Gestionnaire de budget

Travail de Bachelor

Luca Rar

Mai 2022

Supervisé par :

Prof. Dr. Jacques Pasquier
et
Ryan Siow
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Groupe Génie Logiciel
Département d'Informatique
Université de Fribourg (Suisse)

Software Engineering Group

Remerciements

Je souhaite remercier le Prof. Dr. Jacques Pasquier ainsi que Ryan Siow pour leur engagement et leur suivi tout au long de ce travail de bachelor. Ils ont toujours été présents lors des différentes difficultés rencontrées et m'ont toujours orienté dans la bonne direction. Ils m'ont donné l'opportunité de développer une application concrète. Finalement, je souhaite également remercier ma famille et mes amis pour leur soutien et leurs conseils tout au long de ce travail.

Préface

Durant ce 21ème siècle, avec l'essor et l'évolution constante de la technologie, il est primordial d'être toujours connectés dans tous les domaines de la vie. Que ce soit au niveau privé, par exemple pour l'entretien des relations socio-familiales ou pour la gestion du budget, et également au niveau professionnel, actuellement il est nécessaire et important d'avoir à portée de main des outils informatiques qui nous accompagnent dans nos différentes tâches quotidiennes. En plus de l'informatique, l'économie est également un domaine qui dicte nos journées. Ce travail de bachelor présente le développement d'un outil informatique et, plus précisément, d'un gestionnaire de budget. Il s'agit d'une association intéressante et utile entre le monde de l'informatique et le monde de l'économie. Grâce à cet instrument, l'utilisateur a la possibilité d'être toujours connecté avec ses rentrées et ses sorties d'argent. Par ailleurs, grâce à la présence de différentes innovations technologiques, les fonctionnalités proposées dans ce gestionnaire de budget se veulent d'être faciles et ludiques pour l'utilisateur.

Mot-clé : Gestionnaire de budget, Rentrée d'argent, Sortie d'argent, Utilisateur, Diagramme, Application web, REST, API, MongoDB, React

Table des matières

Table des figures

Liste des tableaux

Listings

1
Introduction

1.1. Motivations et objectifs

La gestion des finances est un sujet très vaste. En effet, il existe de nombreuses méthodes afin d'y parvenir au mieux, comme l'utilisation de différents logiciels ou l'avis d'experts dans le domaine. En outre, plus une personne s'intéresse tôt à ce sujet et garde un œil sur ses comptes, plus elle pourra acquérir de bons réflexes et sera à l'aise dans la gestion de ses finances. Pour ce faire, au niveau privé, un gestionnaire de budget remplit parfaitement cette tâche. Il s'agit d'un outil qui doit être simple d'utilisation, ludique et qui assure à l'utilisateur un gain de temps. Par ailleurs, une application répondant à ce besoin de gestion des finances représente une synergie intéressante entre l'informatique et l'économie, deux sujets essentiels dans notre période actuelle. En particulier, ce travail se concentre sur le développement d'une application dite client-serveur de gestion de budget.

L'objectif du projet est d'apprendre et de développer un service client-serveur qui associe l'informatique et l'économie. Pour réaliser cette application, il faut :

1. Définir les différents besoins et fonctionnalités nécessaires au service.
2. Développer un serveur afin de répondre aux différentes requêtes et le lier à une base de données.
3. Développer un client pour permettre une interaction avec le service créé au préalable.

1.2. Structure du rapport

Chapitre 1 : Introduction

En premier lieu, l'introduction décrit les motivations et les objectifs du travail. Par la suite, elle présente la structure du rapport ainsi qu'un résumé de chaque chapitre. Pour terminer, elle définit différentes notations et conventions utilisées tout au long du rapport.

Chapitre 2 : Contexte

Ce chapitre expose tout d'abord l'utilité d'un gestionnaire de budget. Ensuite, différentes applications de gestion de budget sont présentées et décrites. Puis, les fonctionnalités principales de ses différentes applications sont résumées. Par la suite, sont énoncées les fonctionnalités les plus pertinentes qui ont été retenues pour le développement de la nouvelle application. Enfin, des uses cases sont définis afin de simplifier la compréhension du chapitre et des différentes fonctionnalités retenues.

Chapitre 3 : Présentation du point de vue utilisateur final

Ce chapitre commence par une description complète de l'application développée, sans son implémentation. Par la suite, deux scénarios sont présentés, le premier concernant l'ajout d'une nouvelle rentrée d'argent et le deuxième centré sur la modification d'une sortie d'argent. Pour terminer, l'utilisation et la lecture des différents graphiques développés sont présentées : tout d'abord les diagrammes circulaires, puis les diagrammes en bâtons, et enfin, les courbes.

Chapitre 4 : Programmation du serveur

La première partie de ce chapitre est une présentation générale du code du serveur et des différentes technologies utilisées pour sa réalisation. Puis, l'architecture REST, les différents endpoints de l'API Swagger UI ainsi que la base de données MongoDB sont décrits.

Chapitre 5 : Programmation du client

Ce chapitre expose la programmation du client ainsi que les différentes technologies utilisées, comme React, Axios et Chart.js.

Chapitre 6 : Conclusion

Le dernier chapitre débute par une conclusion du rapport, suivie d'une analyse des objectifs fixés. Pour terminer, différentes propositions d'améliorations de l'application sont présentées.

1.3. Conventions

— Le rapport est divisé en 6 chapitres. Chaque chapitre comporte plusieurs sections. Les chapitres 2 et 3 sont subdivisés encore en sous-sections.

— En ce qui concerne le genre, la forme masculine est systématiquement retenue dans le rapport pour des raisons de simplicité. Mais les deux genres sont égaux.

— Les figures, les tableaux, ainsi que les codes sources (listings) présents dans le rapport sont numérotés individuellement par ordre d'apparition.

— Les codes sources présents dans le rapport sont formatés ainsi :

```
1 def division(x, y):
2     result = x / y
3     return result
```

List. 1.1 – Exemple d'un code écrit en Python

2
Contexte

2.1. Principes du gestionnaire de budget

Un gestionnaire de budget est un outil très pratique qui permet à chacun d'avoir une vue d'ensemble sur ses finances. En effet, la gestion des finances est primordiale pour toutes personnes. Par ailleurs, dans un monde qui évolue toujours plus vers l'utilisation des technologies et qui devient toujours plus complexe, il est rassurant de disposer d'un outil informatique simple d'utilisation et qui fournit une aide concrète dans la gestion du budget. Dans ce but, il existe de nombreuses applications mobiles ou des logiciels, plus ou moins complexes, pour répondre aux besoins de chaque personne. Les fonctionnalités de base que l'utilisateur retrouve dans la grande majorité de ces logiciels sont la possibilité d'ajouter, de modifier, de supprimer et d'avoir un aperçu de toutes ses rentrées et sorties d'argent. Ces fonctionnalités lui permettent une utilisation efficace de son outil de gestion. Lorsque l'utilisateur a inséré toutes les données souhaitées, il est intéressant pour lui d'en avoir des visuels concrets. Ainsi, à cet effet, de nombreuses applications lui montrent ses données sous la forme de différents diagrammes, comme par exemple des diagrammes circulaires, des diagrammes en bâtons ou des courbes. La plupart des applications utilisent un ou plusieurs graphiques pour lui garantir une meilleure lisibilité de ses données. Par ailleurs, plusieurs applications permettent à l'utilisateur de filtrer ses dépenses et ses rentrées d'argent selon différents critères, comme par exemple la date de l'inscription, la catégorie inscrite ou le montant ajouté.

Dans ce chapitre, sont d'abord présentées différentes applications qui permettent à l'utilisateur de gérer son budget et également d'avoir un visuel sur les données insérées. Par

la suite, un résumé décrit les fonctionnalités les plus avantageuses pour le développement d'une future application.

2.2. Exemples d'applications déjà existantes

2.2.1. Mint

Mint, ou "Intuit Mint", est une application mobile et un site web de gestion des finances personnelles. Cette application, créée par Aaron Patzer, est l'une des applications les plus connues dans le domaine de la gestion des finances. Elle est uniquement disponible aux États-Unis et au Canada et elle comptait 20 millions d'utilisateurs en 2016 [1, 2, 3]. Mint permet à ses utilisateurs d'atteindre leurs objectifs financiers grâce à des outils et à des budgets personnalisés, à un contrôle de leurs abonnements et à un suivi de leurs dépenses gratuitement. Par ailleurs, l'utilisateur a la possibilité de connecter ses cartes de crédit, ses comptes en banque, ses investissements et ses prêts. Néanmoins, il ne peut pas faire des virements depuis l'application. Il est également possible que plusieurs personnes se connectent au même compte afin qu'elles aient toutes un aperçu des transactions sur un compte partagé. Cette fonctionnalité est destinée principalement aux couples, aux familles ou aux colocataires.

L'application Mint emploie le mot-clé transaction à la place de rentrées d'argent. Ainsi, ce mot-clé est utilisé dans la description de l'application Mint.

L'application pour smartphone comporte 4 pages. Sur la première page, "Overview", l'usager a un visuel, sous la forme d'une courbe, de l'évolution de ses dépenses durant l'année ou durant le mois en cours. En dessous, il peut examiner l'état de tous ses comptes qu'il a liés, comme ses cartes de crédit ou ses comptes bancaires. Par la suite, il a la possibilité d'ajouter une nouvelle transaction, de créer un budget et d'ajouter un nouveau compte. Par exemple, il peut ajouter un nouveau compte American Express et, ainsi, il a la possibilité de consulter chaque nouvelle dépense liée à ce nouveau compte. En dessous des comptes qu'il a associés au préalable, l'usager retrouve la liste complète de toutes ses dépenses. Il peut également filtrer ses transactions ou dépenses et peut effectuer une recherche par catégorie, par période ou par compte.

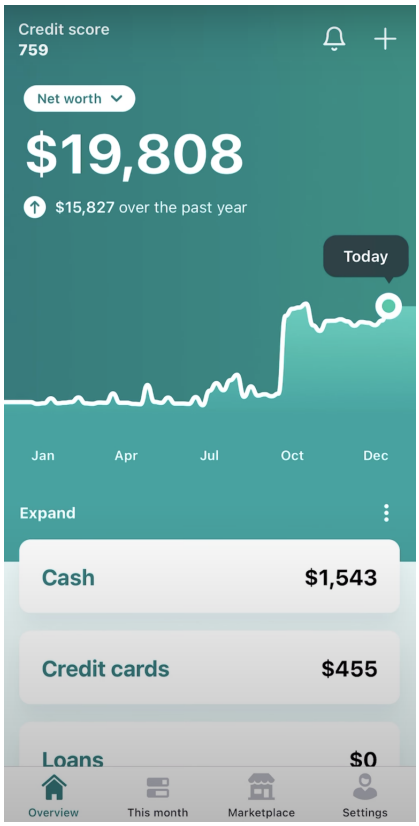

FIGURE 2.1. – Page d'accueil de l'application Mint

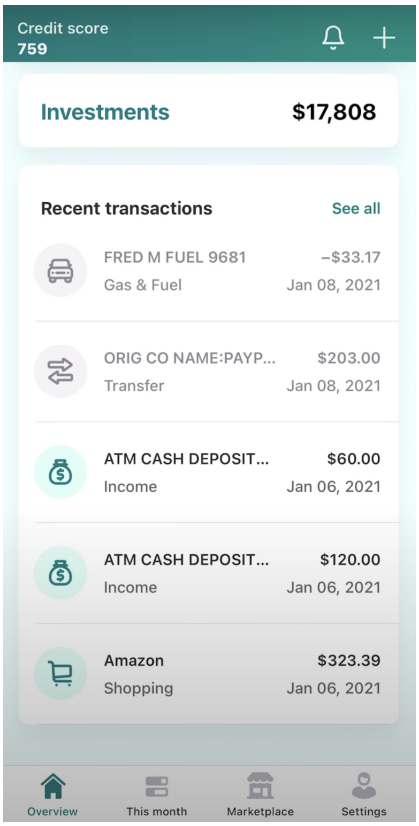

FIGURE 2.2. – Page d'accueil de l'application Mint avec des rentrées et des sorties d'argent

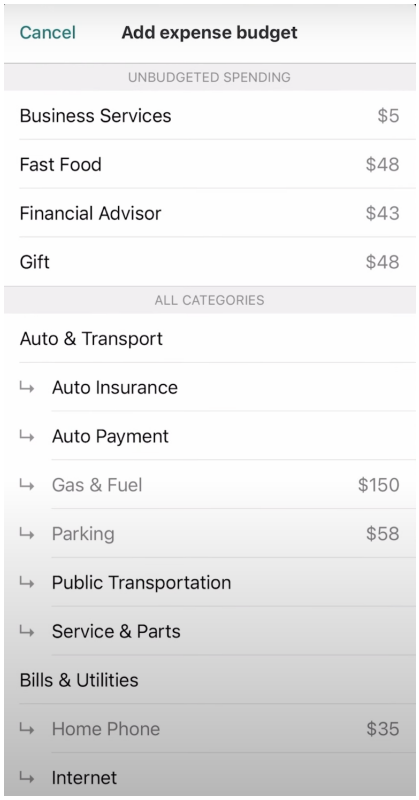

FIGURE 2.3. – Formulaire permettant d'ajouter une nouvelle dépense

Sur la deuxième page, "This month", l'utilisateur retrouve son cash-flow, qui est un résumé de ses transactions et sorties d'argent, sous la forme de diagrammes en bâtons. Il a également la possibilité d'effectuer un tri par période ou par catégorie. S'il se rend dans la section "Upcoming bills", il retrouve trois sous-sections : la sous-section "Subscriptions" pour ses abonnements, la sous-section "Bills" pour ses factures, et enfin, la sous-section "Payments" pour ses paiements. Ces sous-sections l'avertissent s'il doit bientôt régler une facture ou un paiement. Par la suite, il a la possibilité de créer des budgets pour garder un aperçu de ses dépenses et pour se fixer des objectifs d'épargne. Il peut choisir le titre du budget, le montant total, ainsi que la date de lancement.

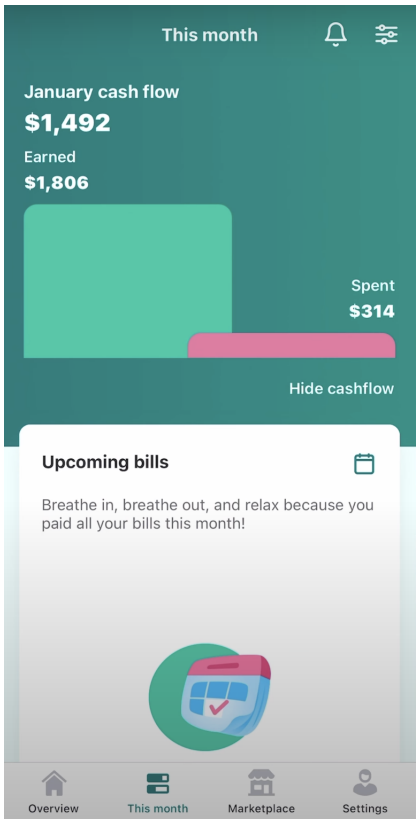

FIGURE 2.4. – Exemple d'un cash flow dans l'application Mint

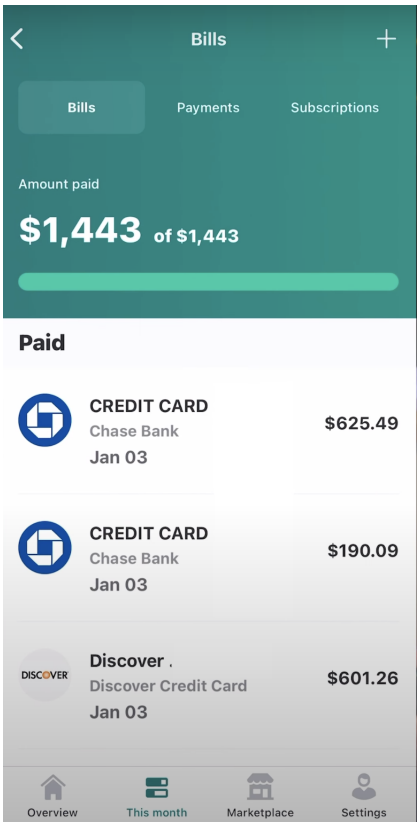

FIGURE 2.5. – Section "Upcoming bills" dans l'application Mint

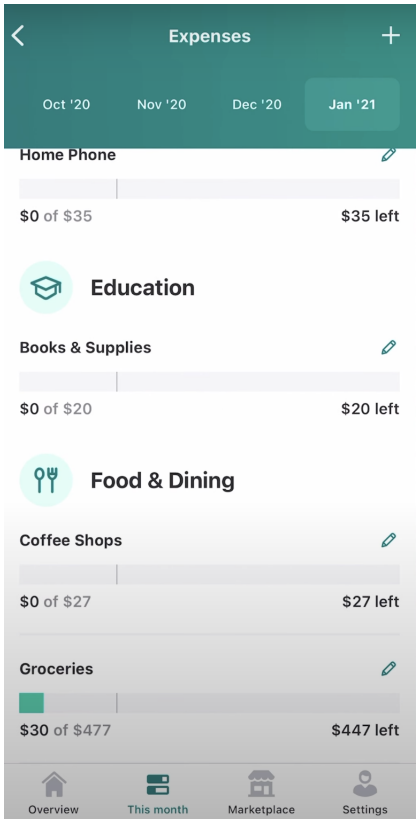

FIGURE 2.6. – Page de gestion des budgets dans l'application Mint

En dessous de la section concernant les budgets, l'utilisateur retrouve un diagramme circulaire avec l'ensemble de ses dépenses, regroupées par catégorie, pour le mois en cours. En cliquant sur le graphique, il peut consulter en détail les dépenses associées à chaque catégorie sous forme d'un diagramme circulaire et d'une liste. Pour terminer, en bas de page, il a la possibilité de définir un objectif d'épargne à atteindre afin, par exemple, de s'offrir une nouvelle voiture.

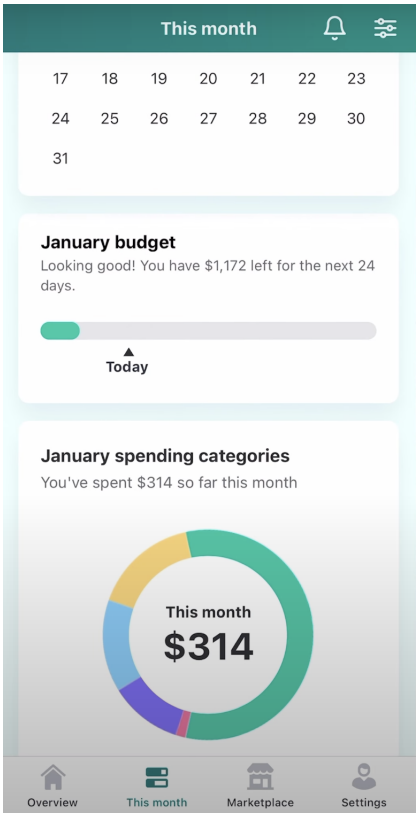

FIGURE 2.7. – Graphique schématisant les dépenses du mois de janvier

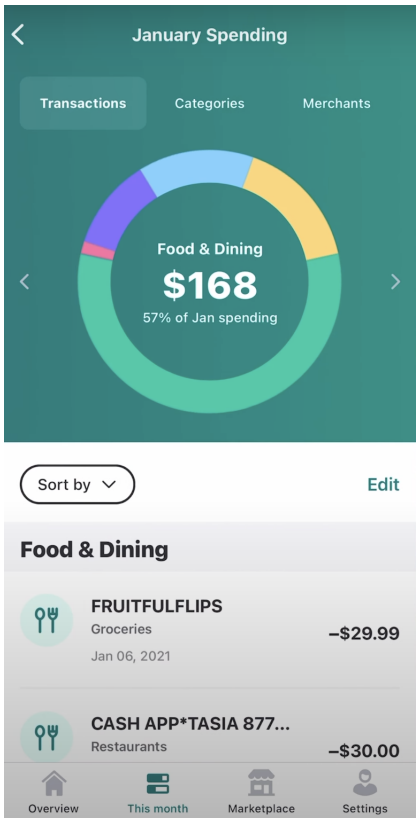

FIGURE 2.8. – Graphique schématisant les dépenses de la catégorie "Food & Dining"

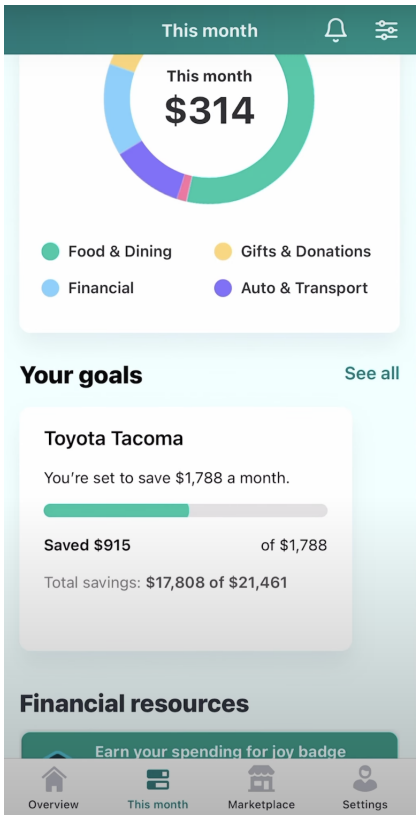

FIGURE 2.9. – Objectif d'épargne fixé

La troisième page regroupe différents articles que Mint publie au sujet de son application, afin d'informer ses utilisateurs sur les nombreuses fonctionnalités proposées par l'application et sur des études en matière de gestion des finances.

Le site web de Mint reprend les principales fonctionnalités de la version mobile avec quelques-unes en plus. Par exemple, l'utilisateur a la possibilité de regrouper ses biens. Ceci lui permet de connaître sa fortune ainsi que d'avoir une estimation faite par le site web de certains de ses biens. À titre d'exemple, chaque année le site web estime la valeur du véhicule de l'utilisateur afin de mettre à jour le total de sa fortune.

