RAPPORT DE PROJET :

GESTIONNAIRE DE BUDGETS

BAUDON Nicolas

Juillet 2020

I/ Problématique:

Comment avoir une vue d'ensemble de nos dépenses pour savoir quelles sont leurs principaux secteurs et ne pas dépasser notre budget ?

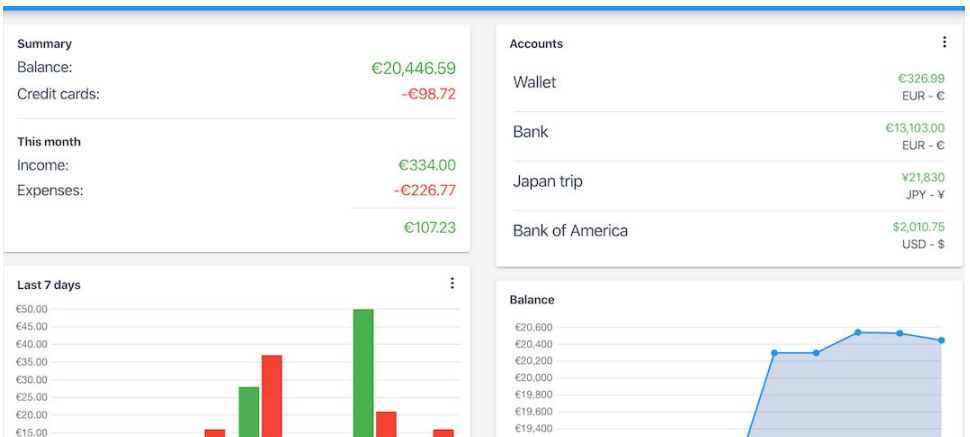
Il est facile de dépenser plus d'argent que l'on ne le pense dans certains type de produits et de services sans forcément se rendre compte des montant totaux.

Mon but est de fourniture à l'utilisateur un outil lui permettant de toujours **avoir un vue global sur ses dépenses**. Pour qu'il puisse toujours savoir dans quels domaines il débourse son argent et pourquoi pas alors, changer ses secteurs de dépenses soi pour faire des économies ou pour utiliser son argent à meilleur escient.

Un **gestionnaire de Budget** semble donc une solution adéquate pour répondre à cette problématique, en organisant les Dépenses par Budget avec un montant maximal pour chaque Budget. Cela permettrait de pouvoir suivre les secteurs de Dépenses au cours du temps

II/ Etat de l'art:

Il exist à l'heure actuel un certain nombre d'outil pour suivre ses dépenses, que ce soit sous forme d'**application mobil et web**.



On peut en retirer quelque **fonctionnalités** comme :

- L'utilisation de **Budget** mensuel total ou par type avec un certain montant qui est renseigné par l'utilisateur.

- La possibilité d'ajouter à ces budget des **Dépenses** caractérisées par un type et un montant.
    - La possibilité de définir une date de répétition d'une **Dépense** pouvant être utile pour une paiment mensuel d'un loyer ou de toute autre charge fixe arrivant de manière **Récurrente**.

- Un **affichage donnant une vue d'ensemble** des dépenses réalisées, permettant une compréhension rapide de l'état de notre budget mensuel.

- **Mise en parallèle des dépenses actuel avec celles des mois précédents**, aidant à voir l'évolution des finances de l'utilisateur.

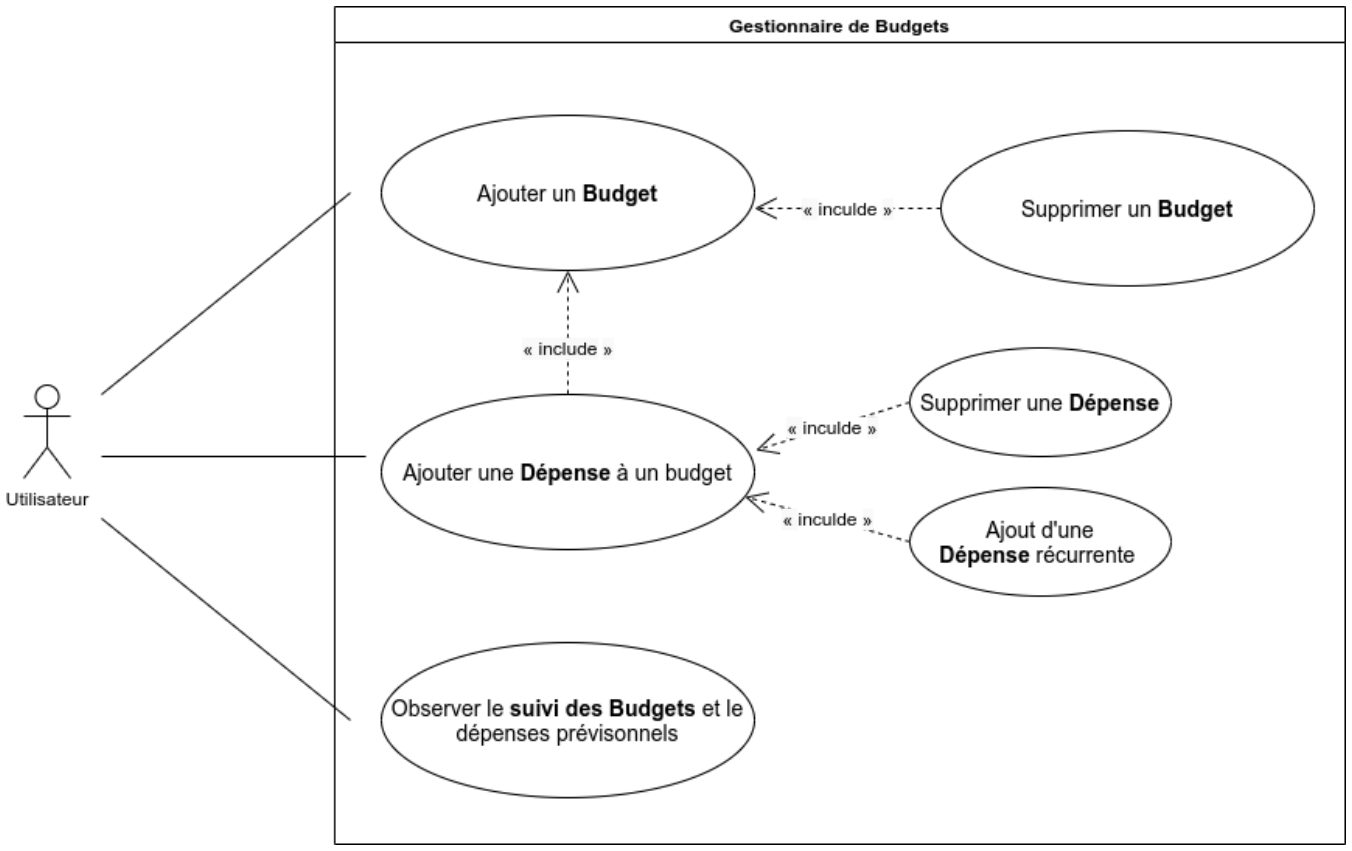
III/ Fonctionnalités:

Apres etude des différentes outils existant une sélection de **fonctionnalités indispensable** pour tous **gestionnaires de budget** resort ainsi que d'autre plus spécifique à la problématique présenté précédemment.

