Table des matières

Liste des tableaux

Liste des figures

1. Introduction

Le sujet de ce mémoire est de trouver le meilleur moyen de représenter de façon fluide des données multidimensionnelles dans un client riche. Afin de comprendre ce but, il est utile de le décortiquer mot par mot.

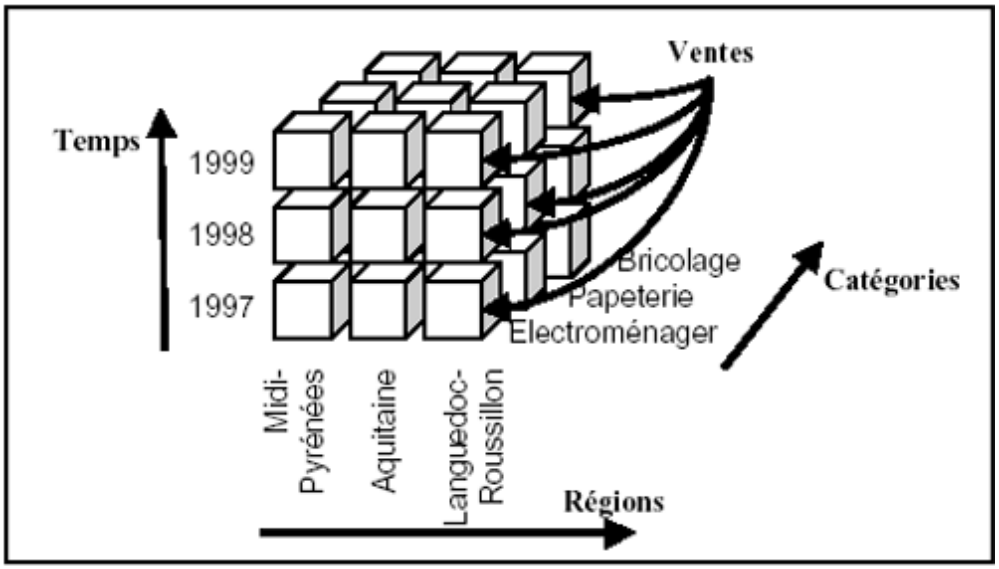
Un client riche est, caricaturalement, un client léger avec l'avantage d'un client lourd. Dans notre cas, nous utilisons un navigateur web (client léger) pour afficher une application complexe de type client-serveur, sans qu'aucun client lourd ne nécessite d'être installé.

La représentation fluide signifie que l'application se doit d'être la plus ergonomique possible, soit rapide et user-friendly.

Les données multidimensionnelles sont un ensemble de données souvent représentées sous forme de cube dont une cellule illustre l'intersection de 2 dimensions.

Une dimension peut illustrer le temps, un lieu ou une catégorie.

Figure 1 : Présentation schématique d'une donnée multidimensionnelle

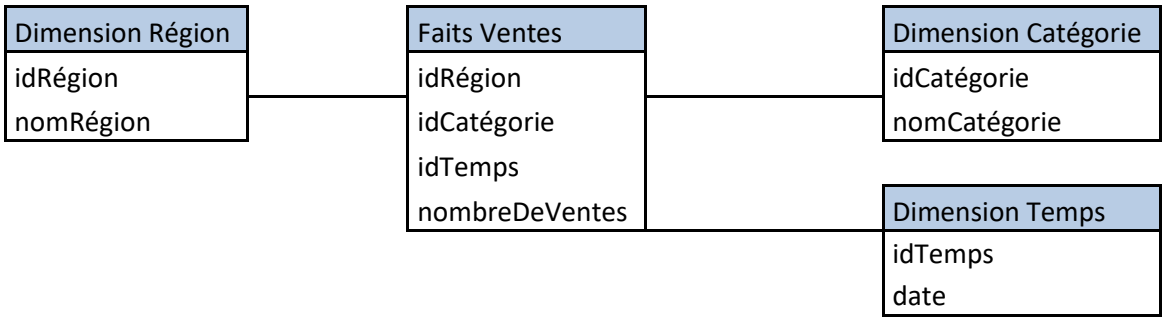


Par exemple, un système de backup possède les fichiers sauvegardés comme première dimension, puis un historique des modifications comme seconde dimension.

Pour visualiser des données multidimensionnelles, un tableau croisé dynamique est généralement suffisant. Dans le cas du schéma ci-dessus, voici comment serait présenté le tableau croisé avec les dimensions de catégories et de régions sur l'axe Y et celle du temps sur l'axe X.

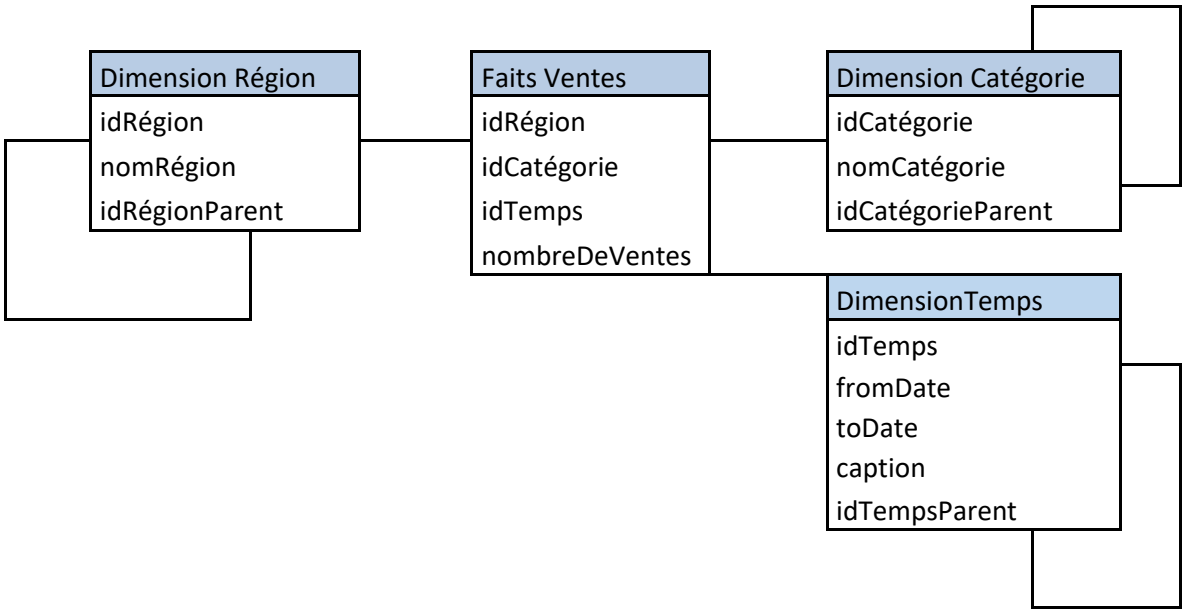
		1995	1996	1997
Frais	IdF	220	265	284
	Province	225	245	240
Liquide	IdF	163	152	145
	Province	187	174	184

Dans le cadre de notre projet, un tableau croisé dynamique n'est pas suffisant. Dans l'exemple ci-dessus, les dimensions de temps et de catégories ne sont pas complexes, or les dimensions utilisées dans notre projet possèdent une structure hiérarchique, sous forme d'arbre : les catégories peuvent avoir des sous-catégories. Pour mieux comprendre cette différence, voici un diagramme représentant les données multidimensionnelles du tableau croisé ci-dessus.



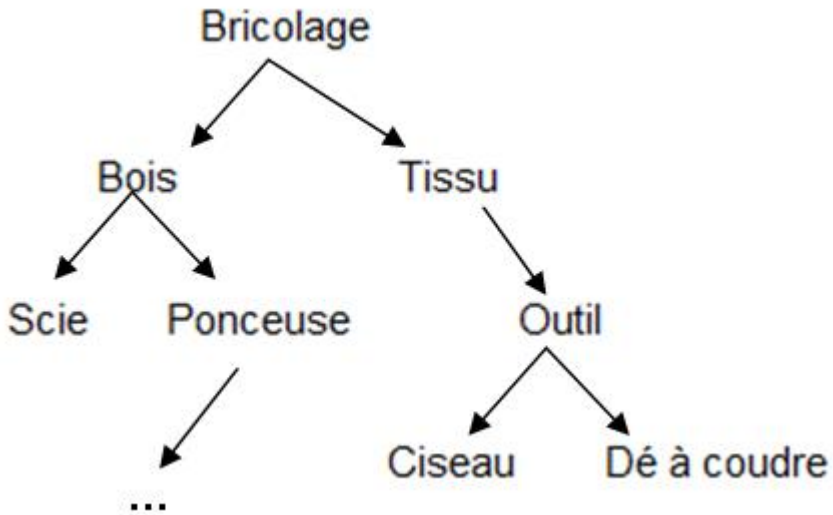
Ici, les dimensions région, catégorie et temps n'ont pas d'éléments fils.

En comparaison, les dimensions temps, région et catégorie possèdent des éléments fils ce qui offre plus de liberté quant à la précision de celles-ci.



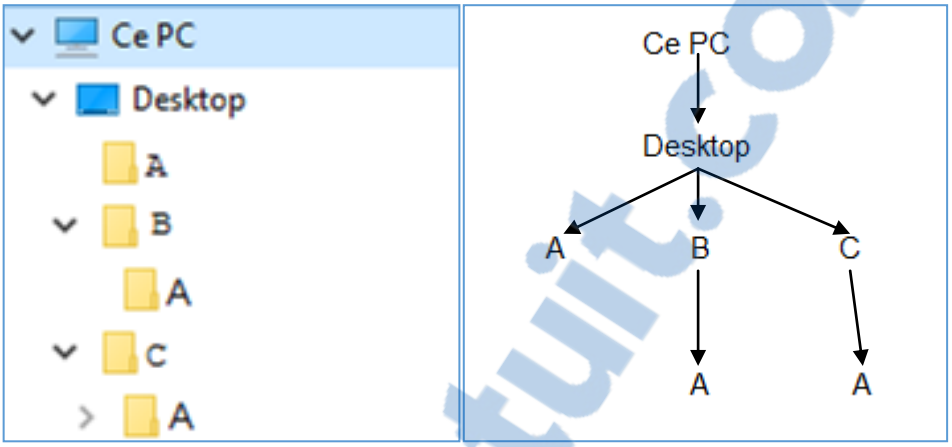
Un tableau croisé de ce type n'est donc pas suffisant pour afficher ces dimensions hiérarchiques, car une dimension pourrait contenir plusieurs niveaux sans limite, au lieu d'un nombre fixe comme dans le cas simple.

Figure 2 : Dimension catégorie sous forme de hiérarchie



Présenter des données structurées en arbre est courant en informatique, par exemple l'arborescence des fichiers de notre ordinateur est sous forme d'arbre.