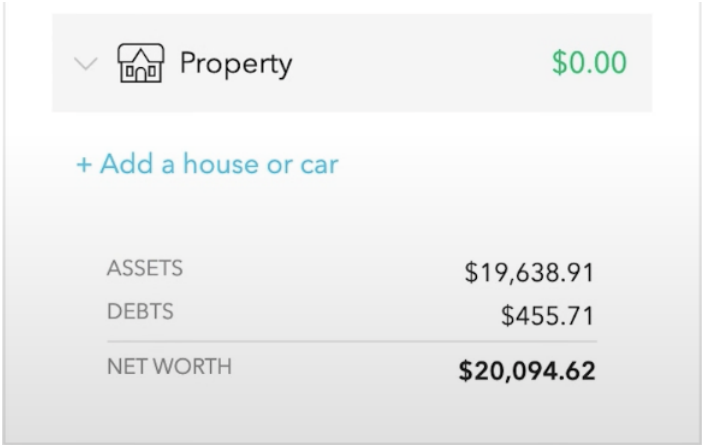

FIGURE 2.10. – Fortune de l'utilisateur

2.2.2. BudgetCH

L'application pour smartphone BudgetCH a été créée par l'Association faîtière Budgetconseil Suisse. Il s'agit d'une association suisse à but non lucratif. En tant que partenaire indépendant, cette association est spécialisée dans les budgets des particuliers depuis plus de 60 ans. Avec ses membres et partenaires, tels que la Coop et YOO AG, elle propose également des informations, des conseils et des opportunités de formation continue pour les particuliers, les professionnels et les entreprises. Cette application mobile est utilisée par près de 45'000 personnes. Elle fonctionne également sans connexion internet. Par ailleurs, il est possible pour un utilisateur de créer un compte gratuit sur l'application. Cela permet que les données enregistrées ne soient pas uniquement stockées en local sur son smartphone, mais qu'elles puissent aussi être restaurées à n'importe quel moment. Par la création d'un compte, plusieurs utilisateurs ont également la possibilité d'utiliser le même compte pour partager des données. Ce cas de figure peut être pratique par exemple pour une famille [5].

Lorsque l'usager lance l'application, apparaît d'abord une page de chargement avec le nom de l'association ainsi que son sponsor principal, à savoir la Coop.

Par la suite, l'utilisateur arrive sur la page principale "Vue d'ensemble". Sur cette première page, il trouve en premier lieu un aperçu, par défaut, de ses dépenses du mois en cours, sous la forme d'un diagramme en bâtons et d'une liste récapitulative. Il a la possibilité de filtrer l'affichage du diagramme ainsi que de la liste selon la période qu'il désire consulter. Par ailleurs, il peut également remplacer le diagramme en bâtons par un diagramme

circulaire. Dans la liste récapitulative, une sortie d'argent est décrite par le logo qui représente la catégorie, par la description, par la date et enfin par le montant.

FIGURE 2.11. – Page d'accueil de l'application BudgetCH

En cliquant sur une dépense, l'utilisateur a la possibilité d'en modifier les informations ou de la supprimer.

FIGURE 2.12. – Formulaire permettant de modifier une dépense dans l'application BudgetCH

Toujours sur la page principale "Vue d'ensemble", en bas de page, l'usager peut créer une nouvelle dépense. Il doit alors choisir une catégorie concernant la dépense parmi 20 catégories prédéfinies, une date ainsi qu'un montant. Seulement le champ "Description" ne doit pas être obligatoirement rempli. Par contre, si l'utilisateur ne remplit pas le champ "Montant", un message d'erreur apparaît. L'application prend en charge uniquement le franc suisse.

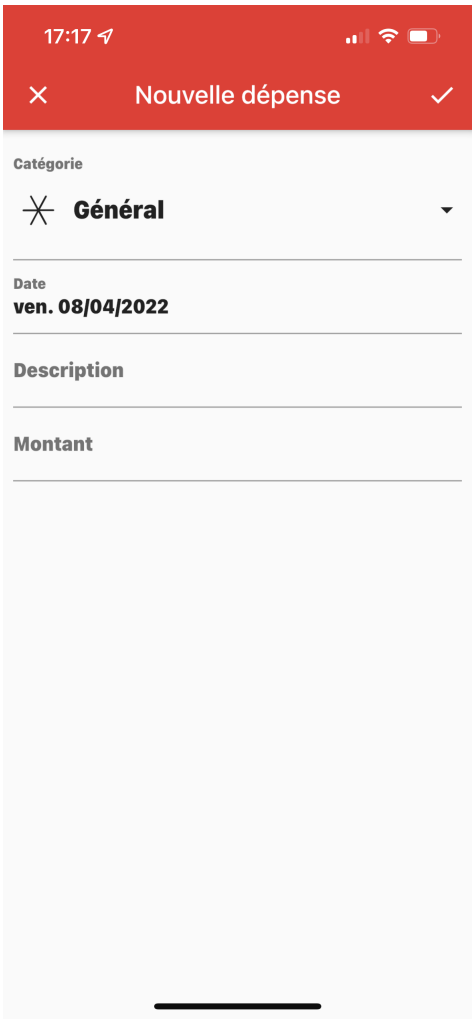

FIGURE 2.13. – Formulaire permettant de créer une dépense dans l'application BudgetCH

Une dernière fonctionnalité à disposition de l'utilisateur sur cette première page est la possibilité d'exporter ses données, par exemple dans un fichier Excel. Cette fonctionnalité est toutefois payante.

La deuxième page s'intitule "Dépenses". Cette page reprend la liste récapitulative des sorties d'argent de la première page. Cependant, l'utilisateur ne peut pas trier les dépenses déjà créées, elles sont toutes listées par ordre de date décroissante. Il a la possibilité d'ajouter une nouvelle dépense, de la modifier et / ou de la supprimer, comme sur la page "Vue d'ensemble". Sur cette deuxième page, se trouvent deux sections : la première intitulée "Paiements individuels" et la deuxième "Ordres permanents". Dans la seconde section, l'usager peut créer une dépense qui est ajoutée hebdomadairement, mensuellement ou annuellement.

FIGURE 2.14. – Page listant toutes les dépenses créées dans l'application BudgetCH

La troisième page se nomme "Budget". Sur cette page, l'usager indique tout d'abord sa situation, à savoir s'il vit seul ou en couple et s'il a des enfants. Par la suite, il spécifie sa condition au niveau du logement : s'il est propriétaire, locataire, ou s'il vit à la maison. Pour terminer, il indique quel est son moyen de transport : s'il utilise une voiture / moto, un vélo / cyclomoteur, ou les transports publics. Puis, il remplit un formulaire au sujet de ses revenus. Il n'est pas obligé de remplir tous les champs de ce formulaire. Une fois que toutes les informations concernant sa situation ainsi que ses revenus ont été rentrées dans l'application, BudgetCH calcule automatiquement des budgets pour chaque catégorie de dépenses. Ces budgets lui permettent de fixer des objectifs de dépenses pour chaque catégorie selon une référence calculée par l'application.

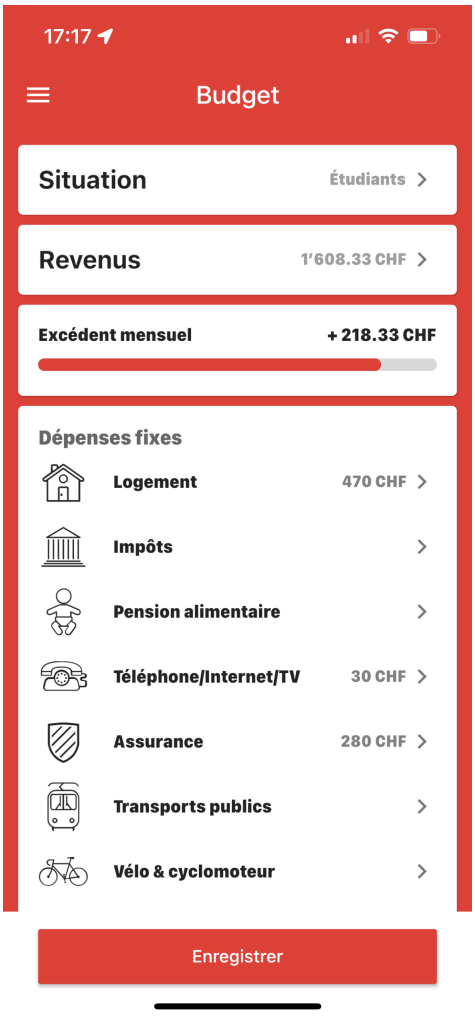

FIGURE 2.15. – Page de gestion des budgets dans l'application BudgetCH

2.2.3. Spendee

Spendee est une application pour smartphone créée par David Neveceral et Jakub Sechter, deux entrepreneurs tchèques. Elle a gagné de nombreux prix comme la "Startup of the Year" en République Tchèque en 2017 et la "CESA Central European Fintech of the Year" en 2019. Cette application permet d'avoir un aperçu de ses dépenses et revenus. Elle permet également de synchroniser directement l'application à un compte en banque. Les banques partenaires en Suisse sont PostFinance, Raiffeisen et St.Galler Kantonalbank. Grâce à cette application, l'usager peut gérer des budgets communs, par exemple avec la famille, des colocataires ou des amis. Par ailleurs, elle lui donne la possibilité de fixer des budgets maximums et des objectifs de dépense [6, 7, 8].

Lorsque l'utilisateur lance l'application, il arrive sur la première page "Historique". Sur cette page, il a la possibilité de consulter l'historique de ses dépenses. Tout en haut de la page figure le total de son budget, à savoir la différence entre ses revenus et ses dépenses. Par la suite, un diagramme en bâton regroupe les rentrées et sorties d'argent créées au préalable. Puis, l'usager retrouve une liste récapitulative de ses dépenses. Dans cette liste, il peut modifier ou supprimer une dépense. Les outils de recherche sur cette page sont la recherche par catégorie ou par description d'une rentrée ou d'une sortie d'argent.

L'utilisateur a également la possibilité d'effectuer une recherche par portefeuille ou par période. Pour terminer, il peut se rendre dans la section "Aperçu des dépenses" afin de consulter ses dépenses sous forme d'une courbe et d'un diagramme circulaire. Dans cette section, il peut également effectuer des recherches par portefeuilles ou par périodes.

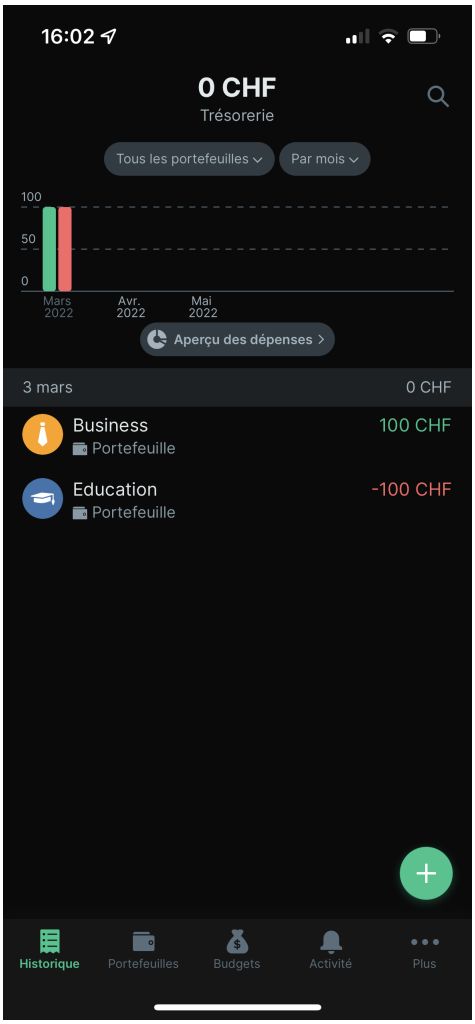

FIGURE 2.16. – Page d'accueil de l'application Spendee

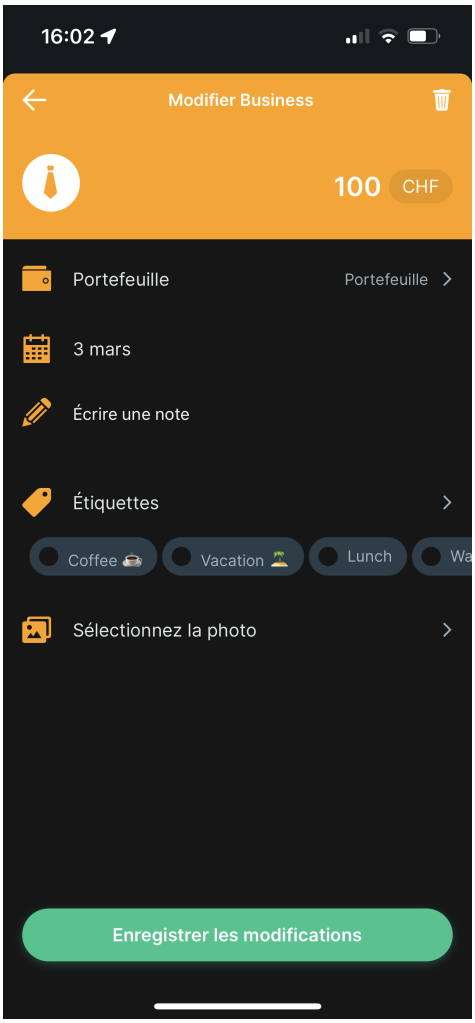

FIGURE 2.17. – Formulaire permettant de modifier une dépense dans l'application Spendee

En bas de la première page, l'utilisateur a la possibilité de créer une nouvelle dépense ou rentrée d'argent. Tout d'abord, il doit choisir une catégorie parmi une liste prédéfinie, ou il peut en créer une personnalisée. Puis, il écrit le montant dans la devise qu'il souhaite et, pour terminer, il choisit dans quel portefeuille cette nouvelle dépense ou rentrée est enregistrée. Il a l'obligation d'attribuer une catégorie, un montant et un portefeuille pour valider la création. Il peut ajouter des éléments facultatifs, comme une date, une note, une étiquette, une photo et / ou une récurrence.

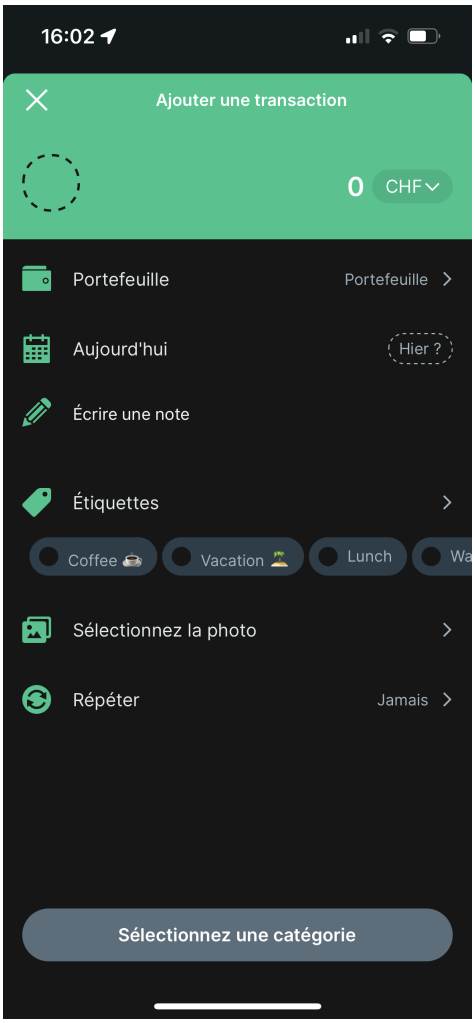

FIGURE 2.18. – Formulaire permettant de créer une dépense dans l'application Spendee

Sur la page suivante, "Portefeuilles", l'utilisateur peut créer différents portefeuilles pour rassembler ses revenus et ses dépenses. Par ailleurs, sur cette page, il peut également connecter son compte bancaire pour simplifier le suivi de ses mouvements d'argent. L'application lui permet de créer un seul portefeuille gratuitement.

Sur la troisième page de l'application, "Budgets", l'utilisateur a la possibilité de créer plusieurs budgets afin de consulter ses épargnes. Pour cela, il doit tout d'abord choisir le nom des budgets. Ensuite il fixe un montant maximum dans une devise. Puis, il peut ajouter un portefeuille dans ces nouveaux budgets. Par la suite, il détermine pour quelle dépense et avec quelle fréquence les budgets sont remis à 0. Pour terminer, il fixe la date du début des budgets. L'application lui permet de créer un seul budget gratuitement.

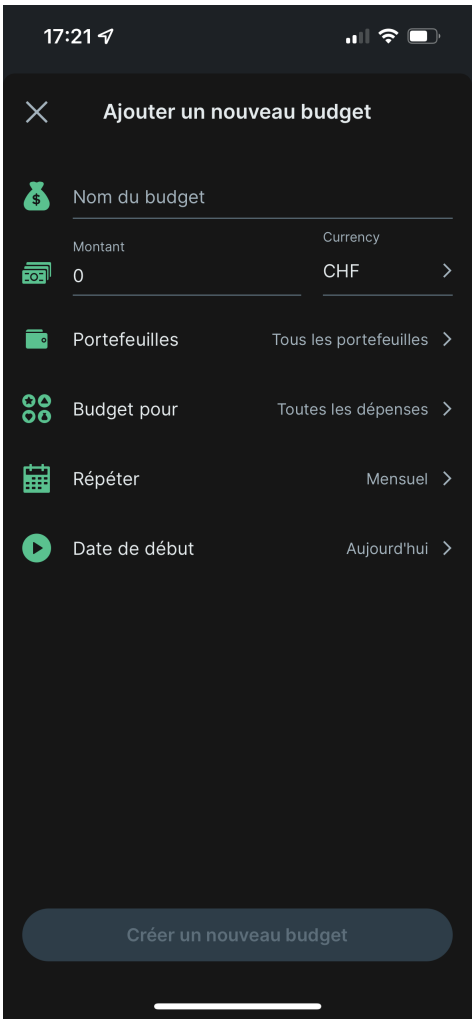

FIGURE 2.19. – Formulaire permettant de créer un budget dans l'application Spendee

La quatrième page de l'application Spendee, "Activité", est consacrée aux derniers articles informatifs publiés par Spendee. En effet, cette application publie régulièrement des articles afin d'informer ses utilisateurs sur les dernières fonctionnalités de l'application disponibles, ainsi que sur des études au sujet de la gestion des finances.

2.2.4. Buddy

Buddy est une application pour smartphone et pour ordinateur. Elle a été créée par l'entreprise Nattkod AB en 2017. Cette application met à disposition de l'utilisateur de nombreuses fonctionnalités afin qu'il garde toujours un œil sur ses finances. En effet, elle lui permet d'ajouter des dépenses, de créer des budgets et de les partager avec d'autres personnes. Buddy compte 1,5 millions d'utilisateurs dans le monde [9]. Pour cette application, le mot-clé transaction désigne les rentrées et les sorties d'argent de l'usager.

Sur la première page, "Transactions", l'utilisateur a un aperçu de ses transactions du mois en cours. En haut de la page, il a la possibilité de choisir un budget défini au préalable qu'il souhaite afficher. Toujours en haut de la page, il peut rechercher une transaction par sa note, son montant ou son nom de catégorie. Par la suite, il retrouve le montant total de ses revenus, de ses dépenses ainsi que son solde du mois en cours. Son solde est le

résultat de la différence entre ses revenus et ses dépenses. Puis, dans les section "Liste" et "Catégories", l'usager peut choisir s'il souhaite examiner toutes ses dépenses et revenus sous la forme d'une liste ou de diagrammes circulaires. Une fois la section choisie, il peut définir le mois dont il désire avoir un aperçu. S'il décide de consulter ses transactions dans la section "Liste", elles sont listées par ordre du dernier ajout sur l'application et sont regroupées par jour de l'ajout. Par ailleurs, l'application calcule le solde de chaque jour. En revanche, s'il préfère examiner ses transactions dans la sections "Catégories", il y a tout d'abord un graphique circulaire suivi d'un résumé de ses dépenses. Enfin, la dernière section, "Partage", permet à plusieurs personnes de gérer un même compte. En effet, cette fonctionnalité permet à chaque personne d'ajouter ses transactions sur un même compte, puis l'application équilibre les dépenses. Cette fonctionnalité de gestion de compte à plusieurs est proposée par exemple pour les couples ou les colocataires et elle est payante.

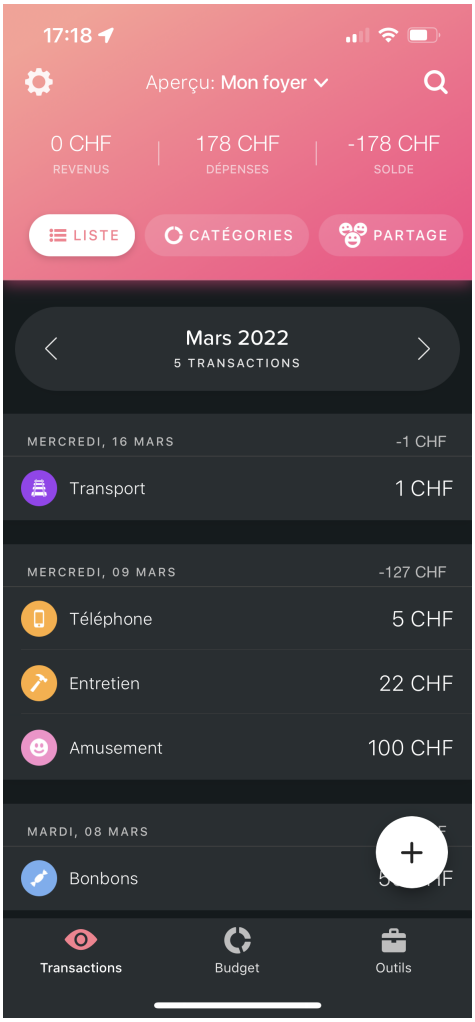

FIGURE 2.20. – Page d'accueil de l'application Buddy

FIGURE 2.21. – Aperçu des dépenses sous forme d'un diagramme dans l'application Buddy

Toujours sur la page "Transactions", l'utilisateur a la possibilité d'ajouter une nouvelle transaction. Tout d'abord, il doit définir le montant de sa transaction. Concernant la devise par défaut, il peut la définir dans les réglages de l'application. Par la suite, il doit choisir s'il s'agit d'une dépense, d'un revenu ou d'un virement. Puis, il peut choisir la catégorie associée à sa nouvelle transaction. Concernant les différentes catégories, il peut soit choisir une catégorie prédéfinie par l'application, soit en créer une personnalisée. Par défaut, la catégorie est "Divers". Une fois la catégorie choisie, il définit depuis quel compte la nouvelle transaction est effectuée et sur quel compte elle est versée, par exemple en cas de virement. Enfin, il a la possibilité d'ajouter une remarque, de choisir une date, qui par défaut est la date actuelle, ainsi que d'inscrire la fréquence de cette nouvelle transaction. Pour modifier ou supprimer une transaction, il lui suffit de sélectionner la transaction souhaitée et un formulaire s'affiche.

