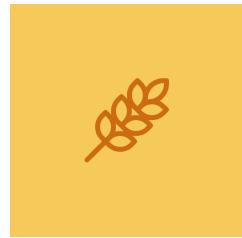


Rapport de projet



Lafo-cheuse

「 Application de gestion de budget pour étudiants 」

GOBLLOT David, VIALA Alexandre

Sommaire

Présentation du groupe	3
Présentation du sujet et objectifs	3
Analyse du besoin	3
Analyse de l'existant	6
Rédaction d'un persona	7
Outils utilisés	8
Diagramme de cas d'utilisation	9
Diagramme SADT	10
Maquettes	11
Modèle de données	13
Architecture de programmation	15
Modélisation PAC	16
Écran d'accueil	16
Écran de budget spécifique	17
Écran d'ajout de dépense/revenu	18
Écran d'ajout de dépense/revenu régulier	19
Écran de choix de catégorie	19
Écran de paramètres	20
Menu de sélection d'écran	20
Diagrammes de contrôle	21
Diagramme de contrôle global	21
Diagramme d'affichage du budget	22
Diagramme d'affichage de budget spécifique	23
Diagramme de l'affichage de l'écran de dépenses/revenus	24
Diagramme de réglage des paramètres	24
Phase de réalisation	25
Résultat final et évaluation	26

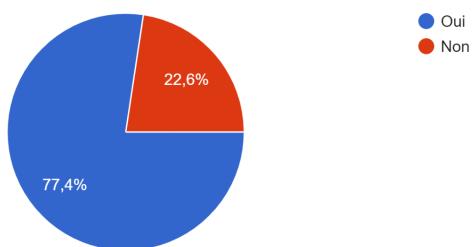
1. Présentation du groupe

Ce projet est réalisé par Alexandre VIALA et David GOBLOT, étudiants en informatique à l'UTBM dans le cadre du cours Interface Homme Machine HM40, en collaboration avec le cours Android development SY43.

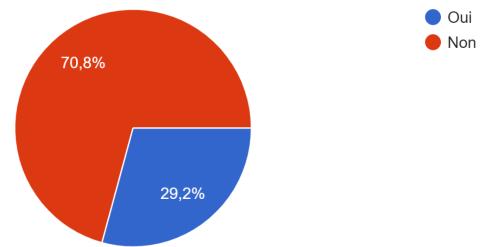
2. Présentation du sujet et objectifs

Le cours de SY43 nous demande de réaliser sous Android une application de gestion budgétaire. Nous avons choisi de lier ce projet au cours de HM40, en faisant un travail d'analyse/conception sur l'interface de l'application et en approfondissant son aspect graphique.

Gérez-vous vos comptes de manière régulière ?
168 réponses



Utilisez-vous une application de gestion budgétaire ?
130 réponses

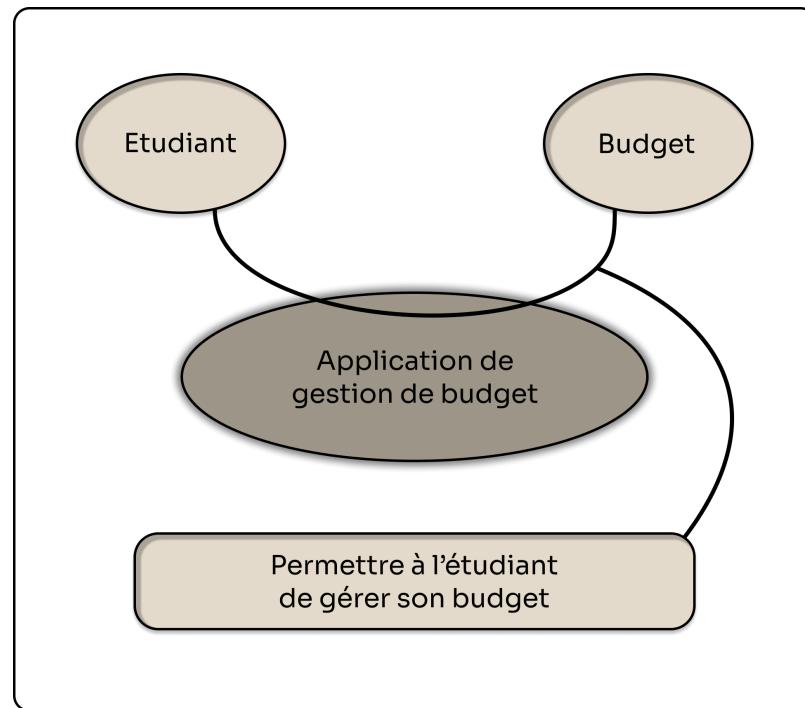


Comme on peut le voir sur ces graphiques tirés d'une enquête donnée aux étudiants de l'UTBM, beaucoup d'étudiants gèrent leur budget mais n'utilisent pas d'application pour cela, il y a donc une vraie demande. L'objectif final est donc que cette application puisse être mise en ligne sur le PlayStore (Plateforme de diffusion d'application Android) afin que des étudiants même hors de l'UTBM puissent la télécharger et l'utiliser au quotidien

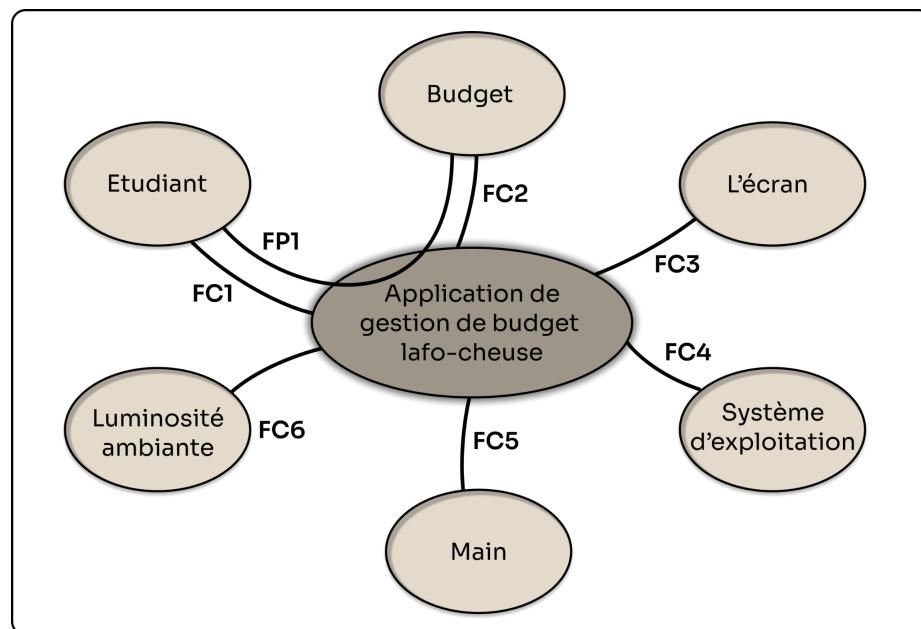
3. Analyse du besoin

Comme dit plus haut, nous avons réalisé une enquête au moyen d'un questionnaire diffusé auprès des étudiants. Nous avons donc basé ce cahier des charges sur les résultats obtenus.

Nous avons en premier lieu étudié le besoin avec un diagramme bête à corne de la méthode APTE.



Une fois notre besoin principal identifié, nous avons déterminé les besoins secondaires au moyen d'un diagramme pieuvre et de fonctions contraintes :



Fonction de Service	Critère	Priorité (F0 -> Plus grande priorité)
FP1 : Gérer son budget	Permettre à l'étudiant de gérer son budget	F0
	Consulter le budget	F0
	Planifier un budget	F0
FC1 : Accéder à l'application	Être visuellement agréable grâce à un design minimaliste (icônes minimalistes, graphiques thématiques, etc.)	F2
	Pas plus de 2-3 écran pour accéder à une information (principale, ajout de dépenses/revenues et paramètre)	F1
	Limiter le nombre d'informations sur la page	F2
	Ajouter des dépenses	F0
	Ajouter des revenus	F0
	Mot de passe pour accéder à l'application (type code PIN à 4 chiffres)	F4
FC2 : Interagir avec son budget	Graphiques claires du solde actuel	F2
	Ajout de dépenses et revenus rapides	F2
	Ecran de planification du budget (dépenses/revenus mensuels)	F1
FC3 : S'adapter à la taille de l'écran	Prendre en compte un écran PC et un écran de téléphone	F5
	S'adapter au mode paysage pour le téléphone	F5
	S'adapter à l'écran de téléphone	F1
FC4 : Prendre en compte le système d'exploitation	Générer des notifications sous Android pour indiquer la fin du mois ou le bilan	F4
	Enregistrer dans une base de données les informations	F1
	Chiffrer la base de données	F3

	Exporter au format csv les données	F3
FC5 : Maniabilité et ergonomie	Travailler l'interface pour qu'elle soit agréable à utiliser, même à une main	F2
	Options d'accessibilités (taille de la police, différent types de mot de passe)	F5
FC6 : S'adapter à la luminosité de l'environnement	Possibilité de passer en Dark Mode	F3

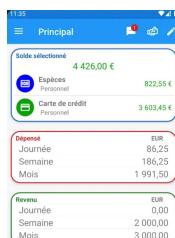
4. Analyse de l'existant

Afin de construire une interface la plus proche des besoins des utilisateurs, nous avons également réalisé une analyse des logiciels existants. Nous avons notamment interrogé les étudiants sur 3 interfaces différentes.



Gestion des dépenses et budget, finances, argent de *Innim Mobile Exp*

Ce premier visuel a été globalement bien apprécié par les utilisateurs. Il présente le bilan sous la forme d'un diagramme circulaire. Le nombre d'informations est réduit à l'essentiel.



Budget - Suivi des dépenses de *Alexey Kobyakov*

Ce deuxième visuel a été beaucoup moins apprécié que les autres. Il présente les données de manière chiffrée sans mise en forme graphique. Les couleurs donnent un effet "scolaire".