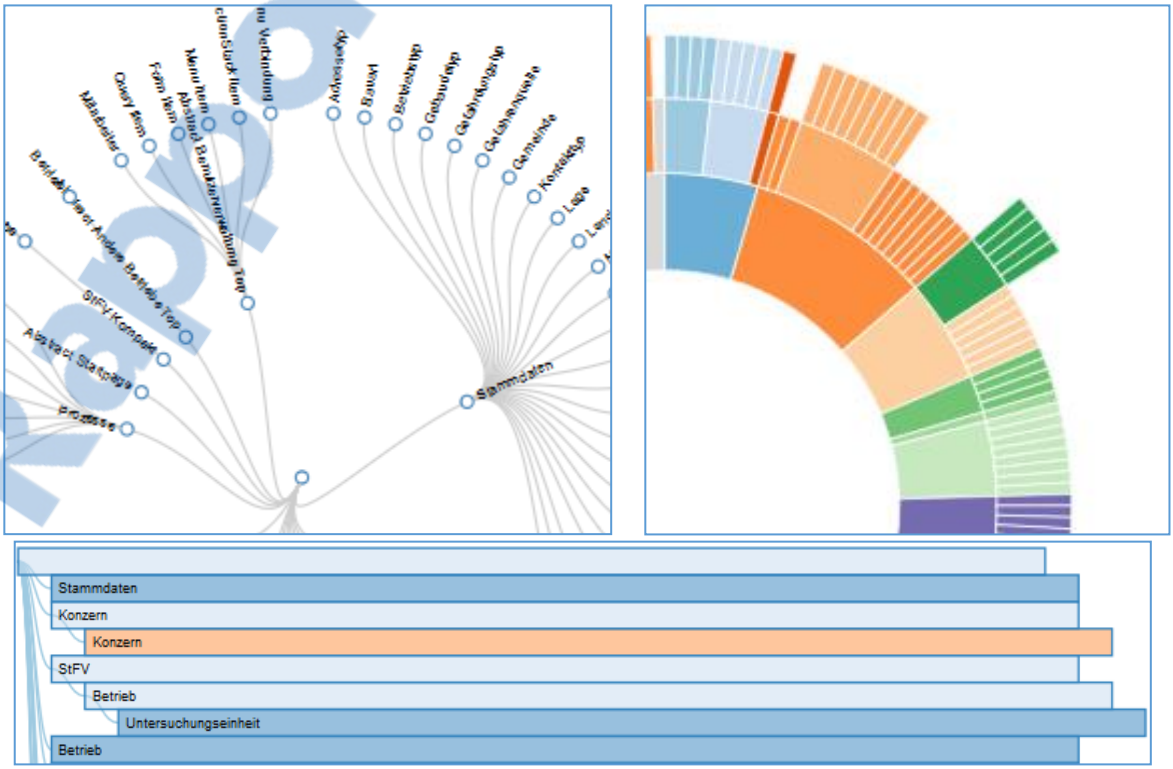
Figure 3 : Présentation d'une arborescence



Cette représentation possède l'avantage de pouvoir permettre d'identifier rapidement une information en profondeur de manière récursive, d'autant plus si l'on ajoute un identifiant à chacun des nœuds ou feuille représentant son chemin : l'identifiant de la dernière feuille A est *PC/Desktop/C/A.*

Il existe de multiples manières de représenter des arbres.

Figure 4 : Exemples de représentation d'une arborescence de données avec la librairie D3.js



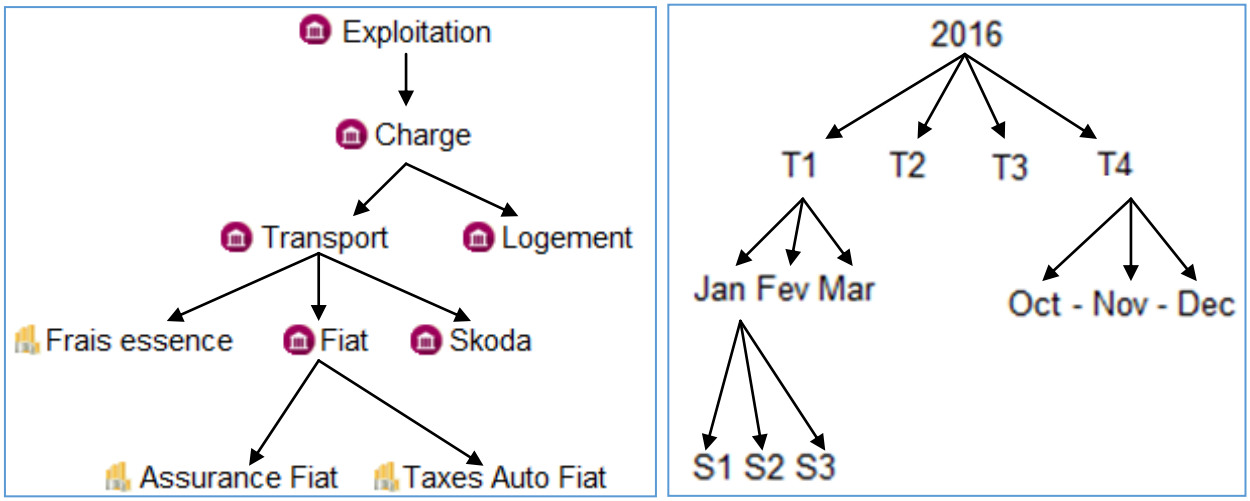
La problématique majeure du projet est qu'il faut gérer deux dimensions structurées de façon hiérarchiques au sein d'un tableau croisé.

2. Besoins et réflexion

2.1 Des données complexes à afficher

Dans le cadre de ce projet, Les données multidimensionnelles sont des écritures comptables dont les dimensions sont le plan comptable et les dates.

Figure 5 : Exemples d'arbres du prototype



Le plan comptable possède des groupes, chacun d'eux possède des sous-groupes et/ou des comptes.

Le calendrier est hiérarchisé en autant d'unité de temps nécessaires : Année – Trimestre - Mois – Semaine – etc.

Une écriture comptable contient une date et appartient à un compte. La mesure à représenter est le calcul du montant total des écritures d'un groupe ou d'un compte dans une fourchette de temps donnée.

Cette mesure doit pouvoir être calculée puis affichée le plus simplement possible dans une fenêtre permettant la navigation dans le plan comptable et dans les unités de temps.

L'outil doit pouvoir interpréter les données sous format JSON. Les données du projet sont rassemblées dans 4 fichiers différents.

Ecritures.json


```
"name": "Ecritures",
"Entry": [
    {
        "entryId": 1,
        "caption": "Courses Migros",
        "date": "2016-05-05T00:00:00",
        "accountId": 8,
        "amount": 83
    },
    {
        "entryId": 2,
        "caption": "Dr Maboule",
        "date": "2016-05-07T00:00:00",
        "accountId": 10,
        "amount": 215
    },
    {
        "entryId": 3,
        "caption": "Courses Coop",
        "date": "2016-05-07T00:00:00",
        "accountId": 8,
        "amount": 65.3
    },
```


Comptes.json


```
"name": "Comptes",
"Account": [
    {
        "accountId": 1,
        "caption": "Assurance Fiat",
        "groupId": 1111
    },
    {
        "accountId": 2,
        "caption": "Assurance Skoda",
        "groupId": 1112
    },
    {
        "accountId": 3,
        "caption": "Taxes Auto Fiat",
        "groupId": 1111
    },
```


Calendrier.json


```
"name": "MasqueCalendrier",
"TimeAggregate": {
    "caption": 2016,
    "timeFrom": "2016-01-01T00:00:00",
    "timeTo": "2016-12-31T23:59:59",
    "TimeAggregate": [
        {
            "caption": "1er trimestre",
            "timeFrom": "2016-01-01T00:00:00",
            "timeTo": "2016-03-31T23:59:59",
            "TimeAggregate": [
                {
                    "caption": "Janvier",
                    "timeFrom": "2016-01-01T00:00:00",
                    "timeTo": "2016-01-31T23:59:59"
                },
                {
                    "caption": "Février",
                    "timeFrom": "2016-02-01T00:00:00",
                    "timeTo": "2016-02-29T23:59:59"
                },
                {
                    "caption": "Mars",
                    "timeFrom": "2016-03-01T00:00:00",
                    "timeTo": "2016-03-31T23:59:59"
                }
            ]
        },
```


Groupes.json


```
"name": "PlanComptable",
"Group": {
    "caption": "Exploitation",
    "order": 1,
    "groupId": 1,
    "Group": {
        "caption": "Charges",
        "order": 1,
        "groupId": 11,
        "Group": [
            {
                "caption": "Transport",
                "order": 1,
                "groupId": 111,
                "Group": [
                    {
                        "caption": "Fiat",
                        "order": 1,
                        "groupId": 1111
                    },
                    {
                        "caption": "Skoda",
                        "order": 2,
                        "groupId": 1112
                    }
                ]
            },
```

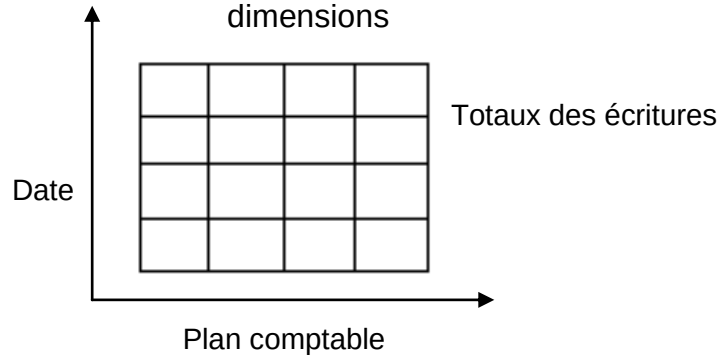
2.2 Choix de la représentation des dimensions hiérarchiques

Le défi principal de ce projet est d'obtenir une interface dynamique, ergonomie et fluide pour afficher les données évoquées précédemment.

Tout d'abord, nous avons cherché un moyen de représenter les dimensions hiérarchiques "plan comptable" et "temps".

Il doit être possible de naviguer dans l'arbre de chacune de ces deux dimensions pour sélectionner l'intervalle de temps (trimestre, année..) et un élément du plan comptable pour filtrer les écritures et calculer les montants totaux de celles-ci.

Figure 6 : Schéma de la représentation des dimensions



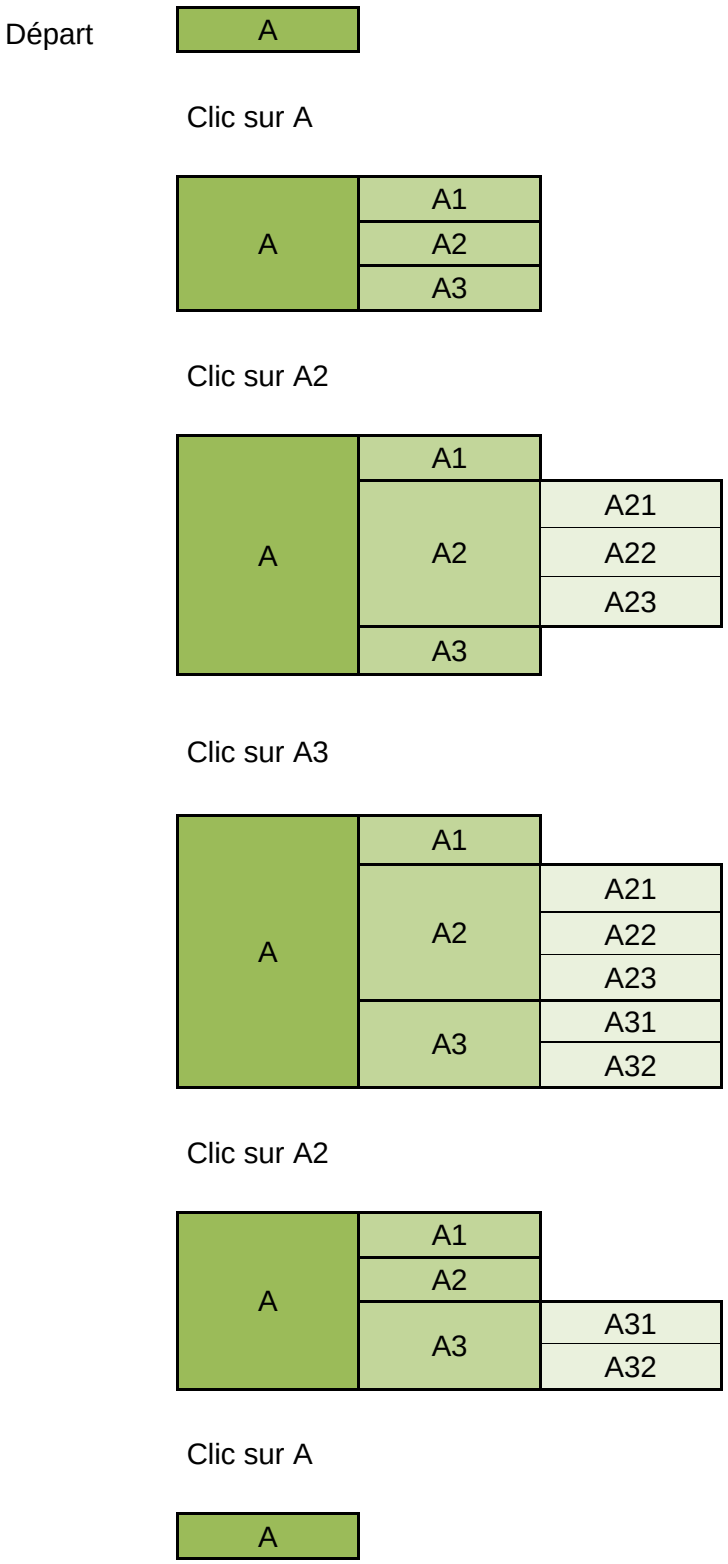
En raison de cette contrainte, il a été choisi d'afficher les dimensions hiérarchiques comme des partitions ouvrables et fermables.