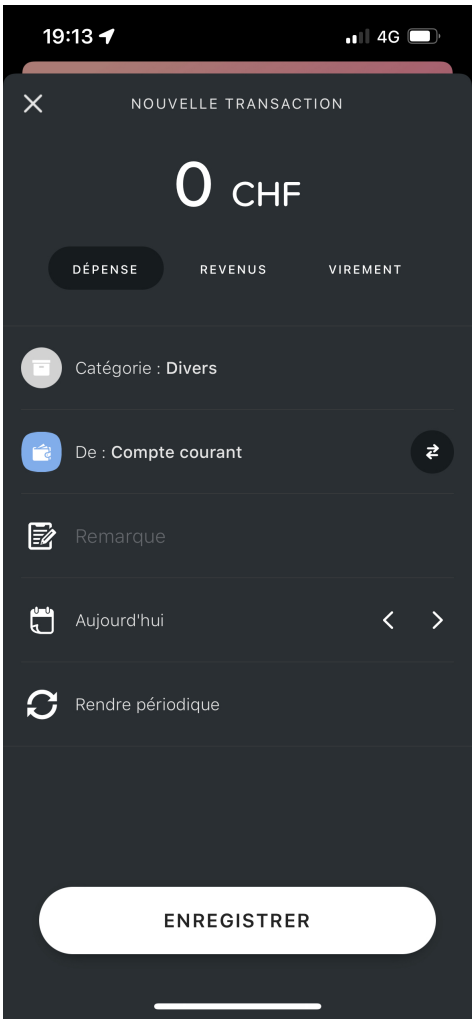

FIGURE 2.22. – Formulaire permettant de créer une dépense dans l'application Buddy

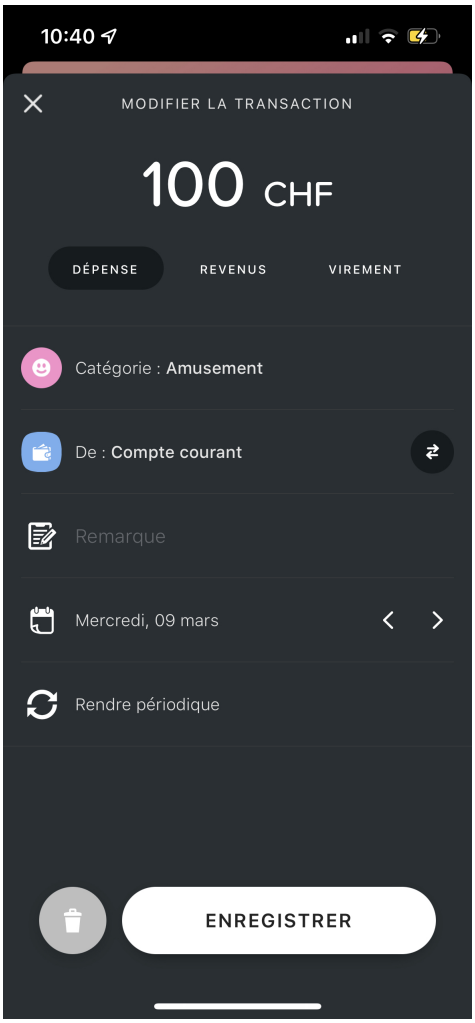

FIGURE 2.23. – Formulaire permettant de modifier une dépense dans l'application Buddy

Dans la page "Budget", l'usager peut modifier le budget sélectionné. Il a la possibilité de modifier le nom du budget, son icône, sa durée, ainsi que son début, et d'inviter des membres de l'application à rejoindre son budget. Sur cette page, il existe 3 sections : "Programmer", "Il reste" et "Informations". Dans la première section, l'utilisateur retrouve ses transactions fixes, comme par exemple le salaire pour ses revenus et ses différents abonnements pour ses dépenses. En rentrant ses transactions fixes, Buddy calcule combien l'utilisateur peut encore dépenser pour ce budget. Par ailleurs, l'application calcule également le pourcentage que représente chaque catégorie de dépense par rapport au revenu.

Dans la section "Il reste", un graphique illustre à l'utilisateur combien il a effectué de dépenses non fixes au cours du mois sélectionné. Pour terminer, il peut modifier ou supprimer une transaction.

Dans la dernière section, "Informations", l'utilisateur a la possibilité de payer pour avoir accès à des informations supplémentaires proposées par l'application Buddy, comme par exemple aux widgets.

Dans la dernière page de l'application, "Outils", il peut gérer son compte utilisateur, paramétrer l'application afin de la personnaliser et exporter des données enregistrées. Pour terminer, il a également la possibilité d'ajouter des membres à ses différents budgets.

2.2.5. Résumé des fonctionnalités

En résumé, les différentes applications décrites dans les sous-chapitres précédents partagent de nombreuses fonctionnalités. La fonctionnalité la plus importante qu'elles ont en commun est la gestion des revenus et des dépenses. En effet, elles permettent toutes à l'utilisateur de créer, de modifier, de supprimer et d'avoir un aperçu de ses rentrées et sorties d'argent. Pour l'ajout d'une nouvelle dépense ou d'une nouvelle rentrée, chaque application a ses spécificités. Certaines donnent à l'usager plus ou moins de liberté dans la création, comme la possibilité de relier une rentrée ou une dépense à une nouvelle catégorie fraîchement créée, ou la possibilité de choisir la devise dans laquelle elle est ajoutée. Un autre point important et ludique à la fois, est que toutes ces applications permettent à l'utilisateur d'avoir un aperçu de ses finances sous forme de graphique, que ce soient des diagrammes en bâtons, des courbes et / ou des diagrammes circulaires. Cela lui permet d'avoir un aperçu simple et rapide de ses finances, ce qui constitue un des objectifs principaux de ces applications de gestion de budget. Un autre outil pratique de ces applications est la possibilité de rechercher des rentrées ou des sorties d'argent par période, par catégorie ou par description. Par ailleurs, certaines applications permettent à l'utilisateur de connecter son compte en banque ou sa carte de crédit afin que les sorties d'argents soient automatiquement créées. Cependant, s'il relie un de ses comptes à l'une des applications, il ne peut pas faire directement des virements via son gestionnaire de budget. De plus, les applications l'aident non seulement à avoir un visuel de ses finances, mais l'assistent également afin d'atteindre des objectifs d'épargne, grâce à la possibilité de créer des budgets. En effet, en créant un budget, l'utilisateur a la possibilité de connaître exactement combien il dépense pour une certaine catégorie ou durant une certaine période. Grâce à la création de budget, il peut également savoir combien il doit épargner pour atteindre un objectif fixé, comme par exemple pour s'acheter une nouvelle voiture. Une autre fonctionnalité partagée par les applications BudgetCH et Buddy, est la possibilité d'exporter les données de l'usager vers d'autres plateformes, comme Microsoft Excel par exemple. Pour terminer, une fonctionnalité proposée par toutes les applications, est la possibilité de partager un compte avec d'autres utilisateurs. Cela permet, par exemple à un couple ou à un groupe de travail, d'avoir un aperçu des transactions en commun.

2.2.6. Fonctionnalités retenues

Comme énumérées précédemment, les applications de gestion de budget possèdent toutes de nombreuses fonctionnalités, similaires ou différentes. De ce fait, il est important de déterminer lesquelles sont les plus importantes pour le lancement d'un nouveau gestionnaire de budget. Tout d'abord, il convient de donner à l'utilisateur la possibilité de gérer manuellement la création, la modification ainsi que la suppression de ses revenus et de ses dépenses. En effet, ces fonctionnalités semblent être les plus importantes et intéressantes pour qu'il puisse gérer au mieux ses dépenses ainsi que ses rentrées d'argent. Par ailleurs, lors de la création d'une rentrée ou d'une sortie d'argent, il est intéressant de lui laisser le choix entre plusieurs catégories. Cela lui permet par la suite de savoir comment sont affectées ces différentes catégories. Une fois les dépenses et les revenus enregistrés dans l'application, il est important pour l'usager d'en avoir un aperçu grâce à des graphiques et / ou des listes. En effet, c'est par ces différents outils qu'il a le meilleur suivi de ses dépenses et de ses rentrées. Comme pour certaines applications décrites précédemment, l'idéal est que l'usager puisse choisir le diagramme qu'il souhaite consulter. Ainsi, en pro-

posant plusieurs graphiques différents, chacun est libre de visionner celui qui lui convient le mieux et celui qui lui fournit le maximum d'informations qui l'intéressent. Étant donné que chaque usager est différent, un type de graphique n'est pas apprécié de la même façon par tout le monde. En l'espèce, pour ce travail de bachelor, les graphiques retenus sont les diagrammes circulaires, les diagrammes en bâtons ainsi que les courbes. Ces graphiques permettent à l'utilisateur d'avoir un visuel de ses finances, mais également de calculer facilement, par exemple, la somme totale des dépenses pour une période définie par lui-même. Enfin, il est primordial qu'il puisse rechercher facilement une rentrée ou une sortie d'argent créée au préalable. Cette fonctionnalité de recherche lui permet de retrouver une dépense ou un revenu dont il souhaite par exemple consulter un détail précis ou pour lequel il souhaite apporter des modifications. Le gestionnaire de budget qui a été créé propose la recherche par catégorie des rentrées et des sorties d'argent, par période et / ou par description. Cette possibilité de recherche concerne également les différents diagrammes proposés, de manière à avoir toujours un aperçu cohérent. En effet, l'utilisateur peut effectuer une recherche pour visionner, par exemple, ses dépenses d'un mois concernant tous ses abonnements et il peut choisir de visualiser cette rechercher sous la forme d'un diagramme circulaire.

2.3. Use cases

Un cas d'utilisation, ou use case, permet d'expliquer facilement le déroulement des tâches effectuées par l'utilisateur sur l'application web. Par ailleurs, un cas d'utilisation est un excellent moyen pour communiquer des idées complexes de manière facile. Du point de vue de l'utilisateur, il montre le comportement du système lorsque ce dernier reçoit et répond à une demande. Un cas d'utilisation commence par exposer le but de l'utilisateur et se termine lorsque celui-ci est atteint. Un cas d'utilisation peut être décrit soit de manière écrite, soit par un outil de modélisation. Dans ce sous-chapitre, les deux possibilités sont présentées.

La Figure 2.24 illustre le diagramme Unified Modeling Language (UML) de gestion des rentrées et des sorties d'argent. L'utilisateur peut créer, supprimer, modifier ou rechercher une rentrée ou une sortie d'argent. Deux cas d'utilisation sont présentés en dessous de la Figure 2.24 par des tableaux. Le premier cas d'utilisation est la création d'une rentrée d'argent sur l'application web, et le deuxième cas d'utilisation est la modification d'une sortie d'argent. Étant donné que la création et la modification d'une rentrée et d'une sortie d'argent sont construites de la même façon, seule la création d'une rentrée d'argent et la modification d'une sortie d'argent sont développées.

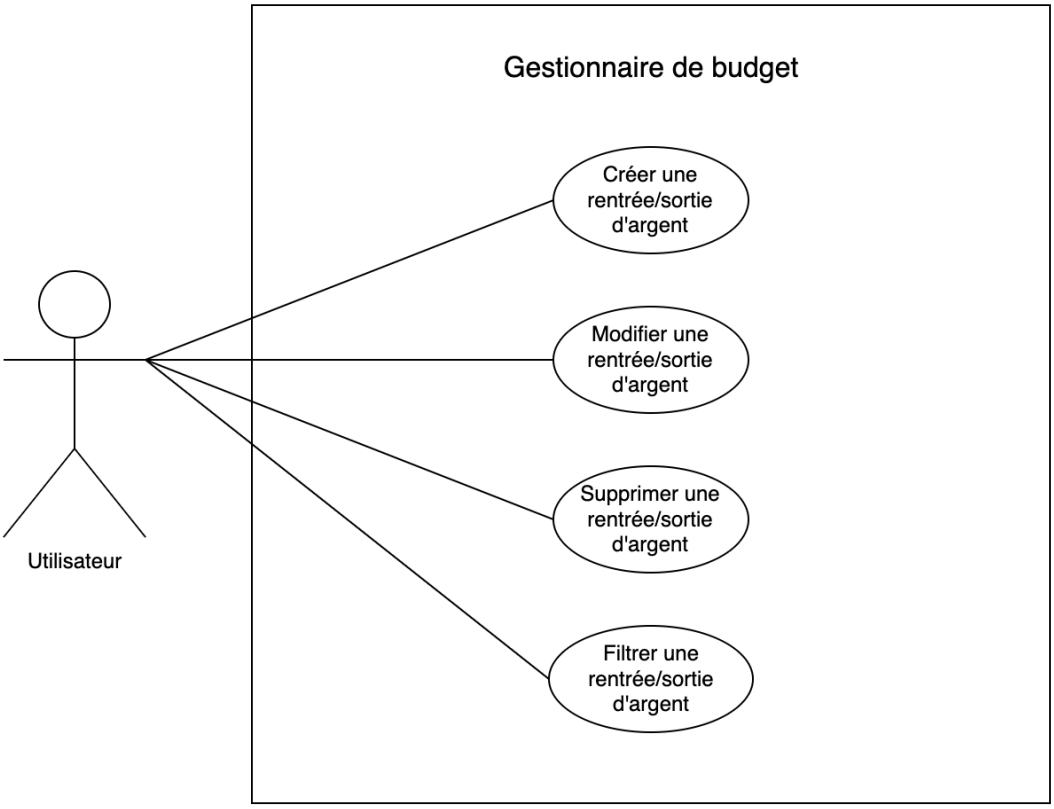

FIGURE 2.24. – Diagramme UML de la gestion des rentrées et des sorties d'argent

Dans la Figure 2.24, le rectangle montre l'application de gestion de budget et permet de définir la portée de celle-ci. Tout ce qui se trouve à l'intérieur du rectangle se produit à l'intérieur de l'application. Le personnage ou l'acteur à côté du rectangle est quelqu'un ou quelque chose qui utilise l'application pour atteindre un objectif. Un acteur peut être une personne, une organisation ou une autre application. Dans notre cas, il s'agit de l'utilisateur qui utilise l'application. Les éléments qui se trouvent dans le rectangle indiquent les fonctionnalités de l'application. Chaque forme ovale représente une action ou un cas d'utilisation qui accomplit une tâche dans l'application. Les lignes reliant l'acteur aux formes ovales représentent des relations et sont appelées association. Elles indiquent une communication ou une interaction basique entre un acteur et une action.

Dans l'application créée, toutes les fonctionnalités se trouvant dans le diagramme UML ont été implémentées.

Titre	Création d'une rentrée d'argent	
Description et but	L'utilisateur doit pouvoir ajouter une nouvelle rentrée d'argent contenant toutes les information requises.	
Acteurs	Utilisateur	
Préconditions	L'utilisateur se trouve sur la page de gestion des rentrées d'argent.	
Post conditions	La rentrée d'argent est créée.	
Déroulement basique	Étape	Action
	1	L'utilisateur clique sur ajouter une nouvelle rentrée d'argent.
	2	L'utilisateur remplit tous les champs du formulaire de création d'une nouvelle rentrée d'argent.
	3	L'utilisateur enregistre la nouvelle rentrée d'argent. | Alternative 1
Déroulement alternatif	Alternative 1 : L'utilisateur ne remplit pas tous les champs du formulaire	
	Étape	Action
	1	L'application web retourne un message d'erreur au champ non remplit.
	2	L'utilisateur retourne à l'étape 2 du déroulement basique.

TABLE 2.1. – Création d'une rentrée d'argent

Titre	Modification d'une sortie d'argent	
Description et but	L'utilisateur doit pouvoir modifier une sortie d'argent en remplissant un formulaire.	
Acteurs	Utilisateur	
Préconditions	L'utilisateur a créé au préalable une sortie d'argent.	
Post conditions	La sortie d'argent sélectionnée est modifiée.	
Déroulement basique	Étape	Action
	1	L'utilisateur clique sur modifier une sortie d'argent.
	2	L'utilisateur modifie les informations souhaitées dans un formulaire.
	3	L'utilisateur enregistre la sortie d'argent modifiée. | Alternative 1
Déroulement alternatif	Alternative 1 : L'utilisateur ne complète pas le formulaire de modification sur l'application.	
	Étape	Action
	1	L'application web retourne un message d'erreur au champ remplit de manière erronée.
	2	L'utilisateur retourne à l'étape 2 du déroulement basique.

TABLE 2.2. – Modification d'une sortie d'argent

3

Présentation du point de vue utilisateur final

3.1. Concept global

Cette application se base sur une architecture composée d'un client, d'un serveur et d'une base de données. Le client effectue des requêtes au serveur et ce dernier a pour rôle de répondre ensuite au client. Ainsi, le serveur est un service pour le client. Le point de vue de l'utilisateur final équivaut au client de l'application, et donc à l'interface utilisateur du gestionnaire de budget. L'interface est utilisée par l'utilisateur. Par ailleurs, comme l'application n'est pas déployée, la page d'accueil est atteignable à l'adresse `http://localhost:3000/accueil`. Chaque page de l'application créée commence par l'Uniform Resource Locator (URL) `http://localhost:3000`. Par exemple, la page concernant la gestion des rentrées d'argent a comme URL `http://localhost:3000/rentregestion`. L'application a été développée en français.

En premier lieu, en haut des différentes pages, se trouve une barre de navigation. Cette barre permet à l'utilisateur d'accéder facilement aux diverses pages de l'application web, à savoir à une page d'accueil, puis à deux sections, une section pour ses rentrées d'argent, et une autre section pour ses dépenses. En cliquant sur les différentes sections, il a la possibilité d'apercevoir les différentes pages qui composent chaque section. Par ailleurs, pour une meilleure compréhension, un logo est situé à côté de chacune des pages des deux sections. Enfin, chaque page contient un message en bas de page.

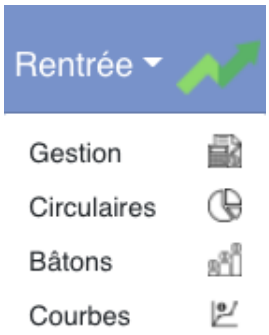

FIGURE 3.1. – Liste de toutes les pages pour les rentrées d'argent

FIGURE 3.2. – Liste de toutes les pages pour les sorties d'argent

Sur la page d'accueil, l'utilisateur a la possibilité de consulter le total de ses rentrées d'argent dans un encadré vert, le total de ses dépenses dans un encadré rouge, ainsi que son budget total dans un encadré variant de couleur. Si son budget total est positif l'encadré est vert, s'il est négatif l'encadré devient rouge. Son budget total est calculé en soustrayant le total de ses rentrées d'argent par le total de ses sorties d'argent. Les totaux varient en fonction des dates que l'usager souhaite afficher, ainsi que de l'ajout, de la modification et de la suppression de rentrées d'argent et de dépenses.

FIGURE 3.3. – Page d'accueil de l'application créée

La section des rentrées d'argent se compose d'une page pour la gestion des revenus et de trois pages contenants chacune deux diagrammes. Dans chacune des pages contenant les diagrammes, en dessous du titre de la page, se trouve par défaut le total des rentrées d'argent de l'utilisateur, depuis le premier jour du mois en cours jusqu'au jour actuel. Étant donné que l'architecture des pages des rentrées et des sorties d'argent est similaire, uniquement l'architecture des pages des rentrées d'argent est développée dans cette section. Par ailleurs, dans toutes les pages de cette application, lorsque l'usager a la possibilité de faire une recherche par date, les dates par défauts sont le premier jour du mois en cours et le jour actuel. Lorsqu'une page a été sélectionnée, en dessous de la barre de navigation, figure toujours le titre de la page.

La première page permet à l'usager de gérer ses rentrées d'argent. Pour cela, il a la possibilité d'ajouter une nouvelle rentrée d'argent en cliquant sur le bouton *Ajouter une nouvelle rentrée*. Lorsqu'il clique sur ce bouton, un formulaire s'ouvre. Ce formulaire d'ajout est décrit en détail dans la prochaine sous-section avec un scénario permettant de créer une rentrée d'argent. Une fois celle-ci créée, elle s'ajoute dans le tableau situé en dessous par ordre décroissant. Ce tableau se compose de quatre colonnes : une pour la catégorie de la rentrée d'argent, une pour son montant, une pour sa description, une pour sa date et, enfin, une contenant deux boutons. Le premier bouton *Modifier* permet de mettre à jour la rentrée d'argent sélectionnée et le deuxième bouton *Supprimer* sert à la supprimer. Lorsque l'utilisateur clique sur le bouton *Modifier*, un formulaire s'ouvre. Ce formulaire de modification est décrit en détail dans la prochaine sous-section avec un scénario permettant de modifier une sortie d'argent. En revanche, pour le bouton *Supprimer*, seulement une demande de confirmation s'affiche et, s'il y a confirmation, la rentrée d'argent est supprimée du tableau. En dessus du tableau, l'usager a la possibilité de trier toutes les rentrées d'argent créées. Il peut les trier par date, par catégorie et / ou par description. Les deux premiers filtres permettent d'obtenir une ou plusieurs rentrées d'argent par date. Le troisième filtre contient une liste de toutes les catégories disponibles. Le quatrième filtre est une barre de recherche permettant de trouver une ou plusieurs rentrées d'argent grâce à leur description. À souligner que pour ce dernier filtre, l'usager a la possibilité ou non d'utiliser des majuscules lors de la recherche. Par exemple, rechercher "Février", "février" ou "féVrIer" donne le même résultat. Afin de lancer une recherche, il suffit de cliquer sur le bouton *Rechercher*. En cliquant sur le bouton *Réinitialiser*, les filtres se remettent à 0 et toutes les rentrées d'argent sont de nouveau visibles dans le tableau.