- Création de **Budget** avec un type et un montant
- Ajout de **Dépenses** avec un type, un montant et un date (automatique)
- Ajout de **Dépenses Récurrentes** mensuel avec un type, un montant
- Visualisation :
    - Progression de chaque budget
    - Progression du budget total
    - Graphique détaillant pour chaque mois les dépenses pour chaque budget
    - ----- affichage montant total du Budget
    - ------- prévision des dépenses récurrentes pour les mois à venir

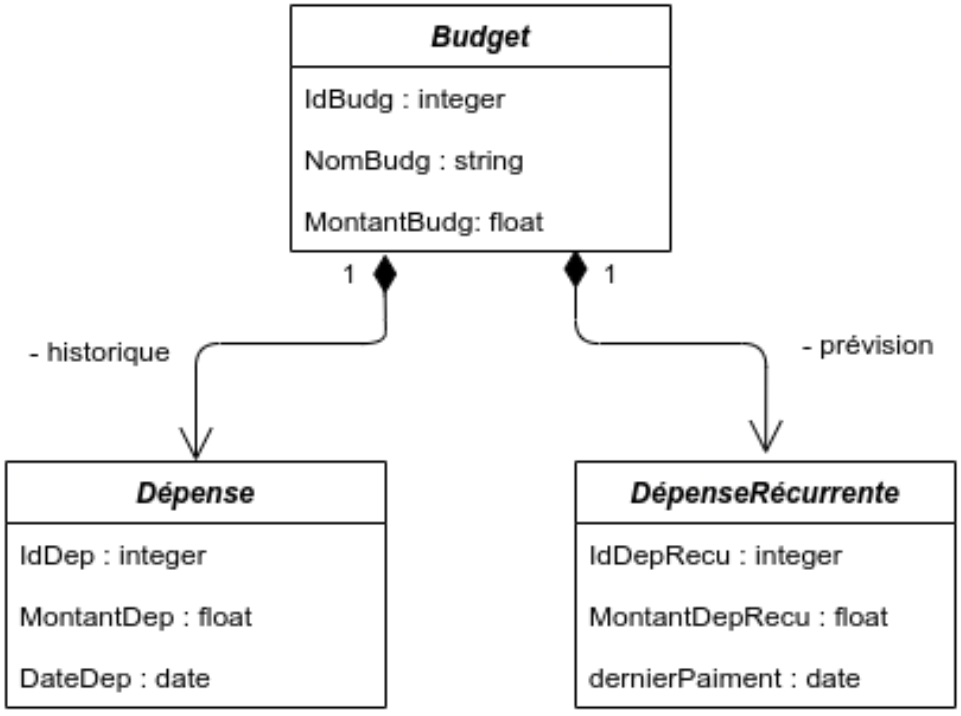
1/ Cas d'utilisations:

On peut alors estimer des **Cas d'Utilisations** qui pourraient être pertinent au projet.



2/ Structure de base de données:

On peut alors déterminer une structure de base de données permettant de réaliser les différentes fonctionnalités.



Budget(**idBudg**, montantBudg, nomBudg)
Depense(**idDep**, montantDep, **#idBudg,** dateDep)
DepenseRecu(**idDep**, montantDep, **#idBudg,** dernierPaiment)

```
sqlite> select * from BUDGETS ;
1|250|Alimentation
2|100|Loisirs
3|250|Loyer
4|50|Economies Vacances
5|50|Transport
6|400|Dépenses Vacances
```

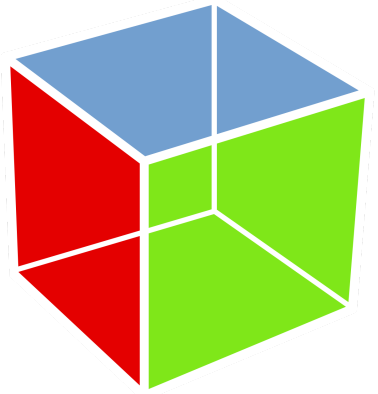
```
sqlite> select * from DEPENSES;
3|10|2|1595596818
5|10|2|1595596825
11|25|4|1595597250
12|100|4|1595597404
13|75|5|1595597413
14|300|6|1595597961
```

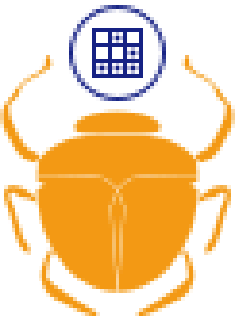
```
sqlite> select * from DEPENSESRECURRENTE;
1|100|1|07
2|250|2|07
3|25|3|07
```

3/ Outils et Technologies utilisées:

La totalité du développement de ce logiciel à été effectué en entièreté sous **Linux.**La contrainte du système d'exploitation et des différentes fonctionnalités envisagées a orienté le choix de la bibliothèque dédié à l'interface graphique. Il en est sortie que **GTK** serait le meilleur choix, GTK n'est pas seulement une bibliothèque mais un ensemble de bibliothèques pour faire une interface, comme par exemple **Cairo** une bibliothèque permettant faire des dessins pour créer des graphique par exemple.

De plus GTK donne la possibilité de générer une interface via un fichier **XML** ce qui facilite sa création et l'arrangement des éléments graphiques.

