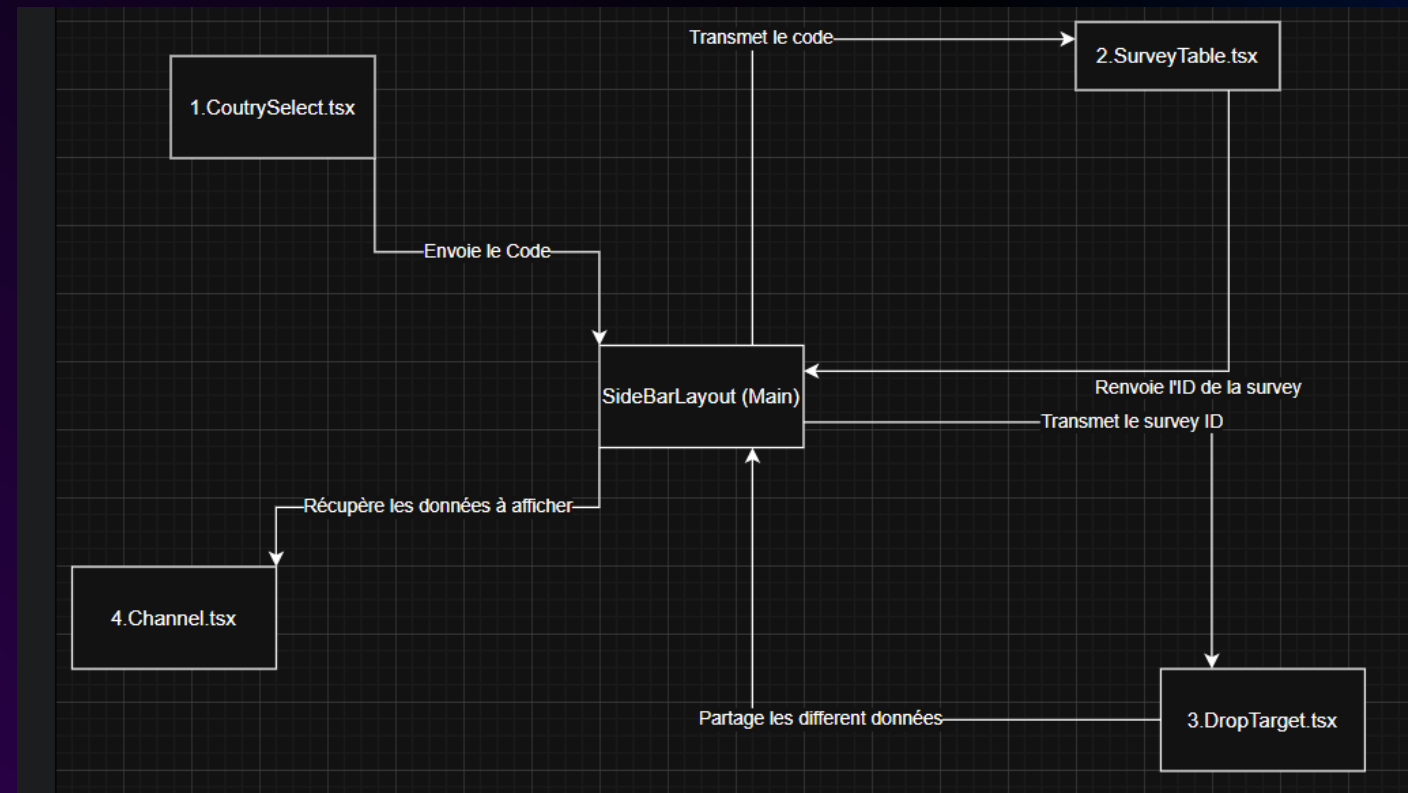
4. ARCHITECTURE

- Pour concevoir cette application, j'ai utilisé plusieurs composant sous React :



Ces différents composants doivent communiquer entre eux pour pouvoir transmettre les différentes informations.

Donc j'utilise un composant « parent » appelé SideBarLayout.



