

- Explication de la problématique X
- État de l'art X
- Fonctionnalité X
- Problèmes rencontrés

I Problématique:

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

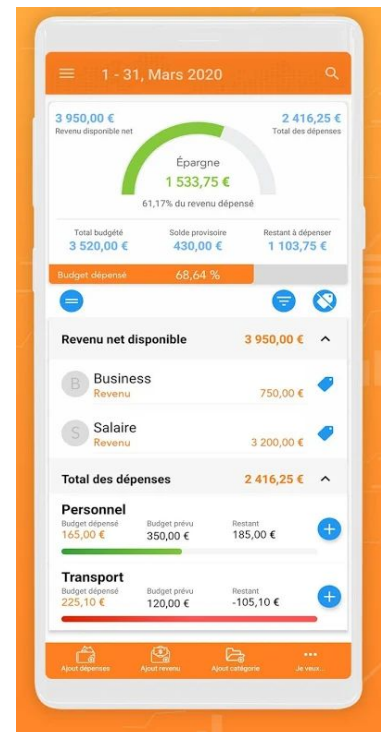
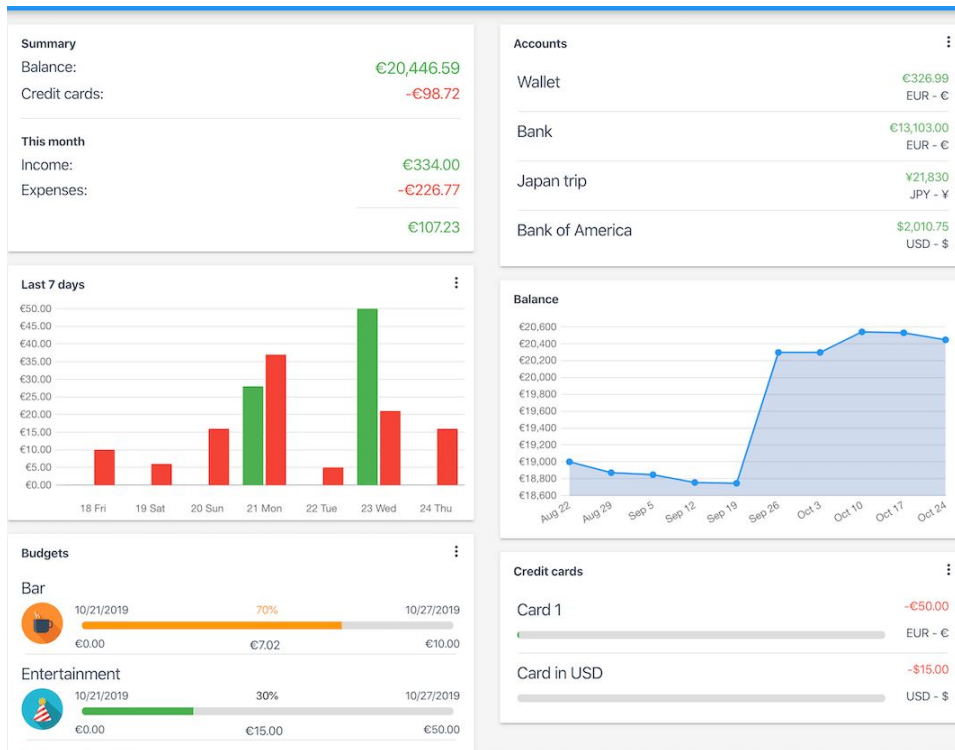
Il est facile de dépenser plus d'argent que l'on ne le pense dans certains type de produits et de services sans forcément se rendre compte des montant totaux.

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 paiement 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.