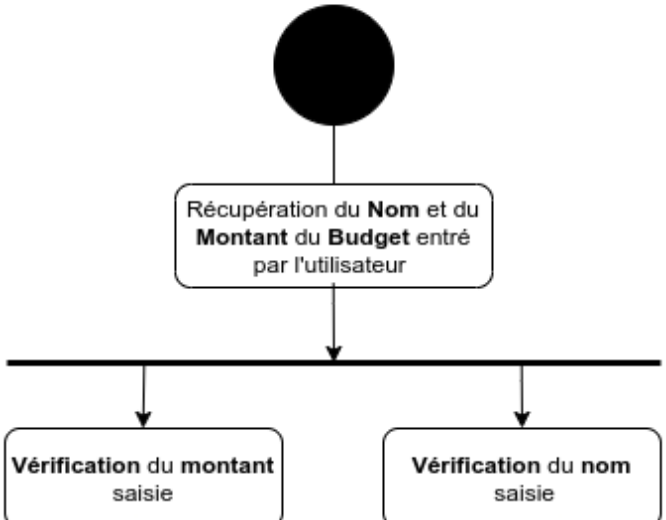




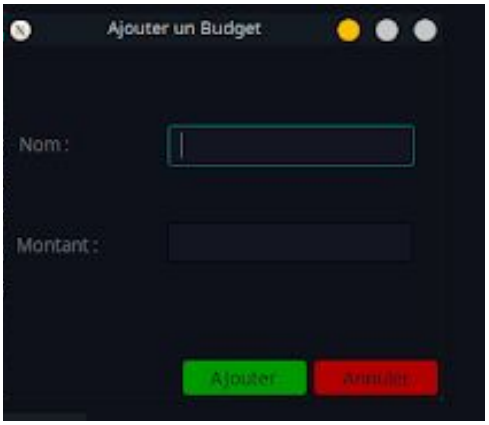
Pour le choix de la bibliothèque chargé de la **base de données** j'ai opté pour **SQLite** qui est une solution légère et simple d'utilisation. Permettant de créer une base de donné facilement sans serveur, elles est simplement stocké dans un fichier text.

4/ Description des Fonctionnalitées:

A/ Création de Budget:



Une fois que l'utilisateur clique sur le bouton d'ajout d'un **Budget** une fenêtre de saisie s'ouvre pour pouvoir rentrer le nom et le montant voulu pour le nouveau **Budget**.
La fonction d'insertion s'exécute une fois le formulaire validé.
En cas d'annulation la fenêtre se ferme et l'ajout est annulé.



La fonction "**budgUnique**" vérifie si le nom d'un budget est unique elle permet de ne pas avoir plusieurs budget avec le mêm nom.
Cette fonction retourne **0** elle ne trouve un budget dans la base avec le même nom et **1** si elle en trouve un **Budget** existant avec le mêm nom dans la base de données.

Fonction **budgUnique**

```c
int budgUnique(char *type){
 sqlite3_stmt *stmt;
 char request[60] = "select count(*) from budgets where typeBudg = '";
 strcat(request, type);
 strcat(request, "'");

 if (sqlite3_prepare_v2(db, request, -1, &stmt, NULL)) {
   printf("ERROR TO SELECT DATA : getBudget\n");
   exit(-1);
 }
 sqlite3_step(stmt);
 return sqlite3_column_int(stmt, 0) - 1;
}
```

Nous utilisons pour insérer un nouveau budget dans la base de données la bibliothèque **SQLite** mise à disposition pour C.
Après avoir créée la requête avec le nom et le montant saisies par l'utilisateur nous utilisons la fonction "**sqlite3_exec()**" qui permet d'exécuter une requête SQL en un ligne.

Insertion d'un **Budget**

```c
char *request = "insert into BUDGETS (montantBudg,typeBudg)values(";
replacechar(montant, ',', '.');
strcat(request, montant);
strcat(request,", '");
strcat(request, nom);
strcat(request,"')"); // Creation de la requete
if(sqlite3_exec(db, request, NULL, NULL, NULL)){ // Insertion
    printf("ERROR IN INSERTION : BUDGET\n"); // Erreur si insertion impossible
}else{
    printf("INSERT BUDGET : %s : %s\n", montant, nom);
    vueBudgets();//Mise à jour de la vue des Budgets
}
```

Une fois le **Budget** inséré dans la base on actualise l'affichage avec la fonction "**vueBudget**" pour le rendre visible sur l'interface.

B/ Ajout d'une Dépense à un Budget:

L'ajout d'une **Dépense** à une **Budget** est en grand partie identique à l'ajout d'un Budget excepté le fait que l'utilisateur doit sélectionner un **Budget associé** à sa **Dépense** dans la liste de ceux qu'il a créé.

Comme pour le Budget une fois les vérifications du validités du montant (> 0) et de l'existence dû **Budget** sélectionné une requête SQL est généré et la nouvelle dépense est inséré dans la base de données. On actualise alors les vues des Budgets, pour afficher le nouveau montant restant, et les Dépenses pour afficher la Dépense ajoutée à l'instant.

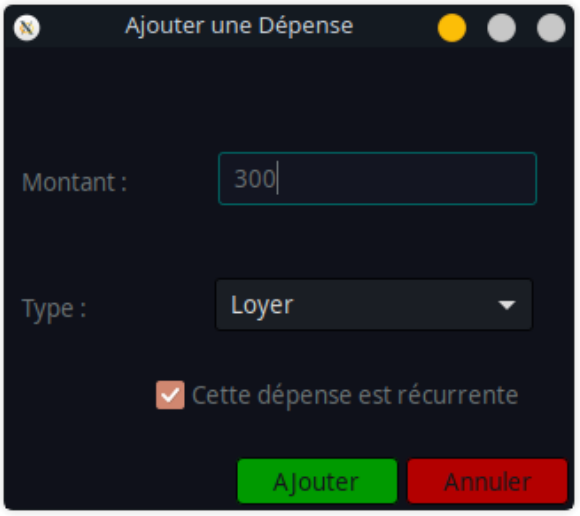
Vérification et Insertion d'une **Dépense**

```
//Si le montant est > 0 et type != null
if (dep.montant > 0 && strcmp(dep.type, "(null)") != 0) {
  char request[80] = "INSERT into DEPENSES (montantDep, idType) values(";
  replacechar(m, ',', '.');
  strcat(request, m);
  strcat(request,", '");
  snprintf(buffId, sizeof(buffId), "%d", idB);
  strcat(request, buffId);
  strcat(request,"')");
  printf("REQUEST: %s", request);
  if(sqlite3_exec(db, request, NULL, NULL, NULL)){
    printf("ERROR IN INSERTION : DEPENSE\n"); // Erreur d'insertion
  }else{
    printf("INSERT : DEPENSE\n");
    vueBudgets(); // actualisation de la vue des Budgets
    vueDepenses(); // actualisation de la vue des Dépenses
  }
```

C/Ajout d'une Dépense Récurrente à un Budget:

Une **Dépense Récurrente** se répétera chaque mois et ajoutera une dépense classique d'un montant donnée au budget au quelle elle est associé.

Lors de la création d'une nouvelle dépense l'utilisateur peut sélectionner une "**CheckBox**" pour que la dépense soit récurrente.



La distinction entre Dépense classique et Dépense Récurrente s'effectue après la validation du formulaire grâce à la fonction "**gtk_toggle_button_get_active**" qui retourne **TRUE** si l'élément "**CheckBox**" est sélectionné.