1Money: dépenses, finances de *PixelRush*

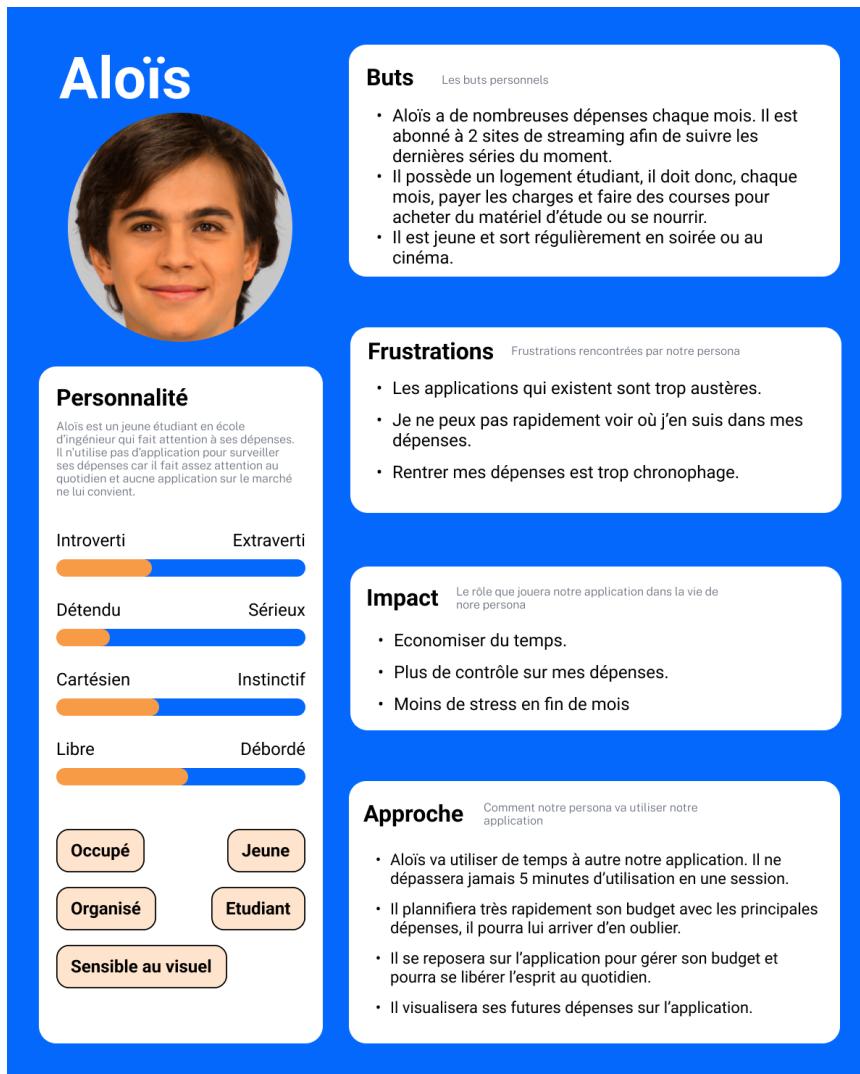
Ce troisième visuel a également été relativement apprécié mais les résultats sont moins unanimes que pour la première interface. Le fait d'afficher beaucoup de données sur l'écran semble diviser.

Nous avons également interrogé les utilisateurs sur les applications de gestion de budget qu'ils utilisaient. Il s'est avéré que les interrogés utilisaient principalement l'application **tricount** ou les intégrations du budget dans les **applications bancaires**.

Nous devrons donc étudier en profondeur ces applications afin de déceler des fonctionnalités communes à ces applications que nous devrons intégrer, voir améliorer, dans notre application.

5. Rédaction d'un persona

Afin de faciliter le développement de notre application, nous avons conçu un persona dont le but est de fournir un profil type d'une personne qui pourrait utiliser notre application.



6. Outils utilisés

- Codage de l'application : **Android Studio** et **Git**



+

- Gestion du projet et planification : **Notion**



- Maquettes graphiques : **Penpot**

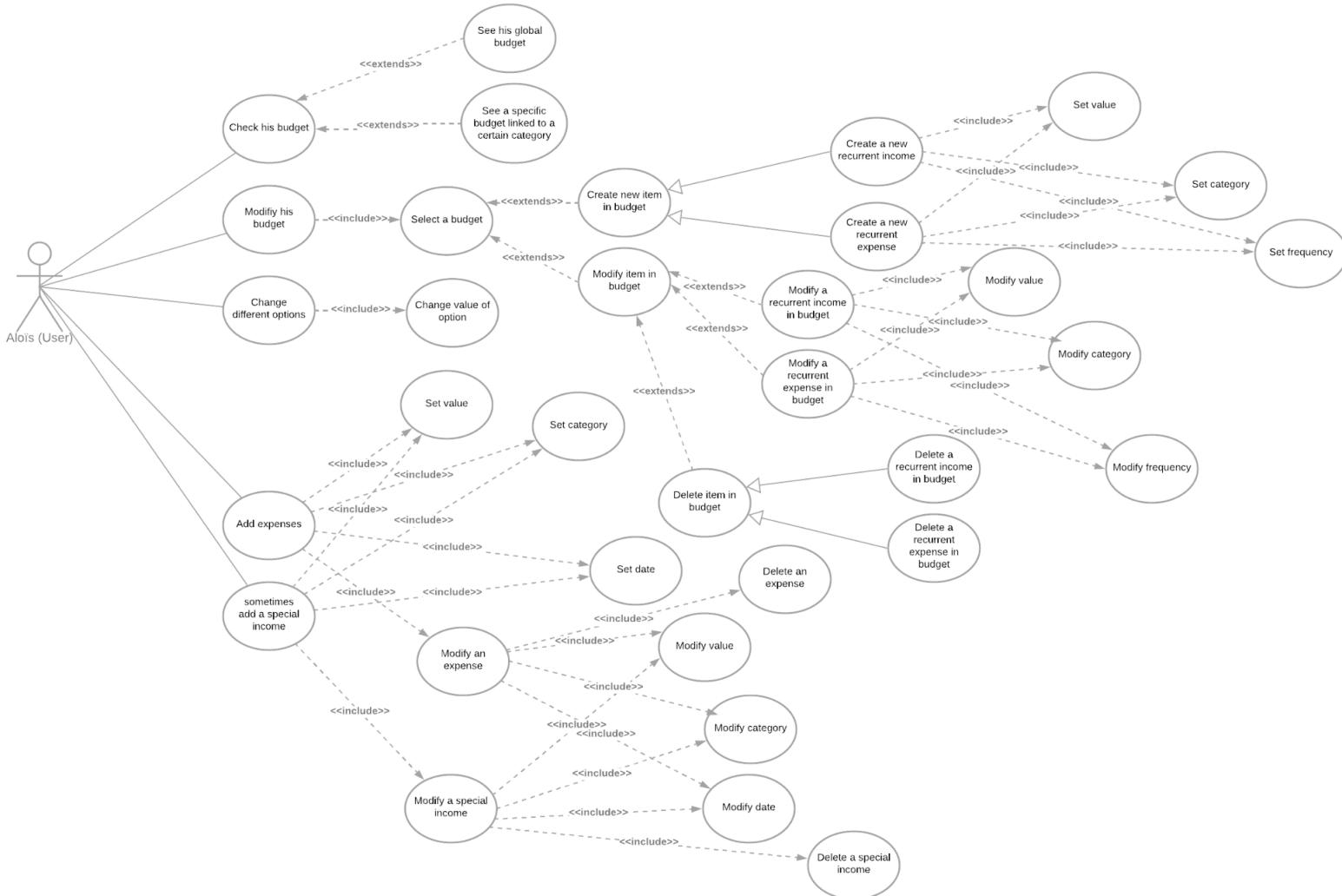


- Modèle de données : **Lucidchart**



7. Diagramme de cas d'utilisation

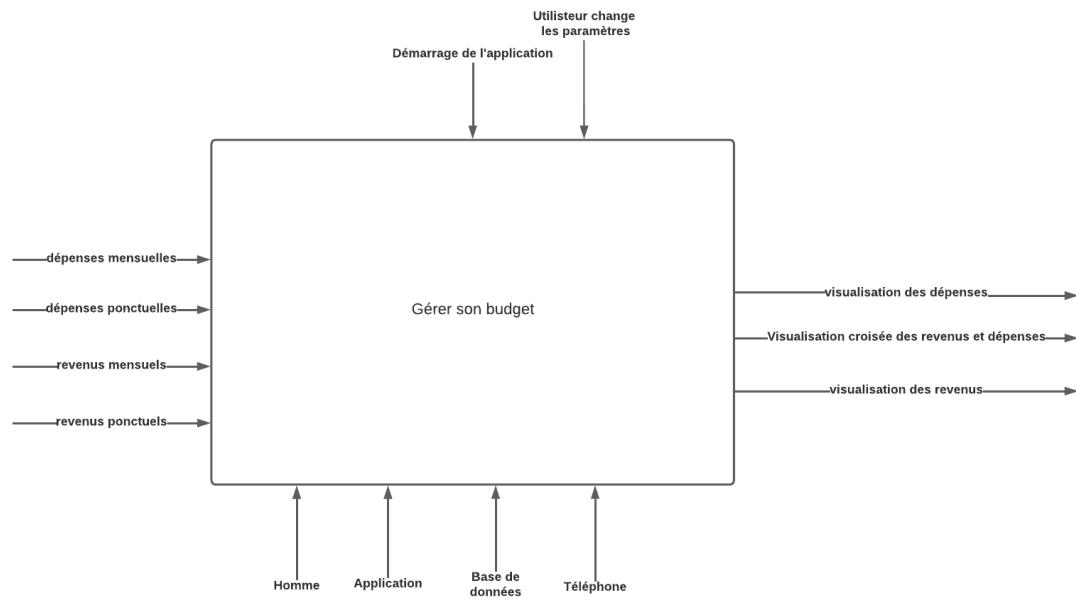
Afin de mieux penser aux fonctionnalités nécessaires à notre utilisateur, nous avons réalisé un diagramme UML de cas d'utilisation



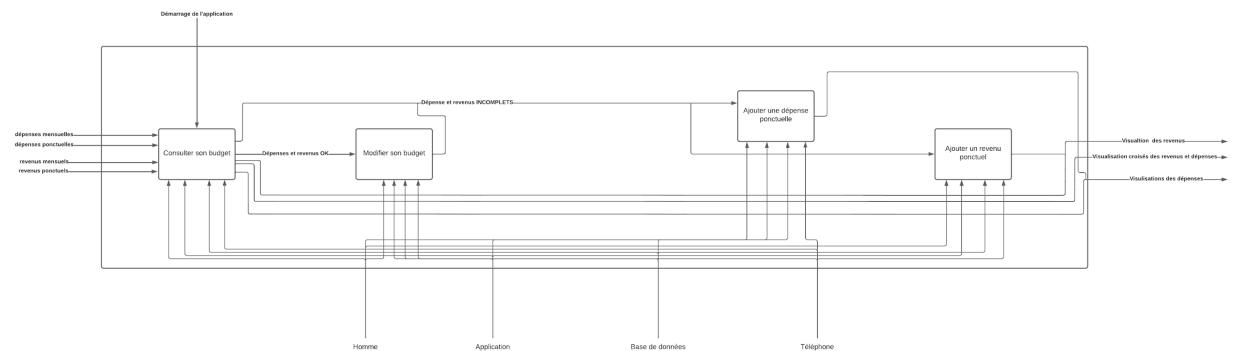
8. Diagramme SADT

Afin de concevoir notre application, nous avons conçu un diagramme SADT de niveau 0 et de niveau 1 afin de connaître grossièrement quelles fonctionnalités nous devons implémenter. Nous avons basé ces diagrammes SADT sur le diagramme de cas d'utilisations vus plus haut.