5. API DOCUMENTATION

L'application récupère dynamiquement des données via des requêtes API. Chaque sélection (pays, enquêtes, cibles et les chaînes) déclenche un appel qui retourne les données correspondantes.

Méthode	URL	Description
GET	/CP/RegionsAndCountries	Renvoie les pays.
GET	/CP/Countries/CountryCode/{}/PagentDefaultSurvey	Renvoie les enquêtes disponibles pour ce Pays.
GET	/CP/Surveys/{}/Targets	Renvoie les cibles disponibles pour l'enquêtes choisie.
POST	/CP/Countries/CountryCode/{}/Targets/{}/PagentTouchpointsForCalc	Crée le tableau contenant les différentes chaînes.

{ } = les données
(ID) à ajouté.



Afin que nous puissions utiliser les différents Endpoint sans nécessairement avoir d'énormes blocs de code. On peut créer un « Service ». Une fonction qui me permet de faire mon appel en quelque ligne.

```
export async function fetchData(endpoint: string): Promise<any> {  
  const headers: Record<string, string> = {  
    Accept: "application/json",  
    "x-dt-auth": `${API_ACCESS_TOKEN}`,  
  };  
  
  const fullUrl = endpoint.startsWith("http")  
    ? endpoint  
    : `${API_URL}${endpoint}`; // J'ajoute l'URL de base si nécessaire  
  
  const response = await fetch(  
    fullUrl, {  
      method: "GET",  
      headers  
    });  
  
  // Vérifier si la réponse est OK (statut HTTP 200-299)  
  if (!response.ok) {  
    throw new Error(`Erreur HTTP : ${response.status}`);  
  }  
  return response.json(); // Récupérer directement en JSON  
  console.log("Réponse de l'API :", response.json()); // Afficher la réponse de l'API dans la console  
}
```

Cette fonction permet de définir (dans un fichier à part) de définir la méthode, un token ou une clé d'accès ainsi que l'URL de l'API en une simple fonction.

N'oubliez pas de définir en amont (dans le fichier de votre fonction) votre clé d'accès ainsi que la base de votre URL

```
const API_ACCESS_TOKEN = "Votre clé ou token d'accès"; // Remplacez par votre clé ou token d'accès
const API_URL = "https://Jusqu'au-Premier-Slash(/)"; // l'URL de votre API
```

Le résultat final sur l'un de composant de l'application ressemble donc à ça :

```
try {
  const data = await fetchData(`/cp/Countries/CountryCode/${countryCode}/PageDefaultSurvey`); // J'appelle ma fonction
  console.log("Réponse de l'API :", data); // Afficher la réponse de l'API dans la console
  if (data) {
    setSurvey(data); // Mettre à jour l'état avec les données récupérées
    console.log("Survey récupéré :", data);
    const surveyID = data.ID; // Récupérer l'ID du survey
    setSelectedSurvey(surveyID); // Mettre à jour l'état avec le pays sélectionné
    onSurveySelect(surveyID); // Appeler la fonction de rappel avec le code du pays sélectionné
  }
} catch (error) {
  // console.error("Erreur lors de la récupération des données :", error);
  // Gérer l'erreur ici, par exemple en affichant un message d'erreur à l'utilisateur
}

getDefaultSurvey(); // Appeler la fonction pour récupérer les données
}, [countryCode]); // Exécuter une seule fois au montage du composant
```

Si tout ce passe
correctement dans votre
console de votre navigateur,
vous devriez voir ceci :

```
► Object i  
  CompanyID: 3  
  CompanyName: "Mediabrandz"  
  CountryCode: "us"  
  CountryID: 21  
  CountryName: "United States"  
  CustomHVA: null  
  DateCreated: "2025-04-29"  
  HasPathwaysData: false  
  HasRecodeData: false  
  HasScoutData: false
```

Cela signifie que l'appelle
est bien partie et que l'on
récupère bien nos données.