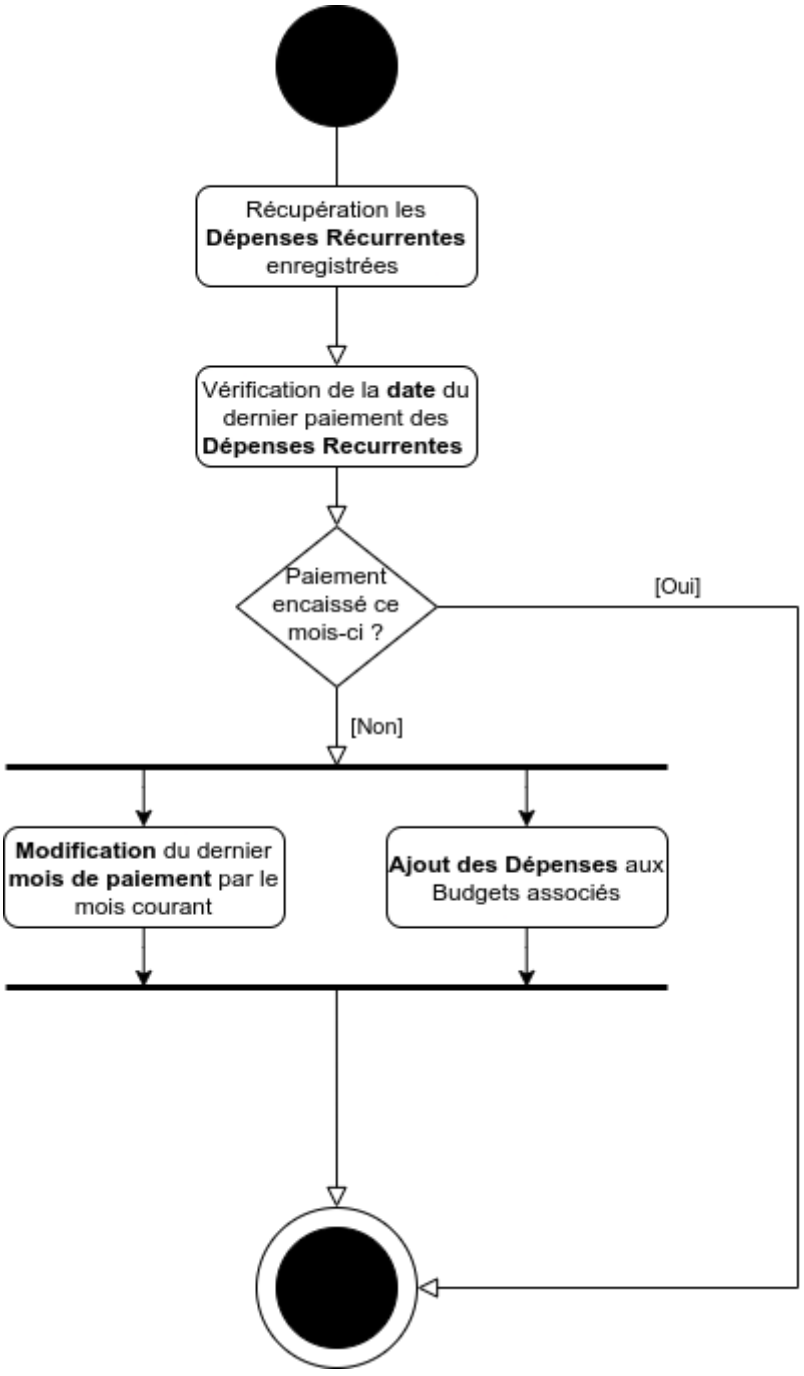
Nous pouvons donc appeler la fonction correspondante au type de **Dépense** soir "**insertDepense**" ou "**insertDepenseRecu**" en leur passant en paramètre le montant entré par l'utilisateur ainsi que l'id associé au Budget sélectionné.

Dépense classique ou **Récurrente**

```
if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(check_box_recu))){
   printf("DEPENSE RECURRENTE\n");
  // Insertion d'une Dépense Récurrente avec son montant et l'id du Budget
   insertDepenseRecu(m, (int)sqlite3_column_int(stmt, 0));
 }else{
   printf("DEPENSE\n");
  // Insertion d'une Dépense classique avec son montant et l'id du Budget
   insertDepense(m, (int)sqlite3_column_int(stmt, 0));
 }
```

D/ Paiement des Dépenses Récurrentes:

Comment expliqué précédemment les **Dépenses Récurrents** sont payées tous les mois.
À chaque lancement du logiciel une vérification est effectuée pour voir si toutes les **Dépenses Récurrentes** enregistrées ont été encaissé ce mois si. Si cela n'a pas été fait alors la **Dépense** est encaissé et le mois du dernier paiement de la Dépense est actualisé.

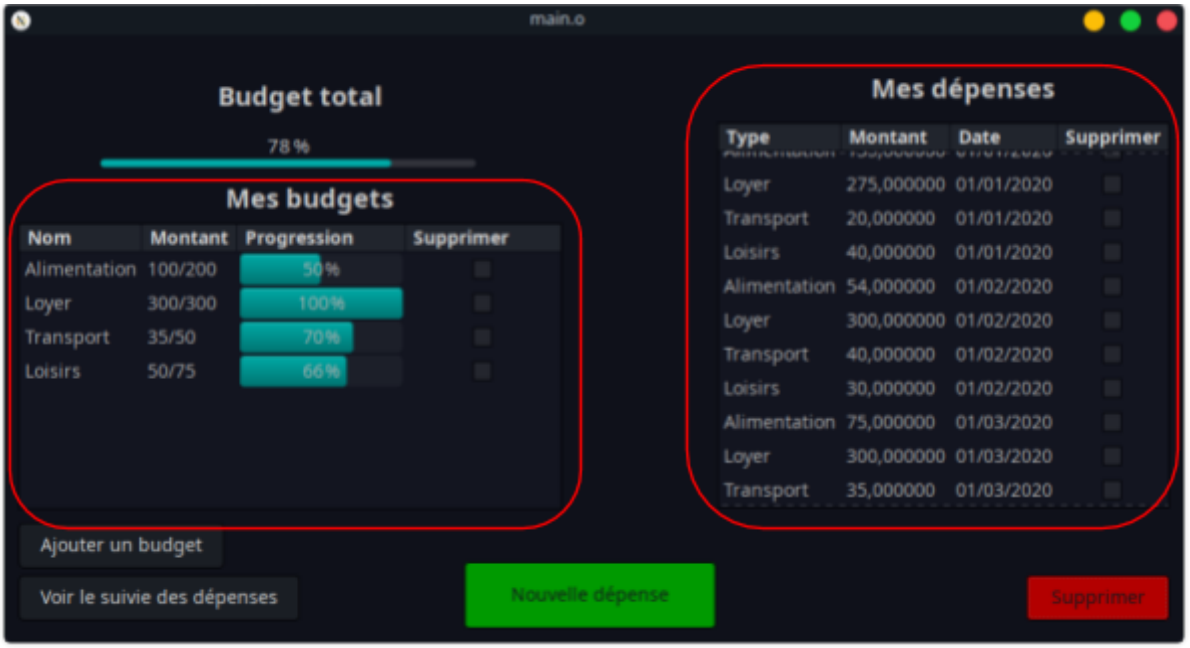


E/ Affichage des Budget et de l'historique des Dépenses:

L'affichage de la liste des **Budgets** créés et de l'historique des **Dépenses** se fait grâce à un élément graphique appelé "**TreeView**" qui est majoritairement utilisé avec une structure de donnée appel "**TreeModel**". Elle permet alors d'afficher des arborescence d'éléments comme par exemple des fichiers et des dossiers.