1. Diagramme A0

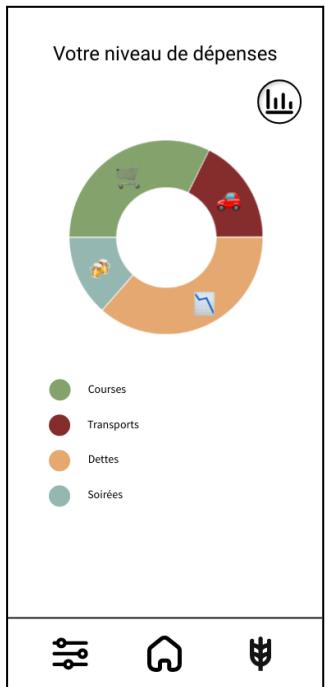


2. Diagramme A1

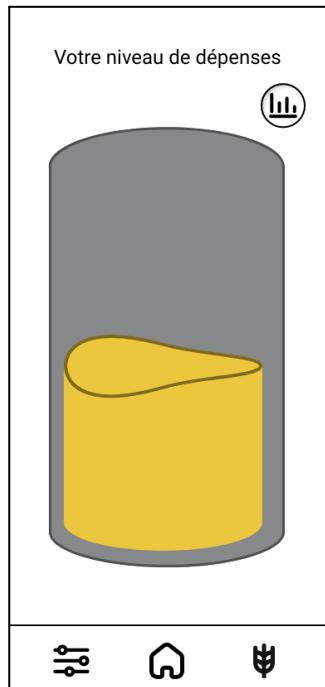


9. Maquettes

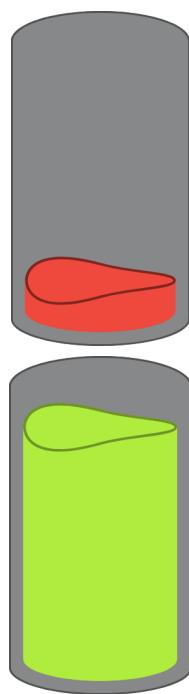
Ecran principal



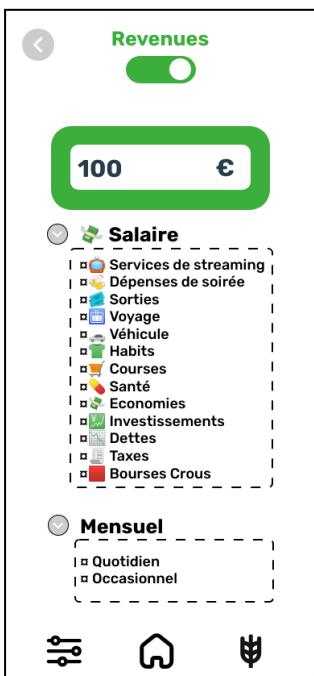
Ecran de budget spécifique



Evolution du budget



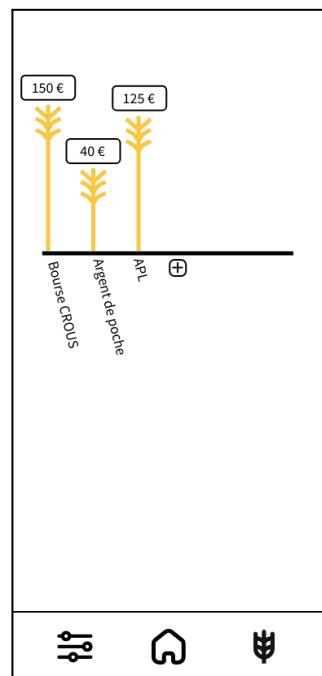
Revenu ponctuel



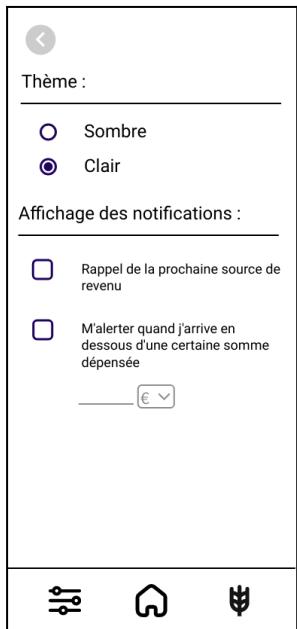
Dépense ponctuelle



Mise en place du budget



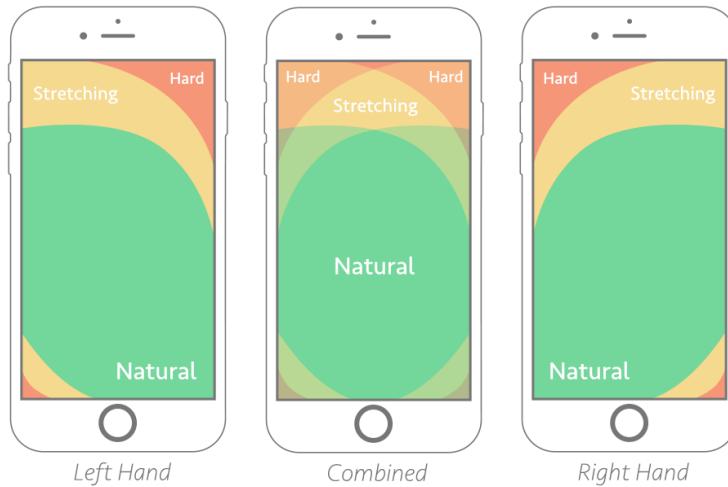
Écran d'options



Nouveau menu



Pour la conception du menu, nous nous sommes appuyés sur la zone d'accessibilité du pouce. En effet, de nombreuses sources s'accordent pour dire que l'espace d'un écran tactile n'est pas toujours très accessible par le pouce. C'est pourquoi, nous avons voulu concevoir un menu qui s'ouvre en bulle pour faciliter la navigation.



source : <https://www.smashingmagazine.com/2016/09/the-thumb-zone-designing-for-mobile-users/>

10. Modèle de données

En nous renseignant sur Android, nous avons pu voir que Android disposait de plusieurs mécanismes pour conserver les données. Afin de conserver correctement les données de l'utilisateur de manière pérenne, il nous était indispensable de stocker ces données dans un fichier.

Nous pouvions utiliser différentes technologies et nous nous sommes donc renseignés sur les différentes possibilités :

- SQLite

Il s'agit de la solution de base pour stocker des données sur Android. Il s'agit d'une version allégée de PostGreSQL. Ce système ne présente malheureusement pas de hash de données. Néanmoins, il est possible de hasher les données depuis Android.

- MariaDB

Il s'agit d'un système de base de données très populaire au sein de la communauté Unix. Néanmoins ce système n'est absolument pas natif à la programmation Android et aucune bibliothèque facile d'utilisation n'existe.

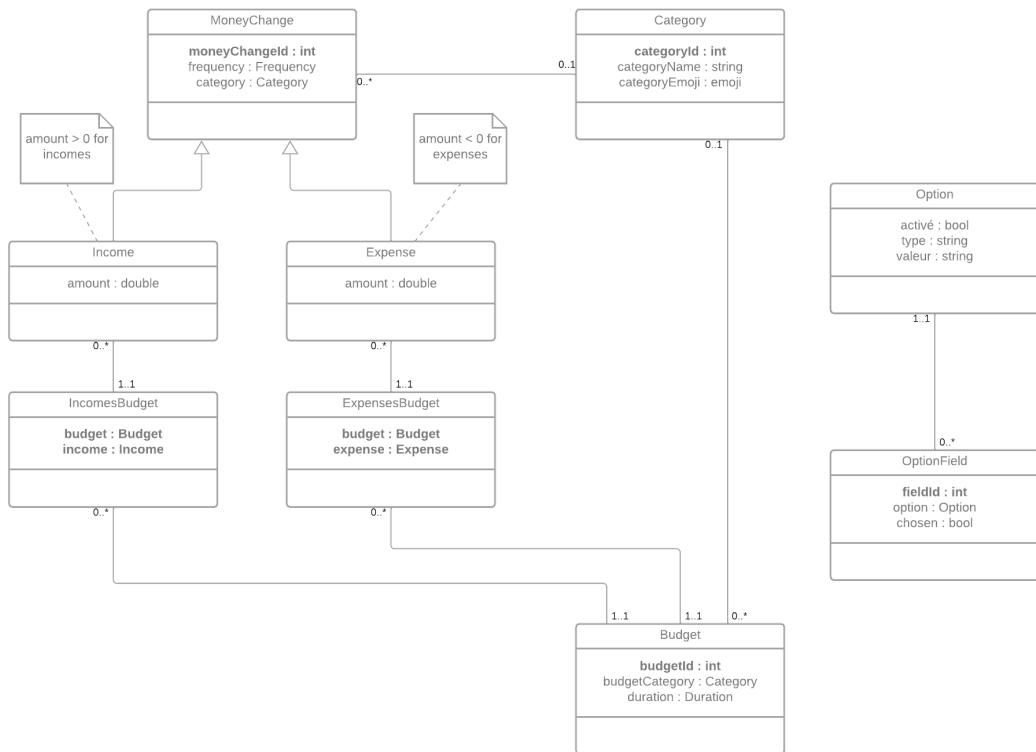
- MongoDB et stockage au format JSON :

Cette technologie est relativement récente et prend le pas de plus en plus sur les bases de données relationnelles. Ces logiciels utilisent des bases de données NoSQL utilisées notamment pour les grandes infrastructures. Ces bases de données sont plus flexibles car ne nécessitent pas de relation entre tables. Elles sont néanmoins moins adaptées lorsque l'on connaît la structure précise des données qu'on veut stocker.