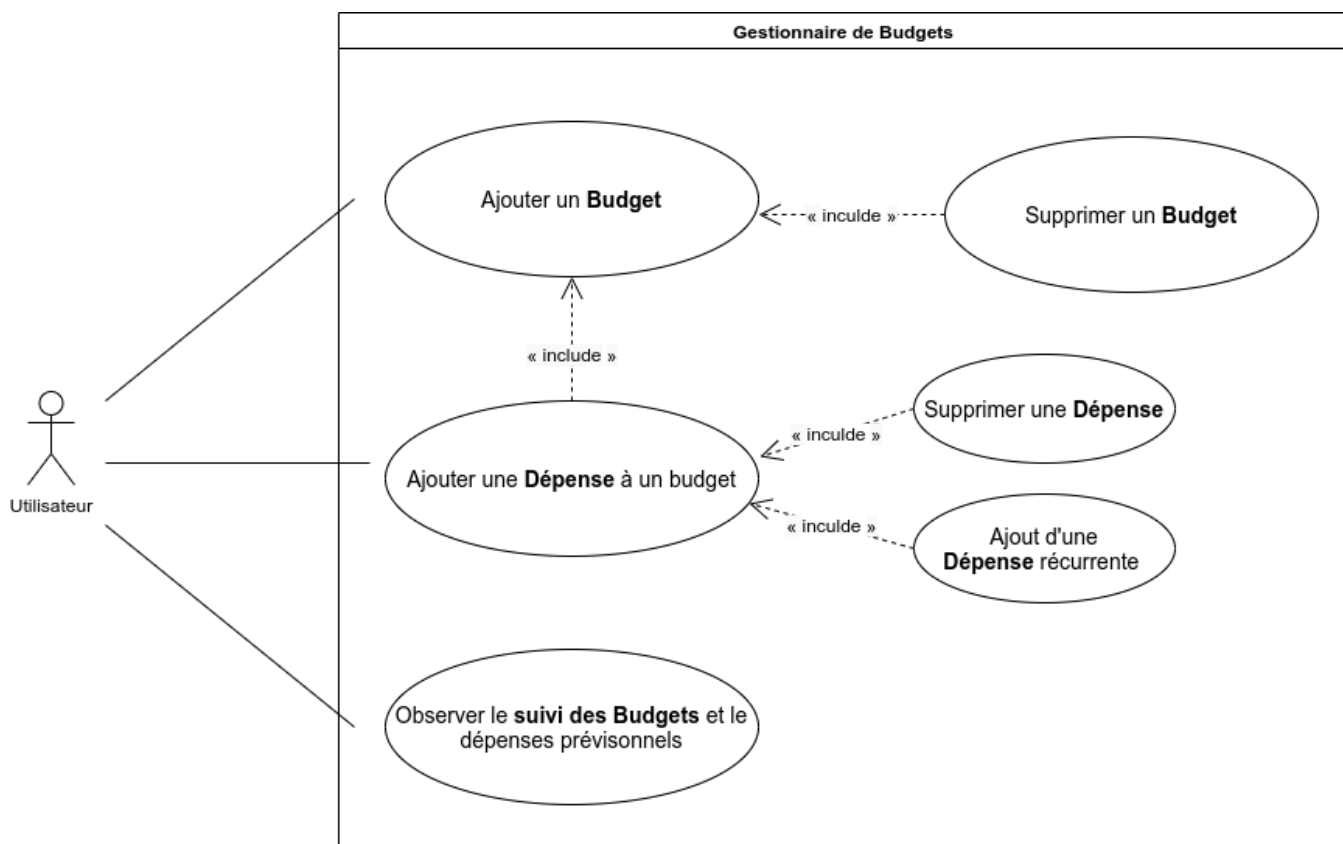
III Fonctionnalités:

Après étude des différents outils existants une sélection de **fonctionnalités indispensables** pour tous **gestionnaires de budget** ressort ainsi que d'autres plus spécifiques à la problématique présentée précédemment.