Dans notre cas nous l'utiliserons avec une structure appelé "**ListStore**" puisque nous n'avons pas besoin d'afficher d'arborescence mais simplement une liste. Cette élément graphique rend l'affichage de liste plus simple et plus ergonomique en effet un certain nombre de fonctions y sont associé pour faciliter l'ajout de ligne, leurs lecture et leurs suppression par exemple.

Nous avons donc deux "**TreeView**" associé chaqu'une à une "**ListStore**", une pour les **Budgets** et l'autre pour les **Dépenses**.



Ces deux éléments sont créé à partir d'un fichier **XML** décrivant à la fois les "**TreeView**" et les "**ListStore**". Ce fichier **XML** décrit plus généralement l'intégralité de l'interface du logiciel mais nous aborderons ce sujet dans une autre partie.

La gestion de ces "**TreeView**" dans le programme est fait grâce à deux fonction : "**vueBudget**" et "**vueDepense**". Ces deux fonctions permet au jaque **Budget** et à chaque **Dépense** de la base de données d'etre afficher sur leur "**TreeView**" respectif.

Les fonctionnalités de cet élément nous permettent d'ajouter dans les ligne d'autres éléments graphique comment de barres de progressions montrant le pourcentage de dépense de chaque **Budget** et des "**CheckBox**" pour choisir les ligne que l'on pourrait vouloir supprimer.

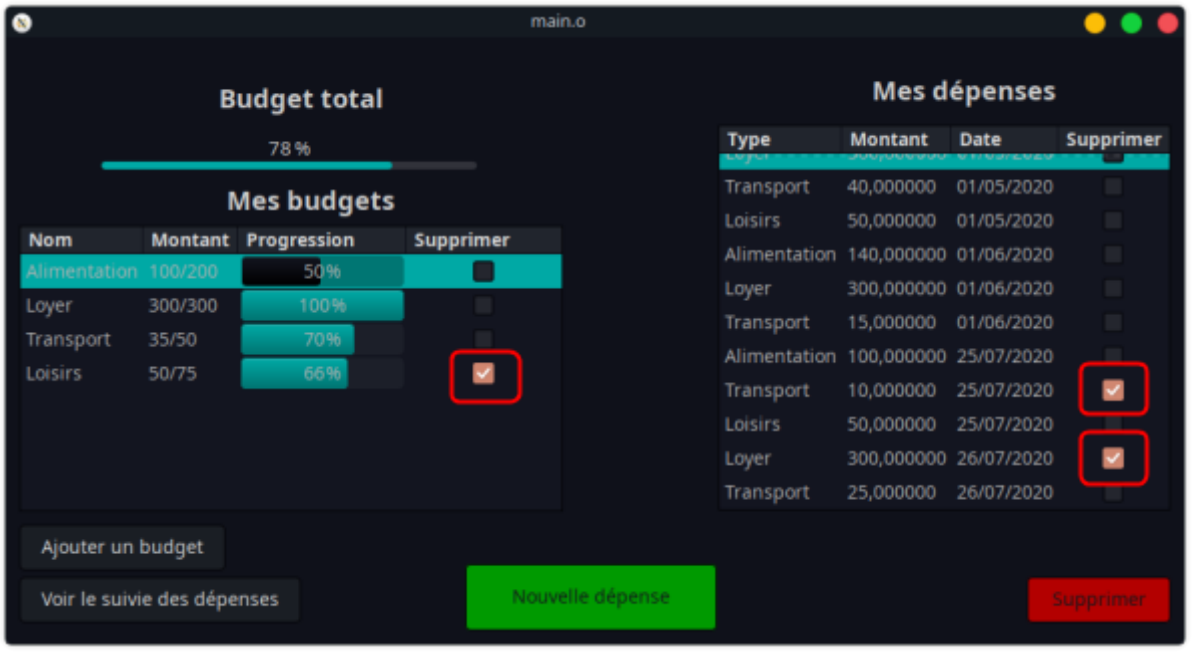
Fonction **vueDepense**

```
void vueDepenses(){
   GtkTreeIter iter;
   sqlite3_stmt *stmt;

   char request[80] = "select * from DEPENSES INNER JOIN BUDGETS on idType = idBudg order
   by dateDep"; // Requête pour récupérer le budget via l'id

   if (sqlite3_prepare_v2(db, request, -1, &stmt, NULL)) {
     printf("ERROR TO SELECT DATA : vueDepenses\n");
     exit(-1);
   }
   gtk_list_store_clear(list_store_dep); // Suppression des Dépenses afficher dans la
   Treeview

   while (sqlite3_step(stmt) == SQLITE_ROW) {
     int id = sqlite3_column_int(stmt, 0); // Variable pour l'id de la Dépense
     double montant = sqlite3_column_double(stmt, 1); // Variable pour le montant
     de la Dépense
     char type[20]; // Variable pour le type de Dépense
     snprintf(type, sizeof(type), "%s", (char *)sqlite3_column_text(stmt, 6));

     char date[20];
     struct tm  ts;
     char  buf[20]; // Récupération de la date d'insertion de la Dépense
     time_t rawtime = atol((char *)sqlite3_column_text(stmt, 3));
     ts = *localtime(&rawtime);
     strftime(buf, sizeof(buf), "%d/%m/%Y", &ts); // Conversion de la date sous
     le format Unix Time au format jour/mois/année
     snprintf(date, sizeof(date), "%s", buf);

     gtk_list_store_append(list_store_dep, &iter);// Ajout d'une nouvelle ligne
     modèle list_store_dep qui est une ListStore
     gtk_list_store_set(list_store_dep, &iter, 0, type, 1, montant, 2, date, 3,
     FALSE, 4, id, -1); // Insertion des donnée dans la ligne
   }
}
```

Pour la fonction "**vueBudget**" les seuls différences sont la récupération du montant total des **Dépense** pour chacun des **Budget** avec la fonction "**getDepensesSumByType"**, ainsi que l'appel à la fonction "**gtk_progress_bar_set_fraction**" qui permet de changer le niveau progression des la barres de progressions.