La deuxième page regroupe deux diagrammes circulaires. Le premier diagramme circulaire réunit toutes les rentrées d'argent par catégorie. Chaque secteur représente une ou plusieurs rentrées d'argent avec la même catégorie, comme par exemple "Salaire". Le deuxième diagramme circulaire permet à l'utilisateur de rechercher plus précisément toutes les rentrées d'argent par catégorie, par description et / ou par date. Ce diagramme lui donne la possibilité d'avoir un aperçu de toutes les rentrées qui appartiennent par exemple à la catégorie "Argent de poche". La troisième et quatrième page de la section des rentrées d'argent contiennent respectivement deux diagrammes en bâtons et deux courbes, montrant les mêmes données que les diagrammes circulaires précédemment décrits. Cela permet à l'usager de choisir le diagramme qu'il préfère pour la visualisation de ses données. Les pages contenants les différents diagrammes sont décrites dans les prochaines sections de ce chapitre.

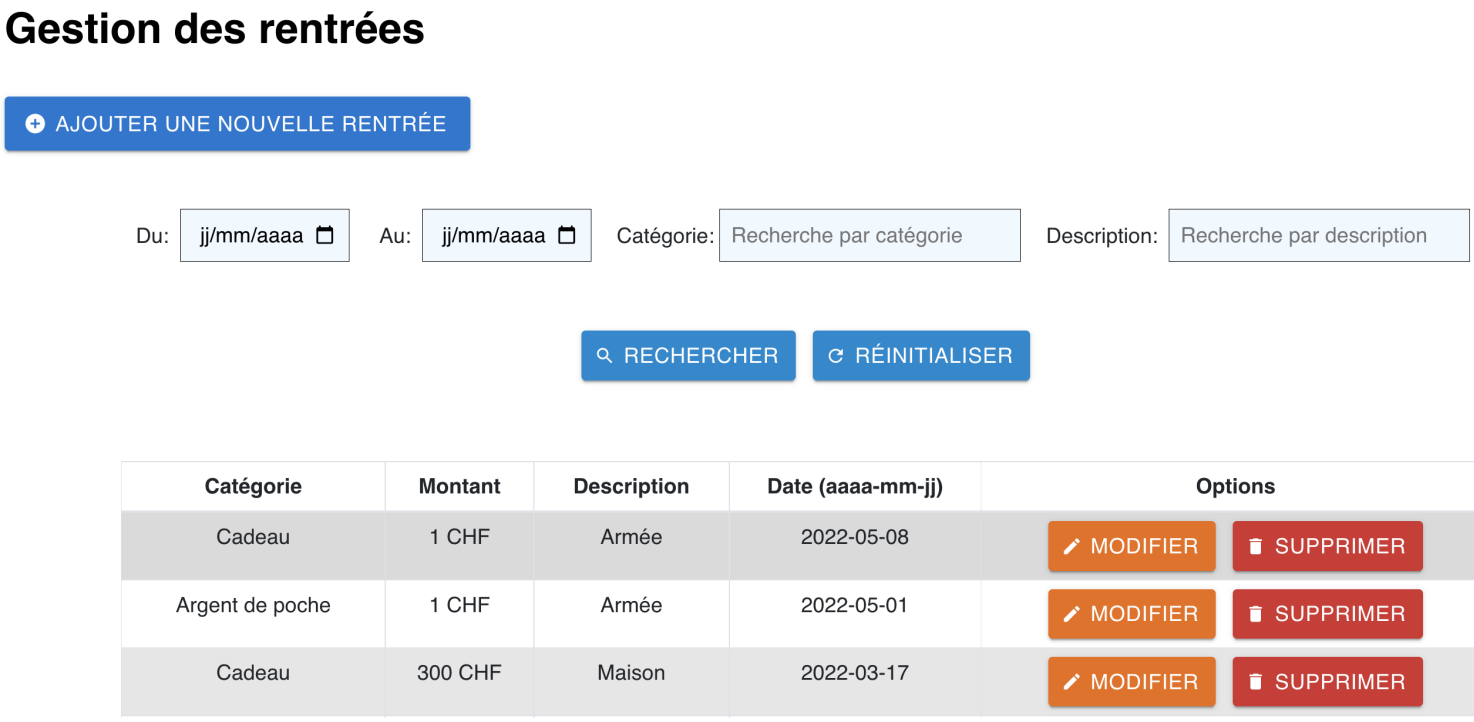

FIGURE 3.4. – Page de gestion des rentrées d'argent

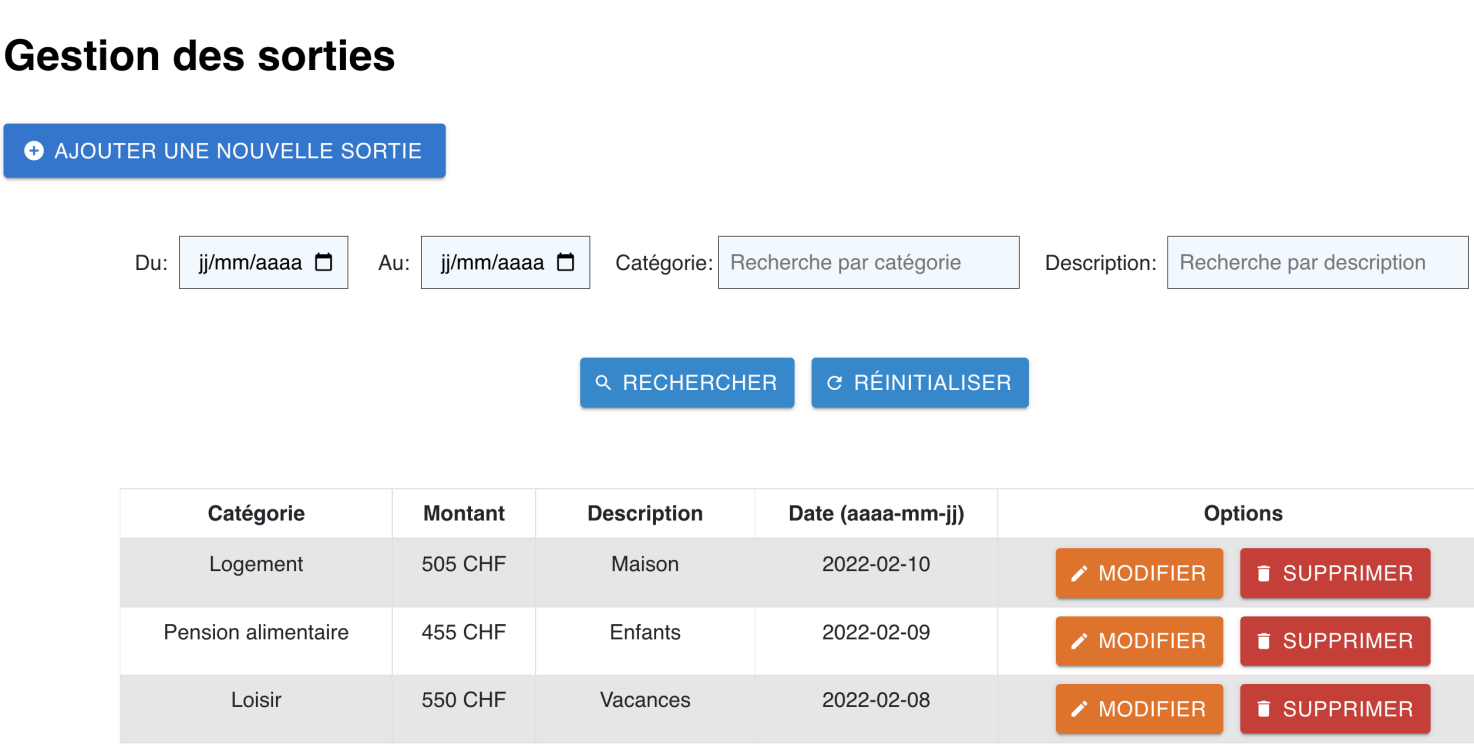

FIGURE 3.5. – Page de gestion des sorties d'argent

3.2. Scénarios

3.2.1. Ajout d'une rentrée d'argent

Ce sous-chapitre décrit précisément comment l'utilisateur a la possibilité d'ajouter une nouvelle rentrée d'argent. Tout d'abord, il se rend à la section "Rentrée", puis à la page "Gestion" se trouvant dans la barre de navigation. Une fois sur cette page, il clique sur le bouton *Ajouter une nouvelle rentrée* et un formulaire s'ouvre comme sur la Figure 3.6. L'usager choisit tout d'abord une catégorie préalablement définie dans une liste. S'il ne trouve pas de catégorie correspondante à sa rentrée d'argent, il peut sélectionner la catégorie "Autre", prévue à cet effet. Ensuite, il indique le montant, donne une description plus ou moins détaillée et choisit la date de la rentrée. À relever que les champs *Montant* ainsi que *Date* acceptent uniquement des chiffres. De plus, dans le champ *Date*, se trouve un calendrier facilitant la recherche d'une date. Pour pouvoir valider le formulaire d'ajout, l'utilisateur doit remplir tous les champs disponibles. À défaut, un message d'erreur apparaît à côté du champ non rempli. Lorsque tous les champs ont été remplis correctement et que l'usager clique sur le bouton *Ajouter*, la nouvelle rentrée d'argent s'ajoute dans le tableau de la page "Gestion" et se classe par ordre de date décroissante. Par ailleurs, un message de succès apparaît en haut de l'écran pour valider l'ajout de la nouvelle rentrée.

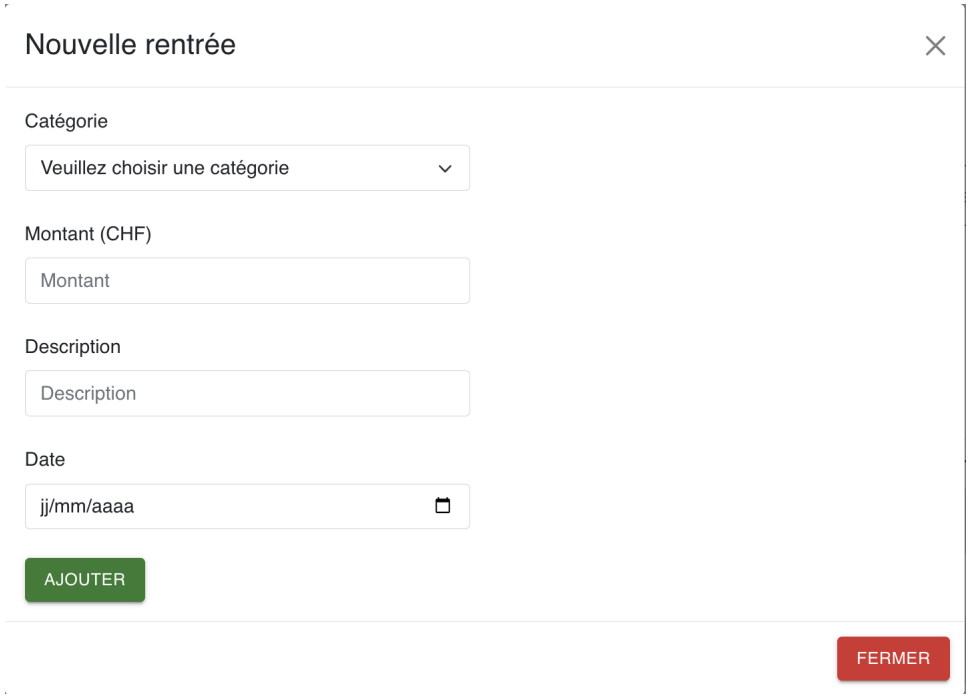

FIGURE 3.6. – Formulaire permettant de créer une rentrée d'argent

3.2.2. Modification d'une sortie d'argent

Lorsque l'utilisateur souhaite mettre à jour une dépense déjà existante, tout d'abord il se rend dans la rubrique "Sortie", puis à la page "Gestion". Ensuite, il clique sur le bouton *Modifier* situé à côté de la dépense qu'il désire modifier et un formulaire apparaît comme sur la Figure 3.7. Dans ce formulaire, il retrouve toutes les informations sur la sortie

d'argent sélectionnée, à savoir sa catégorie, son montant, sa description ainsi que sa date. Cela lui permet de consulter toutes les informations liées à cette dépense et de mettre à jour uniquement ce qu'il souhaite modifier. Pour modifier la catégorie, une liste déroulante est présente pour lui faciliter le choix. Pour les champs du montant et de la date, il peut inscrire uniquement des chiffres. Par ailleurs, dans le champ de la date, se trouve un calendrier facilitant la recherche d'une date. Une fois la ou les modifications effectuées, il est nécessaire de cliquer sur le bouton *Modifier* afin d'enregistrer les changements apportés à la dépense. Un message de succès apparaît en haut de l'écran pour valider la modification de la dépense. Par ailleurs, si l'utilisateur a ouvert par erreur le formulaire de modification d'une dépense, un bouton *Fermer* est disponible permettant de clore le formulaire sans apporter de modification. Pour terminer, lorsque le formulaire de modification est ouvert, si l'usager efface une valeur d'un champ sans la remplacer et qu'il clique sur le bouton *Modifier*, la dépense conserve la valeur précédemment effacée. Cela permet que toutes les dépenses aient chacune une catégorie, un montant, une description et une date.

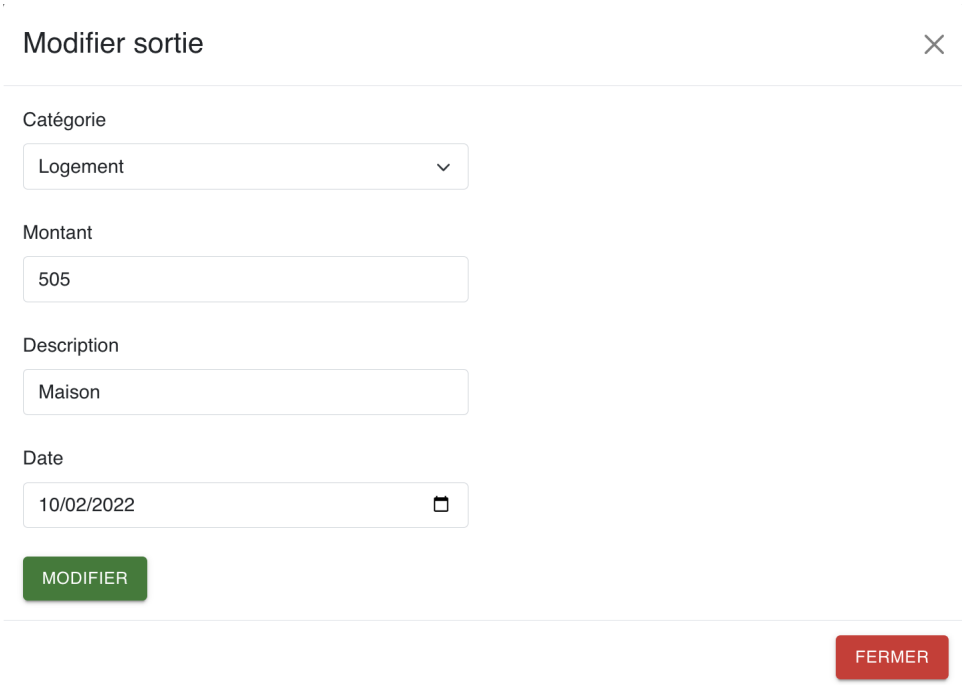

FIGURE 3.7. – Formulaire permettant de modifier une sortie d'argent

3.3. Lecture et utilisation des diagrammes

Les différentes pages contentant les diagrammes ont la même architecture. Cependant, chaque diagramme a ses spécificités et ses manières de représenter les rentrées et les sorties d'argent. En effet, l'utilisateur a la possibilité de choisir quel type de diagramme il préfère utiliser pour la lecture de ses données, en choisissant la page à consulter. Ceci lui permet d'utiliser facilement tous les diagrammes. Dans les prochaines sous-sections, sont décrits les diagrammes circulaires pour les rentrées d'argent, les diagrammes en bâtons pour les sorties d'argent et enfin les courbes pour les rentrées d'argent.

3.3.1. Diagrammes circulaires : rentrées d'argent

En haut de la page des diagrammes circulaires pour les rentrées d'argent, se trouve le total des rentrées d'argent dans un encadré vert. Cet encadré ne change pas de couleur. Le premier diagramme circulaire montre à l'utilisateur le total de chacune des catégories de rentrées d'argent. Chaque catégorie est définie par une couleur pour une meilleure lecture des données. Le choix des couleurs a été primordial. En effet, pour assurer une lisibilité optimale il est important que deux catégories côte à côte aient deux couleurs bien distinctes. Pour les rentrées d'argent, il y a 10 catégories différentes. De ce fait, il a fallu jouer sur les nuances des couleurs pour que l'usager puisse lire convenablement les données sur le diagramme. Par exemple, pour les catégories "Argent de poche", "Dividende / Intérêt", "Micro-service", "Pension alimentaire" et "Salaire", il a fallu trouver des couleurs suffisamment variées afin que, même si ces catégories se retrouvent à côté au niveau du digramme, l'utilisateur puisse, d'un coup d'œil, en saisir les différences. Ce digramme se lit dans le sens des aiguilles d'une montre. Sur le premier graphique, l'usager a la possibilité de sélectionner des dates pour lesquelles il souhaite un visuel grâce aux outils de recherche situés au-dessus du diagramme. Une fois les dates choisies, il appuie sur le bouton *Rechercher* pour valider son choix. En modifiant les dates, le total situé tout en haut de la page s'adapte également. Toujours sur le premier diagramme, si l'usager souhaite cacher une catégorie, il a la possibilité de cliquer sur ladite catégorie située dans la légende du graphique. En outre, à côté de chaque secteur du diagramme, le montant total de la catégorie est affiché. Par ailleurs, si l'utilisateur met son curseur sur un secteur du diagramme, il est en mesure de consulter la couleur associée à la catégorie, le titre de la catégorie ainsi que le montant de la catégorie. La page du diagramme circulaire pour les sorties d'argent contient également deux graphiques. Le premier graphique circulaire repose sur les mêmes principes précédemment décrits. Les différences principales sont l'encadré en haut de page qui est rouge pour la symbolique d'une sortie d'argent et les diverses catégories qui sont représentées. Étant donné qu'il y a plus de catégories pour les dépenses que pour les rentrées d'argent, il a fallu ajouter des nouvelles couleurs et faire en sorte qu'elles soient toujours lisibles pour l'utilisateur.

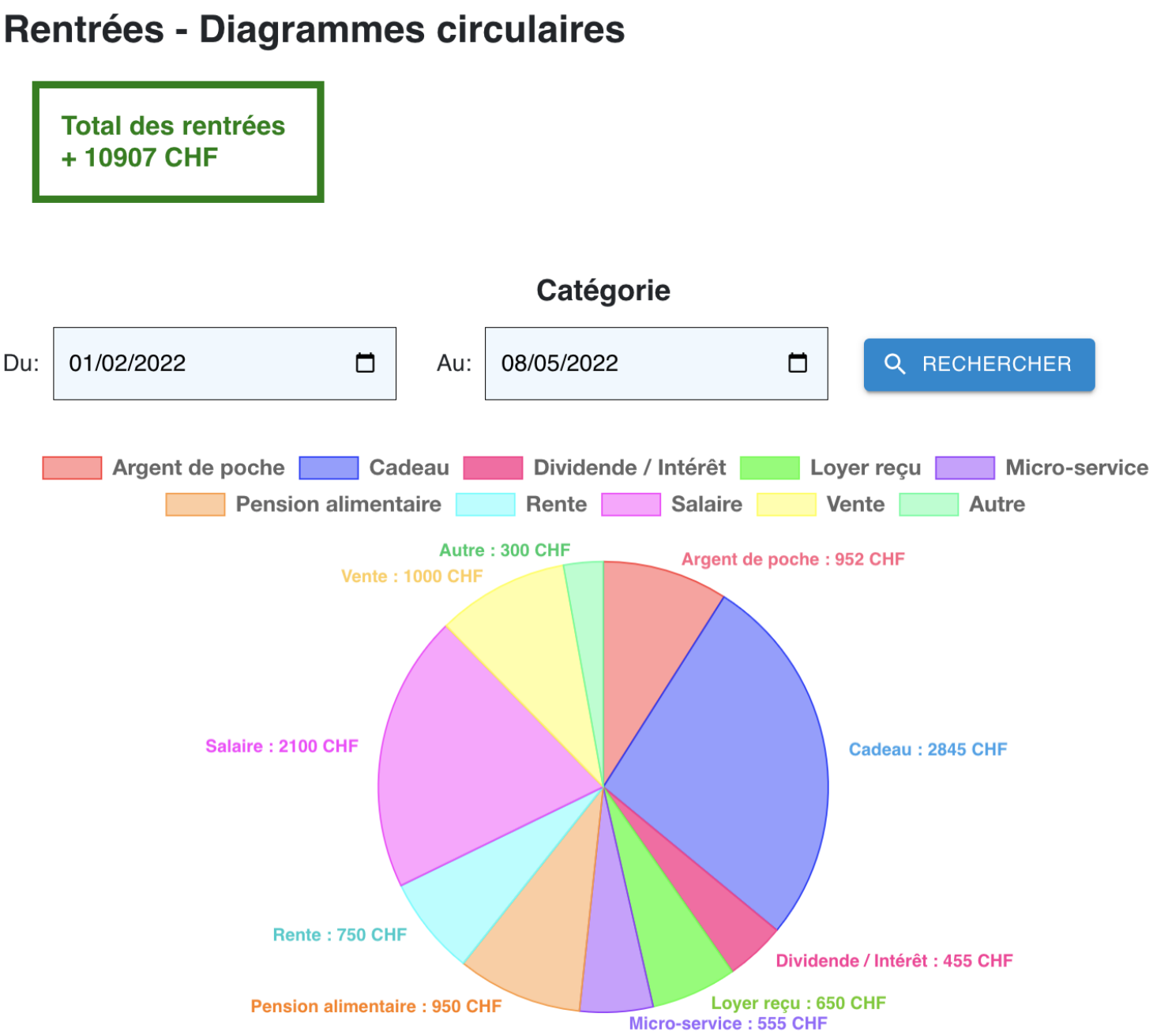

FIGURE 3.8. – Premier diagramme circulaire pour les rentrées d'argent

Le deuxième graphique circulaire affiche l'ensemble des rentrées d'argent que l'utilisateur a créé. Cependant, il ne regroupe pas les rentrées d'argent par catégorie. Ce deuxième diagramme se lit également dans le sens des aiguilles d'une montre et permet à l'usager d'afficher toutes les rentrées d'argent ayant la même catégorie et / ou la même description à un intervalle désiré. Ceci est possible grâce aux outils de filtrage situés sous le titre du deuxième diagramme. La recherche par catégorie se fait grâce à une liste déroulante de toutes les catégories. La liste est organisée par ordre alphabétique. Pour la recherche par description, l'usager doit écrire la description correspondante à une ou à plusieurs rentrées. Par ailleurs, il a la possibilité ou non d'utiliser des majuscules lors de la recherche. Par exemple, écrire "c" ou "C" donne le même résultat. Une fois les différents outils de filtration remplis, il clique sur le bouton *Rechercher* afin de lancer la recherche.