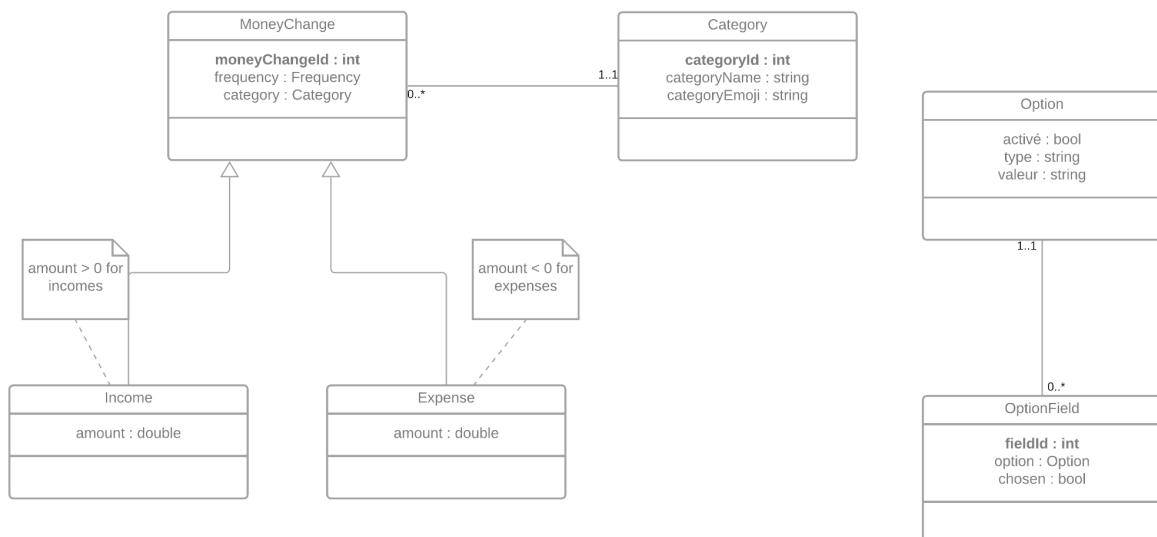
- PostGreSQL :

C'est un système de base de données relativement conséquent qui présente beaucoup de fonctionnalités. Des bibliothèques simples d'utilisation existent sur Android afin d'implémenter de telles bases de données.

Nous avons finalement choisi de nous orienter vers une base de données SQLite car cette dernière est implémentée de base dans Android. Nous avons pour cela utilisé la bibliothèque Room qui permet de manipuler des bases de données grâce à une abstraction ORM.



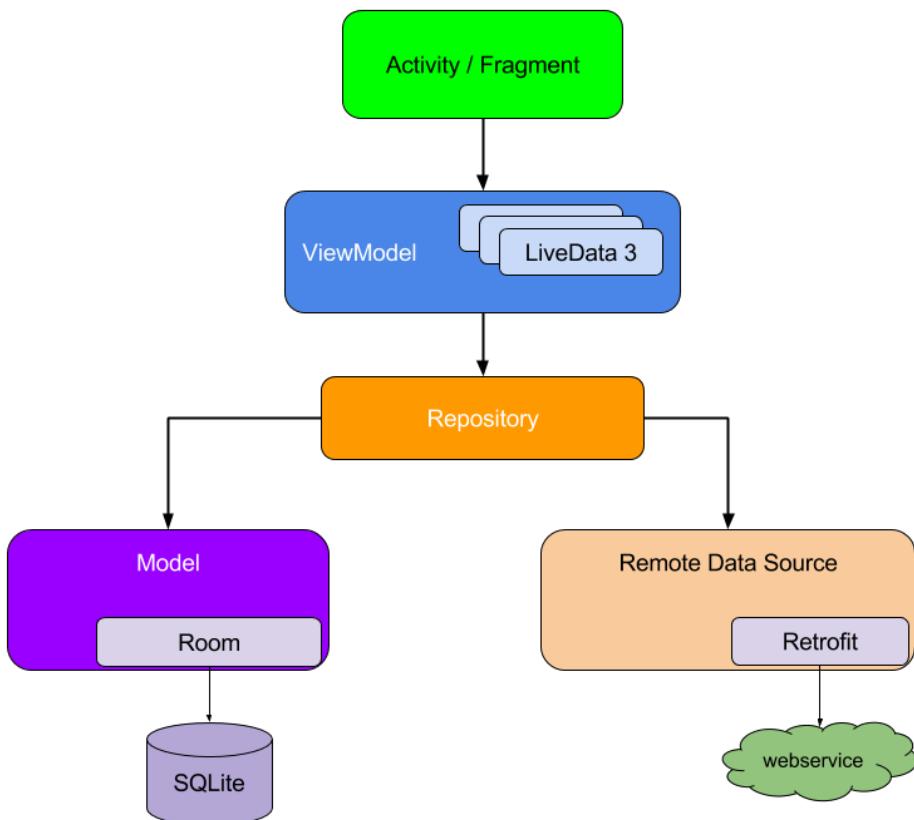
Une fois la programmation commencée, nous nous sommes rendu compte que ce modèle de données pouvait être simplifié :



11. Architecture de programmation

La programmation Android est très normalisée. Nous pensions d'abord programmer selon une logique MVC (Model View Controller). Néanmoins, il s'agit d'une technique de programmation qui n'est plus recommandée.

Les recommandations de Google pour la programmation sous Android sont d'adopter une variante de l'architecture Model View ViewModel (MVVM). Cette variante se nomme l'Android Architecture Components.



source : [Découvrez l'Architecture Components - Gérez vos données localement pour avoir une application 100 % hors-ligne - OpenClassrooms](#)

La programmation sous Android nécessite de disposer de plusieurs fichiers différents :

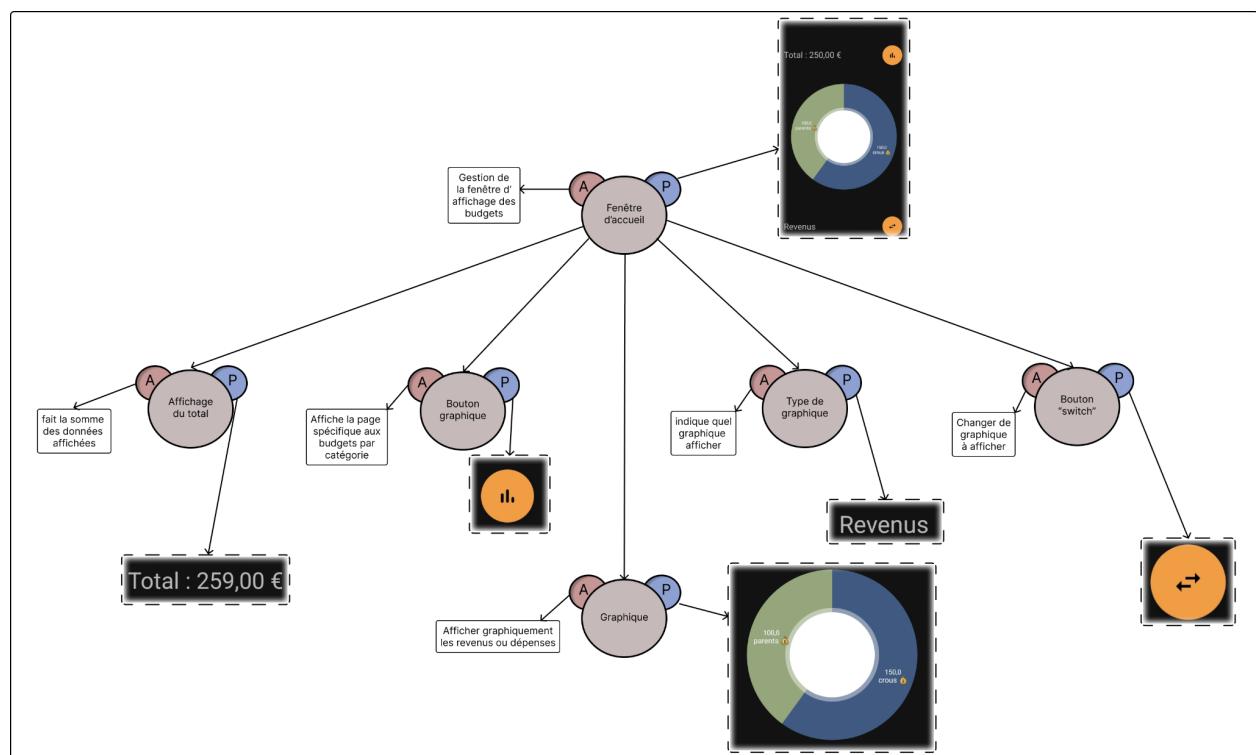
- **des fichiers de layout xml** : il s'agit de fichiers qui contiennent la déclaration visuelle de l'application, un peu comme le ferait un fichier html.
- **des activités** : il s'agit de classes java (ou kotlin) qui vont faire office de backend au fichier xml. Ils agissent comme des contrôleurs.
- **des fragments** : les fragments sont comme des activités mais ils sont néanmoins utilisés comme des composants unitaires. On va notamment les utiliser quand l'on veut utiliser une fonctionnalité spécifique dans plusieurs activités.

- **des ViewModel** : Ces composants vont fournir les données au contrôleur au format graphique afin que l'activité puisse disposer de données graphiques.
- **Un Repository** : Le repository est l'élément qui permet l'interaction entre les modèles et l'interface. Il va permettre d'interagir avec les sources distantes (API de sites web par exemple) ou les sources locales.
- **Un Model** : Le modèle est l'élément qui va stocker toutes nos données. Il est représenté sous forme de class java (ou kotlin) avec des instructions spécifiques à la bibliothèque Room. Cette dernière nous permet de créer une abstraction pour générer une base de données SQLite. On communique ensuite avec la base toujours par abstraction DAO.