F/ Suppression de Budget et Dépense

Comme vue précédemment il est possible de **supprimer** des **Budgets** ou des **Dépenses** en sélectionnant le “**CheckBox**” associé à ligne.



Une fois les lignes voulu sélectionnées l’utilisateur doit confirmer la suppression en cliquant sur le bouton “**Supprimer**”.

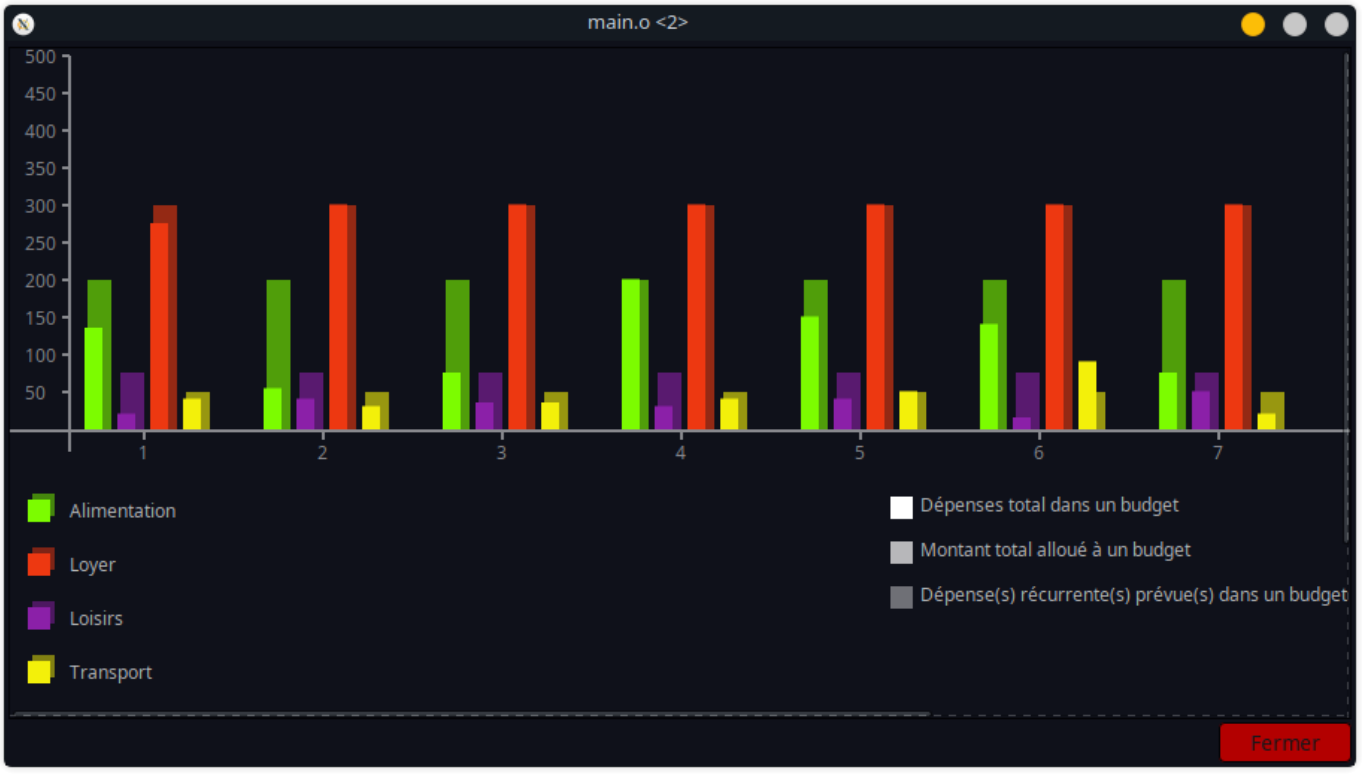
À chaque fois que l’utilisateur clique sur le bouton **Supprimer,** les fonctions “**supprRowBudg**” et “**supprRowDep**” parcoure les ligne de Budget et de Dépense et pour chaque ligne sélectionnée les fonction “**deleteBudg**” ou “**deleteDep”** sont appelé pour supprimer le Budget ou la Dépense correspondant dans la base de donnée.

En sachant que quand un **Budget** est supprimer toutes le **Dépenses** classique et **Récurrente** associées le sont aussi pour éviter tous problèmes.

Une tous les éléments supprimer de la base ont peut actualiser l’affichage des Budgets et des Dépenses grâce aux fonctions “**vueBudg**” et “**vueDep**”.

G/ Suivi des Dépenses et des Budget:

Comment mentionné en introduction le but de ce logiciel est de **fournir un suivie des Dépenses** en fonction de leur type pour savoir où nous dépensons notre argent. L'implémentation d'un **graphique** pour **suivre l'évolution de nos Dépenses** semble donc parfaitement adapté pour répondre à cette problématique.