En cliquant sur la case représentant une dimension, celle-ci doit pouvoir se partitionner pour laisser apparaître d'éventuels éléments fils.

En cliquant à nouveau sur case développée, les éléments fils disparaissent

La figure 8 explique l'enchainement de ces actions et leur répercussion sur l'affichage des dimensions.

Figure 7 : Explication du développement des partitions



En intégrant dans un tableau croisé les dimensions hiérarchiques, les entêtes des lignes et des colonnes peuvent être partitionnées.

Figure 8 : Représentation schématique du tableau croisé avec les entêtes partitionnables

		B			
		B1		B2	B3
		B11	B12		
A	A1	1	2	3	4
	A2	5	6	7	8
	A3	9	0	1	2

Sur l'axe vertical, les groupes et les comptes sont affichés. Sur l'axe horizontal, les unités de temps.

Cette représentation a l'avantage d'être familière et simple d'utilisation, car il suffit de cliquer sur les cases pour descendre ou remonter dans l'arbre et mettre à jour les montants. Cependant, la taille du tableau peut rapidement devenir problématique si les dimensions hiérarchiques ont beaucoup d'éléments fils.

Figure 9 : Démonstration du problème de taille où le tableau dépasse la zone de travail

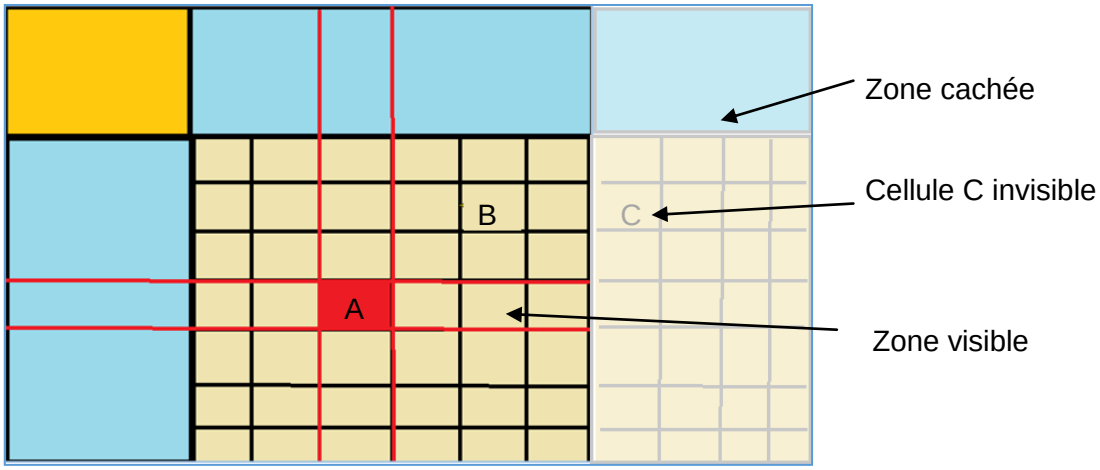
		B															
		B1		B2		B3		B4		B5		B6		B7		B8	
		B11	B12	B11	B12	B11	B12	B11	B12	B11	B12	B11	B12	B11	B12	B11	B12
A	A1	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
	A2	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8
	A3	9	0	1	2	9	0	1	2	9	0	1	2	9	0	1	2
	A4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
	A5	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8
	A6	9	0	1	2	9	0	1	2	9	0	1	2	9	0	1	2

Pour éviter ce problème, une solution est de n'afficher qu'une vue partielle du tableau et de laisser l'utilisateur s'y déplacer pour afficher les informations dont il a besoin.

2.3 Affichage partiel du tableau

En fixant une taille maximale en hauteur et en largeur pour la partie visible du tableau, seul une partie des cellules et des entêtes sont affichés.

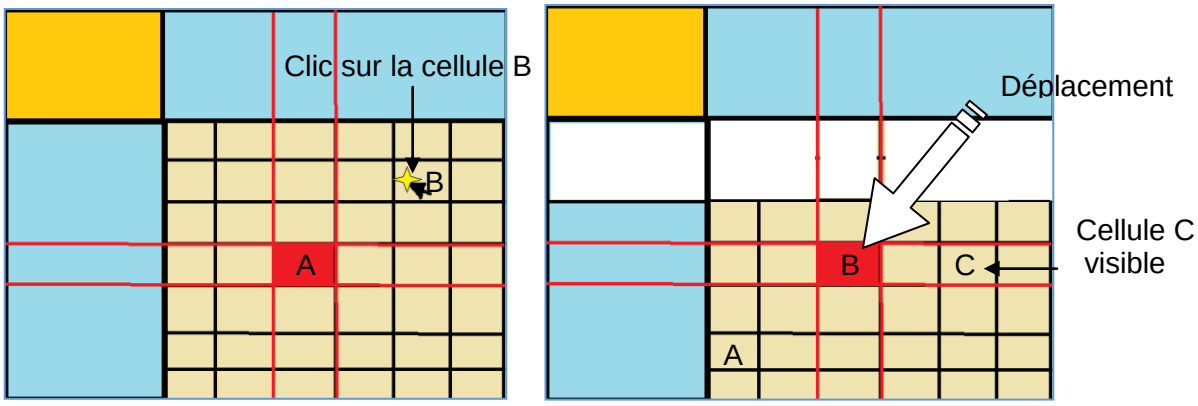
Figure 10 : Affichage partiel du tableau



L'utilisateur doit disposer d'un moyen simple et rapide d'accéder aux cellules cachées.

Lorsque l'utilisateur clique sur une des cellules, le tableau entier se déplace pour que la cellule en question se retrouve au centre de la zone visible, permettant ainsi à l'utilisateur de lire les données des cellules avoisinantes.

Figure 11 : Déplacement dans le tableau



Les entêtes suivent le déplacement des cellules. Même si toutes les cellules du tableau ne sont pas affichées, elles sont rapidement atteignables.

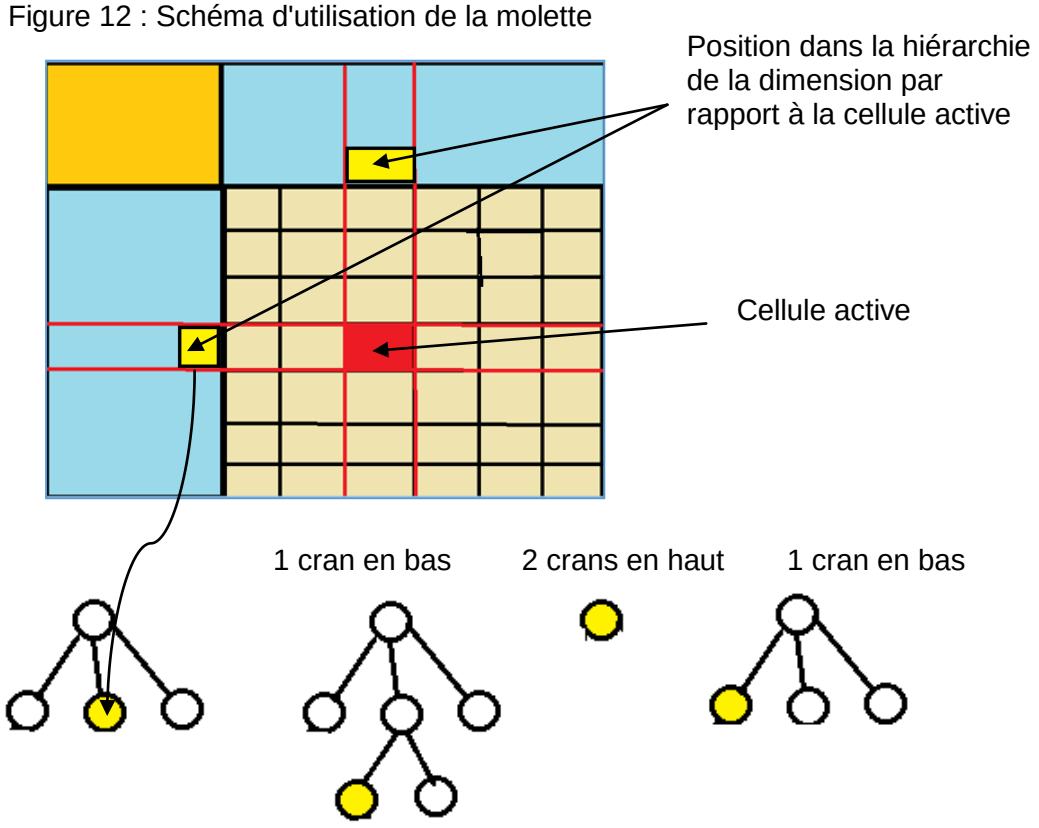
2.4 Utilisation de la molette

Quand l'utilisateur clique sur l'entête des lignes ou des colonnes pour développer ou réduire les dimensions hiérarchiques, des informations apparaissent ou disparaissent dans le tableau. Là cellule active peut disparaître ou ne plus être au centre. Pour que l'utilisateur ne perde pas sa position dans le tableau, il faut qu'il puisse développer ou réduire les dimensions sans cliquer dessus.

Une idée intuitive est d'utiliser la molette de la souris, car celle-ci est souvent utilisée pour "zoomer" ou "dézoomer". Une rotation de la molette vers le bas développe les dimensions de la cellule active et une rotation vers le haut les réduit.

Ci-dessous, un exemple de mouvement dans l'arbre d'une dimension hiérarchique à l'aide de la molette. Lors d'un développement d'une branche, tous les fils de celle-ci sont visibles, mais le premier est sélectionné. Lors d'une fermeture d'une branche, tous les fils de la branche parente sont cachés.

Figure 12 : Schéma d'utilisation de la molette



Afin de rendre l'action indépendante pour chaque dimension, "Shift + Molette" est utilisé pour naviguer dans la dimension calendrier et la molette seule pour la dimension du plan comptable.

2.5 Affichage des écritures

Les cellules du tableau affichent le total des montants des écritures comptables d'un groupe ou d'un compte pour une fourchette de date donnée. Comme les écritures comptables ne sont pas visibles directement, il faut un moyen de les afficher.

Le tableau étant de taille fixe, il est difficile de dégager de l'espace pour présenter les écritures comptables. De ce fait, la solution la moins contraignante est de mettre en place une nouvelle fenêtre d'affichage où sont listées les écritures comptables de la cellule active. Cette fenêtre est amovible afin de permettre à l'utilisateur de choisir sa position et ainsi palier au problème d'espace.

Lors d'un changement de cellule active, les écritures comptables se mettent à jour dans cette fenêtre.

3. Recherche de solutions existantes

Il est intéressant de se renseigner pour voir s'il existe une solution qui répond ou se rapproche des besoins présentés précédemment. Les solutions existantes qui satisfont aux mieux les besoins sont celles qui proposent une gestion de tableaux croisés dynamiques.

3.1 Solutions gratuites

3.1.1 PivotTable JS

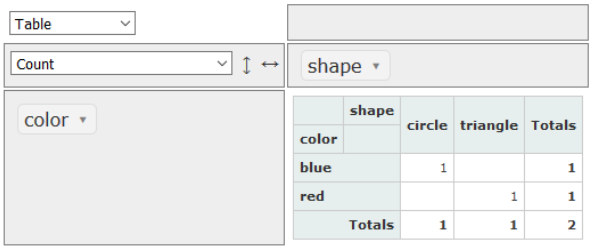
PivotTable.js est une implémentation en javascript de tableau croisé créée par Nicolas Kruchten.

Le tableau croisé est avec ou sans interface utilisateur :

Figure 14 : Tableau sans IU

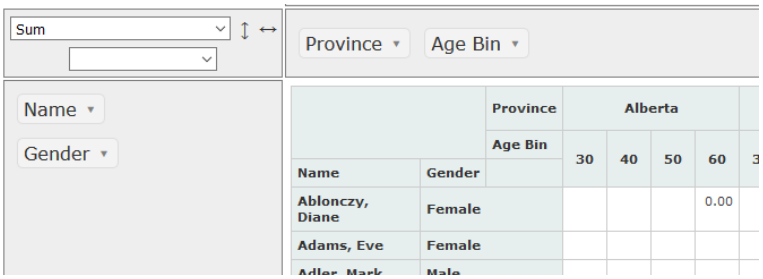
color	shape	circle	triangle	Totals
blue		4		4
red			6	6
Totals		4	6	10

Figure 13 : Tableau avec IU



Les dimensions peuvent être ajoutées ou supprimées à souhait pour chaque axe dans le cas ou l'interface utilisateur est disponible.