Voici un petit tableau regroupant quelque
erreur en cas de problème ainsi que leur
cause :

Requête	Code d'erreur	Cause
GET	404	La ressource demandée n'existe pas
POST	400	Le corps de la requête est mal formé
GET	500	Problème coté serveur
POST	409	La donnée envoyée existe déjà
GET	401	L'utilisateur n'est pas authentifié
POST	422	Le format de données est invalide

6. DÉPANNAGE

Problème	Solution
Aucunes données affichées	Vérifié les logs et les données de l'API
Erreur (404 not found)	Vérifié si l'ID envoyé est valide.
L'application est lente	Optimiser les requêtes et limiter les renders inutiles
Problèmes CORS	Configurer un PROXY ou les autorisations du Back.

FAQ

• **Comment ajouter un nouveau pays à l'application ?**

→ Il faut modifier l'API, du côté serveur.

• **Puis-je filtrer les channels affichés ?**

→ Oui, en ajoutant des paramètres supplémentaires aux requêtes API ou en filtrant côté front-end.

• **Que faire en cas d'erreur 500?**

→ L'erreur vient du serveur. Il faut vérifier les logs backend et essayer plus tard.

• **Comment optimiser les appels API ?**

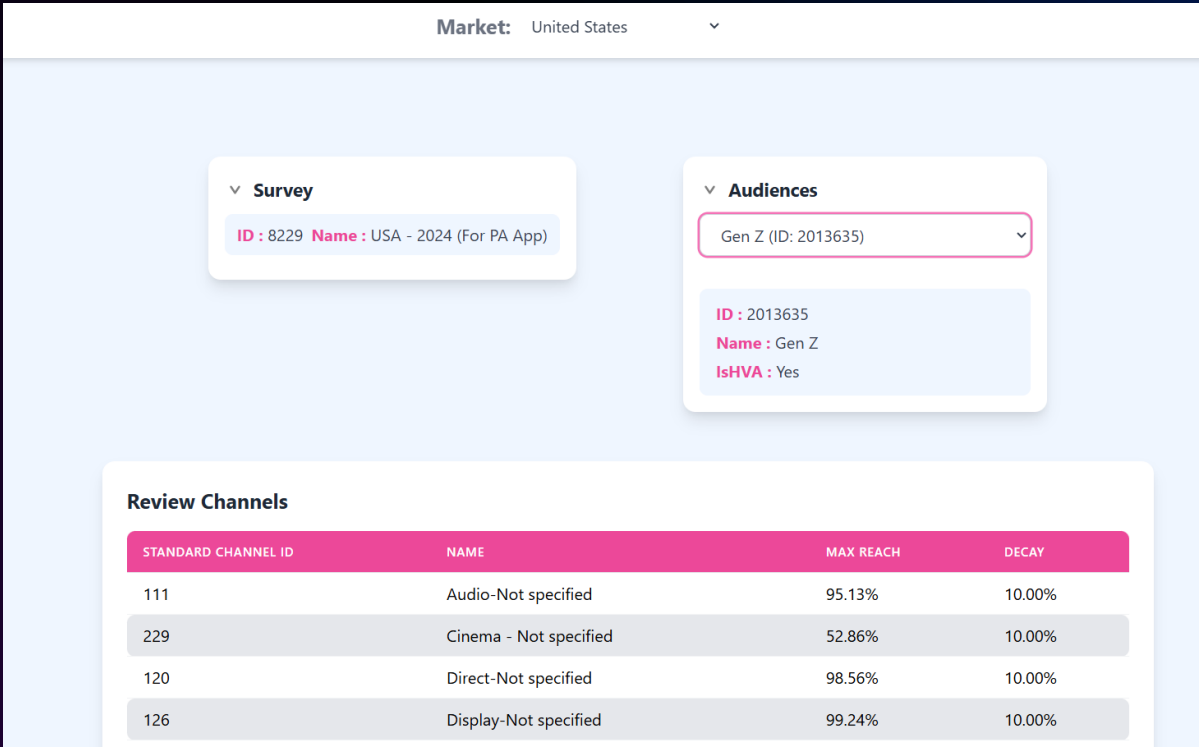
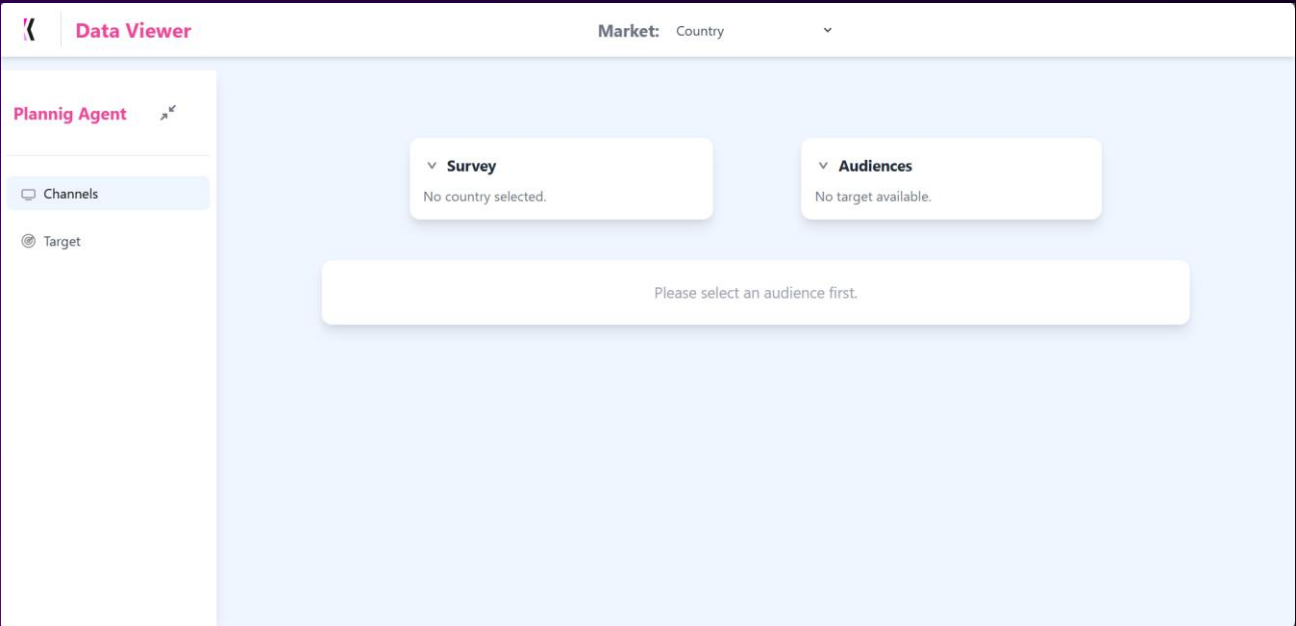
→ En utilisant **React Query** ou **use Memo** pour éviter les requêtes répétitives

7. RENDU VISUEL

Voici un rendu visuel de la première page
avec et sans pays sélectionner :

Avec :