- Création de **Budget** avec un type et un montant
- Ajout de **Dépenses** avec un type, un montant et une 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

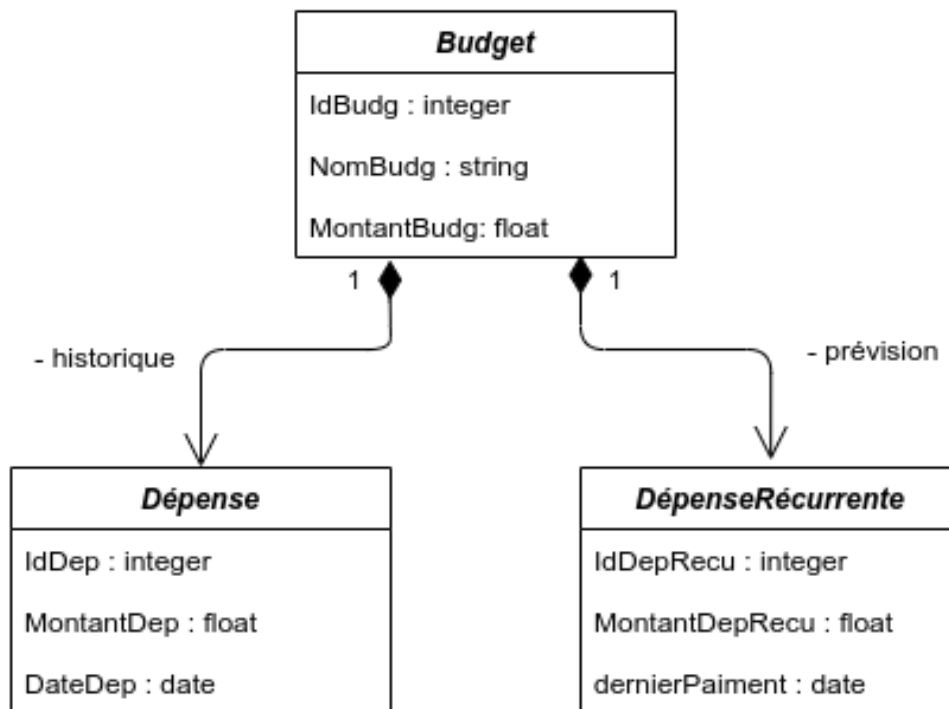
Cas d'utilisations:

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



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, dernierPaiement)

```

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
  