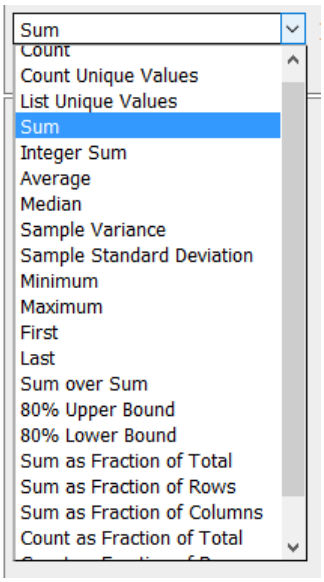
Figure 16 : Exemple avec 2 dimensions par axes



De plus, il est facile de modifier le calcul des données croisées avec une liste d'opération très complète.

Il est aussi possible de trier les informations par ordre alphabétique ou numérique.

Figure 15 : Opérations possible



Les avantages de cet outil sont d'abord ses multiples possibilités d'importation de données : CSV, JSON, Array, une table sous forme html.

Ensuite, il est très facile de déplacer ou combiner les dimensions sur les axes du tableau.

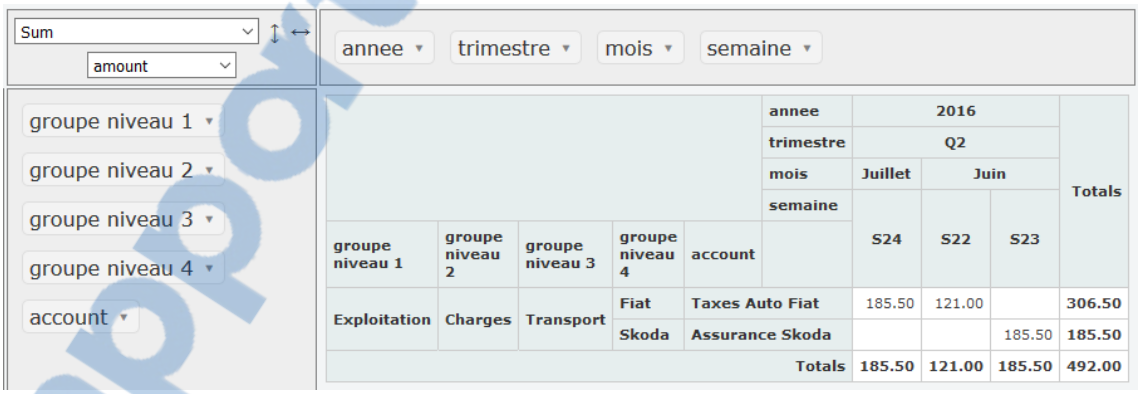
L'inconvénient majeur de cette solution est la taille du tableau qui prend devient rapidement ingérable avec de trop nombreuses données.

Un autre inconvénient est que cette solution ne gère pas les données de notre projet. Il faut rassembler celles-ci en un seul fichier et "aplatir" les dimensions hiérarchiques ce qui fait perdre leur intérêt.

Exemple de nos données multidimensionnelles après conversion


```
"account": "Taxes Auto Fiat",
"groupe niveau 1" : "Exploitation",
"groupe niveau 2" : "Charges",
"groupe niveau 3" : "Transport",
"groupe niveau 4" : "Fiat",
"entryId": 23,
"caption_ecriture": "Impôt Fiat",
"date": "2016-06-07T00:00:00",
"amount": 121,
"annee" : "2016",
"trimestre" : "Q2",
"mois" : "Juin",
"semaine" : "S22"
```


Figure 17 : PivotTable avec les données utilisées pour le prototype



					annee	2016		Totals
					trimestre	Q2		
					mois	Juillet	Juin	
					semaine	S24	S22	S23
groupe niveau 1	groupe niveau 2	groupe niveau 3	groupe niveau 4	account				
Exploitation	Charges	Transport	Fiat	Taxes Auto Fiat		185.50	121.00	306.50
			Skoda	Assurance Skoda			185.50	185.50
					Totals	185.50	121.00	185.50	492.00

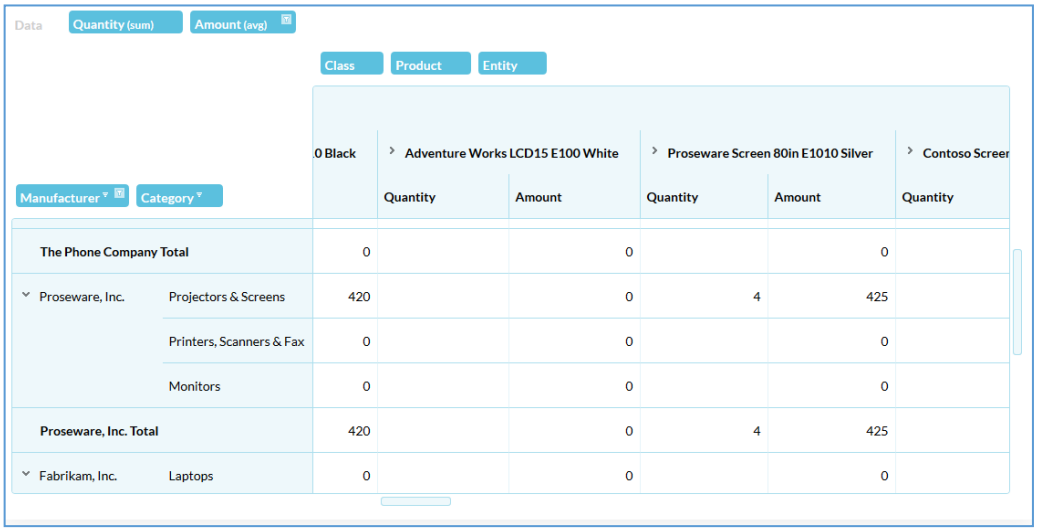
3.1.2 orb.js

Orb.js est un composant JavaScript/HTML5 utilisant React JS, une librairie JavaScript développée par Facebook pour construire des interfaces utilisateurs.

Les fonctionnalités de cette librairie sont similaires à PivotTable, tout comme ses avantages et inconvénients.

Contrairement à PivotTable, le tableau gère un surplus de données avec un scroll horizontal et vertical qui ne cache pas les entêtes des lignes et colonnes.

Figure 18 : Présentation Orb.js



Le tableau possède une toolbar qui permet de développer les entêtes des lignes et des colonnes, d'afficher ou cacher les totaux et d'exporter le tableau en format xls (Excel).

Figure 19 : toolbar d'Orb.js

Pour résumer orb JS est assez similaire à pivotTable, mais gère mieux les problèmes de tailles et propose une toolbar facilitant la navigation et l'exploration de données. Néanmoins cette librairie ne répond pas à nos besoins car elle ne gère pas les dimensions hiérarchiques.

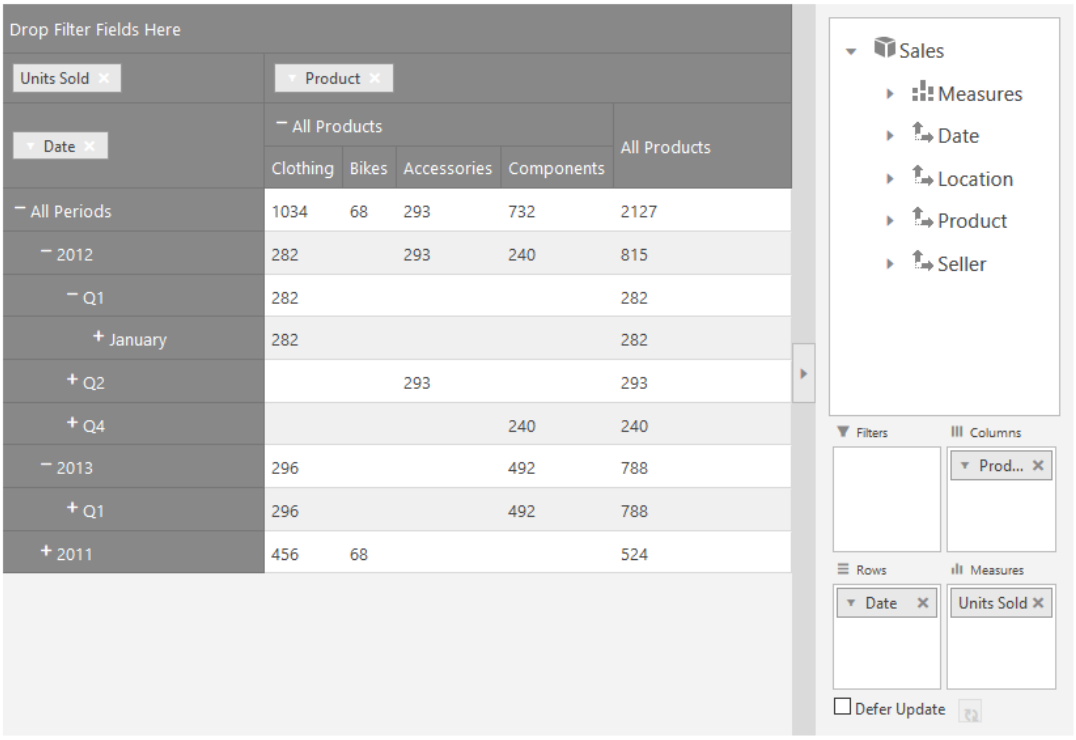
3.2 Solutions payantes

3.2.1 Ignite UI – igPivotView

Ignite UI est une librairie de composant JavaScript/HTML5 développée par Infragistic, une entreprise spécialisée dans le développement informatique depuis plus de 27 ans.

La librairie propose un composant igPivotView définit comme étant un tableau croisé capable de gérer des données multidimensionnelles.

Figure 20 : Présentation igPivotView de Ignite UI



Cette solution permet la construction de dimension hiérarchique mais ne supporte pas des données déjà hiérarchisées comme celles de notre projet.

La représentation du tableau est intéressante car elle affiche les totaux de chaque fils et parent, mais surcharge le tableau d'informations, le rendant moins lisible.

Un scroll est présent pour se déplacer dans un tableau de grande taille, mais cette méthode à l'inconvénient de cacher les dimensions à gauche lors d'un scroll à droite.

Il est impossible d'obtenir d'avantages d'informations sur les valeurs calculées, car aucune interaction n'est possible avec une cellule.

3.2.2 DevExtreme – PivotGrid

PivotGrid est un outil proposé par DevExtreme. Le système est compatible avec jQuery, Knockout, AngularJS et ASP.NET.

Les tableaux de grandes tailles sont gérés avec un scroll qui ne déplace pas les entêtes des lignes et des colonnes.

Figure 21 : Présentation PivotGrid de DevExtreme

		2013				2013 Total	2014				2014 Total
		Q1	Q2	Q3	Q4		Q1	Q2	Q3	Q4	
Africa	Cairo (EGY)	$3,310	$8,440	$3,340	$5,400	$20,490	$5,020	$6,370	$5,230	$5,640	$22,260
	Pretoria (ZAF)	$3,040	$6,830	$5,380	$7,880	$23,130	$4,740	$5,230	$7,110	$5,860	$22,940
Africa Total		$6,350	$15,270	$8,720	$13,280	$43,620	$9,760	$11,600	$12,340	$11,500	$45,200
Asia	Beijing (CHN)	$12,270	$15,780	$18,780	$25,140	$71,970	$18,960	$21,420	$26,190	$13,800	$80,370
	Seoul (KOR)	$11,825	$17,275	$10,050	$8,500	$47,650	$17,450	$15,500	$16,850	$13,425	$63,225
	Tokyo (JPN)	$14,310	$22,200	$14,250	$18,810	$69,570	$19,410	$11,370	$19,470	$7,020	$57,270
Asia Total		$38,405	$55,255	$43,080	$52,450	$189,190	$55,820	$48,290	$62,510	$34,245	$200,865
Australia	Melbourne (AUS)	$11,340	$14,130	$9,915	$10,350	$45,735	$7,035	$13,770	$7,635	$9,390	$37,830
	Sydney (AUS)	$20,540	$9,240	$12,280	$13,680	$55,740	$11,460	$8,800	$6,480	$11,560	$38,300
Australia Total		$31,880	$23,370	$22,195	$24,030	$101,475	$18,495	$22,570	$14,115	$20,950	$76,130

Il y un temps de chargement d'environ une seconde à chaque action sur les dimensions ce qui impacte la fluidité de la navigation.