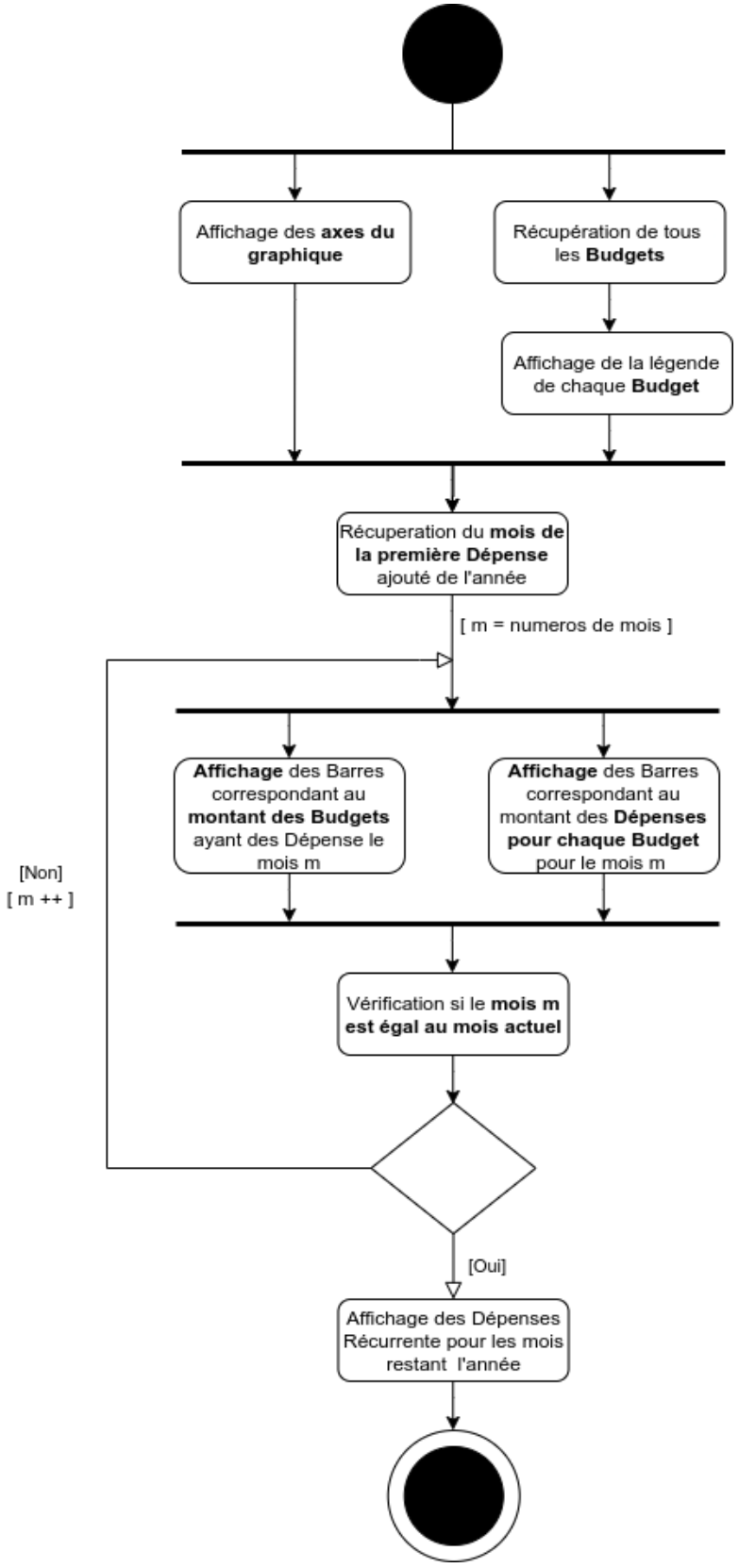


Comment on peut le voire le graphique de suivi est composé de plusieur éléments.
Chaque mois est représenté par une graduation sur l'axe des abscisse, on retrouve les **Budgets présentés sous forme de barre** de couleur.

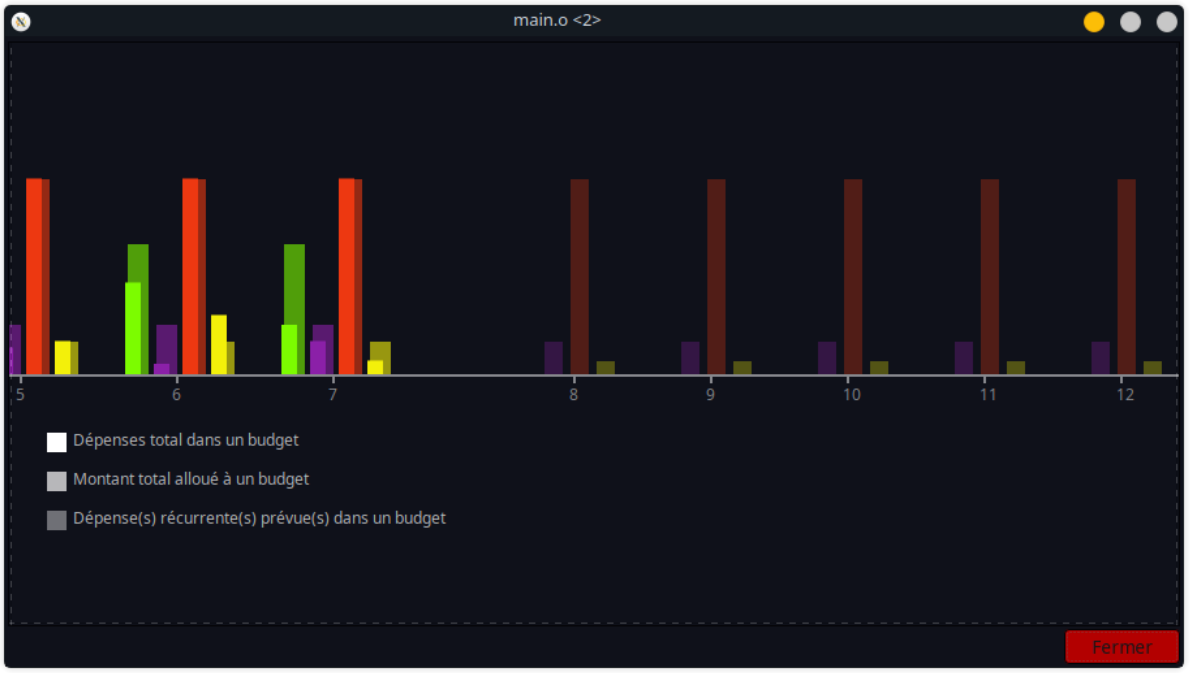
La barre au premier plan de couleur plus foncé représente **le montant des Dépenses** pour le mois dans le Budget et la barre en arrière plan le **montant total associé au Budget**. Cela permet de savoir si il nous reste encors dans l'argent dans ce Budget.

Le graphique commence avec le premier de l'année ayant des Dépenses d'enregistrés, cela permet d'avoir un suivi des Dépenses sur l'année en cours .

Ce **Diagramme d'Activité** décrit la création de l'affichage dû suivi des Dépenses:



L’autre fonctionnalité de ce graphique est d’afficher une **prévision des Dépenses** à venir. Ces Dépenses à venir sont les Dépenses Récurrente enregistré par l'utilisateur elles sont payées **chaque mois** et peuvent donc être prévues. Elles sont affichées avec une couleur correspondant à leur **Budget** associé mais en plus claire pour pouvoir les différencier.