```

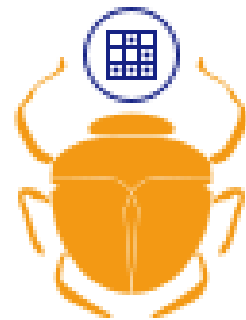
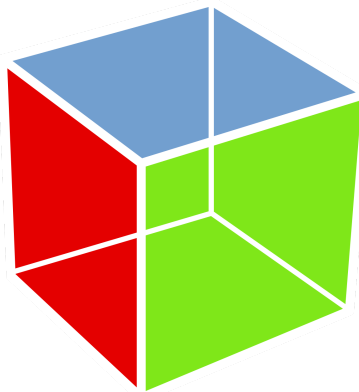
```

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

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.

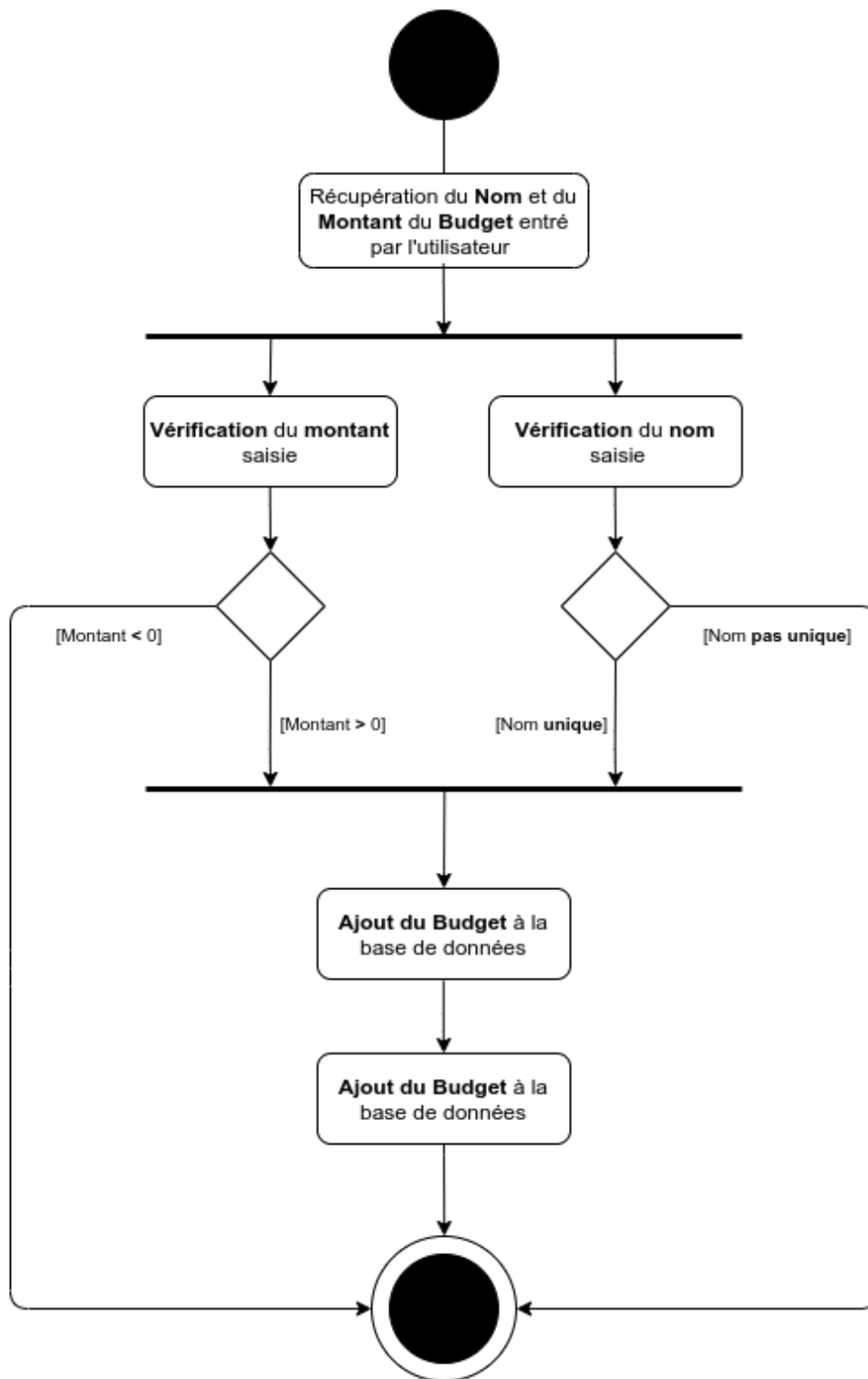
De plus GTK donne la possibilité de générer une interface via un fichier **XML** ce qui facilite sa création et l'arrangement des éléments graphiques.



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 données facilement sans serveur, elle est simplement stockée dans un fichier text.



1/ 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.

The screenshot shows a dialog box titled 'Ajouter un Budget'. It contains two input fields: 'Nom :' and 'Montant :'. At the bottom, there are two buttons: 'Ajouter' (green) and 'Annuler' (red).

La fonction "**budgUnique**" vérifie si le nom d'un budget est unique elle permet de ne pas avoir plusieurs budget avec le même 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 même nom dans la base de données.

Fonction **budgUnique**

```
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 saisis par l'utilisateur nous utilisons la fonction "**sqlite3_exec()**" qui permet d'exécuter une requête SQL en un ligne.

Insertion d'un **Budget**

```
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.

2/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 de la validité du montant (> 0) et de l'existence du **Budget** sélectionné, une requête SQL est générée et la nouvelle dépense est insérée 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
    }
}
```

2/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.