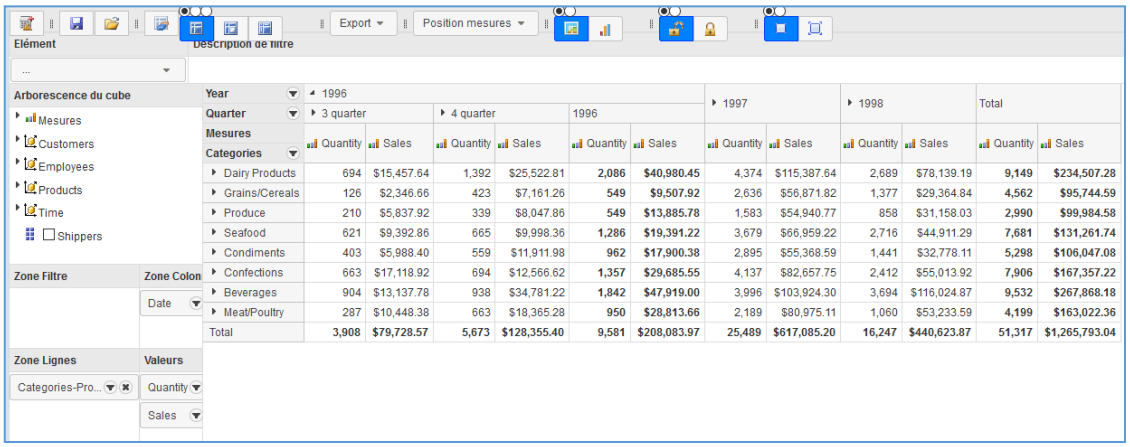
Le tableau est exportable en fichier xls (Excel).

3.2.3 Radar Cube

Radar Cube est un outil développé en ASP.NET par Radar-Soft.

Il propose la possibilité de sauvegarder en format XML l'état du tableau et de pouvoir le recharger antérieurement.

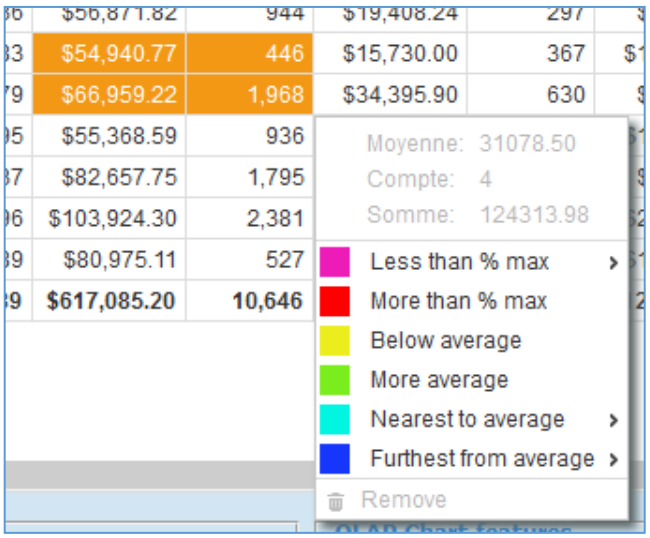
Figure 22 : Présentation RadarCube



Year	1996						1997		1998		Total	
Quarter	3 quarter		4 quarter		1996							
Mesures	Quantity	Sales	Quantity	Sales	Quantity	Sales	Quantity	Sales	Quantity	Sales	Quantity	Sales
Categories												
Dairy Products	694	$15,457.64	1,392	$25,522.81	**2,086**	**$40,980.45**	4,374	$115,387.64	2,689	$78,139.19	**9,149**	**$234,507.28**
Grains/Cereals	126	$2,346.66	423	$7,161.26	**549**	**$9,507.92**	2,636	$56,871.82	1,377	$29,364.84	**4,562**	**$95,744.59**
Produce	210	$5,837.92	339	$8,047.86	**549**	**$13,885.78**	1,583	$54,940.77	858	$31,158.03	**2,990**	**$99,984.58**
Seafood	621	$9,392.86	665	$9,998.36	**1,286**	**$19,391.22**	3,679	$66,959.22	2,716	$44,911.29	**7,681**	**$131,261.74**
Condiments	403	$5,988.40	559	$11,911.98	**962**	**$17,900.38**	2,895	$55,368.59	1,441	$32,778.11	**5,298**	**$106,047.08**
Confections	663	$17,118.92	694	$12,566.62	**1,357**	**$29,685.55**	4,137	$82,657.75	2,412	$55,013.92	**7,906**	**$167,357.22**
Beverages	904	$13,137.78	938	$34,781.22	**1,842**	**$47,919.00**	3,996	$103,924.30	3,694	$116,024.87	**9,532**	**$267,868.18**
Meat/Poultry	287	$10,448.38	663	$18,365.28	**950**	**$28,813.66**	2,189	$80,975.11	1,060	$53,233.59	**4,199**	**$163,022.36**
Total	**3,908**	**$79,728.57**	**5,673**	**$128,355.40**	**9,581**	**$208,083.97**	**25,489**	**$617,085.20**	**16,247**	**$440,623.87**	**51,317**	**$1,265,793.04**

Une autre fonctionnalité intéressante est la possibilité de pouvoir sélectionner une zone dans les mesures afin d'y lire quelques informations supplémentaires comme la moyenne, la somme ou le nombre de comptes.

Figure 23 : Exemple de cellules selectionnées



Comme il s'agit d'une solution payante, il est compliqué de tester au-delà des fonctionnalités de démonstration et il est difficile de savoir si cette solution gère les fichiers JSON à structure hiérarchique.

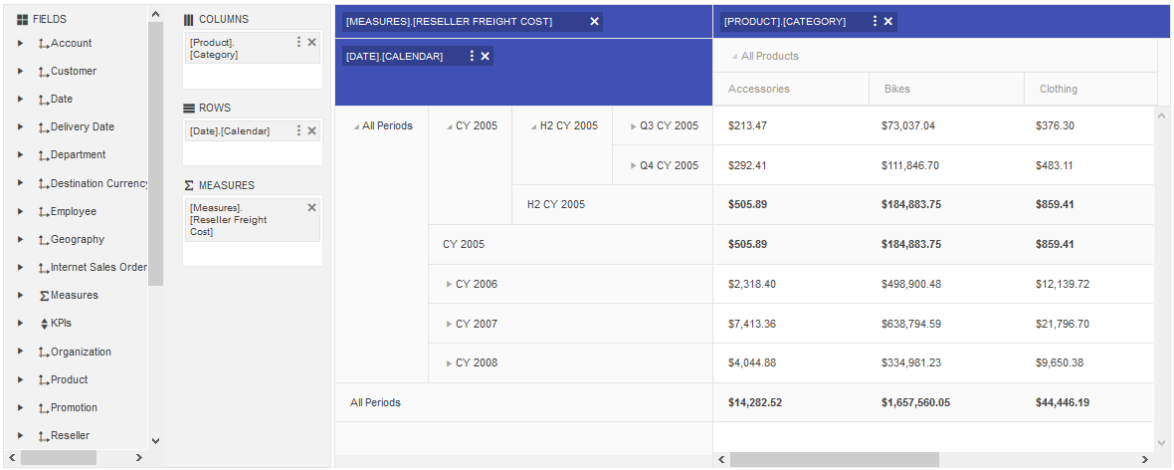
L'interface, bien que complète manque d'ergonomie (trop d'options, de fenêtres…)

3.2.4 PivotGrid - Kendo UI

Kendo UI est présenté comme étant la plus complète librairie d'interface utilisateur pour des applications pour client riche. La librairie est compatible avec Jquery, React, Angular et Vue.

Dans sa liste de composant, elle propose un tableau croisé nommé PivotGrid.

Figure 24 : PivotGrid de Kendo UI



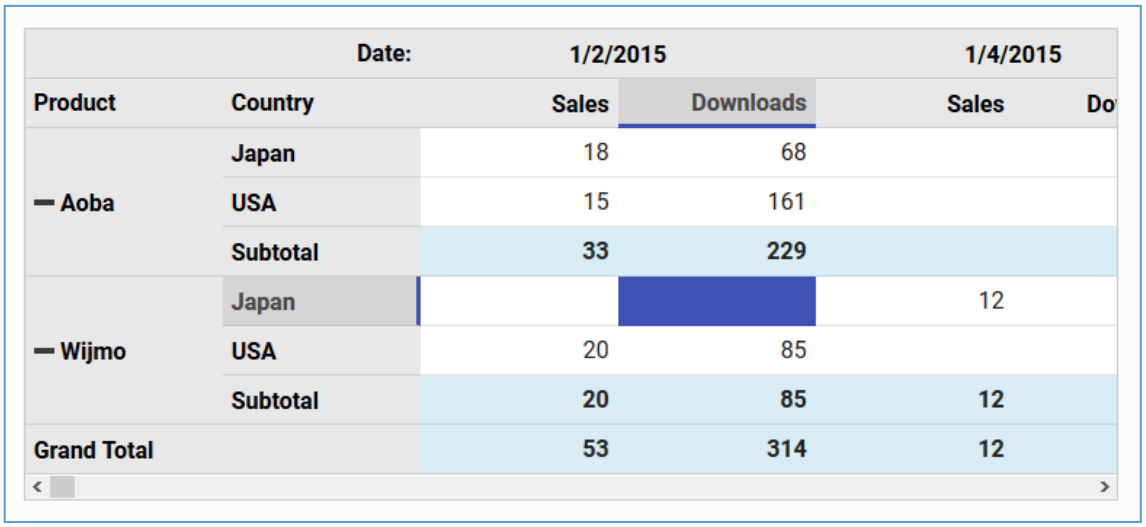
Cette solution propose des fonctionnalités similaires aux précédentes solutions, si ce n'est en plus la possibilité d'exporter le tableau en PDF (en plus d'xls).

Il n'y a pas assez d'informations quant au format des données supportées par défaut.

3.2.5 Wijmo Olap

Wijmo.OLAP est un module Angular permettant d'ajouter un tableau croisé à une application JavaScript.

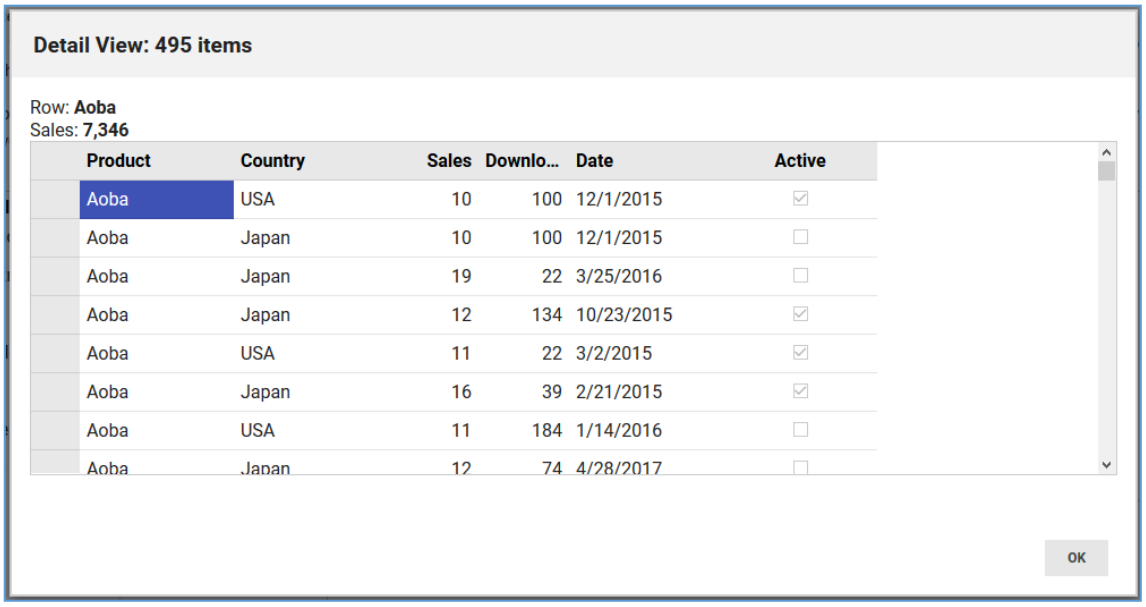
Figure 25 : Présentation Wijmo Olap



	Date:	1/2/2015		1/4/2015	
Product	Country	Sales	Downloads	Sales	Do
− Aoba	Japan	18	68		
	USA	15	161		
	Subtotal	33	229		
− Wijmo	Japan			12	
	USA	20	85		
	Subtotal	20	85	12	
Grand Total		53	314	12	

Moins complet que les autres solutions, il présente toutefois une caractéristique intéressante : lorsqu'on clique sur une cellule du tableau, une fenêtre s'ouvre avec les détails de celle-ci.