Pour effacer une catégorie ou une description choisie dans la barre de recherche, il peut supprimer les filtres qu'il a appliqué, puis il clique de nouveau sur le bouton *Rechercher*. Ce procédé permet d'afficher encore toutes les rentrées d'argent à la période indiquée. A côté de chaque secteur du diagramme, se trouve la description de la rentrée d'argent. Ceci permet à l'usager un meilleur aperçu lorsqu'il trie le graphique par catégorie. S'il déplace son curseur sur un secteur du diagramme, il peut observer la couleur de la rentrée d'argent, sa description et son montant. Par ailleurs, s'il souhaite cacher l'une des rentrées d'argent, il a la possibilité de cliquer sur la description correspondante se trouvant dans la légende du diagramme. Les couleurs utilisées pour ce diagramme sont des verts différents générés aléatoirement afin que chaque rentrée d'argent puisse être lue plus aisément par l'utilisateur. La seule différence notable entre le deuxième diagramme circulaire des rentrées d'argent et celui des sorties d'argent, est que pour ce dernier les couleurs utilisées sont différents rouges, également générés aléatoirement.

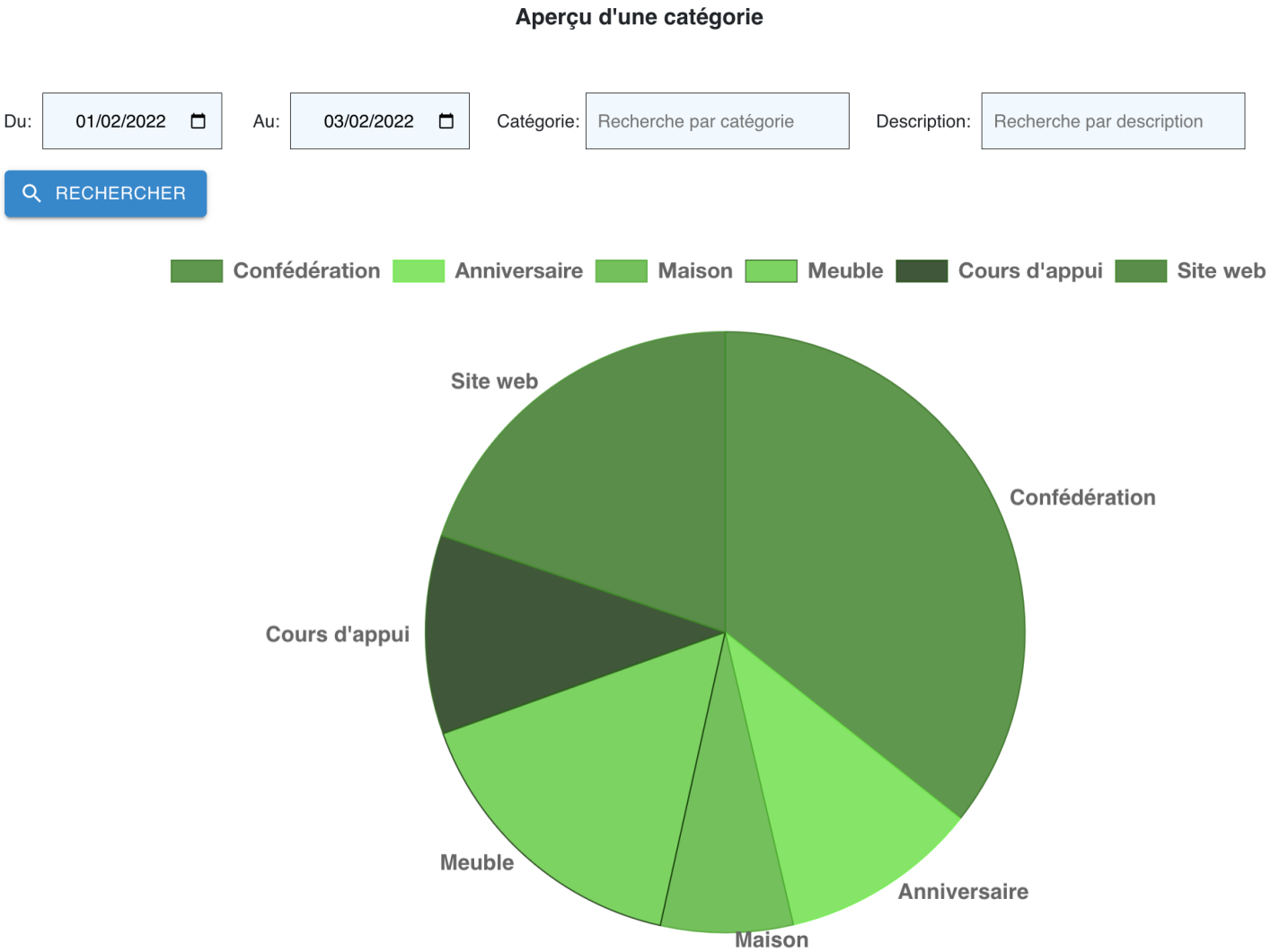

FIGURE 3.9. – Deuxième diagramme circulaire pour les rentrées d'argent

Le tableau en fin de page est lié au deuxième diagramme circulaire. Il permet à l'utilisateur de consulter toutes les informations de la ou des rentrées d'argent qui sont représentées dans le graphique, à savoir la catégorie, le montant, la description ainsi que la date. Le tableau affiche toutes les rentrées d'argent par date de défaut. Les rentrées d'argent sont triées par date décroissante. Lorsque l'usager utilise les différents outils de filtrage mis à disposition, le tableau s'adapte également automatiquement selon les résultats de la recherche. Si l'utilisateur efface les filtres de catégorie et de description, le tableau s'ajuste également pour reprendre son état initial.

Catégorie	Montant	Description	Date (aaaa-mm-jj)
Salaire	1000 CHF	Confédération	2022-02-03
Cadeau	300 CHF	Anniversaire	2022-02-03
Loyer reçu	200 CHF	Maison	2022-02-02
Vente	450 CHF	Meuble	2022-02-02
Salaire	300 CHF	Cours d'appui	2022-02-01
Micro-service	555 CHF	Site web	2022-02-01

FIGURE 3.10. – Tableau en fin de page des diagrammes circulaires pour les rentrées d'argent

3.3.2. Diagrammes en bâtons : sorties d'argent

Tout en haut de cette page, dans un encadré rouge, l'utilisateur peut lire le total de ses dépenses. Cet encadré rouge ne change également pas de couleur. Le premier diagramme en bâtons montre à l'usager le total de chacune des catégories des sorties d'argent et reprend les mêmes fonctionnalités de base que le premier diagramme circulaire pour les rentrées d'argent. Sur l'axe des abscisses du diagramme, l'utilisateur retrouve toutes les catégories des sorties d'argent. Sur l'axe des ordonnées, se trouve une échelle de montant qui s'ajuste automatiquement en fonction du montant des catégories. Puis, au-dessus de chaque bâton, l'usager retrouve le montant total des catégories représentées. Lorsqu'il met son curseur sur l'une des barres, il peut lire le titre de la catégorie, la couleur associée à cette catégorie ainsi que le montant total. À la différence des diagrammes circulaires, il ne peut pas cacher une ou des catégories sur les digrammes en bâtons.

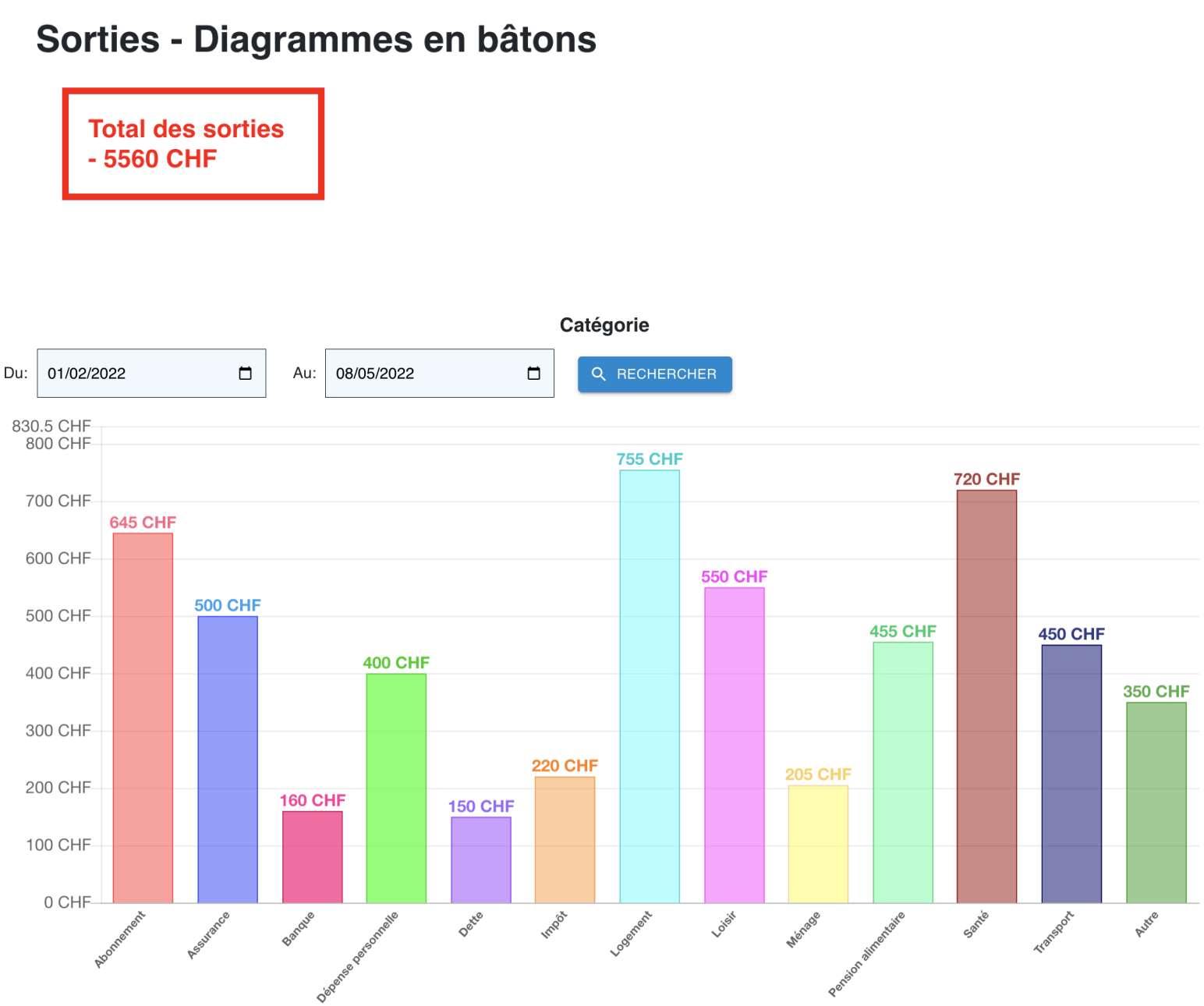

FIGURE 3.11. – Premier diagramme en bâtons pour les sorties d'argent

Tout comme le deuxième diagramme circulaire des rentrées d'argent, le deuxième diagramme en bâtons permet à l'utilisateur d'avoir un aperçu de l'ensemble de ses dépenses à une période donnée. Il a également la possibilité de rechercher des sorties d'argent par dates, par catégorie et / ou par description. Le principe de recherche par filtre et de suppression des filtres est le même que pour le deuxième diagramme circulaire. Comme pour le premier diagramme en bâtons, sur l'axe des abscisses du deuxième graphique, l'usager retrouve les descriptions de ses différentes sorties d'argent. Sur l'axe des ordonnées, il y a une échelle de montant qui s'ajuste aussi automatiquement. Pour le choix des couleurs, toutes les sorties ont un rouge généré aléatoirement pour permettre une meilleure lecture des informations.

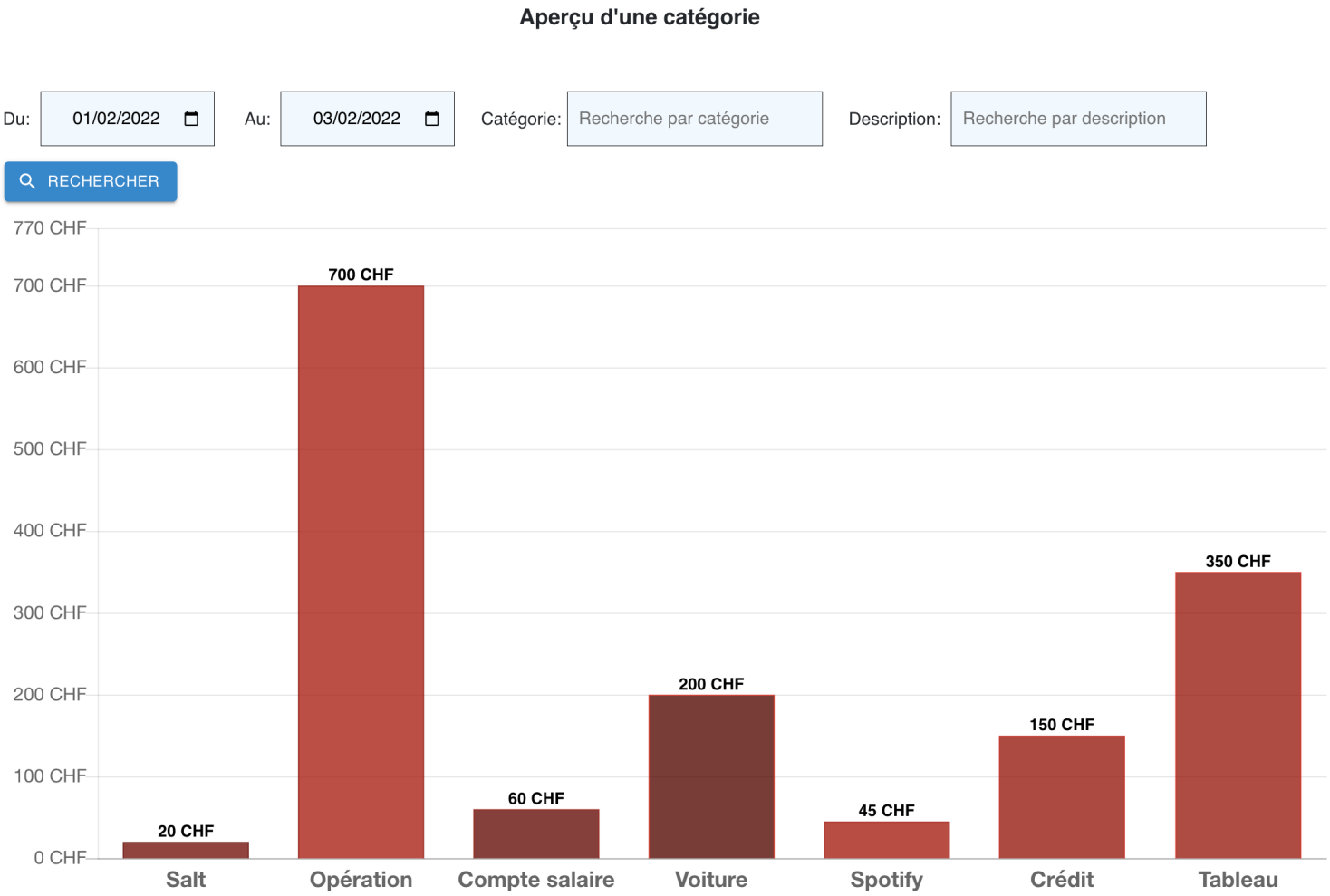

FIGURE 3.12. – Deuxième diagramme en bâtons pour les sorties d'argent

Le tableau en fin de page reprend la même architecture et les mêmes fonctionnalités que le tableau qui se trouve à la page des digrammes circulaires pour les rentrées d'argent. L'unique différence est que la sortie d'argent située tout à gauche du deuxième graphique figure tout en haut du tableau.

Catégorie	Montant	Description	Date (aaaa-mm-jj)
Impôt	20 CHF	Salt	2022-02-03
Santé	700 CHF	Opération	2022-02-03
Banque	60 CHF	Compte salaire	2022-02-02
Impôt	200 CHF	Voiture	2022-02-02
Abonnement	45 CHF	Spotify	2022-02-02
Dette	150 CHF	Crédit	2022-02-01
Autre	350 CHF	Tableau	2022-02-01

FIGURE 3.13. – Tableau en fin de page des diagrammes en bâtons pour les sorties d'argent

3.3.3. Courbes : rentrées d'argent

Les derniers diagrammes que l'utilisateur peut consulter sont des courbes. La structure de la page reste globalement la même que celle des autres pages comprenant des diagrammes. A savoir, l'utilisateur retrouve en haut de la page un encadré vert totalisant le montant des rentrées d'argent, puis deux diagrammes, dont le premier montrant l'évolution totale de chaque catégorie, et le deuxième affichant la totalité des rentrées d'argent à une période choisie. Les axes des abscisses et des ordonnées ont la même architecture que les axes utilisés pour les diagrammes en bâtons. La couleur choisie pour la courbe est le vert, symbolisant une rentrée d'argent. Si l'usager place son curseur sur l'un des points présents sur la courbe, il a la possibilité de lire le nom de la catégorie, la couleur de la courbe, ainsi que le montant total de la catégorie.

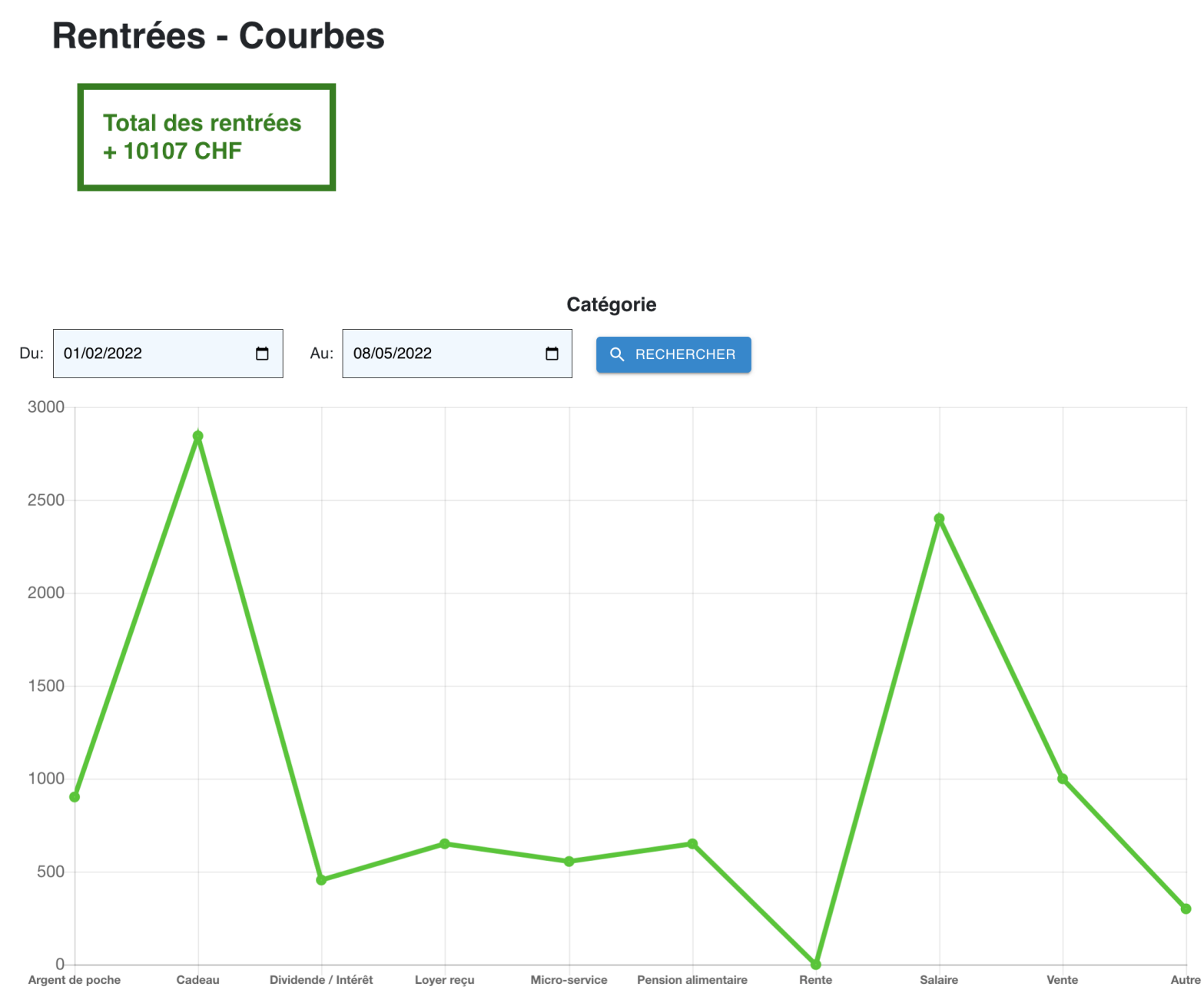

FIGURE 3.14. – Première courbe pour les rentrées d'argent

Le deuxième graphique repose aussi sur la même structure et les mêmes fonctionnalités que les autres diagrammes présents dans les différentes pages de l'application web.