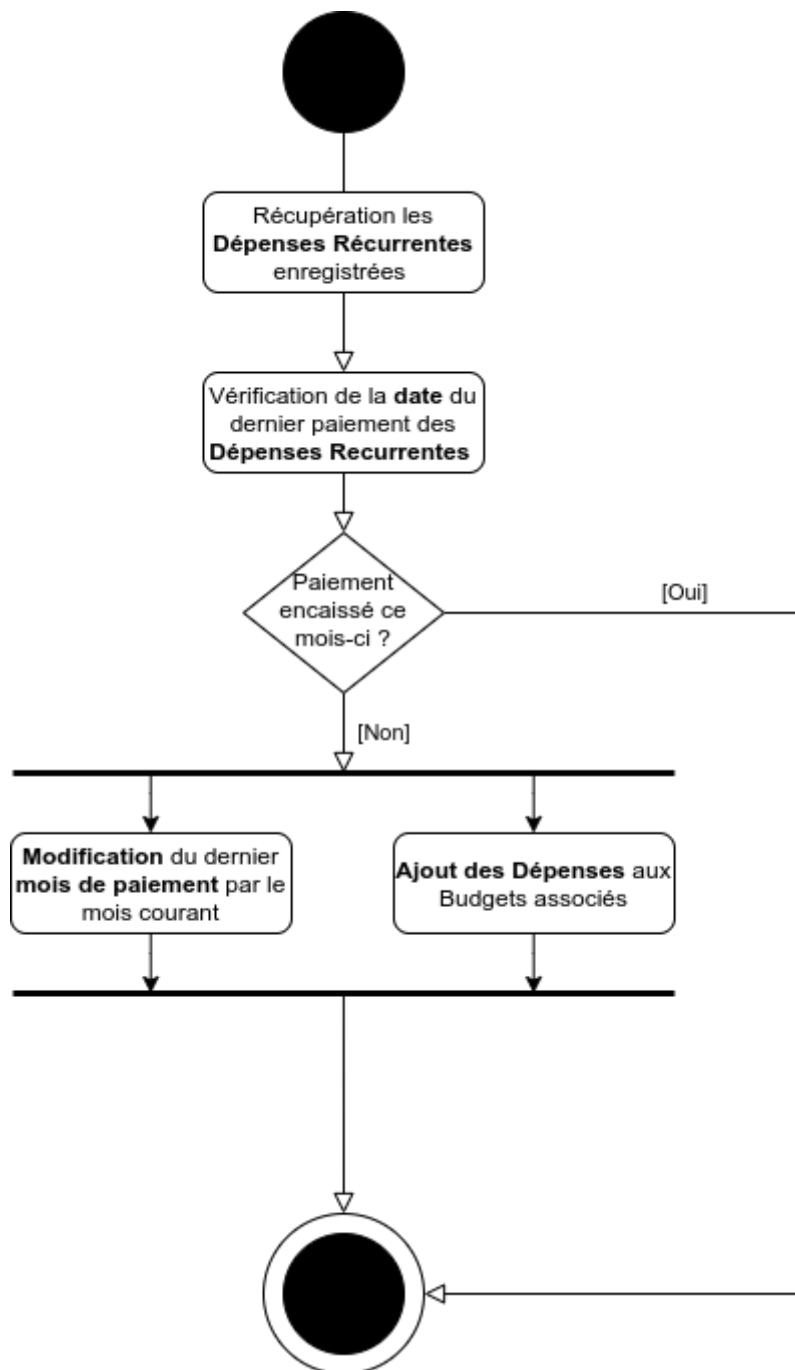
Nous pouvons donc appeler la fonction correspondante au type de **Dépense** soit **"insertDepense"** ou **"insertDepenseRecu"** en leur passant en paramètre le montant entré par l'utilisateur ainsi que l'id associé au Budget sélectionné.

Dépense classique ou Récurrente

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

3/ Paiement des Dépenses Récurrentes:

Comment expliqué précédemment les **Dépenses Récurrentes** 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.