Figure 26 : Exemple d'affichage des détails d'une cellule



3.3 Bilan des solutions existantes

Les solutions gratuites et les solutions payantes ne proposent pas la gestion de véritable dimensions avec des hiérarchies; les dates et des "sous-dates" (Année – Trimestre – Mois) sont simulées avec plusieurs filtres superposés.

Pour gérer nos données hiérarchiques telles que le plan comptable, il faudrait alors ajouter des attributs fixes "groupe niveau1", "groupe niveau 2", "groupe niveau 3", etc, ce qui enlève l'avantage des dimensions hiérarchiques.

La fluidité fait défaut à la plupart des solutions qui ont un délai de chargement à chaque action. Seules RadarCube et Wijmo Olap proposent des interactions avec les données calculées du tableau, pour afficher les détails ou des statistiques.

Plusieurs solutions offrent les possibilités d'exporter le tableau en xls ou en PDF mais c'est RadarCube qui est plus complet que les autres en offrant la possibilité de sauvegarder l'état du tableau.

Aucune~~s~~ des solutions ne propose un système ressemblant exactement à nos besoins. Les meilleures solutions adaptables à nos besoins s'avèrent payantes. Leur prix élevé justifie l'élaboration d'une nouvelle solution.

Tableau 1 : prix des solutions

Solution	Prix unitaire	Prix entreprise
PivotGrid - Dev-Extreme	Gratuit (non-commercial)	499.99 $
Radar Cube - RadarSoft	1'499 $	2'999 $
PivotGrid – Kendo UI	899 $	-
igPivotView – Ignite UI	695 $	-
Wijmo Olap	895 $	-

S'il fallait retenir des idées d'amélioration pour notre futur prototype, il s'agirait en priorité de pouvoir gérer les dimensions que l'on souhaite avoir en entête des colonnes et lignes et par conséquent de pouvoir en importer plus de 2.

Il serait également intéressant de pouvoir gérer des dimensions autres que celle d'un plan comptable et de pouvoir exporter le tableau en xls ou en pdf.

4. Réalisation du prototype

Comme aucune solution existante ne répond entièrement aux besoins du projet, la réalisation d'un prototype s'impose.

4.1 Choix des technologies

Le prototype est réalisé en langage Web avec le couple classique HTML5 + CSS3 + JS. La question du choix technologique se porte sur les librairies, framework et alternatives à JavaScript qui seront utilisées dans le cadre du développement.

4.1.1 JQuery

JQuery est une bibliothèque JavaScript, créée par John Resig en 2006; elle a pour but de faciliter la manipulation du DOM côté client. Elle possède de nombreuses fonctionnalités simplifiées comme le parcours du DOM, les animations, les requêtes Ajax ou la manipulation de la CSS.

Cette bibliothèque est facile à utiliser car il suffit d'inclure dans notre page web sa version minifiée de 92 ko, téléchargeable sur le site officiel.

Voici un exemple de requête Ajax en JQuery à l'aide de la méthode getJSON. Cette méthode est utilisée dans notre prototype pour récupérer les données des fichiers JSON.

```
$.getJSON(parametres.data_ecritures, (data) => {
    $(data.Entry).each(function () {
        tabEcritures.push(new ecriture(this.capti
    })
}),
```

Maintenant, l'exemple d'une méthode similaire sans JQuery, on remarque que pour la même fonctionnalité, il y a beaucoup plus de lignes de code.

```
var request = new XMLHttpRequest();
request.open('GET', 'ecriture.json', true);
request.onload = function() {
    if (request.status >= 200 && request.status < 400){
        var cv = JSON.parse(request.responseText);
    }
};
request.send();
```

L'utilisation de cette librairie offre une grande diversité de fonctionnalité sur le DOM sans surcharger le code. JQuery agit comme une surcouche qui simplifie l'utilisation de JavaScript.

4.1.2 TypeScript

TypeScript est un langage de programmation créé par Microsoft qui fonctionne comme un sur-ensemble de JavaScript : le code JS est utilisable en TS mais TS recompile son code en JS pour être interprété par le navigateur.

TypeScript, comme son nom l'indique propose avant tout de pouvoir typer ses variables mais aussi de pouvoir créer des classes, des sous-classes, des interfaces et d'autres fonctionnalités utiles difficilement réalisables en JS pur.

Voici l'exemple d'une classe du prototype codée en TypeScript avec ses attributs, son constructeur et une méthode, similaire à du Java ou du C#.

```
class ecriture {
    nom : string;
    date : Date;
    montant : number;
    account_id : number;
    constructor(nom : string, date : Date, montant : number, account_id : number) {
        this.nom = nom;
        this.date = date;
        this.montant = montant;
        this.account_id = account_id;
    }
    public toString() : string {
        return "<p>"+this.date.toLocaleDateString('fr-FR')+" " +
            ""+this.nom+ " "+this.montant.toFixed(2)+"</p>";
    }
}
```

Voici maintenant, son équivalent en JS pur après la compilation TS -> JS à l'aide du module "amd".

```
var ecriture = (function () {
    function ecriture(nom, date, montant, account_id) {
        this.nom = nom;
        this.date = date;
        this.montant = montant;
        this.account_id = account_id;
    }
    ecriture.prototype.toString = function () {
        return "<p>" + this.date.toLocaleDateString('fr-FR') + " " +
            "" + this.nom + " " + this.montant.toFixed(2) + "</p>";
    };
    return ecriture;
}());
```

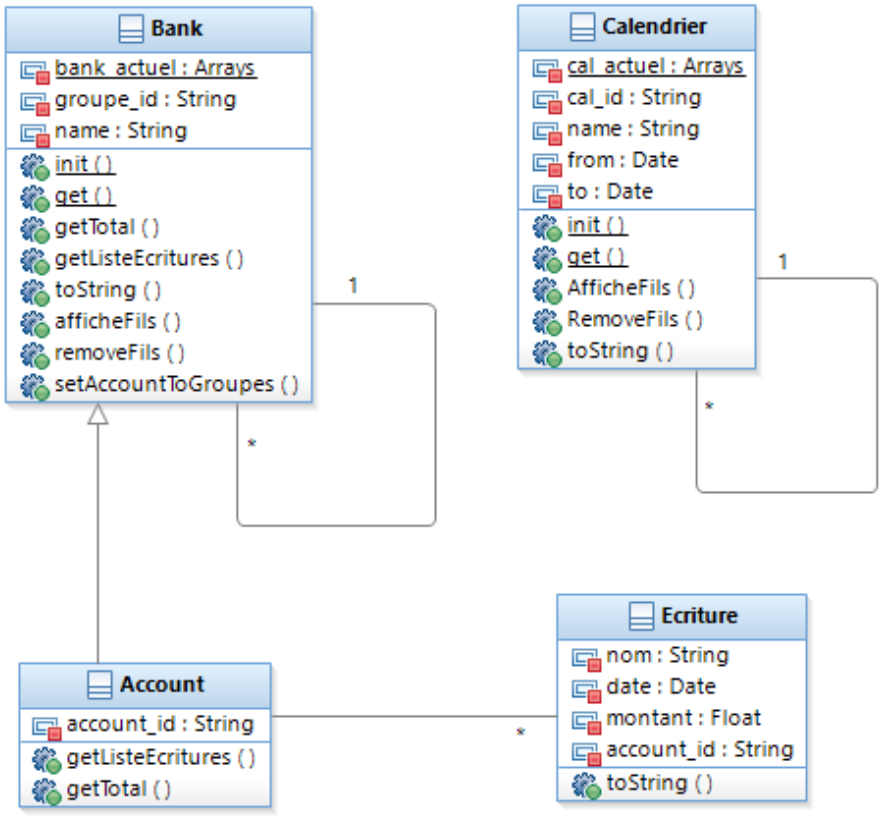
Le JS est visuellement moins compréhensible et pas typé.

L'exemple ci-dessus est simple mais avec du code plus complexe comme des sous-classes, il devient vite illisible. En TS, il y a tous les outils pour faire de la programmation objet.

4.2 Implémentation

4.2.1 Diagramme UML des classes

Figure 27 : Diagramme de classe du prototype



Un groupe est représenté par une instance de la classe "Bank". Il contient une liste d'objets "Bank" représentant ses éléments fils, un identifiant unique et un nom. La classe "Account" est une sous classe de "Bank" et représente un compte. Un compte est obligatoirement le dernier élément de la hiérarchie, il n'a pas d'élément fils. Un compte possède une liste d'écritures comptable.

Les méthodes "getTotal" et "getListeEcritures" de la classe "Account" utilisent le polymorphisme. "getTotal" parcourt de manière récursive la liste des éléments fils et additionne les totaux. "getListeEcritures" parcourt de la même manière les éléments fils et concatène les écritures dans une chaine de caractères. Dès l'instant ou les méthodes sont appelées sur un compte, le total ou la liste des écritures liées à celui-ci sont retournés.

Les méthodes statiques "init" permettent d'instancier des objets de classe "Bank" ou "Calendrier" en utilisant les données JSON qui leur sont passées en paramètre.

4.3 Calcul des montants

Les attributs statiques "bank_actuel" et "cal_actuel" des classes "Bank" et "Calendrier" sont des listes contenant les instances "Bank" et "Calendrier" actuellement utilisés pour les calculs affichés dans le tableau croisé.

Figure 28 : Explication des attributs "bank_actuel" et "cal_actuel"

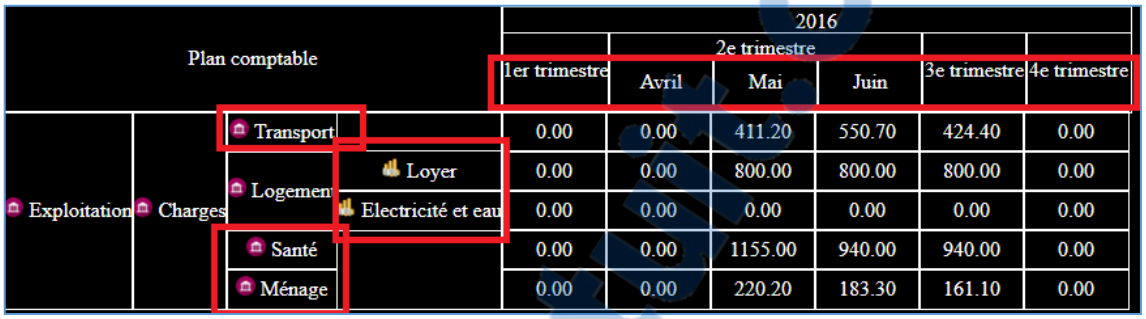


Tableau 2 : contenu des variables "bank_actuel" et "cal_actuel" selon la figure 29

bank_actuel	cal_actuel
Transport	1er trimestre
Loyer	Avril
Électricité et eau	Mars
Santé	Juin
Ménage	3e trimestre
	4e trimestre

Avec ces listes, le calcul du contenu des cellules du tableau croisé se fait à l'aide de deux boucles imbriquées qui parcourent les objets "Bank" contenus dans la liste "bank_actuel" et les objets "Calendrier" contenus dans la liste "cal_actuel".