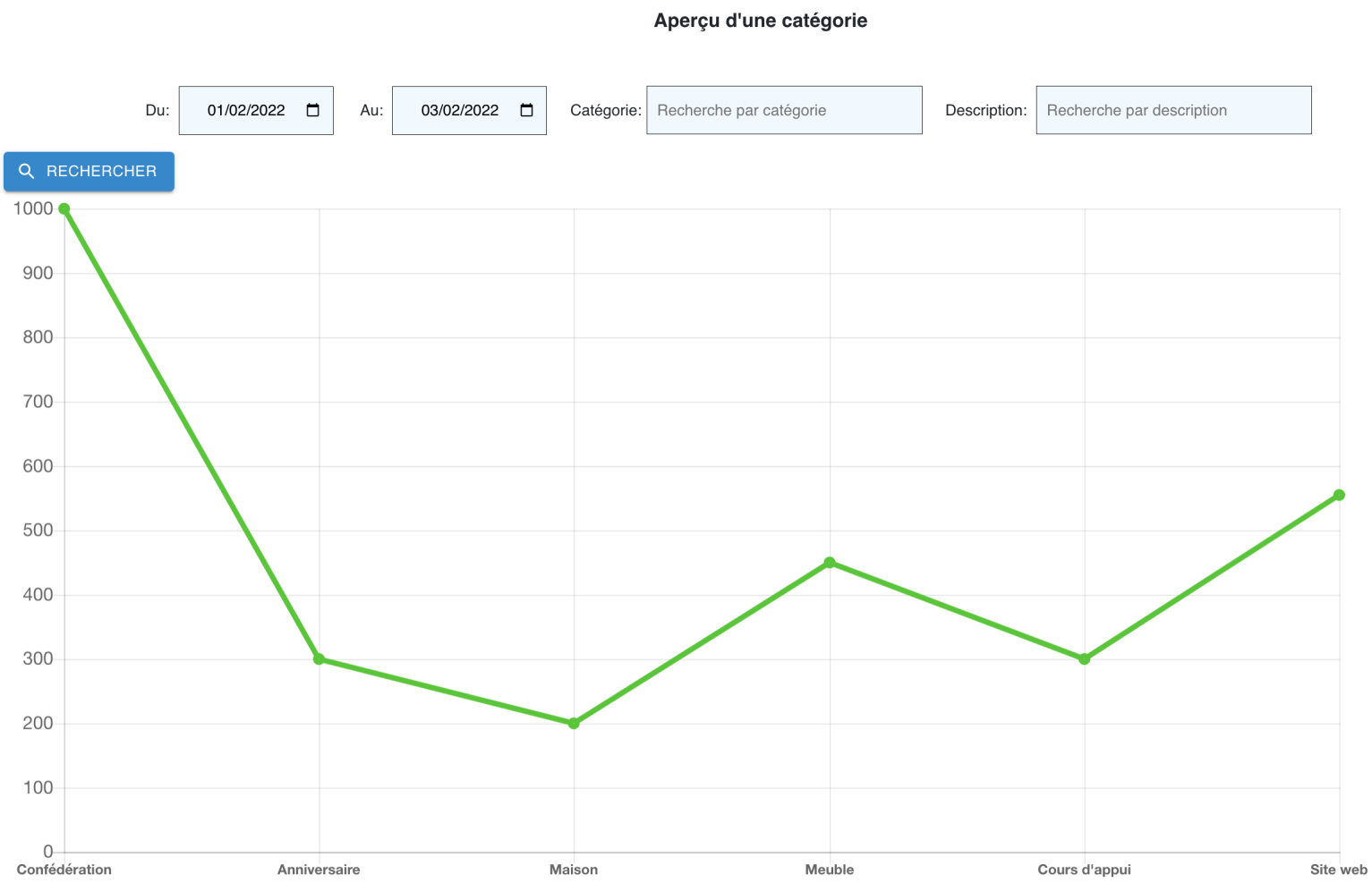

FIGURE 3.15. – Deuxième courbe pour les rentrées d'argent

Le tableau en fin de page repose sur la même architecture que le tableau se trouvant à la page des digrammes en bâtons pour les sorties d'argent.

Catégorie	Montant	Description	Date (aaaa-mm-jj)
Salaire	1000 CHF	Confédération	2022-02-03
Cadeau	300 CHF	Anniversaire	2022-02-03
Loyer reçu	200 CHF	Maison	2022-02-02
Vente	450 CHF	Meuble	2022-02-02
Salaire	300 CHF	Cours d'appui	2022-02-01
Micro-service	555 CHF	Site web	2022-02-01

FIGURE 3.16. – Tableau en fin de page des courbes pour les rentrées d'argent

4
Programmation du serveur

4.1. Présentation générale

Le serveur représente l'outil employé par le client pour interagir avec la base de données. En premier lieu, il est important de définir l'architecture ainsi que les technologies utilisées pour le développement du serveur. L'architecture Representational State Transfer (REST) convient parfaitement à ce type d'application, étant un service client-serveur. En effet, cette application exécute les opérations Create, Read, Update, Delete (CRUD) et elle n'a pas besoin de maintenir l'état du client. L'architecture REST développée est présentée dans la prochaine section de ce chapitre. Concernant les technologies utilisées, le serveur est développé dans le langage de programmation Python et notamment avec l'aide du framework `aiohttp`. Ce framework prend en charge les côtés serveur du protocole Hypertext Transfer Protocol (HTTP) et client, tout en étant asynchrone [10]. En effet, il permet de développer des serveurs et des clients asynchrones [11]. Pour cette application, ce framework est uniquement utilisé pour développer le serveur.

Un framework asynchrone est très pratique pour un service client-serveur. En effet, il permet de continuer une opération, même si une ou plusieurs opérations précédentes ne sont pas terminées [12]. Par exemple, pour une telle application, cela a l'avantage de charger une page sans que toutes les informations ne soient obtenues. La syntaxe asynchrone en Python se base sur l'utilisation de deux mots-clés, `async` et `await`. Une fonction déclarée avec le mot-clé `async` est appelée une coroutine [13]. Une coroutine peut également être un objet retourné par une fonction coroutine si elle est employée avec le mot-clé `await`.

Concernant le code du serveur, tout est regroupé dans le fichier `Database.py`. Cela permet d'avoir l'ensemble du code à un même endroit. Le code source se trouve sur le repository Github [1].

[1] `https://github.com/Manga3000/Gestionnaire-de-budget`

Le code du serveur est divisé en quatre parties : tout d'abord les différents modules et librairies importés et utilisés, ensuite les méthodes pour effectuer des requêtes, puis les chemins associés à chaque méthode, et pour terminer le code permettant le démarrage du serveur.

Tout d'abord, les principaux modules et frameworks utilisés sont `json`, `aiohttp`, `motor`, `datetime` et `aiohttp_swagger`. Le module `json` permet l'encodage d'objet en Python au format JSON [14]. Ce format d'encodage de données léger s'inspire de la syntaxe des objets dans le langage de programmation JavaScript. Tous les objets (rentrées et sorties d'argent) dans cette application sont encodés en JSON. La nécessité du module `aiohttp` a été décrite précédemment. De plus, deux sous-modules `aiohttp` sont utilisés, `aiohttp_cors` et `aiohttp_swagger`, qui permettent de faire le lien avec Cross-Origin Resource Sharing (CORS) et Swagger. CORS dénote la situation dans laquelle un domaine appelant une ressource est différent du domaine partageant ladite ressource [15]. C'est notamment le cas de cette application. En effet, le navigateur (client) communique avec le serveur. La librairie `aiohttp_cors` permet d'implémenter la prise en charge du partage de ressources d'origine croisée, ou CORS, pour un serveur HTTP asynchrone utilisant le framework `aiohttp` [16]. Le module `aiohttp_swagger` permet de générer une documentation de l'Application Programming Interface (API) pour serveur construit avec `aiohttp` grâce à la technologie Swagger. Ce dernier module est présenté dans la troisième section de ce chapitre. Le module `motor` permet d'utiliser Motor, qui est un pilote (driver) Python asynchrone pour la connection avec la base de données MongoDB. Motor est également développé plus en détail dans la dernière section de ce chapitre concernant MongoDB. Pour terminer, le module `datetime` offre des classes permettant de manipuler les heures et les dates [17].

Ensuite, les méthodes développées sont issues de la partie du code concernant les sorties d'argent. Étant donné que les méthodes pour les rentrées d'argent suivent la même architecture que les sorties d'argent, uniquement les méthodes des sorties d'argent sont développées. De plus, toutes les méthodes développées sont des méthodes asynchrones. Les chemins associés à chaque méthode sont développés dans le paragraphe dédié aux chemins.

La première méthode de requête employée, `get_all_sorties`, permet de rechercher toutes les sorties d'argent enregistrées dans la base de données et de les retourner toutes. Il s'agit d'une méthode GET. Les sorties d'argent sont enregistrées dans un tableau. Cela figure à la ligne 4 du Listing 4.1.

```
1 async def get_all_sorties(request):
2     sorties = []
3     cursor = collection_Sorties.find({})
4     async for document in cursor:
5         sorties.append(document)
6     return web.json_response(sorties)
```

List. 4.1 – Méthode `get_all_sorties`

La deuxième méthode, `get_sortie_by_filtre`, est également une méthode GET. La différence avec la première méthode, est que celle-ci retourne également toutes les sorties d'argent enregistrées dans la base de données, mais uniquement celles qui correspondent aux filtres appliqués. En effet, dans cette méthode, les sorties d'argent sont filtrées par catégorie, par description et par date. L'utilisation de `request.rel_url.query` permet d'obtenir en paramètres des chaînes de caractères dans une requête à partir d'une URL

en utilisant le framework `aiohttp`. Cela permet concrètement d'appliquer des filtres de recherche lors d'une requête HTTP GET. Pour la recherche par date, deux variables sont créées pour enregistrer des dates. Ces deux variables sont `from_date` et `to_date`. La méthode `strftime()` employée renvoie une à chaîne de caractère représentant la date à l'aide de l'objet datetime [18]. Ces deux variables sont ensuite utilisées avec deux opérateurs de comparaison propres à MongoDB, à savoir `$gte` et `$lte`. Ces deux opérateurs permettent de retourner ensuite des dates plus grandes que (`$gte`) et plus petites que (`$lte`) selon la requête envoyée. Ensuite, la recherche par expression régulière, RegEx, est utilisée afin d'appliquer les filtres par catégorie et par description, grâce aux deux variables `categorie` et `description`. Cette recherche est directement employée dans la syntaxe de MongoDB. L'option `i` permet de trouver des expressions régulières comportant des lettres en minuscules et en majuscules. Comme pour la première méthode GET, les sorties d'argent sont enregistrées dans un tableau à la ligne 15 du Listing 4.2.

```
1  async def get_sortie_by_filtre(request):
2      sorties = []
3
4      from_date = request.rel_url.query['from']
5      to_date = request.rel_url.query['to']
6      categorie = request.rel_url.query['categorie']
7      description = request.rel_url.query['description']
8
9      from_date = datetime.datetime.now().strftime(from_date)
10     to_date = datetime.datetime.now().strftime(to_date)
11     criteria = {"$gte": from_date, "$lte": to_date}
12
13     cursor = collection_Sorties.find({'Categorie': {'$regex' : '.*' + categorie + '.*'
           , '$options': 'i'}, 'Description': {'$regex' : '.*' + description + '.*', '
           $options': 'i'}, 'Date': criteria})
14
15     async for document in cursor:
16         sorties.append(document)
17     return web.json_response(sorties)
```

List. 4.2 – Méthode `get_sortie_by_filtre`

La troisième méthode, `get_categorie_of_sortie_by_date`, est la dernière méthode GET de cette application concernant les sorties d'argent et comporte deux fonctionnalités. La première fonctionnalité regroupe toutes les sorties d'argent enregistrées dans la base de données par catégorie, puis calcule la somme de chaque catégorie ainsi que la somme totale des catégories. Chaque catégorie vaut initialement 0. Ensuite, au fur et à mesure que des sorties d'argent avec une certaine catégorie sont créées, la somme de chaque catégorie augmente. Pour terminer, le total de toutes les catégories est calculé. La deuxième fonctionnalité filtre chaque catégorie par date, suivant le même principe de recherche que la méthode `get_sortie_date` décrite précédemment, pour retourner toutes les catégories selon une période choisie. Les sorties d'argent par catégorie sont enregistrées dans un dictionnaire à la ligne 1 du Listing 4.3.

```
1  async def get_categorie_of_sortie_by_date(request):
2      categories = {}
3      [...]
4      async for document in cursor:
5
6          if document["Categorie"]== "Abonnement":
```

```
7           abonnement = abonnement + document["Montant"]
8       elif document["Categorie"]== "Assurance":
9           assurance = assurance + document["Montant"]
10      elif document["Categorie"]== "Banque":
11          banque = banque + document["Montant"]
12  [...]
13          total = total + document["Montant"]
14
15  categories["Abonnement"] = abonnement
16  categories["Assurance"] = assurance
17  categories["Banque"] = banque
18  [...]
19  categories["Total"] = total
20
21  return web.json_response(categories)
```

List. 4.3 – Méthode `get_categorie_of_sortie_by_date`

La quatrième méthode, `create_sortie`, est une méthode POST et permet de créer une nouvelle sortie d'argent et de l'enregistrer dans la base de données. Tout d'abord, chaque attribut, comme la catégorie, le montant, la description ainsi que la date, est vérifié qu'à la création, il soit du bon type. En effet, il faut que la catégorie, la description ainsi que la date d'une nouvelle sortie d'argent soient des chaînes de caractères, et que le montant soit un chiffre entier ou à virgule. Si un attribut est assigné incorrectement, un message d'erreur est retourné. Un identifiant propre est associé à chaque dépense. À chaque nouvel objet créé, l'identifiant est incrémenté. Cela permet à chaque objet d'avoir premièrement un identifiant unique, et deuxièmement un identifiant plus simple à lire que les identifiants de base créés par MongoDB. Une fois la nouvelle sortie d'argent créée, elle est enregistrée dans la base de données à la ligne 29.

```
1  async def create_sortie(request):
2      data = await request.json()
3
4      if 'Categorie' not in data:
5          return web.json_response({'error': '"Categorie" is a required field'})
6      if not type(categorie) is str:
7          return web.json_response({'error': '"Categorie" must be a string with at least
               one character'})
8      categorie = data['Categorie']
9
10     if 'Montant' not in data:
11         return web.json_response({'error': '"Montant" is a required field'})
12     if not isinstance(montant, (int, float)):
13         return web.json_response({'error': '"Montant" must be a number with at least
               one number'})
14     data['Montant'] = float(data['Montant'])
15     montant = data['Montant']
16
17     if 'Description' not in data:
18         return web.json_response({'error': '"Description" is a required field'})
19     if not type(description) is str:
20         return web.json_response({'error': '"Description" must be a string with at
               least one character'})
21     description = data['Description']
22
23     if 'Date' not in data:
```

```
24          return web.json_response({'error': '"Date" is a required field'})
25      if not type(date) is str:
26          return web.json_response({'error': '"Date" must be a string with at least one
                character'})
27      date = data['Date']
28
29      sortie = {"_id" : new_id, "Categorie": categorie, "Montant": montant, "Description
            ": description, "Date": data['Date']}
30      await collection_Sorties.insert_one(sortie)
31
32      return web.Response(content_type="application/json")
```

List. 4.4 – Méthode `create_sortie`

La cinquième méthode, `update_sortie`, est une méthode PATCH et permet de mettre à jour une sortie d'argent déjà existante dans la base de données et de la retourner ensuite. Cette méthode est notamment utilisée si l'utilisateur a mal rempli les différents attributs lors de la création d'une sortie d'argent. De plus, cette méthode permet à l'usager de modifier un ou plusieurs attributs en même temps. Les lignes 1 à 2 du Listing 4.5 permettent de s'assurer que le type du montant sauvegardé est bien de type float. En effet, si cette vérification n'est pas effectuée, le montant retourné pourrait être de type chaîne de caractère suite à une modification. Si la dépense ne se trouve pas dans la base de données avant une mise à jour, un message d'erreur est retourné. Dans le cas contraire, la sortie d'argent sélectionnée est modifiée puis retournée.

```
1  async def update_sortie(request):
2      if 'Montant' in data:
3          data['Montant'] = float(data['Montant'])
4
5      document = await collection_Sorties.find_one(json.loads(sortie_id))
6
7      if document is None:
8          return web.json_response({'error': 'Sortie not found'}, status=404)
9      else:
10         await collection_Sorties.update_one({'_id': id}, {'$set': data})
11
12     new_document = await collection_Sorties.find_one(json.loads(sortie_id))
13
14     return web.json_response(new_document)
```

List. 4.5 – Méthode `update_sortie`

Enfin, la sixième méthode, `remove_sortie`, est une méthode DELETE et permet de supprimer une sortie d'argent à la fois. Tout d'abord, la sortie doit se trouver dans la base de données, sinon un message d'erreur est retourné. Ensuite, si la sortie se trouve effectivement dans la base de données, elle est supprimée.

```
1  async def remove_sortie(request):
2      document = await collection_Sorties.find_one(json.loads(sortie_id))
3
4      if document is None:
5          return web.json_response({'error': 'Sortie not found'})
6      else:
7          await collection_Sorties.delete_one(json.loads(sortie_id))
8
```

```
9     return web.Response(status=204)
```

List. 4.6 – Méthode `remove_sortie`

Pour les chemins associés aux différentes méthodes décrites précédemment, seules les méthodes `update_sortie` et `remove_sortie` ont comme chemin `"sortie/id"`. En effet, ces deux méthodes ont besoin de connaître l'identifiant de la sortie d'argent à modifier ou à supprimer. De plus, les deux dernières méthodes GET, à savoir les méthodes `get_sortie_by_filtre` et `get_categorie_of_sortie_by_date` ont différents paramètres dans leur chemin. La méthode `get_sortie_by_filtre` prend en paramètre deux dates, une pour le début et une pour la fin d'une période, une catégorie, ainsi qu'une description permettant de filtrer les sorties d'argent présentes dans la base de données. La méthode `get_categorie_of_sortie_by_date` prend en paramètre deux dates également, une pour le début et une pour la fin d'une période, donnant accès aux sorties d'argent regroupées par catégorie à une période choisie et présentes dans la base de données. Les méthodes `get_all_sorties` et `create_sortie` ne prennent pas de paramètre lors de leur exécution.

```
1 cors.add(app.router.add_get('/sorties', get_all_sorties))
2 cors.add(app.router.add_post('/sortie', create_sortie))
3 cors.add(app.router.add_patch('/sortie/{id}', update_sortie))
4 cors.add(app.router.add_delete('/sortie/{id}', remove_sortie))
5 cors.add(app.router.add_get('/sortiee', get\_sortie\_by\_filtre))
6 cors.add(app.router.add_get('/sortie', get\_categorie\_of\_sortie\_by\_date))
```

List. 4.7 – Chemins des sorties

Pour terminer, afin de démarrer le serveur, il faut tout d'abord créer une instance d'application, comme à la première ligne du Listing 4.8. Puis, la deuxième ligne du Listing 4.8 permet d'exécuter l'application. Ces deux lignes permettent de démarrer le serveur à l'adresse : *http ://127.0.0.1 :8080*.

```
1 app = web.Application()
2 web.run_app(app, host="127.0.0.1")
```

List. 4.8 – Démarrage du serveur

4.2. Principes généraux de l'architecture REST

Ce travail repose sur l'architecture REST. Cette architecture permet de construire un service web. Un service web permet d'échanger des données et d'interagir avec d'autres applications web [19]. Il existe d'autres services web comme REST qui sont connus, comme par exemple Simple Object Access Protocol (SOAP) ou HTTP [20]. Un service web développé sur cette architecture se nomme également service web RESTful. Un service web repose sur trois étapes :

1. Le client effectue une requête HTTP.
2. La requête HTTP est envoyée à un serveur.
3. Le serveur produit une réponse à la requête HTTP.