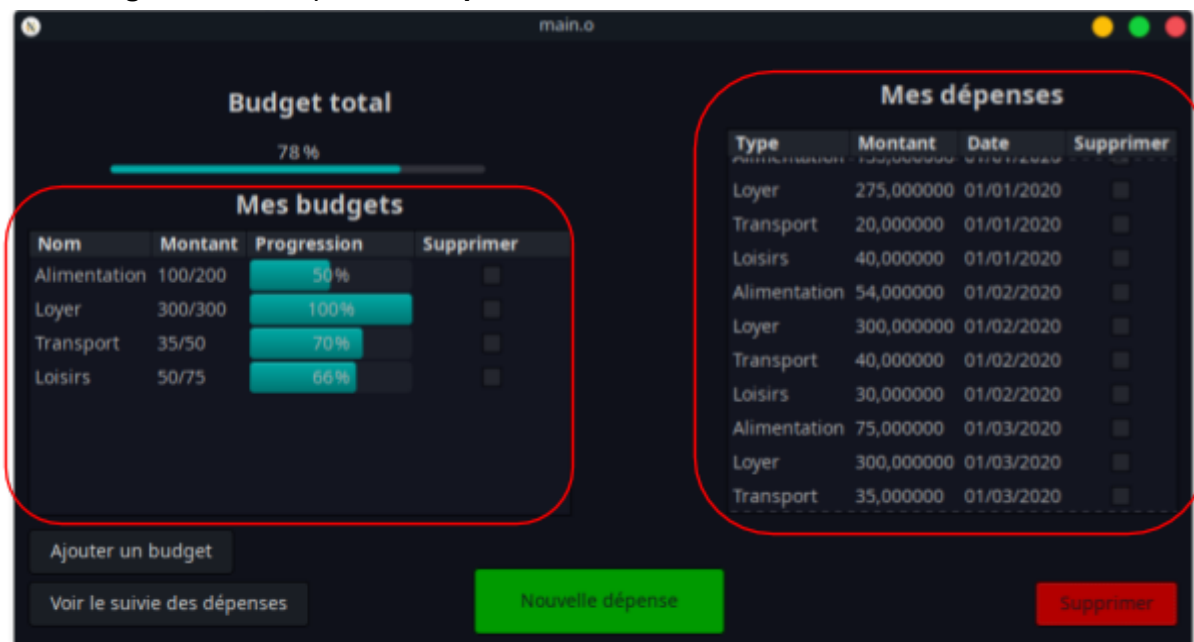


4/ Affichage des Budget et de l'historique des Dépenses:

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 chaque'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 chaque **Budget** et à chaque **Dépense** de la base de données d'être 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.

5/ 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.

The screenshot displays a web application window titled 'main.o'. It features two primary data sections: 'Budget total' and 'Mes dépenses'.

Budget total: A progress bar indicates 78% completion. Below it, the 'Mes budgets' table lists the following data:

Nom	Montant	Progression	Supprimer
Alimentation	100/200	50%	<input type="checkbox"/>
Loyer	300/300	100%	<input type="checkbox"/>
Transport	35/50	70%	<input type="checkbox"/>
Loisirs	50/75	66%	<input checked="" type="checkbox"/>

Below the table are buttons: 'Ajouter un budget' and 'Voir le suivi des dépenses'.

Mes dépenses: A table lists the following data:

Type	Montant	Date	Supprimer
Transport	40,000000	01/05/2020	<input type="checkbox"/>
Loisirs	50,000000	01/05/2020	<input type="checkbox"/>
Alimentation	140,000000	01/06/2020	<input type="checkbox"/>
Loyer	300,000000	01/06/2020	<input type="checkbox"/>
Transport	15,000000	01/06/2020	<input type="checkbox"/>
Alimentation	100,000000	25/07/2020	<input type="checkbox"/>
Transport	10,000000	25/07/2020	<input checked="" type="checkbox"/>
Loisirs	50,000000	25/07/2020	<input type="checkbox"/>
Loyer	300,000000	26/07/2020	<input checked="" type="checkbox"/>
Transport	25,000000	26/07/2020	<input type="checkbox"/>

At the bottom right is a 'Supprimer' button. A central green button is labeled 'Nouvelle dépense'.