Voici le pseudo code de la méthode d'affichage des montants

```
Fonction affichageMontant()
   élémentHtml = ""
     Pour x de 0 jusqu'à la taille de bank_actuel avec un pas de 1
       Pour y de 0 jusqu'à la taille de cal_actuel avec un pas de 1
          total <- bank_actuel[x].getTotal(cal_actuel[y].from, cal_actuel[y].to)
          élémentHtml += cellule avec le total
       Fin pour
     Fin pour
   affiche élémentHtml
Fin Fonction
```

4.4 Initialisation des instances à partir des fichiers JSON

Voici un échantillon de données JSON avec lesquelles le calendrier est initialisé.

Les données étant hiérarchisée et sans limite, la récursivité s'impose car le nombre d'élément fils est inconnu.


```json
{
    "name": "MasqueCalendrier",
    "TimeAggregate": {
        "caption": 2016,
        "timeFrom": "2016-01-01T00:00:00",
        "timeTo": "2016-12-31T23:59:59",
        "TimeAggregate":   {
                "caption": "1er trimestre",
                "timeFrom": "2016-01-01T00:00:00",
                "timeTo": "2016-03-31T23:59:59",
                "TimeAggregate": [
                    {
                        "caption": "Janvier",
                        "timeFrom": "2016-01-01T00:00:00",
                        "timeTo": "2016-01-31T23:59:59"
                    },
                    {
                        "caption": "Février",
                        "timeFrom": "2016-02-01T00:00:00",
                        "timeTo": "2016-02-29T23:59:59"
                    },
                    {
                        "caption": "Mars",
                        "timeFrom": "2016-03-01T00:00:00",
                        "timeTo": "2016-03-31T23:59:59"
                    }
                ]
            }
        }
}
```

Le champ "TimeAggregate" contient un ou plusieurs éléments fils. Si un élément fils ne contient pas de champs TimeAggregate, cela signifie qu'il est le dernier élément d'une branche de l'arbre, soit une feuille.

Pour initialisé les objets correspondant à ces données JSON, une méthode "init" existe dans les classes "Bank" et "Calendrier". Cette méthode parcourt de manière récursive l'arbre des données JSON.

Voici le pseudo code de la méthode récursive "getFils" utilisée dans la méthode "init".

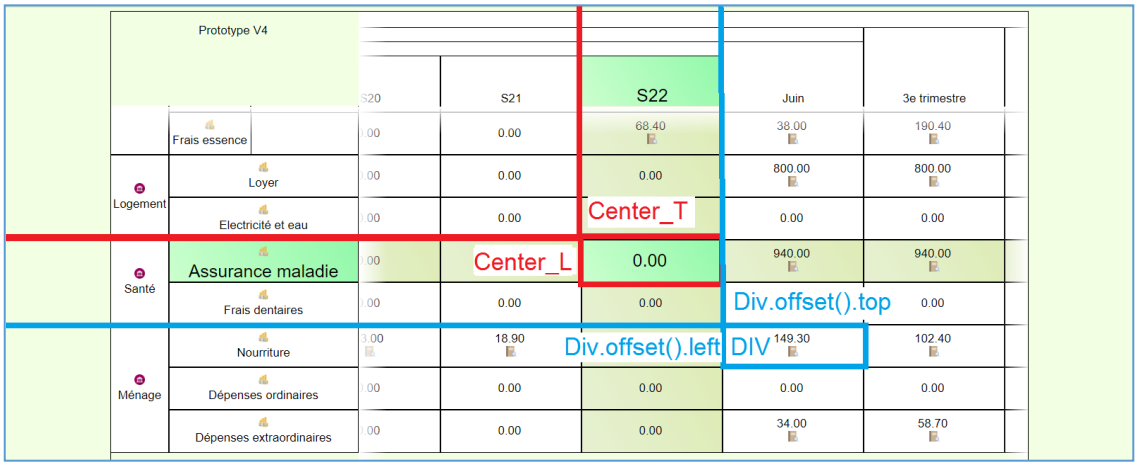
b est l'instance de la classe "Calendrier" parente.

x est le contenu du champ "TimeAggregate" d'une entrée JSON représentant les éventuels fils de *b*

```
Fonction getFils(b,x)
   Si x est une liste
      Pour i = 0 jusqu'à la taille de x avec un pas de 1
            Si x[i].TimeAggregate existe
               b.fils.push(getFils(new calendrier(x[i]), x[i].TimeAggregate))
            Sinon
               b.fils.push(new calendrier(x[i]))
            Fin Si
       Fin pour
   Sinon
      b.fils.push(getFils(new calendrier(x), x.TimeAggregate))
   Fin Si
   retourne b
Fin Fonction
```

4.5 Déplacement du tableau des montants

Le déplacement du tableau est un point complexe du projet car il nécessite de bien gérer les informations de position des éléments du DOM.



Les variables offset.top et offset.left correspondent à la position de la cellule sélectionnée qui doit être déplacée au milieu définit par les variables center_L et center_T. Deux méthodes sont créées pour connaître le déplacement en pixel à effectuer, "Deplace_left" et "Deplace_top".

Deplace_left teste si la position de la cellule sélectionnée est à gauche ou à droite du centre puis calcule sa distance par rapport au centre. Le résultat est négatif ou positif suivant la position de la cellule. Le système est identique pour Deplace_top qui gère la hauteur de la cellule.

Après avoir calculé ces deux valeurs, elles sont envoyées en paramètre de la méthode Deplace_tableaux qui utilise la méthode "animate" de Jquery pour déplacer les dimensions du plan comptable et du calendrier puis le tableau des mesures.

Voici le pseudo code de la méthode Deplace_left

```
Fonction Deplace_left(cellule)
    position = cellule.offset().left
    Si position est < ou = center_l
        position <- center_l – position
    Sinon
        position <- (position – center_l) * -1
    Fin Si
    Retourne position
Fin Fonction
```

4.6 Affichage et fermeture des éléments fils

L'affichage et la fermeture des éléments fils des dimensions hiérarchiques est fait à partir de la méthode "afficheFils" et "removeFils" de la classe "Calendrier" ou "Bank" qui prennent en paramètre la cellule du calendrier à ouvrir ou fermer. La principale difficulté est la mise à jour de la variable "cal_actuel" qui devra contenir les nouveaux éléments déployés.

Tableau 3 : évolution des valeurs de "cal_actuel" après déployement d'un élément parent de la dimension

Avant l'ouverture	Après l'ouverture
1er trimestre id = 0	1er trimestre id = 0
2ème trimestre id = 1	Avril id = 10
3ème trimestre id = 2	Mai id = 11
4ème trimestre id = 3	Juin id = 12
	3ème trimestre id = 2
	4ème trimestre id = 3

Voici le pseudo code de la méthode "AfficheFils"

p est l'élément parent à déployer

```
Fonction AfficheFils()
  tableau_temporaire = []
  Pour i = 0 jusqu'à  la taille de cal_actuel avec un pas de 1
     Si l'identifiant de cal_actuel[i] est égal à l'identifiant de l'instance de p
          Pour y = 0 jusqu'à  la taille de p.fils avec un pas de 1
             tableau_temporaire.push(p.fils[y])
          Fin Pour
       Sinon tableau_tepmoraire.push(cal_actuel[i])
       Fin Si
    Fin Pour
    cal_actuel <- tableau_temporaire
Fin Fonction
```

Pour la fermeture d'un élément parent d'une dimension, la difficulté est qu'il faut remplacer plusieurs instance de la liste de la variable "cal_actuel" par une seule.

Tableau 4 : évolution de la liste cal_actuel après fermeture d'un élément

Avant la fermeture	Après la fermeture
1er trimestre id = 0	1er trimestre id = 0
Avril id = 10	2ème trimestre id = 1
Mai id = 11	3ème trimestre id = 2
Juin id = 12	4ème trimestre id = 3
3ème trimestre id = 2	
4ème trimestre id = 3	

Voici le pseudo code de la méthode removeFils sans le code d'affichage

p est l'élément parent à fermer

```
Fonction RemoveFils()
    tableau_temporaire = []
    replace = vrai
    Pour i = 0 jusqu'à  la taille de cal_actuel avec un pas de 1
        Si l'identifiant de cal_actuel[i]  ne commence pas par l'identifiant de l'instance p
            tableau_tepmoraire.push(cal_actuel[i])
        Sinon
            Si replace = vrai
                tableau_temporaire.push(p)
                replace <- faux
            Fin Si
        Fin Si
    Fin Pour
    cal_actuel <- tableau_temporaire
Fin Fonction
```

5. Utilisation du prototype

5.1 Pré-requis

- Jquery
- Fluidtable.js
- Fluidtable.css

5.2 Paramètres

Le plugin est appelé avec la méthode fluid_table(), voici ses paramètres :

Tableau 5 : paramètres de la méthode fluid_table()

Paramètre	Défaut	Description
Base_div	$("#fluid_table")	Conteneur du tableau
Width	1400	Largeur du tableau en px
Height	750	Hauteur du tableau en px
Data_calendrier	Json/calendrier.json	Données calendrier
Data_ecritures	Json/ecritures.json	Données écritures
Data_groupes	Json/groupes.json	Données groupes
Data_comptes	Json/comptes.json	Données comptes

5.3 Exemple d'utilisation

```
<body>
    <div style='margin-top : 30px;' id="fluid_table"></div>
    <script type="text/javascript" src="node_modules/jquery/dist/jquery.js"></script>
    <script type="text/javascript" src="fluidtable.js"></script>
    <script>
        fluid_table({
            'base_div' : $('#fluid_table'),
            'width' : 1400,
            'height' : 700,
            'data_calendrier' : 'json/Calendrier.json',
            'data_ecritures' : 'json/Ecritures.json',
            'data_groupes' : 'json/Groupes.json',
            'data_comptes' : 'json/Comptes.json'
        });
    </script>
</body>
```

6. Visuels du prototype

Voici une liste de capture d'écran du prototype de son état initial à un état avancé.

Figure 29 : Affichage initial du prototype après clic sur la seule case montant

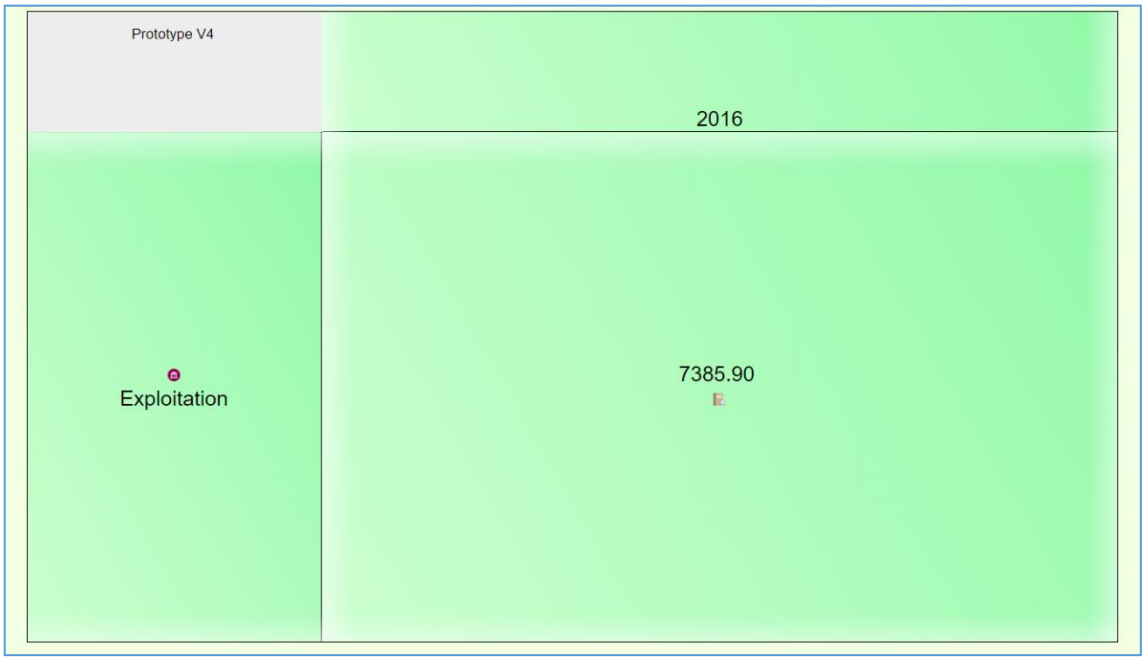
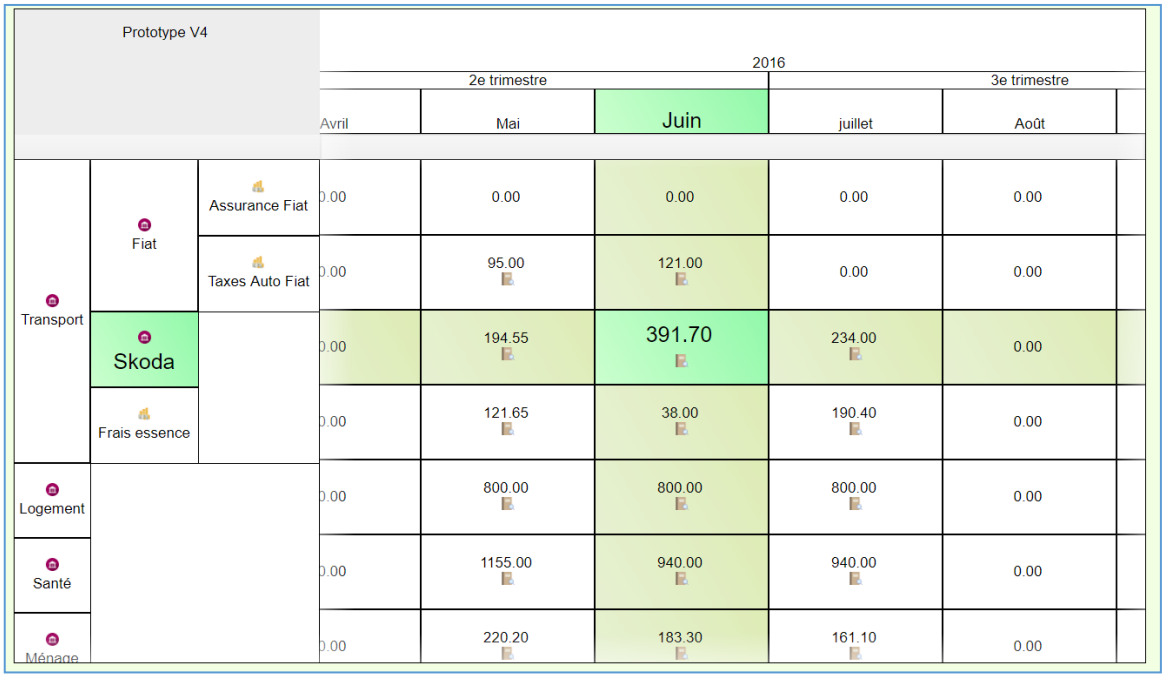