REST définit 4 principes architecturaux principaux [21, 22, 23] :

1. Ressource : tous les éléments utilisés dans l'architecture REST sont des ressources, comme les fonctionnalités et les données. Par exemple, des documents ou des images peuvent être des ressources REST.
2. Client–serveur : le modèle de conception d'un client et d'un serveur indique qu'il n'y a pas de dépendances entre le client et le serveur, grâce à l'interface uniforme. De ce fait, le client et le serveur peuvent par exemple être remplacés et développés indépendamment, tant que l'interface reste inchangée. De plus, cette séparation de développement implique que le client n'est pas concerné par le stockage des données, qui demeure interne au serveur. Ce fait améliore la portabilité du client. De l'autre côté, le serveur ne dépend lui non plus de l'interface utilisateur, ou du client, et cela permet de rendre le serveur plus simple et plus évolutif.
3. Sans état (*stateless*) : ce principe signifie que toutes les requêtes, du client au serveur, sont sans état. En effet, il est nécessaire que dans les requêtes envoyées par le client, figurent toutes les informations essentielles afin que le serveur puisse comprendre et répondre aux requêtes. De ce fait, le serveur ne conserve aucune requête précédemment envoyée et il est donc appelé (*stateless*). Ce dernier traite indépendamment chaque requête et la traite comme nouvelle.
4. Interface uniforme : ce principe simplifie l'architecture REST, ce qui permet au client et au serveur d'évoluer séparément et d'augmenter la visibilité des interactions. De plus, elle spécifie aussi l'interface entre le client et le serveur.

Il existe d'autres principes architecturaux, cependant ils sont moins importants que les quatre principes précédemment décrits.

Comme décrit précédemment dans les principes architecturaux, les échanges entre le client et le serveur doivent se faire sans état, et il s'agit du protocole de communication HTTP qui gère cette communication. De ce fait, HTTP est utilisé pour l'application de ce travail et ses méthodes principales sont [24, 25] :

— POST : Cette méthode permet de créer (*create*) des données uniques.

— GET : Cette méthode permet de lire (*read*) et de récupérer des données reçues par un serveur.

— PUT : Cette méthode permet de mettre à jour (*update/replace*) des données. La spécificité de cette méthode est qu'elle modifie tous les champs d'une donnée. De plus, elle est souvent comparée à la méthode POST, car en modifiant tous les champs d'une donnée, cela peut être considéré comme la création d'une nouvelle donnée.

— PATCH : Cette méthode permet également de mettre à jour (*update/modify*) des données comme la méthode PUT. La différence est qu'avec la méthode PATCH, il est possible de modifier un ou des champs d'une donnée.

— DELETE : Cette méthode permet de supprimer (*delete*) une donnée. Une donnée supprimée ne peut plus être lue ou modifiée.

Il existe de nombreuse autres méthodes HTTP, comme les méthodes OPTIONS, HEAD, CONNECT ou TRACE, qui ne sont pas décrites, car moins importantes. Les méthodes HTTP utilisées pour ce travail sont les méthodes POST, GET, PATCH et DELETE.

4.3. Endpoints avec SwaggerUI

Un service web RESTful nécessite une API. En effet, tout service web est en réalité une API, tandis que le contraire n'est pas toujours le cas. Une API est une interface permettant à différentes applications web de communiquer sans connaître les détails de leur mise en œuvre, comme dans notre cas entre le client et le serveur. Une API peut également être vue comme un contrat avec une documentation précise qui créée un accord entre les deux parties. Les routages utilisés dans le serveur sont les mêmes routages que ceux utilisés dans l'API, également appelés *endpoints*. Un *endpoint*, point de terminaison, est l'emplacement où une API envoie les différentes demandes au serveur [26, 27]. Pour documenter une API, il existe plusieurs technologies, comme Swagger, Postman ou Apigee. Pour ce travail, la technologie Swagger a été utilisée et Swagger UI est son interface utilisateur [28]. De plus, la librairie `aiohttp_swagger` permet d'associer le framework `aiohttp` à la technologie de Swagger. Cette association permet, comme dit dans la première section de ce chapitre, de générer une documentation de l'API pour un serveur développé grâce au framework `aiohttp` en utilisant Swagger. Cette technologie assure deux fonctionnalités primordiales :

1. Avoir un aperçu de tous les *endpoints* développés.
2. Pouvoir effectuer des tests des différentes fonctionnalités développées fournies par le serveur, permettant ainsi de déceler des potentielles erreurs de programmation.

La Figure 4.1 montre un aperçu de l'interface de l'application web développée. Les deux sections contiennent les détails des différents *endpoints* associés aux chemins du serveur.

FIGURE 4.1. – Page d'accueil de l'interface utilisateur Swagger

Lorsqu'une section est choisie, par exemple *Sorties*, s'affiche alors un premier niveau de détail avec les différents *endpoints*. Chaque *endpoint* a un code couleur bien précis afin de se différencier. Les requêtes GET sont en bleu, HEAD en jaune, POST en vert, DELETE en rouge et PATCH en orange. Les méthodes HTTP HEAD sont affichées après chaque requête GET, car elles permettent d'économiser de la bande passante lorsque les méthodes GET procèdent au téléchargement d'une ressource volumineuse [29]. À côté de chaque requête, se trouve un descriptif de son utilisation. La section *Rentrées* se base sur la même architecture que la section *Sorties*.

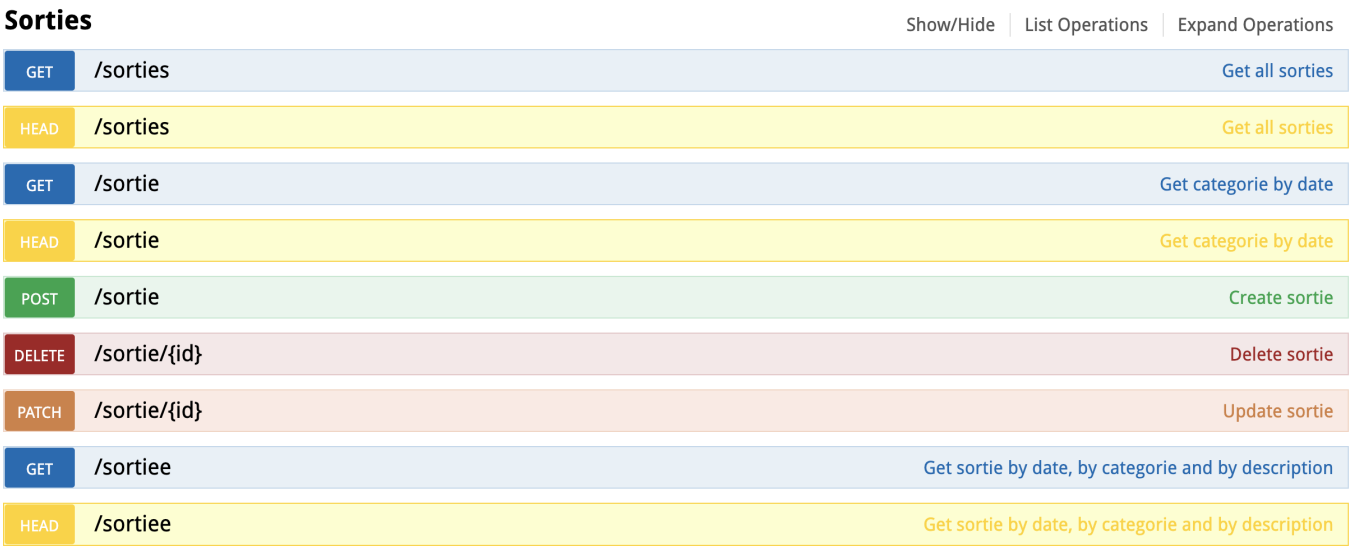

FIGURE 4.2. – Liste de tous les *endpoints* de la section Sorties

Pour afficher le second niveau de détail d'un certain *endpoint*, il suffit de sélectionner celui qui est souhaité. Tous les *endpoints* sont séparés en deux parties distinctes.

Dans la première partie d'un *endpoint*, se trouve une description de celui-ci avec divers paramètres obligatoires ou non (`path` ou `query`) afin d'effectuer la requête en question, et un corps de la demande (`request body`). Un corps de la demande doit se trouver dans une requête POST ou PATCH. En effet, pour créer ou modifier une ressource, il faut impérativement déterminer son contenu. Si un ou plusieurs paramètres sont présents, il est nécessaire de donner une définition pour chaque paramètre. Par exemple, il faut spécifier le type, tel que entier, chaîne de caractère, etc., ainsi que le genre, `path` ou `query`. Les paramètres `path` sont des parties changeantes d'une URL. Généralement, ils sont employés pour pointer vers une ressource en particulier, comme un identifiant. Une URL peut avoir différents paramètres de chemin d'accès, chacun entouré par des accolades. Les paramètres `query` sont le type de paramètre le plus fréquent et ils apparaissent à la fin d'une URL, après un point d'interrogation. Ces paramètres de requête ne sont pas obligatoires.

Dans la deuxième partie d'un *endpoint*, sont listées les différentes réponses possibles, avec chacune une description, ainsi que le code d'état HTTP assimilé. Ces codes d'état HTTP sont composés de trois chiffres. Les codes d'état commençant par 1 indiquent une réponse informatives, ceux commençant par 2 une réponse de succès, par 3 un message de redirection, par 4 une erreur du client et, pour terminer, par 5 une erreur du serveur [30]. Pour cette application, les codes 200, 201, 400 et 404 ont été utilisés. Le code 200 signifie que la requête a réussi. De plus, la signification du succès avec le code 200 diffère selon la méthode HTTP utilisée. Le code 201 indique que la requête a réussi et qu'une nouvelle ressource a été créée. C'est notamment le cas pour une méthode POST. Les codes 400 et 404 indiquent respectivement que la requête envoyée au serveur n'a pas été comprise, à cause d'une syntaxe invalide, ou que le serveur n'a pas trouvé la ressource exigée. Ce dernier code d'état HTTP est très populaire grâce à son apparition courante sur le web.

Par exemple, la Figure 4.3 montre le deuxième niveau de détail de la première méthode GET pour les sorties d'argent. En premier lieu, il y une description de ce qui se passe

lorsque la méthode est appelée sur le *endpoint*, en l'occurrence le retour de toutes les sorties d'argent enregistrées dans la base de données.

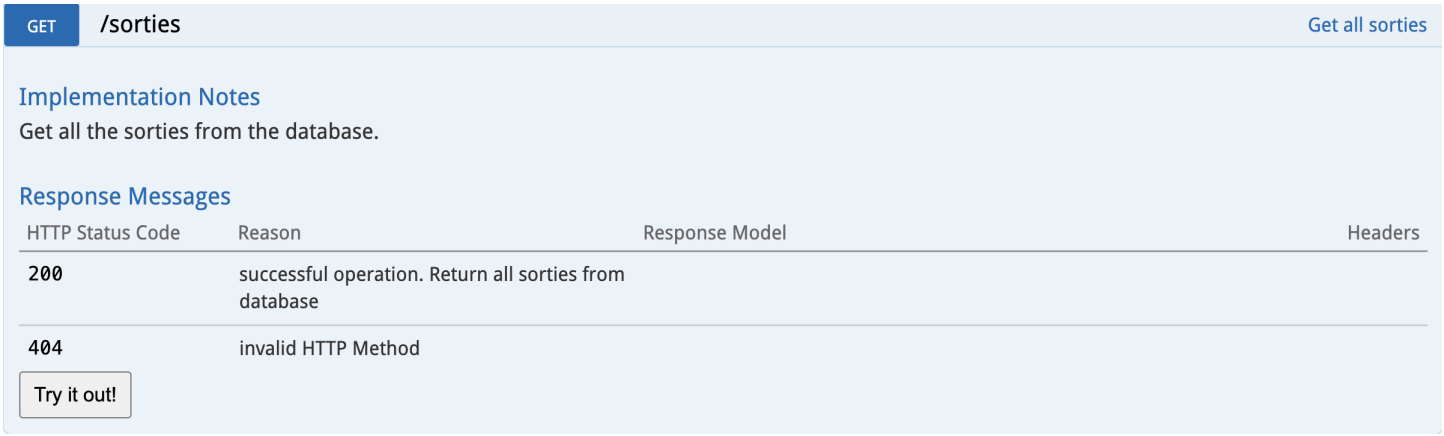

FIGURE 4.3. – Requête GET retournant toutes les sorties

Dans la deuxième partie, sont affichées les réponses de la méthode GET. En appuyant sur le bouton *Try it out !*, tout d'abord est affichée la ligne de commande *Curl* qui permet de récupérer le contenu d'une ressource informatique, ensuite l'URL de la requête, puis la réponse avec toutes les sorties d'argent enregistrées dans la base de données, et enfin le code d'état, 200 si c'est un succès ou 404 s'il y a un problème. Si le code d'état HTTP 200 est envoyé, les sorties d'argent sont structurées à l'aide de schémas et sont composées d'un identifiant, d'une catégorie, d'un montant, d'une description et d'une date.

Curl

```
curl -X GET --header 'Accept: application/json' 'http://127.0.0.1:8080/sorties'
```

Request URL

```
http://127.0.0.1:8080/sorties
```

Response Body


```
[
  {
    "_id": 1,
    "Categorie": "Abonnement",
    "Montant": 14,
    "Description": "Netflix",
    "Date": "2022-01-01"
  },
  {
    "_id": 2,
    "Categorie": "Assurance",
    "Montant": 450,
    "Description": "Maladie",
    "Date": "2022-01-07"
  },
  {
    "_id": 3,
    "Categorie": "Banque",
    "Montant": 100,
    "Description": "Compte épargne",
```


Response Code

```
200
```

Response Headers


```
{
  "content-length": "3393",
  "content-type": "application/json; charset=utf-8",
  "date": "Wed, 11 May 2022 16:52:02 GMT",
  "server": "Python/3.9 aiohttp/3.7.4.post0"
}
```


FIGURE 4.4. – Réponse de la requête GET retournant toutes les sorties

La page Swagger UI de cette application se trouve à l'adresse `http://127.0.0.1:8080/api/doc/`, une fois que le serveur a été au préalable démarré.

4.4. MongoDB

Lorsqu'on parle de programmation d'un serveur, il est primordial d'y associer une base de données. Il existe de nombreux Système de Gestion de Base de Données (SGDB), tels que MongoDB, PostgreSQL ou MySQL. Parmi ces différents systèmes, une distinction essentielle est le langage de programmation utilisé. En effet, des systèmes utilisent Structured Query Language (SQL) comme langage informatique, ou `query language`, permettant d'ajouter, de supprimer, de modifier ou de rechercher des données dans des bases de données dites relationnelles. Les autres systèmes emploient leurs propres langages, comme

c'est le cas pour MongoDB qui utilise MongoDB Query Language (MQL) [31]. Ces autres systèmes de gestion de bases de données qui n'utilisent pas SQL sont appelés des systèmes de gestion bases de données Non-Structured Query Language (NoSQL), pour langage de requête non structuré [32]. Le système de gestion de bases de données choisi pour cette application est MongoDB, créé vers le milieu des années 2000, et le framework utilisé avec est Python, grâce à Motor. Un serveur asynchrone développé en Python se connecte à MongoDB avec l'aide d'un pilote officiel (driver), comme Motor. L'avantage d'utiliser le langage de programmation Python avec MongoDB est que ce dernier conserve les données en format JSON proche du format du dictionnaire employé par le langage Python [33]. Les données stockées par MongoDB sont aussi appelées des documents.

Afin d'effectuer une liaison entre le serveur et la base de données, il faut spécifier l'URL de connexion à la base de données dans le fichier `Database.py`. Cela crée ce qu'on appelle un client. Ensuite, il est nécessaire de créer une base de données contenant deux collections, une pour les rentrées et une pour les sorties d'argent. Ces deux collections stockent par la suite toutes les données. Pour créer les deux collections, il faut transmettre leurs noms à la base de données.

```
1 client = motor.motor_asyncio.AsyncIOMotorClient('mongodb+srv://Luca:<password>cluster1
      .l6wxv.mongodb.net/Budget?retryWrites=true&w=majority&ssl=true&ssl_cert_reqs=
      CERT_NONE')
2
3 db = client['Budget']
4
5 collection_Entrees = db['Entrees']
6
7 collection_Sorties = db['Sorties']
```

List. 4.9 – Création d'une base de données et de deux collections

Il existe de nombreuses méthodes pour effectuer les opérations CRUD propres à MongoDB [34]. Afin d'ajouter des documents et donc d'en créer, il convient d'utiliser, soit la méthode `insertOne(document)` qui permet d'ajouter un document, soit la méthode `insertMany(documents)` pour en ajouter plusieurs d'un coup. Ces deux méthodes prennent en paramètre un ou plusieurs documents à ajouter à la collection. La première méthode a été privilégiée. Pour lire les documents déjà existants dans la base de données, il faut employer la méthode `find(criteria, projection)`. Pour cette méthode, il est également possible d'ajouter des filtres de requête avec la recherche d'expressions régulières, ou RegEx, comme décrit dans la première section de ce chapitre, ou d'ajouter des opérateurs de comparaison qui permettent d'identifier les documents qu'on souhaite renvoyer. Les opérateurs de comparaison employés sont `$gte` et `$lte`, également décrits dans la première section de ce chapitre. Ils sont présents lorsqu'il s'agit d'afficher des rentrées ou des sorties d'argent par date. Ensuite, pour modifier un ou des documents, les méthodes disponibles sont `updateOne(filter, action)` ou `updateMany(filter, action)`. La première méthode a également été privilégiée. De plus, il est possible d'ajouter des filtres ou des opérateurs de comparaison pour identifier les documents que l'on souhaite mettre à jour. Ces filtres et opérateurs de comparaison emploient la même syntaxe que pour la méthode de lecture des documents déjà existants dans la base de données. Enfin, pour supprimer un ou des documents, les méthodes sont `deleteOne(filter)` ou `deleteMany(filter)`. La première méthode a été privilégiée. De plus, comme pour lire ou modifier un document, il est possible d'appliquer des filtres et des opérateurs de comparaison afin de supprimer un document en particulier.

5
Programmation du client

5.1. Présentation générale

Le client est l'interface utilisateur (UI) de cette application. Il permet d'envoyer des requêtes au serveur et de recevoir des réponses de celui-ci. La technologie employée pour développer le client est React, qui est un framework JavaScript, au même titre que Vue.js et AngularJS. Le code source du client se trouve sur le repository GitHub [1]. Seul le dossier `node_modules` ne s'y trouve pas, à cause de sa trop grande taille en terme de stockage. Les fichiers `package.json` et `package-lock.json` sont en quelque sorte similaires à ce dossier. En effet, le dossier `node_modules` comprend tous les modules dont le client dépend. Le fichier `package.json` contient l'ensemble des modules essentiels au client. Le fichier `package-lock.json` est généré automatiquement à chaque modification de l'arborescence du dossier `node_modules` ou du fichier `package.json`.

Le client échange avec le serveur grâce à des requêtes et cela est possible avec l'aide de la librairie Axios. Le fonctionnement de cette librairie est développé dans la troisième section de ce chapitre. Pour afficher des données reçues du serveur sous la forme de différents diagrammes, le client a besoin de la librairie Chart.js. Ce point est développé dans la dernière section de ce chapitre.

L'architecture principale du client se trouve dans le dossier `src` et se compose d'un dossier `components` où se trouvent les fichiers codes ou composants React, d'un dossier `images` contenant une image utilisée dans le client, et de différents fichiers qui sont décrits par la suite. Le dossier `components` contient tous les fichiers `.js`, correspondant chacun à une page du client, à l'exception du fichier `Navigation.js`, qui permet de créer une barre de navigation pour l'application, et du fichier `FooterComponent.js`, qui sert à afficher un message en bas de toutes les pages du client. Dans le dossier `components`, il y a

[1] `https://github.com/Manga3000/Gestionnaire-de-budget`

deux fichiers, `Rentre.js` et `Sortie.js`, qui permettent la création, la modification, la suppression et la visualisation des rentrées et des sorties d'argent. C'est également dans ces deux fichiers que les différentes requêtes sont envoyées vers le client et que les réponses sont réceptionnées. Toujours concernant l'architecture du client, le fichier `App.js`, qui ne se trouve pas dans le dossier `components`, mais à sa source, est le composant principal de React et il contient tous les autres composants React se trouvant dans le dossier `components`. Le fichier CSS, `App.css`, contient les styles appliqués aux composants. Tous les autres fichiers présents dans le dossier `src` sont secondaires à la création d'un client mais veillent à son bon fonctionnement.

5.2. React

React est une librairie JavaScript développée par Facebook depuis 2013 et elle permet la création facile d'applications web avec une ou plusieurs pages [35]. Il est également possible de créer des applications mobiles avec React grâce à *React Native*. React est le framework le plus utilisé pour le développement web [36].

React utilise des composants autonomes permettant de créer des interfaces d'utilisateurs simples ou complexes, selon le type d'application souhaitée. Un composant React possède des attributs et une logique spécifiques. Un composant est également un morceau de code indépendant et réutilisable. Aussi, il a le même objectif que les méthodes JavaScript, mais fonctionne de manière isolée et renvoie du Hypertext Markup Language (HTML). Un composant maintient un état local, grâce à `this.state`, et peut également recevoir des données, grâce à `this.props`. En utilisant React, l'ajout, la modification et la suppression de composants sont facilités. Une méthode `render()` est aussi implémentée dans les composants React et elle permet de prendre des données en input et de retourner en output ce qui doit être affiché. Par ailleurs, l'architecture classique d'un composant React commence par une partie écrite en JavaScript, pour le script, une partie écrite en HTML, pour le template, et une partie en CSS, pour le style de la page. Il est également possible de dédier une page à part à du style CSS, permettant ainsi d'appliquer un même style à plusieurs composants, comme c'est le cas pour le fichier `App.css` [37].