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

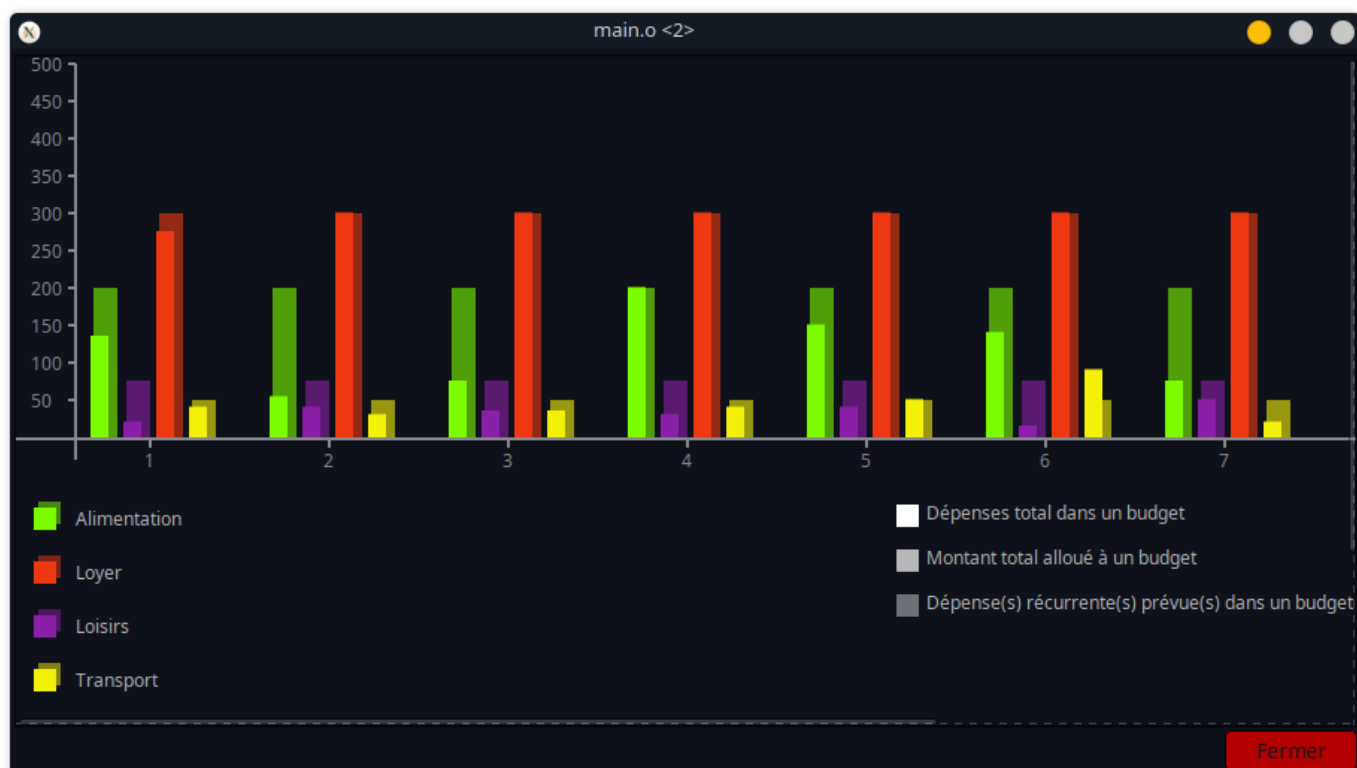
À chaque fois que l'utilisateur clique sur le bouton **Supprimer**, les fonctions “**supprRowBudg**” et “**supprRowDep**” parcourt les ligne de Budget et de Dépense et pour chaque ligne sélectionnée les fonction “**deleteBudg**” ou “**deleteDep**” sont appelé pour supprimer le Budget ou la Dépense correspondant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

6/ 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.