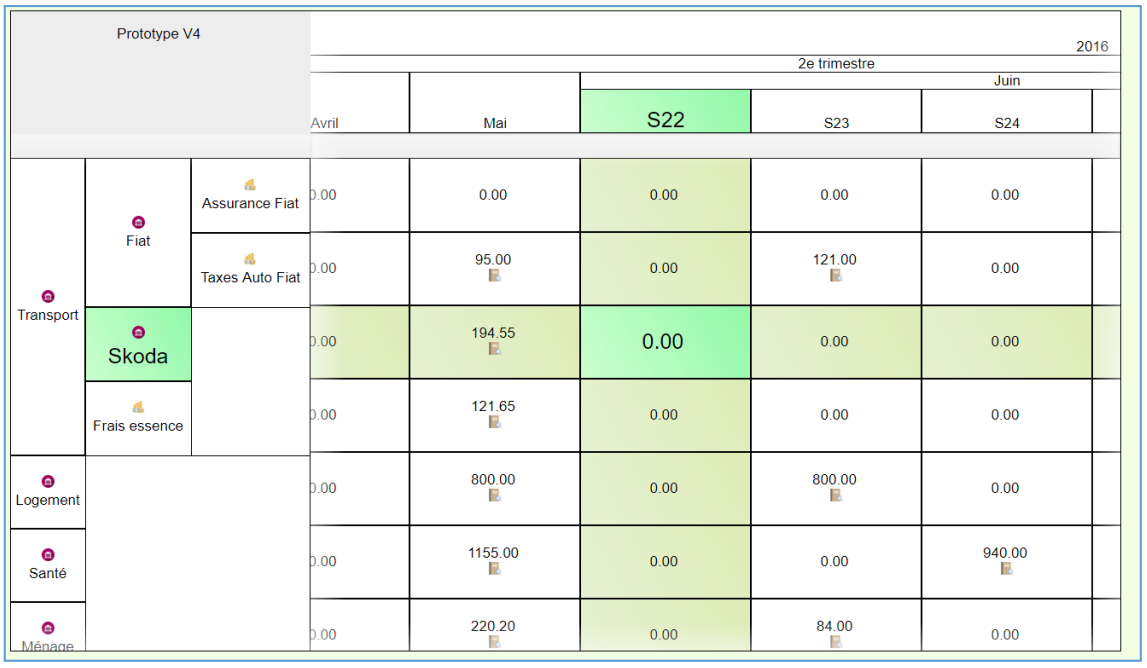
Figure 30 : Affichage après déroulement maximum des groupes avec la molette

Figure 31 : Affichage après la sélection de la cellule Skoda - Juin



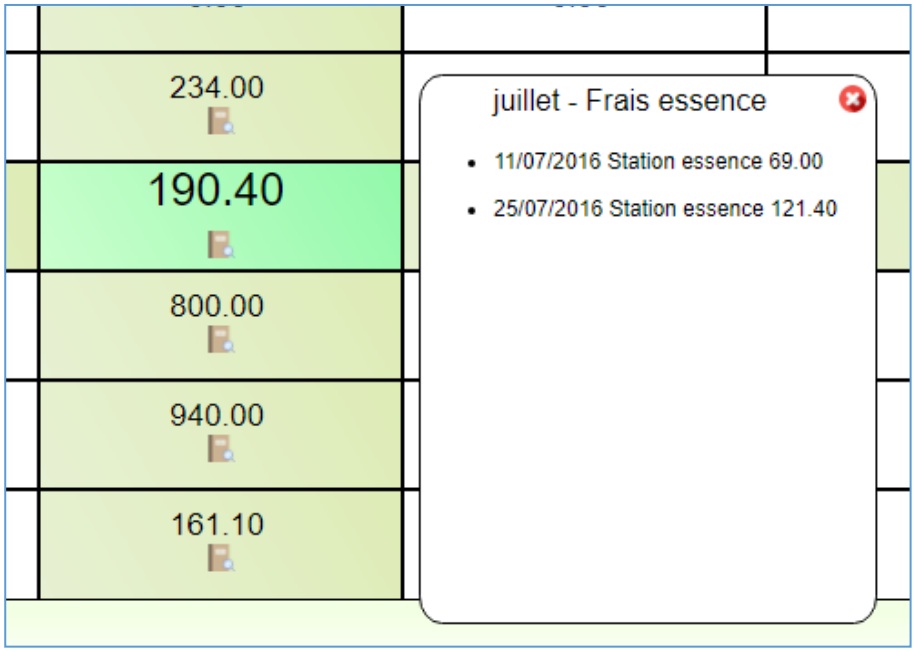
Prototype V4			2016				
			2e trimestre			3e trimestre	
			Avril	Mai	Juin	juillet	Août
Transport	Fiat	Assurance Fiat	0.00	0.00	0.00	0.00	0.00
		Taxes Auto Fiat	0.00	95.00	121.00	0.00	0.00
	Skoda		0.00	194.55	391.70	234.00	0.00
	Frais essence		0.00	121.65	38.00	190.40	0.00
Logement			0.00	800.00	800.00	800.00	0.00
Santé			0.00	1155.00	940.00	940.00	0.00
Ménage			0.00	220.20	183.30	161.10	0.00

Figure 32 : Affichage après le déroulement d'un cran avec la molette de la cellule Skoda – Juin



Prototype V4			2016				
			2e trimestre				
					Juin		
			Avril	Mai	S22	S23	S24
Transport	Fiat	Assurance Fiat	0.00	0.00	0.00	0.00	0.00
		Taxes Auto Fiat	0.00	95.00	0.00	121.00	0.00
	Skoda		0.00	194.55	0.00	0.00	0.00
	Frais essence		0.00	121.65	0.00	0.00	0.00
Logement			0.00	800.00	0.00	800.00	0.00
Santé			0.00	1155.00	0.00	0.00	940.00
Ménage			0.00	220.20	0.00	84.00	0.00

Figure 33 : Affichage des écritures de Juillet – Frais essence



Le dernier prototype est testable à cette adresse :

http://adrien.dallinge.ch/tb/prototype/v5 .

7. Conclusion

Si les tableaux croisés dynamiques sont dans la plupart des cas adaptés pour traiter des données multidimensionnelles, ils ne conviennent pas à tous les types de dimensions hiérarchiques.

Il convient de différencier les dimensions hiérarchiques de longueur définie et celle de longueur indéfinie comme dans notre projet. Si le premier cas est traité soit par une hiérarchisation de plusieurs dimensions différentes (catégorie – sous-catégorie) ou de manière automatisée suivant les solutions (dates affichées de manière hiérarchique), le second cas, avec le nombre de niveau indéfini, crée une problématique qu'aucune des solutions évaluées ne résout.

Pourtant, pouvoir afficher notre type de dimension complexe s'avère très utile, notamment pour l'affichage d'un plan comptable comme il est le cas dans ce travail.

De ce fait, la création d'un prototype qui répond spécifiquement aux besoins est justifiée. Si celui-ci n'est pas exempte de défauts, il a le mérite d'apporter une solution à l'affichage de nos dimensions hiérarchiques.

Il pourrait être intéressant de terminer complètement le prototype pour pouvoir le mettre à disposition en open-source pour évaluer les besoins de la communauté.

La principale difficulté de ce travail a été d'avoir à trouver la manière la plus précise, mais toujours compréhensible, d'expliquer la problématique des données multidimensionnelles à dimensions hiérarchiques. Cette étude m'a permis d'améliorer mes connaissances dans ce domaine.

Le développement du prototype m'a été très enrichissant, tout d'abord par l'utilisation du langage TypeScript et aussi par l'utilisation de concepts avancés de programmation, tels que la récursivité ou le polymorphisme, étudiés tout au long de mon parcours académique.

Pour finir, la réalisation de ce mémoire fut passionnante, il clôt de manière positive mon cursus à la HEG. Ce fût une expérience très enrichissante qui, nulle doute, me servira lors de mon futur parcours professionnel.

Bibliographie

SANTEL LEBORGNE Mathieu, *Entrepôt de Données* [consulté le 1 septembre 2017] Disponible à l'adresse : http://igm.univ-mlv.fr/~dr/XPOSE2005/entrepot/multidim.html

Concept Forge, *tree-radial* [consulté le 3 Août 2017] Disponible à l'adresse :

http://blog.conceptforge.ch/wp-content/uploads/2012/09/tree-radial.html

BOSTOCK Mike, *Sunburst Partition* [consulté le 3 Août 2017] Disponible à l'adresse : https://bl.ocks.org/mbostock/4063423

BOSTOCK Mike, *Partition* [consulté le 3 Août 2017] Disponible à l'adresse : https://bl.ocks.org/mbostock/2e73ec84221cb9773f4c

KRUCHTEN Nicolas, *PivotTable.js* [consulté le 5 Août 2017] Disponible à l'adresse : https://pivottable.js.org/examples/

Nnajm, *Orb.js* [consulté 6 Août 2017] Disponible à l'adresse : http://orbjs.net/

INFRAGISTICS, *Ignite UI – igPivotView* [consulté 6 Août 2017] Disponible à l'adresse : http://www.igniteui.com/help/api/2014.2/ui.igPivotView

DevExtreme, *Olap Data Source* [consulté le 7 Août 2017] Disponible à l'adresse :

https://js.devexpress.com/Demos/WidgetsGallery/Demo/PivotGrid/OLAPDataSource/jQuery/Light/

RadarSoft, *Radar Cube* [consulté le 8 Août 2017] Disponible à l'adresse : http://aspnet-webforms.demos.radar-soft.com/Demos/HtmlOLAPAnalysis.aspx

Kendo UI, *PivotGrid* [consulté le 8 Août 2017] Disponible à l'adresse :

http://demos.telerik.com/kendo-ui/pivotgrid/index

Olap 101, *Wijmo.OLAP* [consulté le 11 Août 2017] Disponible à l'adresse :

http://demos.wijmo.com/5/angular/OlapIntro/OlapIntro/