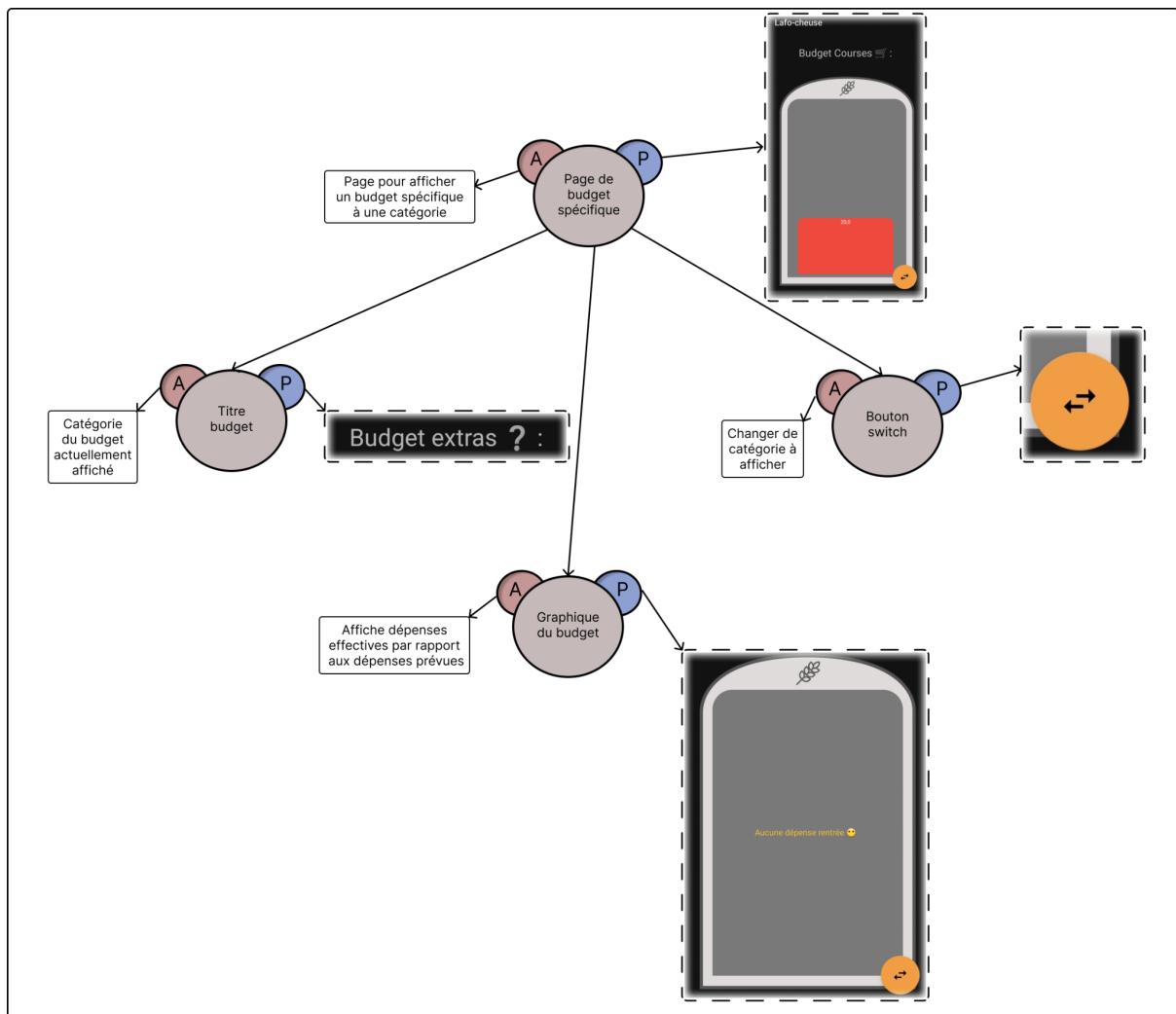
12. Modélisation PAC

Afin de formaliser notre conception et la structure de nos écrans, nous avons conçu plusieurs diagrammes PAC. Nous avons ainsi pu formaliser la syntaxe et la sémantique de notre interface pour les différents écrans prévus.