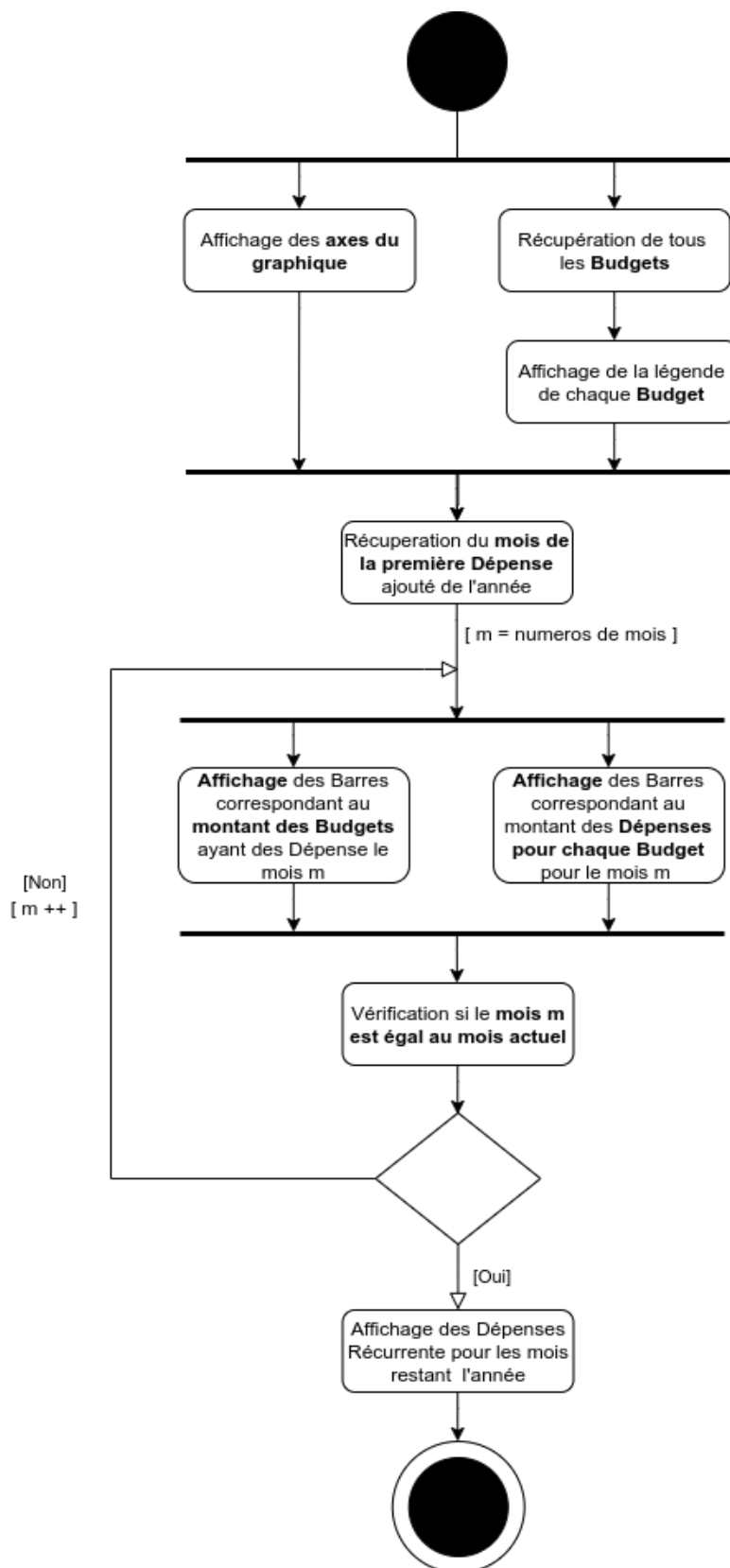


Comment on peut le voir le graphique de suivi est composé de plusieurs éléments. **Chaque mois est représenté par une graduation** sur l'axe des abscisses, on retrouve les **Budgets présentés sous forme de barres** de couleur.

La barre au premier plan de couleur plus foncée 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 encore 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.