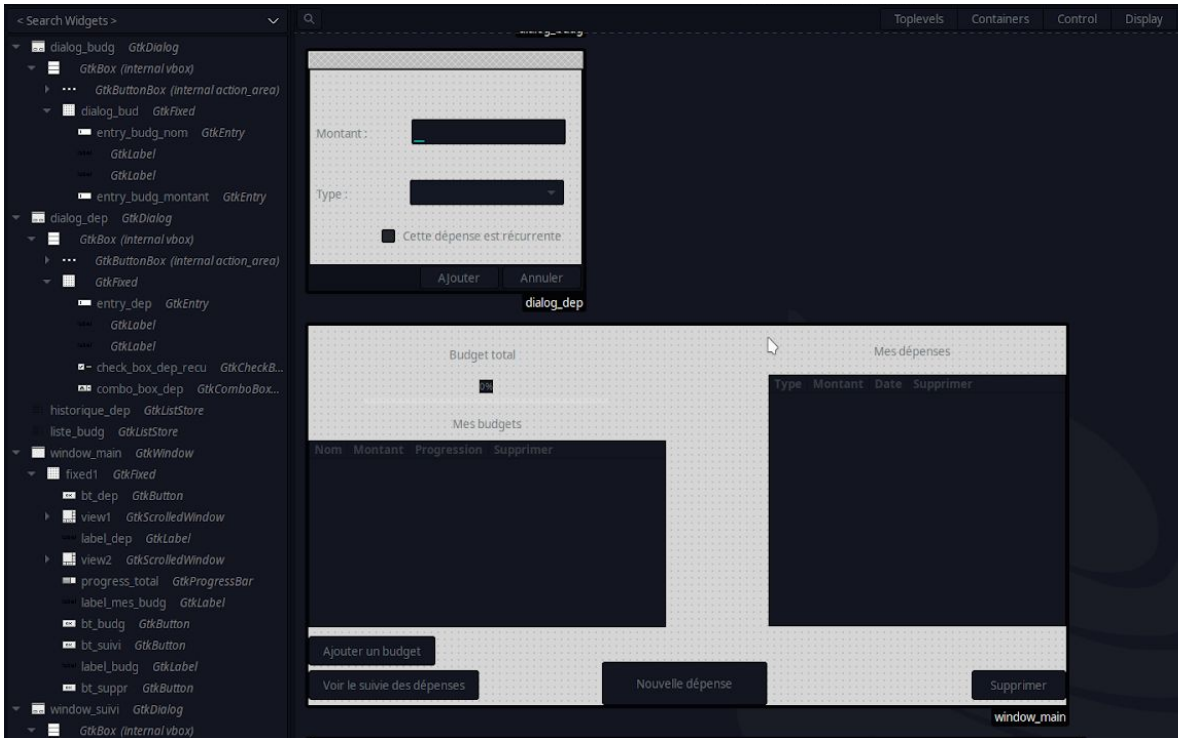


IV/ Problèmes Rencontrés:

- La première difficulté de ce projet fut le **langage de programmation choisie** : **C**. En effet bien qu'ayant déjà utilisé ce langage pendant le première année d'IUT **ce n'était que pour faire des programme basique** pour apprendre les bases du langage. Avant de commencer ce projet j'ai du faire de **nombreuse recherches**, relire mes anciens cours et faire de nombreux tests pour me remémorer le fonctionnement de **C,** comment les avec pointeurs et le fonctionnement de programmes avec plusieurs fichiers.

  De plus je n'avais jamais fait de programme en C utilisant une **interface graphique** ou une **base de donnée**. J'ai du apprendre tout cela par moi même en lisant beaucoup de **documentation** et en regardant un certain nombre de **tutoriel** en ligne pour apprendre ces nouvelles choses.

- Un des autre problème rencontré fut la difficulté de créer une interface ergonomique. Bien que grâce à GTK nous pouvons générer une interface avec un fichier **XML** cela reste compliqué et difficile d'ajouter et personnaliser des éléments graphiques.
  Après des recherches, il se trouve qu'il existe un logiciel de création d'interface appelé **Glade**, il permet de générer un fichier **XML** à partir d'une interface créé. Cela rend le processus de création et de modification de l'interface plus simple et plus rapide.



- La partie la plus complexe de ce projet fut sûrement la création du **graphique de suivi des Dépense**, en effet j'ai du faire de nombreuses recherche pour choisir la meilleur solution. Après avoir lu la documentation de **GTK** au sujet de de l'élément graphique "**GtkdrawingArea**" qui permet de créer des éléments éléments graphique personnalisé j'ai choisie cette élément pour contenir le graphique de **suivi de dépense.** Grâce à la bibliothèque **Cairo** intégré à **GTK** nous pouvons dessiner ce que nous voulons dans cet élément "**GtkDrawingArea**"

  Après avoir trouvé cette technique pour créer le graphique il faut de le **dessiner** le graphique cela fut **long à faire du au nombre d'éléments** à dessiner. Il faut pour chaque mois dessiner toutes les **barres**, soit deux par budget, une pour le montant du budget et une pour le montant des dépenses dans ce Budget, ainsi que les **axes** et leur **graduation** et les montant des Dépenses à venir.

Affichage des barres de **Budget / Dépenses**

```
gdk_cairo_set_source_rgba (cr, &color);
cairo_fill (cr);
cairo_rectangle (cr, (nbBudg - 1) * 22 + 52 + (moisCourant * 40) +
(nbBudgAfficherTotal * 20), 255, 16,  -((budgMontant + 1) / 2));
gdk_rgba_parse (&color, colorsTab[sqlite3_column_int(stmt2 , 1) - 1]);
color.alpha = 0.6;
gdk_cairo_set_source_rgba (cr, &color);
cairo_fill (cr);
```

Le plus difficile fut que pour chaque mois il faut prendre en compte le **nombre de mois précédent** avec le **nombre de Budget précédemment afficher** pour ne pas avoir d'**erreur d'affichage** comme la **superposition** de plusieur barres de Budget.
C'est grâce au trois variable : **nbBudg**, **moisCourant** et **nbBudgAfficherTotal** on peut éviter la superposition des barres et les affichers à la bonne position.

La première variable **nbBudg** elle contient le nombre de Budget déjà affichés ce mois si, pour permettre de séparer les barres de budget d'un même mois de 22 pixels pour plus de visibilité.

La seconde variable **moisCourant** elle à la valeur du nombre de mois déjà affiché, pour permettre elle de séparer chaque **Mois** les un des autre de 40 pixels**.**

La dernière variable **nbBudgAfficherTotal** et celle qui fut le plus difficil à définir, elle contient le **nombre de Budget total** déjà affiché sur le graphique. Ce qui est essentiel pour avoir un bonne affiche sans superposition.

V/ Conclusion:

Pour conclure la problématique annoncée au début à été en grande partie répondue par le logiciel et ses fonctionnalité. Nous avons un logiciel capable d'**enreV Conclusion:gistrer des Budget** créé par l'utilisateur pour tout type de Dépense que qu'il souhaite suivre. Il peut y **ajouter des Dépenses** et ainsi voit combien il dépense dans chaque secteur sur le mois. De plus le **graphique de suivi** des Budgets permet d'avoir une **vue plus globale sur ses Dépenses** au cours du temps. Pour finir les **Dépenses Récurrentes** permettent d'avoir un logiciel plus intéressant permettant de prévoir les futures dépenses qui reviennent tous les mois et de les anticiper.

Pour finir j'aimerais remercier; Mme.Illina, M.Mangeol les autres personnes qui m'ont permis de faire de travail de remplacement et m'ont aidé au cours de ce projet et à le mener à bien.