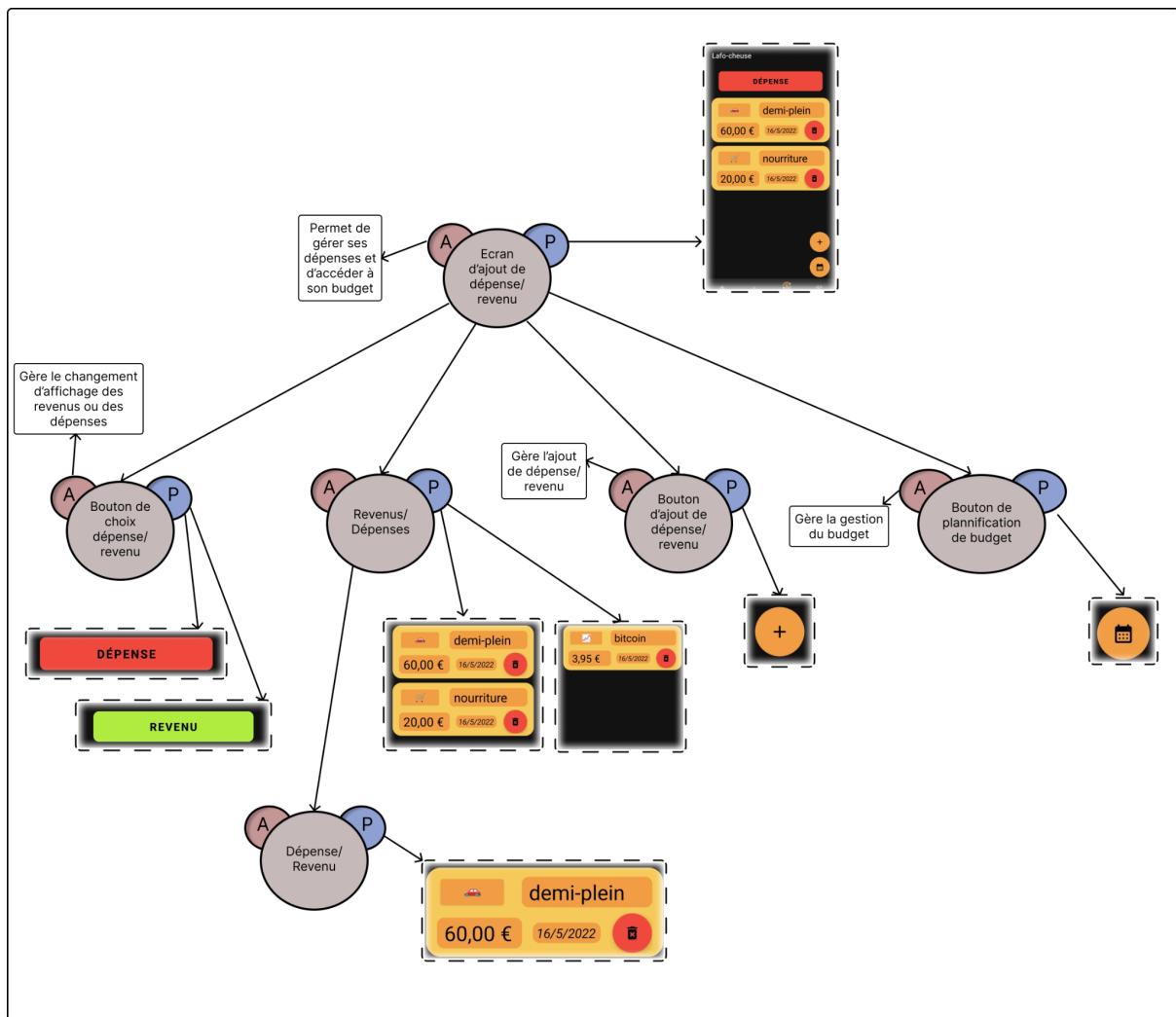
1. Écran d'accueil



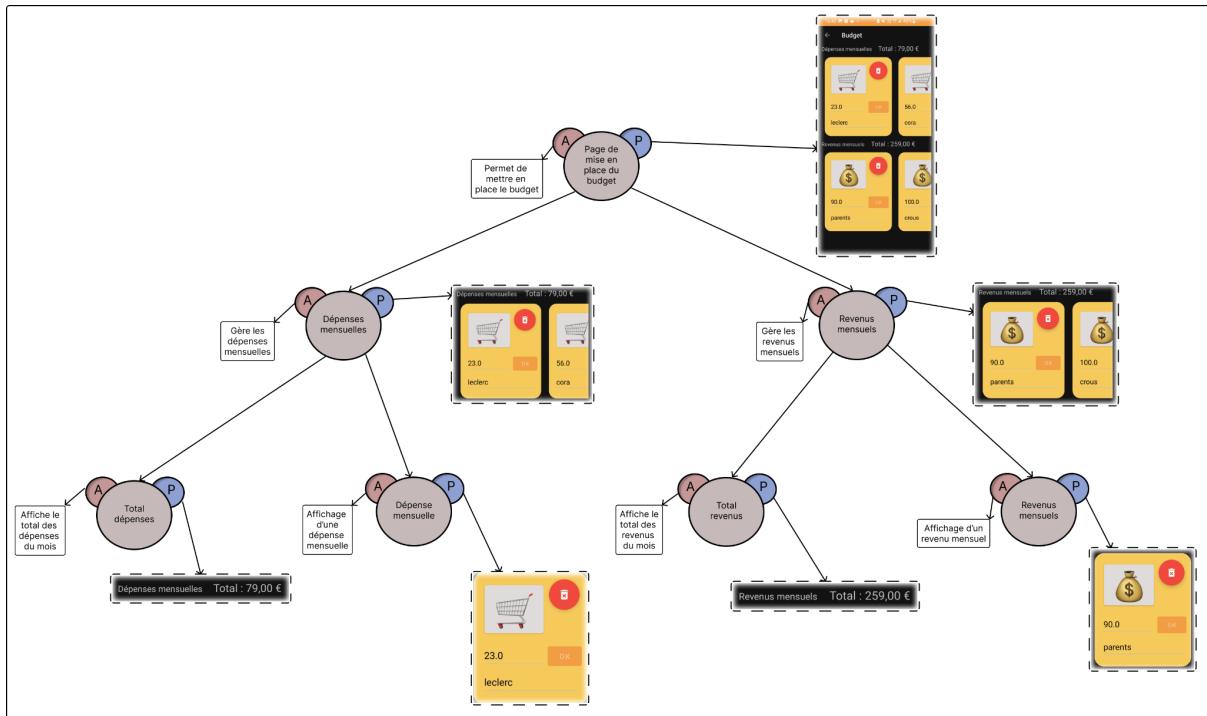
2. Écran de budget spécifique



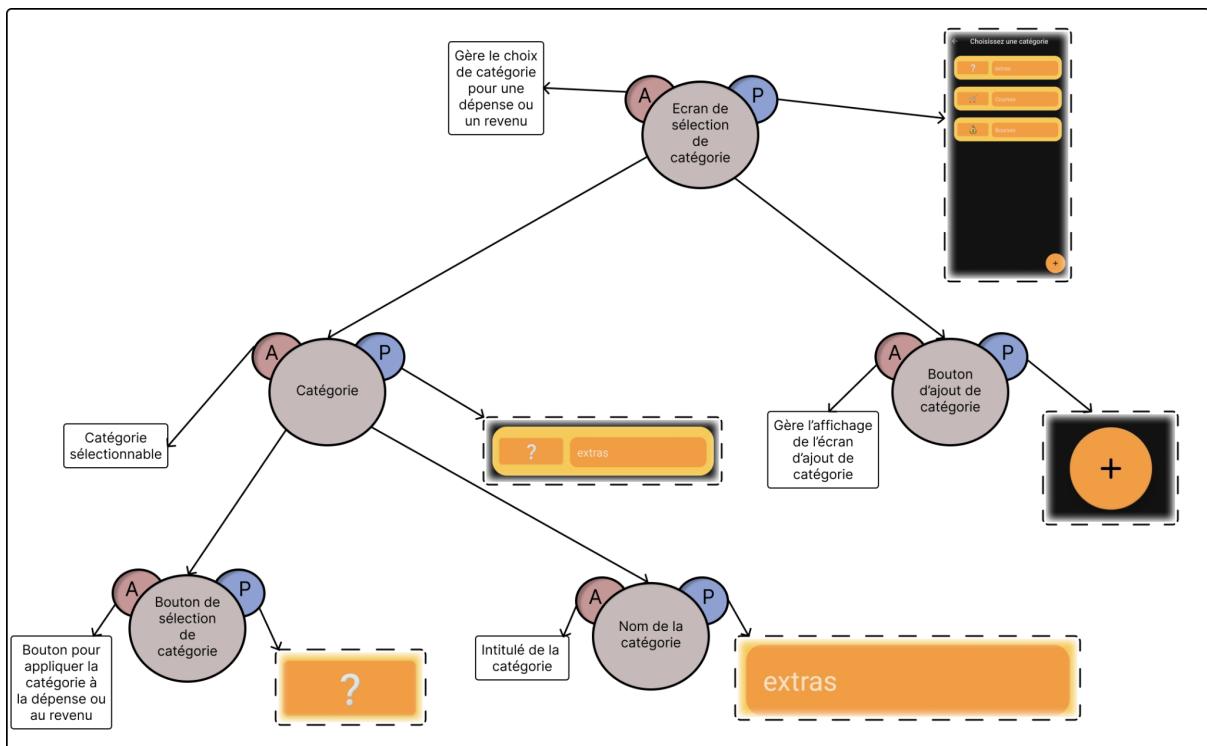
3. Écran d'ajout de dépense/revenu



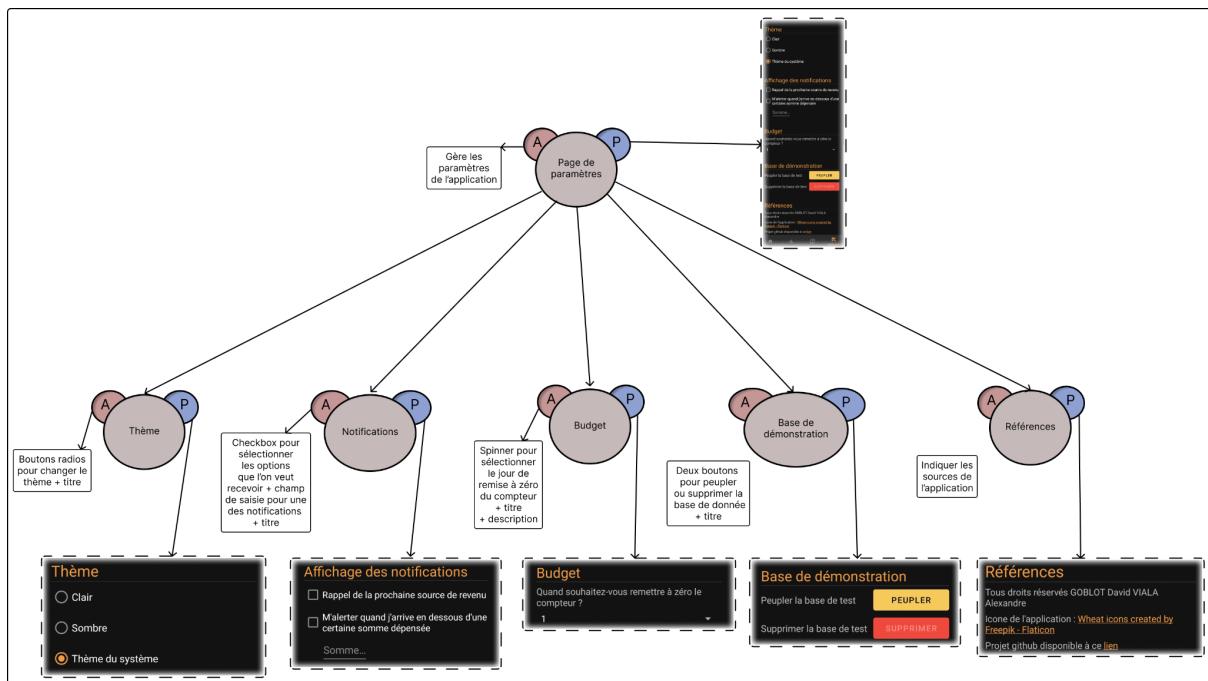
4. Écran d'ajout de dépense/revenu régulier



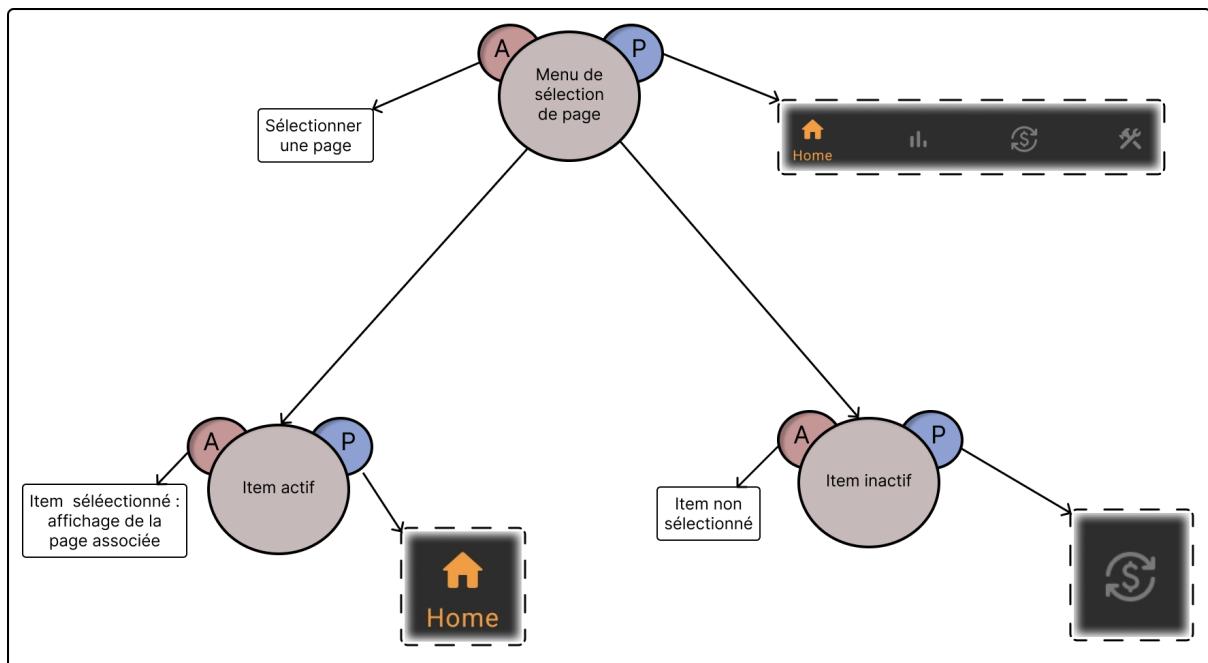
5. Écran de choix de catégorie



6. Écran de paramètres



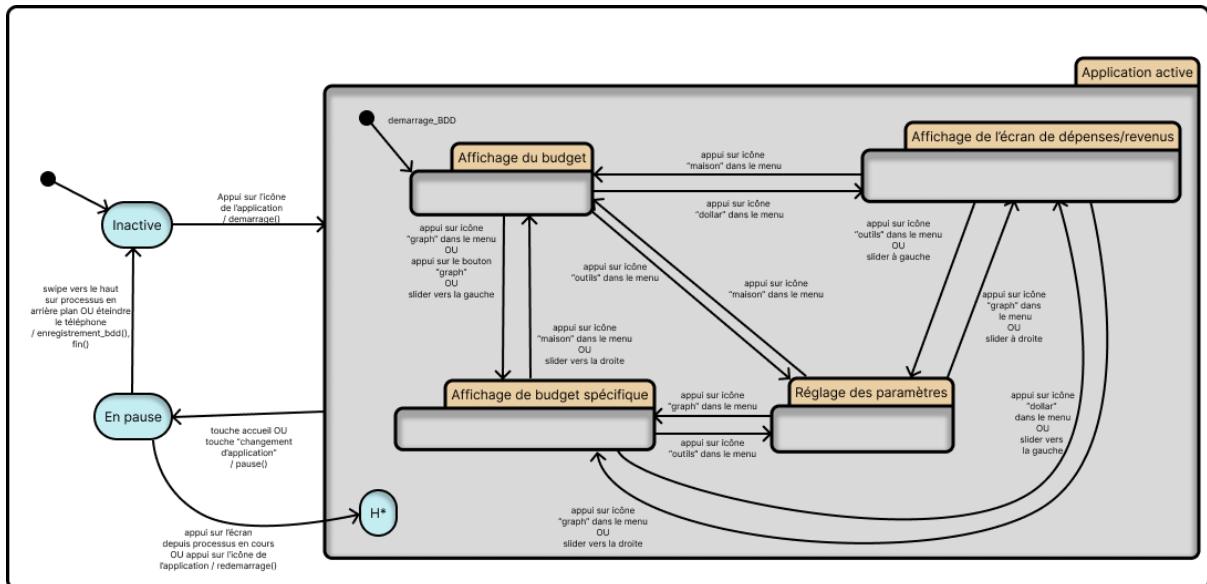
7. Menu de sélection d'écran



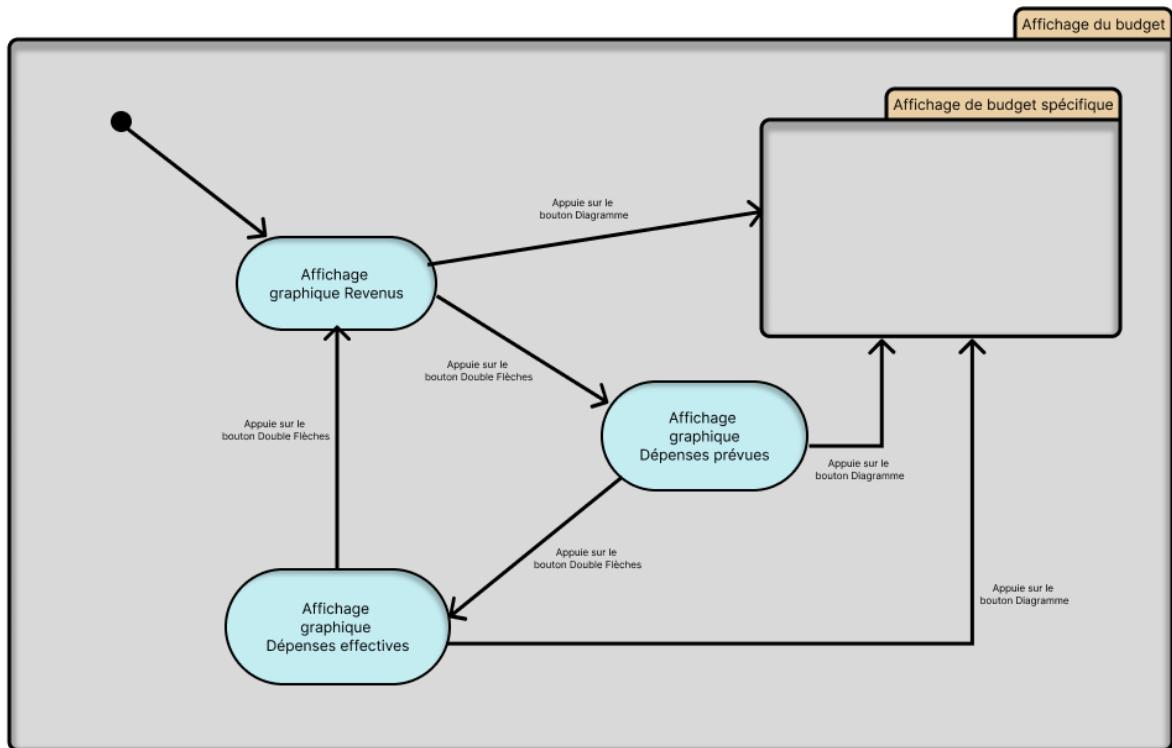
13. Diagrammes de contrôle

Afin de finaliser notre analyse fonctionnelle de notre application, nous avons réalisé des diagrammes de contrôles pour expliquer les fonctionnalités principales de notre application. Nous avons donc réalisé un premier diagramme expliquant le fonctionnement global de l'application puis 4 diagrammes expliquant les différentes fonctionnalités.