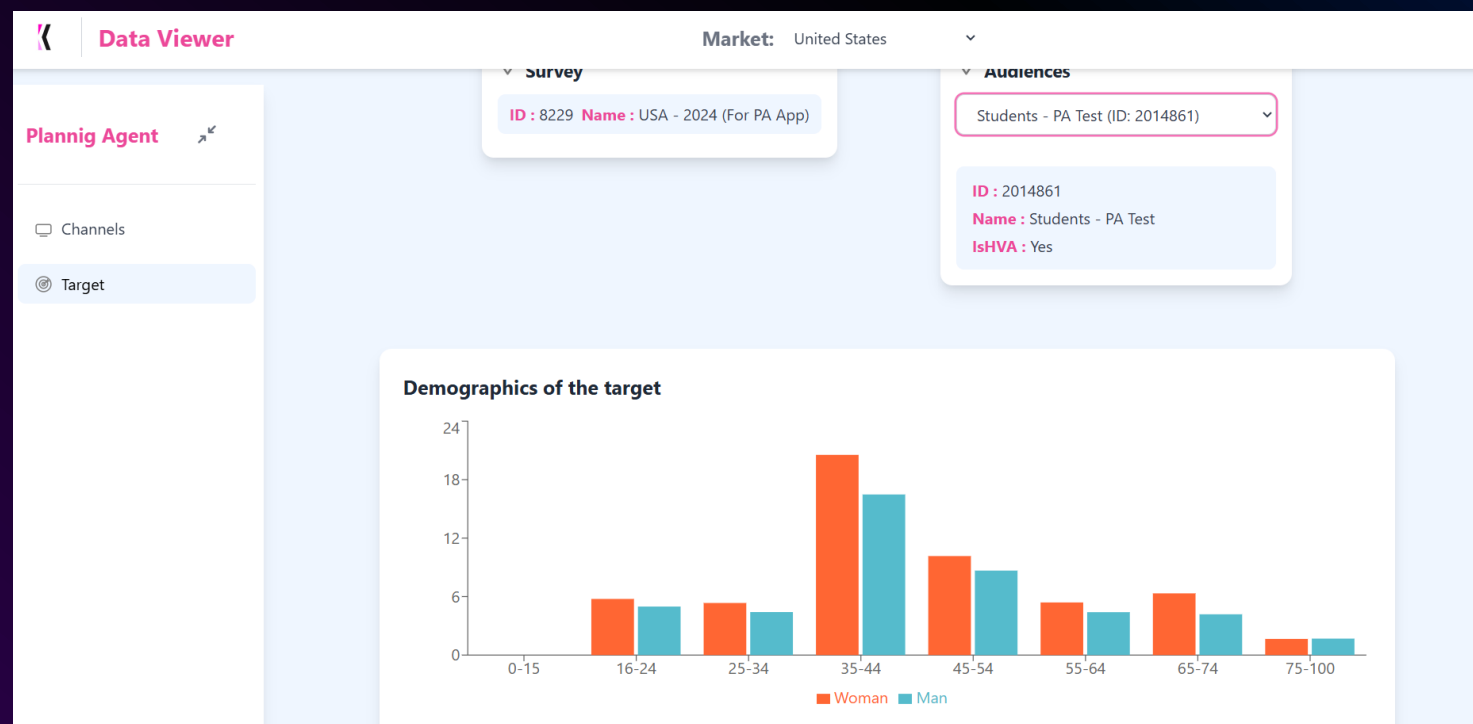
Sans :



Et enfin la deuxième page, où le tableau de statistique c'est transformer en graphique, représentant différent pourcentage en fonction des sexes et des tranches d'âge :

L'onglet à gauche est la « SideBar » qui permet de naviguer sur 2 pages différente sans changer de page.

Il est le parent de tous les composants présent dans l'application React (SidebarLayout).



8.CONCLUSION

1- Ce projet React apporte une solution efficace et dynamique pour l'affichage de données médiatiques en fonction des choix de l'utilisateur. Grâce à une **architecture bien structurée**, une **gestion optimale des états** et une **interface intuitive**, il permet une exploration fluide des informations pertinentes.

2- L'application est conçue pour être **fonctionnelle et claire**, répondant aux besoins des utilisateurs sans nécessiter de modifications futures. Cependant, elle pourrait servir de base à d'éventuels développements, comme une amélioration de l'interface, une optimisation des requêtes ou une intégration de nouvelles fonctionnalités.

3- Enfin, cette documentation vise à offrir une **compréhension complète** du fonctionnement du projet, de son **architecture technique** à son **interaction utilisateur**. Avec ces éléments, toute personne découvrant le projet pourra facilement le prendre en main et comprendre sa structure.

 **Merci d'avoir exploré cette documentation et bonne découverte du projet !**

MERCI D'AVOIR LU

Campillo Jordan

0648436340

Jduquesne17@gmail.com