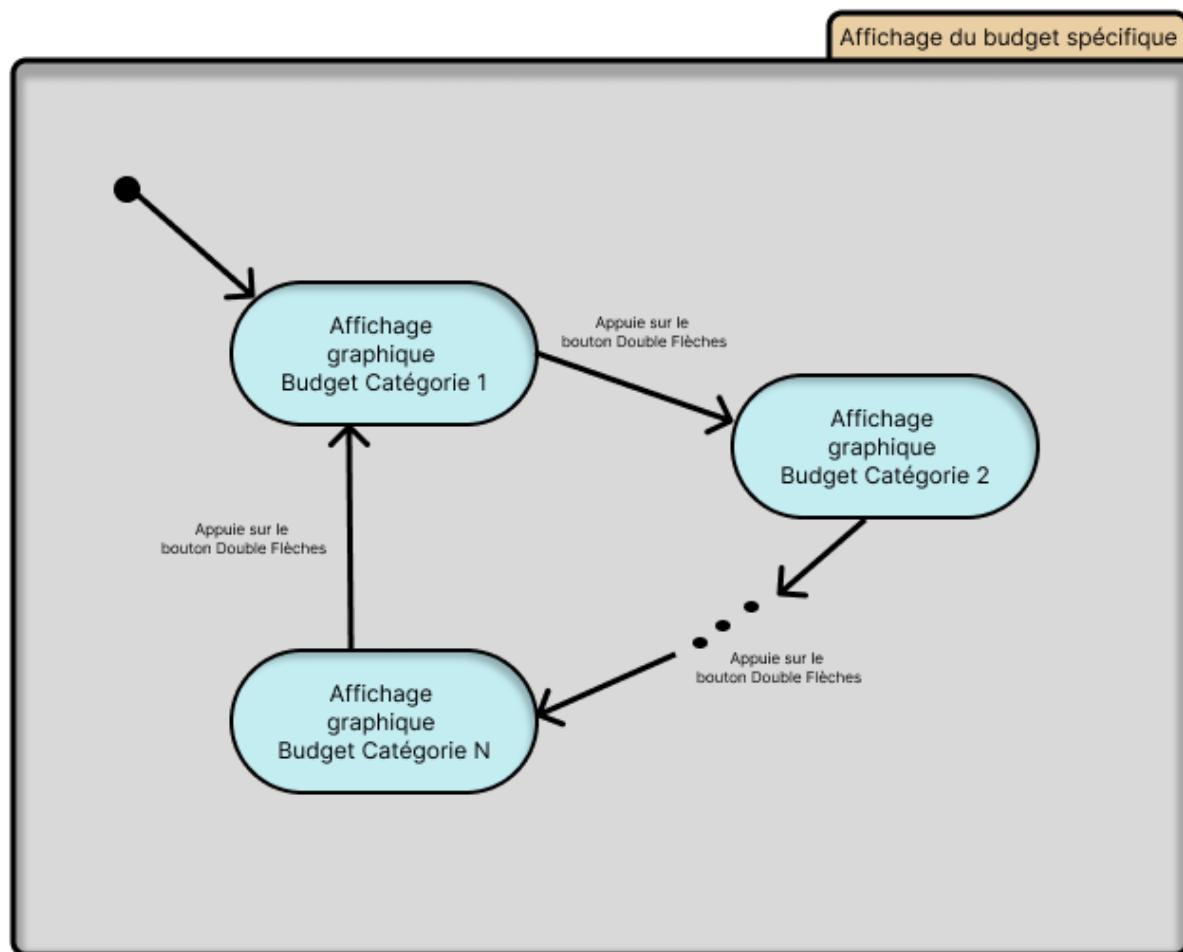
1. Diagramme de contrôle global



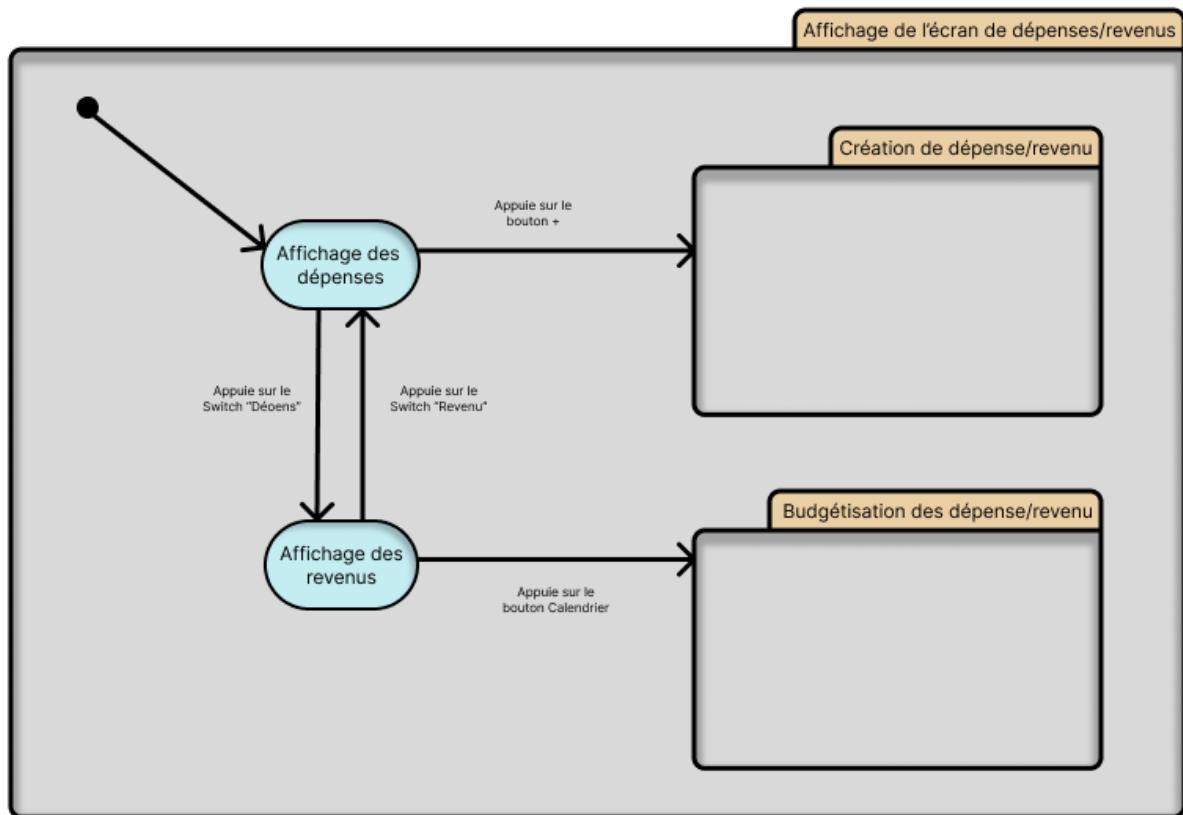
2. Diagramme d'affichage du budget



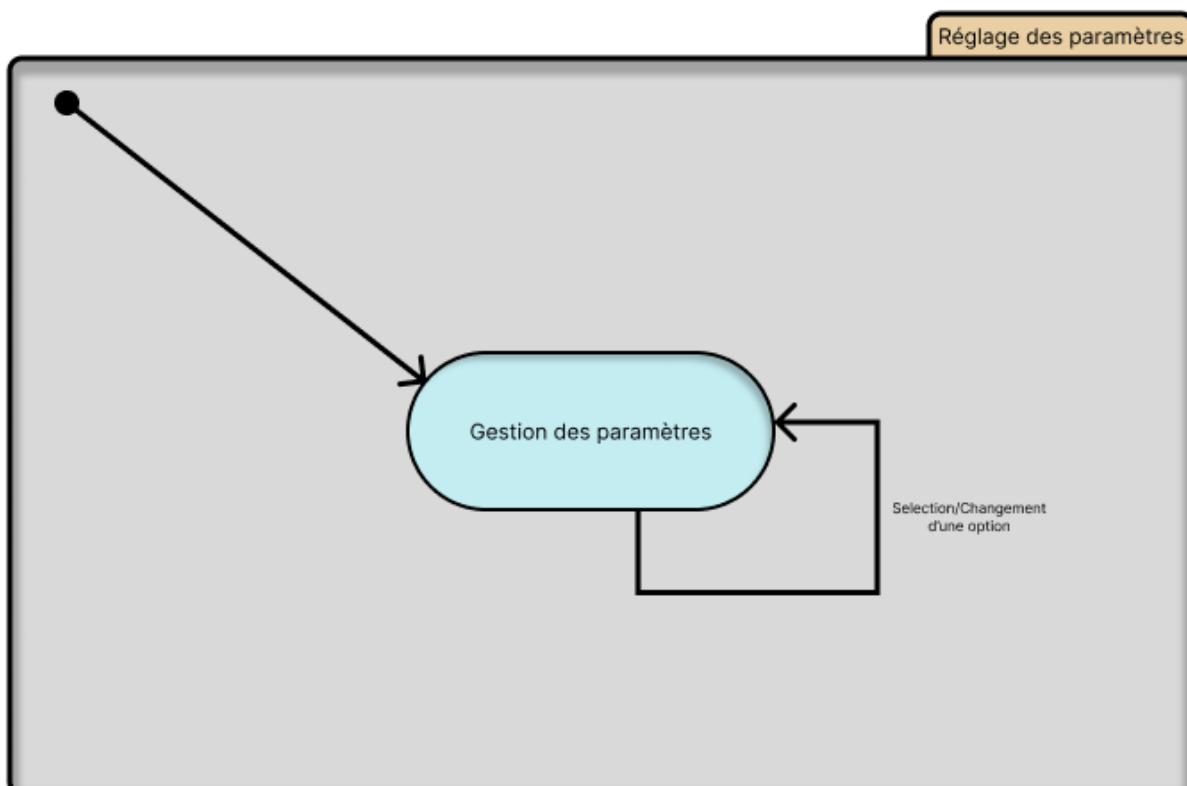
3. Diagramme d'affichage de budget spécifique



4. Diagramme de l'affichage de l'écran de dépenses/revenus



5. Diagramme de réglage des paramètres



14. Phase de réalisation

Une fois que le fonctionnement de l'application était planifié, nous avons commencé la phase de réalisation. Pour cela, nous avons organisé notre travail à l'aide du logiciel Notion. Nous avons ainsi planifié les tâches que nous devions réaliser en suivant la méthode AGILE.

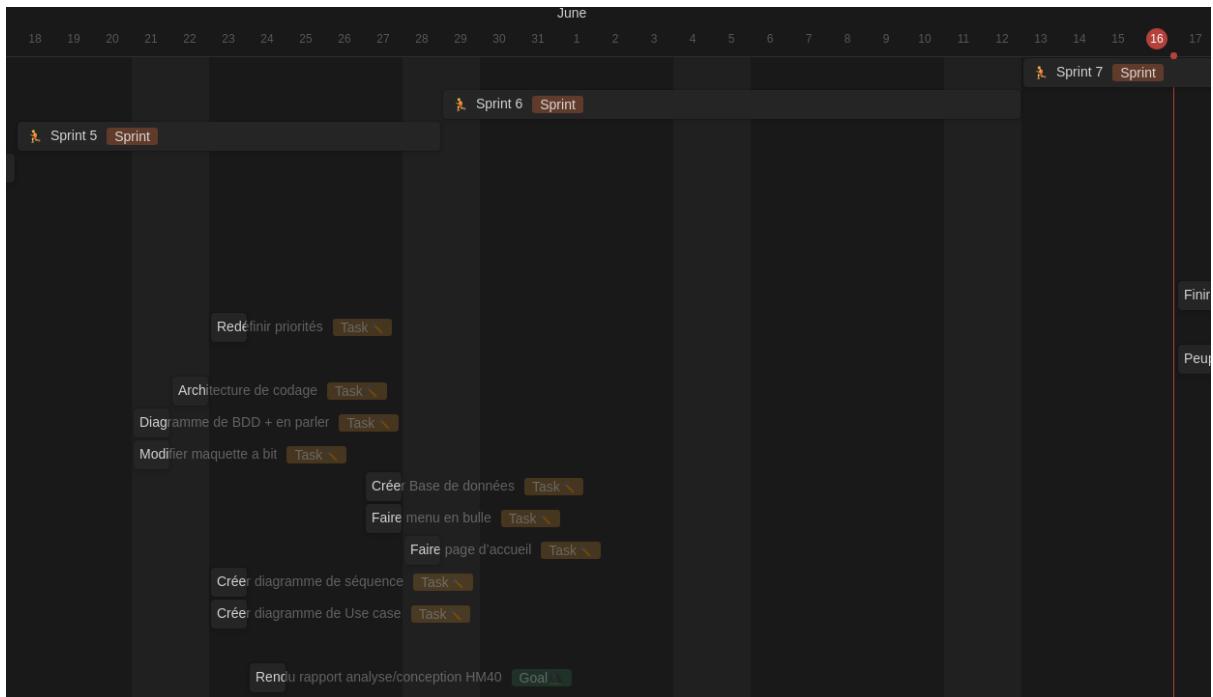


Figure : Extrait du projet Notion à l'aide duquel nous nous sommes organisés

Nous avons réalisé notre application en suivant les précepts de notre cahier des charges en suivant la méthode conception centrée utilisateur. Pour programmer, nous avons utilisé le logiciel Android Studio, un IDE créé conjointement par Google et JetBrain. Il nous a permis de travailler facilement avec Github et git en nous permettant de documenter chaque changement infinitésimal sur notre code.

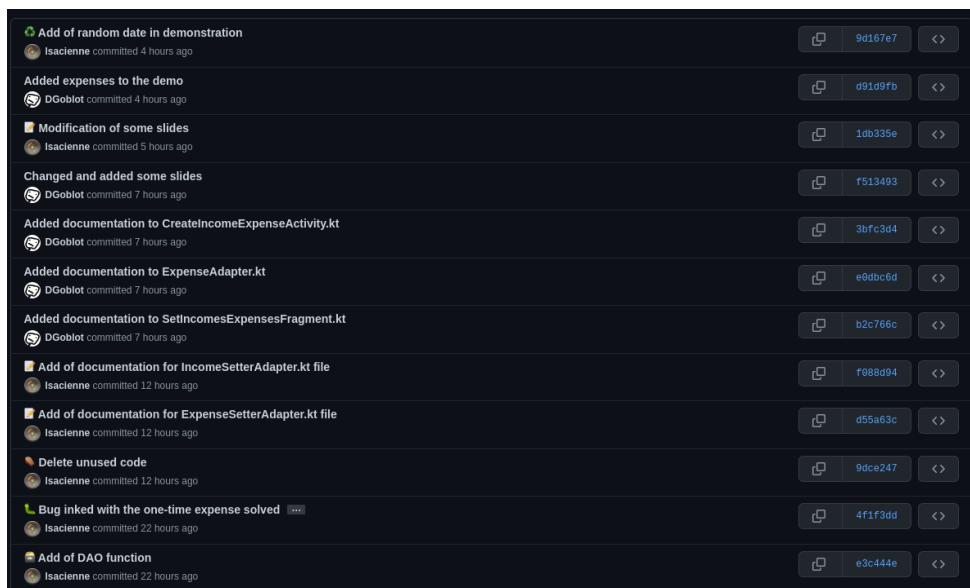


Figure : Extraits des derniers commits réalisés

L'intégralité de notre code est disponible à l'adresse suivante : [lafo-cheuse](#).

15. Résultat final et évaluation

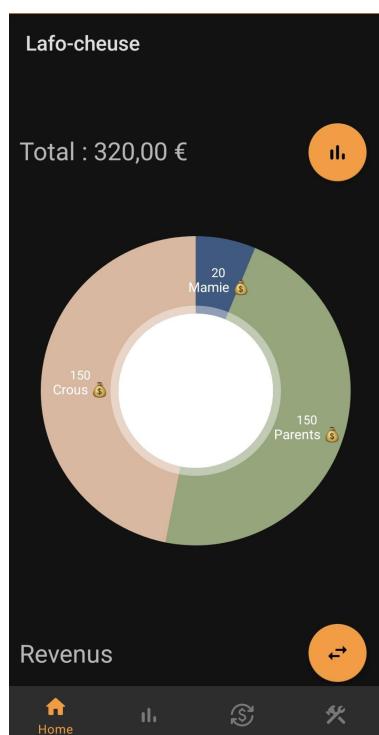
Finalement, nous avons pu réaliser notre application en répondant aux besoins de la plupart des critères de notre cahier des charges. Les quelques critères que nous n'avons pas pu satisfaire sont :

- La gestion de mot de passe pour accéder à l'application
- La gestion de différents types d'écrans et de différents supports (mode paysage sur portable, PC)
- La génération de notifications
- Le chiffrement de la base de données
- L'exportation de la base de données au format csv

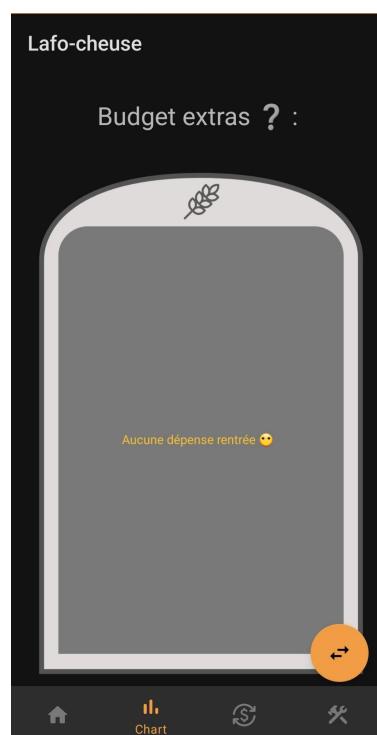
Ces critères avaient tous une priorité relativement faible par rapport à l'ensemble des fonctionnalités que nous voulions inclure.

Nous avons, en fin de compte, intégré les interfaces que nous avions maquettées au sein de notre application tout en ajoutant toute la logique de fond.

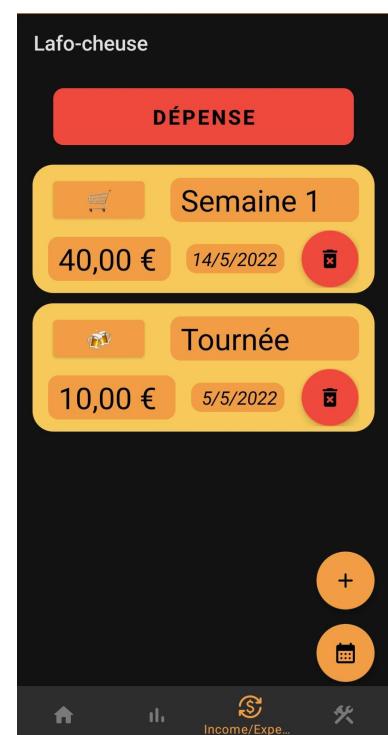
Écran du budget principal



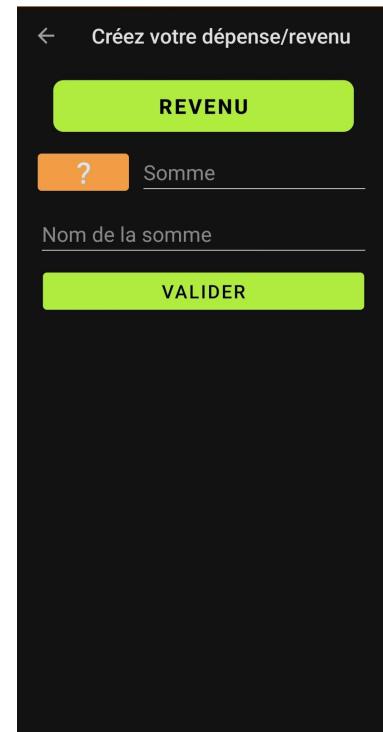
Écran de budget spécifique



Écran de dépenses/revenus



Écran de paramètres



Écran de budgétisation

Écran de choix de catégorie

Écran d'ajout de catégorie