Le composant qui est décrit par la suite représente la page d'accueil du client. Tout d'abord, se trouve la partie script du composant, écrite en JavaScript. Le script décrit et initialise les différentes propriétés et le comportement du composant de la page d'accueil. Tout d'abord, différents éléments sont importés dans ce composant, comme la librairie Axios. Ensuite, il est nécessaire d'utiliser la méthode `constructor()`, qui permet de créer et d'initialiser un objet, ainsi qu'une instance de la classe [38]. Dans le Listing 5.1, les attributs `chartData`, `start`, `end`, `totalrentre` et `totalsortie` sont des attributs initialement vides. Ces attributs sont définis dans l'objet `state`. Des attributs définis dans cet objet peuvent contenir des informations qui changent au fil du temps du composant, et tout changement dans cet attribut modifie le comportement du composant. Par la suite, vient l'emploi de la méthode `componentDidMount()`. Cette méthode est appelée par React, et permet de récupérer les données reçues depuis l'API. Cette méthode permet également de changer l'état de l'application une fois que tout le composant est rendu correctement, en appelant les méthodes définies à l'intérieur de celle-ci [39]. C'est le cas avec les méthodes `getRentre()` et `getSortie()`. Ensuite viennent les méthodes `getRentre()` et `getSortie()`. Ces deux méthodes permettent d'envoyer des requêtes

vers le serveur et de recevoir les réponses de celui-ci. Tout d'abord, pour les deux méthodes, la date du premier jour du mois en cours et la date actuelle sont définies, en reprenant les attributs `start` et `end`, prédéfinis dans le `constructor()`. Dans l'instruction `if`, les dates choisies par l'utilisateur dans le client sont stockées dans les variables `this.state.start` et `this.state.end`. Ensuite, viennent les différentes requêtes au serveur grâce aux fonctions GET de la librairie Axios. Les requêtes sont d'abord stockées dans la variable `coin`, puis dans la variable `this.state.totalrentre`. Pour terminer avec la partie script, la méthode `set.State()` annonce à React que le composant a besoin d'être rafraîchi une fois l'état de la page mis à jour. C'est de cette façon que l'interface utilisateur est mise à jour en réaction à des réponses réseau ou à des événements. Dans l'exemple, la méthode `set.State()` indique que les variables `this.state.totalrentre` et `this.state.totalsortie` sont les événements qui nécessitent une mise à jour de l'interface utilisateur. Cela permet d'afficher sur la page d'accueil le total des rentrées et des sorties d'argent de l'utilisateur à une période prédéfinie.

```
export class Accueil extends Component {
    constructor(){
        super();
        this.state = {
            chartData:{},
            start: '',
            end: '',
            totalrentre: null,
            totalsortie: null,
        };
    }

    componentDidMount() {
        this.getRentre();
        this.getSortie();
    }

    getRentre() {
        axios.get('http://127.0.0.1:8080/entre', {params: {from: this.state.start, to:
            this.state.end}})
        .then(res => {
            const coin = res.data;
            this.state.totalrentre = coin.Total;

            this.setState({
            chartData: {
                data: this.state.totalrentre,
            }
            });
        });
    }

    getSortie() {
        axios.get('http://127.0.0.1:8080/sortie', {params: {from: this.state.start, to:
             this.state.end}})
        .then(res => {
            const coin = res.data;
            this.state.totalsortie = coin.Total;

            this.setState({
                chartData: {
```

```
40                  data: this.state.totalsortie,
41              }
42          });
43      });
44  }
45 }
```

List. 5.1 – Script du fichier `Accueil.js`

La dernière partie du composant pour la page d'accueil est construite en majorité avec HTML, pour le template, et avec CSS, pour le style. Le template définit comment est construite la page, tandis que le style apporte une touche plus esthétique à la page.

Du côté du template, les attributs globaux tels que `id` ou `defaultValue`, définis dans la balise HTML `input`, reprennent des éléments définis dans la partie écrite en Javascript. La page commence par deux sélecteurs de dates, afin que l'utilisateur puisse choisir pour quelle période il souhaite visionner son budget. Ces deux sélecteurs de dates sont définis à la ligne 5, par les attributs de types `type="date"`. Ils sont tous les deux construits avec HTML, CSS et des éléments définis dans le script, comme `defaultValue=this.state.start` et `defaultValue=this.state.end`, qui reprennent la date du début et la date de la fin de la période choisie. Ensuite, un bouton permet à l'utilisateur d'enregistrer les deux dates choisies dans les sélecteurs de dates. Ce bouton est relié aux deux méthodes `getRentre()` et `getSortie()` par l'événement `onClick`. Puis, le module `startIcon` permet d'afficher un logo de recherche. Ce module est issu de la librairie MUI, qui est le framework React UI le plus connu au monde. Cette librairie est notamment utilisée par Spotify, Amazon, Netflix ou encore la NASA. Par la suite, la page se compose de trois encadrés, définis aux lignes 9 à 11. Ces trois encadrés sont aussi composé avec HTML, CSS et des éléments définis dans le script. Pour le dernier encadré, une instruction `if` est employée. Cette instruction définit la couleur du dernier encadré si la différence entre les rentrées et les sorties d'argent est plus grande que 0. Si le résultat est plus grand que 0, le style applique une couleur verte à l'encadré, autrement il lui applique une couleur rouge. Pour terminer, sur cette page d'accueil se trouve la barre de navigation présente dans toutes les pages du client ainsi qu'un message en bas de page.

```
1  render(){
2          return (
3              <div className="margin background">
4                  <div className="titre1">
5                  Du:<input className="search mt-5 mr-4" id="start" type="date"
                       defaultValue={this.state.start}/> Au:<input className="search mr-4"
                       id="end" type="date" defaultValue={this.state.end}/>
6                  <Button className="button" variant="contained" color="info" onClick={()
                       => { this.getRentre(); this.getSortie();}} startIcon={<SearchIcon
                       />}>Rechercher</Button>
7                  </div>
8                  <div className="flexbox-container">
9                      <h5 className="boxtotalrentre">Total des rentrees {this.state.
                           totalrentre} CHF</h5> <h1 className="maths">-</h1>
10                     <h5 className="boxtotalsortie">Total des sorties {this.state.
                           totalsortie} CHF</h5> <h1 className="maths">=</h1>
11                     <h5 className="boxtotal" style={{color: this.state.totalrentre -
                           this.state.totalsortie > 0 ? "green" : "red", borderColor: this.
                           state.totalrentre - this.state.totalsortie > 0 ? "green" : "red"
                           , backgroundColor: this.state.totalrentre - this.state.
```

```
                        totalsortie > 0 ? "#CCFFCC" : "#FFCCCC"}}>Total du budget {this.
                        state.totalrentre - this.state.totalsortie} CHF</h5>
12                  </div>
13              </div>
14          )
15      }
16  }
```

List. 5.2 – Template du fichier `Accueil.js`

Au niveau du style, la propriété `className` permet d'utiliser du CSS pour toute la balise. Par exemple, la deuxième propriété `className` applique le style `titre1`, qui est implémenté dans le fichier `App.css`. La syntaxe d'un style CSS nécessite de définir si le style est appliqué à une balise HTML de manière globale dans le client, ou si l'on souhaite appliquer un style seulement à certaines balises. Pour la création du style, entre deux accolades, il faut définir la ou les propriétés à modifier selon la ou les valeurs que doit prendre la balise. Par exemple, les différentes propriétés modifiées pour le style `titre1` sont les suivantes. Tout d'abord, il est spécifié qu'une certaine marge à gauche est nécessaire. Ensuite, la taille et la couleur de la police d'écriture du titre est ajustée afin qu'elle soit plus lisible sur la page d'accueil.

```
1  .titre1 {
2    margin-left: 20px;
3    font-size: 20px;
4    font-weight: bold;
5    color: white;
6  }
```

List. 5.3 – Style `titre1`

5.3. Axios

Axios est une librairie JavaScript qui permet de réaliser des requêtes HTTP vers des *endpoints* d'une API [40]. Dans chaque fichier `.js`, se trouvent plusieurs méthodes contenant différentes requêtes axios. Seulement les deux fichiers `FooterComponent.js` et `Navigation.js` ne contiennent pas de méthodes utilisant la librairie Axios. Dans les fichiers `Rentre.js` et `Sortie.js`, il y a 5 fonctions Axios différentes :

- La première retourne toutes les rentrées et sorties d'argent (GET).
- La deuxième permet de créer une rentrée ou une sortie d'argent (POST).
- La troisième permet de supprimer une rentrée ou une sortie d'argent (DELETE).
- La quatrième retourne toutes les rentrées et sorties d'argent selon leur date, leur catégorie ou leur description (GET).
- La cinquième permet de mettre à jour une rentrée ou une sortie d'argent (PATCH).

Ces 5 fonctions sont toutes asynchrones et envoient des requêtes GET, POST, DELETE ou PATCH au serveur.

Pour la première fonction Axios, il faut tout d'abord définir la méthode HTTP à utiliser, dans ce cas une méthode GET, puis passer en paramètre l'URL permettant à la fonction de faire une requête vers l'API. La suite de la fonction permet, en premier lieu, de retourner

les données en cas de réussite de la requête, ou de retourner un message d'erreur en cas d'échec de la requête. La fonction utilisant la méthode HTTP DELETE, permettant de supprimer des données, est construite sur la même architecture que cette première fonction.

```
componentDidMount = async () => {
    await axios.get('http://127.0.0.1:8080/sorties')
    .then(response => {
        this.setState({posts: response.data});
    })
    .catch(error => {
        console.log("Il y a une erreur", error)
    })
}
```

List. 5.4 – Méthode GET permettant de retourner toutes les sorties d'argents présentes dans la base de données

```
deleteSortie = async (sortieid) => {
    if(window.confirm('Etes-vous sur de vouloir supprimer cette sortie ?')) {
        await axios.delete('http://127.0.0.1:8080/sortie/' + sortieid)
        .then(response => response.data)
        .then((data) => {
        this.componentDidMount()
        })
        .catch(function (error) {
        console.log(error);
        });
    }
}
```

List. 5.5 – Méthode DELETE permettant de supprimer une sortie d'argent présente dans la base de données

Pour la fonction utilisant la méthode HTTP POST, permettant d'ajouter des données, il est nécessaire de passer en paramètre tous les éléments fondamentaux pour la création d'une rentrée ou d'une sortie d'argent, à savoir, la catégorie, le montant, la description, ainsi que la date, en plus de l'URL permettant d'effectuer une requête à l'API. La quatrième et cinquième fonction sont construites sur la même architecture que cette fonction en utilisant les méthode HTTP GET et PATCH, et en passant des données supplémentaires en paramètre.

```
sendHandler = async (e) => {
    await axios.post('http://127.0.0.1:8080/sortie', newSortie)
        .then(response => response.data)
        .then((data) => {
            this.componentDidMount();
        })
        .catch(error => {
            console.log(error)
        })
}
```

List. 5.6 – Méthode POST permettant de créer une sortie d'argent dans la base de données

```
getSortieDate = async () => {
    await axios.get('http://127.0.0.1:8080/sortiee', {params: {from: this.state.start,
         to: this.state.end, categorie: this.state.categorie, description: this.state.
        description}})
    .then(response => {
        this.setState({posts: response.data})
    })
    .catch(error => {
        console.log("Il y a une erreur", error)
    })
}
```

List. 5.7 – Méthode GET permettant de retourner une ou des sorties d'argent présentes dans la base de données selon les filtres appliqués

```
updatePost = async () => {
    await axios.patch('http://127.0.0.1:8080/sortie/' + this.state.sortieid, newSortie
        )
    .then(response => response.data)
    .then((data) => {
           this.componentDidMount();
    })
    .catch(function (error) {
        console.log(error)
}
```

List. 5.8 – Méthode PATCH permettant de modifier une sortie d'argent présente dans la base de données

Dans les fichiers contenant les diagrammes, il y a 2 fonctions Axios différentes qui utilisent des méthodes HTTP GET. La première retourne toutes les rentrées et sorties d'argent, regroupées par catégorie et par date, tandis que la deuxième retourne toutes les rentrées et sorties d'argent selon leur date, leur catégorie ou leur description. Dans le fichier `Accueil.js`, il y a également 2 fonctions Axios différentes. La première fonction retourne le total des rentrées d'argent selon une période choisie et la deuxième retourne le total des dépenses, également selon une période choisie [41].

5.4. Chart.js

Chart.js est une librairie JavaScript open source, créée en 2013, qui permet la visualisation de données, et qui est similaire à Chartist et à Google Charts. C'est l'une des librairies graphiques JavaScript les plus populaires. Elle prend en charge 8 types de graphiques différents, par exemple des barres, des tartes ou des lignes, et tous ces graphiques sont réactifs. Chart.js peut être utilisée dans une application JavaScript très basique ou avec un framework tel que React.js [42].

Pour qu'un gestionnaire de budget soit clair et lisible, il est primordial de bien choisir quel type de diagramme afficher, ainsi que les données à présenter à l'utilisateur. Comme vu dans le chapitre concernant d'autres gestionnaires de budget, les graphiques qui répondent au mieux à cette attente sont les tartes, les barres et les lignes.

Les différents éléments à fournir à Chart.js afin de créer un graphique sont :

— Définir où le graphique est dessiné.
— Définir quel type de graphique est dessiné.
— Définir des données, des étiquettes et d'autres options.

Les fichiers où Chart.js a été utilisé pour la création de graphiques sont `RentrePie.js`, `RentreBar.js`, `RentreLine.js`, `SortiePie.js`, `SortieBar.js` et `SortieLine.js`. Ces fichiers comprennent la même architecture, avec chacun des spécificités selon le type de diagramme employé.

Pour spécifier où les diagrammes sont dessinés dans l'application, le type de diagramme et diverses options, cela est défini dans la partie écrite en HTML, le template. Par exemple, dans le fichier `RentrePie.js`, en utilisant la balise `<Pie/>`, on crée un diagramme circulaire. Ensuite, la variable `data` récupère les données reçues du serveur pour les afficher dans le diagramme. Puis, différentes options sont également définies dans le template, comme la couleur et la taille de la police utilisée autour des graphiques, la personnalisation des axes des abscisses et des ordonnés, ou la durée de l'animation de l'affichage du diagramme. Selon le type de diagramme choisi, comme les barres ou les lignes, il est intéressant de fixer que l'axe des ordonnés commence à 0. L'échelle de l'axe des ordonnés est établie automatiquement par Chart.js. Cependant, il est également possible de la personnaliser.

```
<Pie
data={this.state.chartData}
    options={{
        legend: {
            labels: {
                fontSize: 15,
                fontStyle: 'bold'
            },
        },
                fontColor: [...],
                fontFamily: 'Helvetica',
                fontStyle: 'bolder',
            },
        },
    }}
/>
```

List. 5.9 – Balise `<Pie/>` avec différentes options

Pour spécifier les données qui sont affichées sur les diagrammes, cela est défini dans la partie écrite en JavaScript, le script. La méthode Axios récupère les données stockées dans la base de données en effectuant des requêtes et les affiche dans le diagramme. Dans la méthode `set.State()`, il faut définir la légende du graphique employé dans la variable `labels`, les données récupérées du serveur dans la variable `data` ainsi que les couleurs des secteurs et des bords des secteurs utilisées pour le diagramme.

```
this.setState({
    chartData: {
    labels: [...],
    datasets: [
                {
```

```
6                   data: this.state.m,
7                   backgroundColor: [...],
8                   borderColor: [...],
9               }
10          ]
11      }
12  });
```

List. 5.10 – Déclaration de la légende du digramme ainsi que des couleurs utilisées pour le diagramme

6
Conclusion

6.1. Résultats

Ce travail a consisté à créer une application de gestion de budget ainsi qu'à rédiger un rapport de description et d'analyse de l'application créée. Tout d'abord, il a fallu étudier et évaluer les besoins ainsi que les fonctionnalités nécessaires à la réalisation du gestionnaire de budget. Une fois cette analyse effectuée, il a été primordial de considérer la structure des données, leurs relations entre elles, ainsi que de déterminer toutes les technologies à utiliser. Le client a été développé en utilisant React et la base de données choisie, MongoDB, est une base de données NoSQL. Pour établir le lien entre le client et la base de données, a été développé un serveur dans le langage de programmation Python et a été respectée l'architecture REST.

Tous les objectifs fixés au préalable ont été atteints. En effet, l'application de gestion de budget permet à l'utilisateur de créer, modifier, supprimer et lister la globalité de ses rentrées et sorties d'argent. Par ailleurs, il a également la possibilité de les filtrer par catégorie, par description et par date. A relever également que l'application propose plusieurs types de diagrammes différents, tant pour les rentrées que pour les sorties d'argent, afin d'assurer à l'utilisateur un meilleur visuel de toutes ses données. Ainsi, toutes les caractéristiques prévues ont été développés dans l'application de gestion de budget et son fonctionnement est assuré.

6.2. Améliorations possibles

L'application de gestion de budget créée constitue une base de départ solide à partir de laquelle plusieurs développements futurs sont possibles. Trois axes différents seraient à privilégier.

Tout d'abord, l'authentification de plusieurs utilisateurs serait un aspect intéressant à développer. En effet, cette fonctionnalité permettrait à plusieurs personnes d'utiliser l'application et de constituer des groupes, comme par exemple le groupe des membres d'une famille, des colocataires, ou des collègues de travail.

En deuxième lieu, étant donné qu'il s'agit d'une application web, la sécurité des données et l'accessibilité sécurisée à l'application sont très importantes et leur développement apporterait à l'application une valeur ajoutée certaine. En effet, actuellement, les cyberattaques se multiplient toujours davantage et il serait essentiel de garantir la sécurité

de l'utilisateur et des informations qu'il transmet à l'application. Par exemple, lorsque l'utilisateur se connecte à l'application, une authentification à deux facteurs serait un moyen pour garantir la sécurité des données et l'accessibilité sécurisée à l'application. Par ailleurs, la protection des données est un sujet extrêmement important et sensible qui est constamment au centre des débats actuels.

Enfin, un troisième axe intéressant à développer serait la possibilité d'exporter les données sauvegardées dans l'application. En effet, l'utilisateur pourrait avoir besoin d'exporter ses données afin par exemple de les imprimer pour en garder une trace papier, de les partager dans un document Excel ou de les transmettre à une tierce personne. Pour tous ces motifs, l'exportation des données contenues dans l'application pourrait constituer un aspect intéressant à développer.

A

Acronymes courants

API Application Programming Interface
CORS Cross-Origin Resource Sharing
CRUD Create, Read, Update, Delete
HTML Hypertext Markup Language
HTTP Hypertext Transfer Protocol
JSON JavaScript Object Notation
MQL MongoDB Query Language
NoSQL Non-Structured Query Language
SGDB Système de Gestion de Base de Données
SOAP Simple Object Access Protocol
SQL Structured Query Language
REST Representational State Transfer
UML Unified Modeling Language
URL Uniform Resource Locator

B

Licence de la documentation

Bibliographie

[1] "Mint official web site." https://mint.intuit.com/ (dernière consultation le Mars 6, 2022). 4

[2] "Mint on App Store." https://apps.apple.com/us/app/mint-budget-planner-tracker/id300238550 (dernière consultation le Mars 6, 2022). 4

[3] "Wikipedia web page of Mint." https://en.wikipedia.org/wiki/Intuit_Mint (dernière consultation le Mars 6, 2022). 4

[4] "Mint Budgeting App Review 2021 (Brand NEW Features!)." https://www.youtube.com/watch?v=WG2L1PQgLwg&t=111s (dernière consultation le Mars 6, 2022).

[5] "BudgetCH official web site." https://budgetberatung.ch/budget-app (dernière consultation le Mars 3, 2022). 11

[6] "Spendee official web site." https://www.spendee.com/ (dernière consultation le Mars 4, 2022). 16

[7] "SPENDEE." https://www.lesnumeriques.com/telecharger/spendee-31940 (dernière consultation le Mars 4, 2022). 16

[8] "How to start with Spendee." https://medium.com/spendee/how-to-start-with-spendee-d7c4e96052e5 (dernière consultation le Mars 4, 2022). 16

[9] "Buddy official web site." https://www.buddy.download/ (dernière consultation le Mars 9, 2022). 20

[10] "AIOHTTP Framework." https://techdocs.broadcom.com/us/en/ca-enterprise-software/it-operations-management/dx-apm-saas/SaaS/implementing-agents/python-agent/Python-Agent-Extensions/AIOHTTP-Framework.html (dernière consultation le Avril 4, 2022). 44

[11] "An Intro to aiohttp." https://dzone.com/articles/an-intro-to-aiohttp#:~:text=One%20such%20package%20is%20aiohttp%20which%20is%20an%20HTTP%20client,Server%20WebSockets%20and%20Client%20WebSockets (dernière consultation le Avril 5, 2022). 44

[12] "Guide de la programmation asynchrone dans Python." https://espresso-jobs.com/conseils-carriere/guide-de-la-programmation-asynchrone-dans-python/ (dernière consultation le Avril 4, 2022). 44

[13] "Coroutines et tâches." https://docs.python.org/fr/3/library/asyncio-task.html (dernière consultation le Avril 5, 2022). 44

[14] "json — Encodage et décodage JSON." https://docs.python.org/fr/3/library/json.html (dernière consultation le Avril 5, 2022). 45

[15] "CORS Explained + Enable in Python Projects." https://dev.to/ninahwang/cors-explained-enable-in-python-projects-1i96 (dernière consultation le Avril 5, 2022). 45

[16] "Aiohttp_cors." https://github.com/aio-libs/aiohttp-cors (dernière consultation le Avril 5, 2022). 45

[17] "datetime — Types de base pour la date et l'heures." https://docs.python.org/fr/3/library/datetime.html (dernière consultation le Avril 5, 2022). 45

[18] "Python strftime()." https://www.programiz.com/python-programming/datetime/strftime (dernière consultation le Avril 5, 2022). 46

[19] "L'architecture REST expliquée en 5 règles." https://blog.nicolashachet.com/developpement-php/larchitecture-rest-expliquee-en-5-regles/ (dernière consultation le Mars 30, 2022). 49

[20] "Comment fonctionne un Web Service - Qu'est-ce qu'un Web Service ?." https://www.oracle.com/fr/cloud/definition-web-service/ (dernière consultation le Mars 30, 2022). 49

[21] "Learn REST : A RESTful Tutorial." https://www.restapitutorial.com/index.html (dernière consultation le Mars 30, 2022). 50

[22] "What is REST." https://restfulapi.net/ (dernière consultation le Mars 30, 2022). 50

[23] "Representational State Transfer (REST)." https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (dernière consultation le Mars 30, 2022). 50

[24] "Méthodes de requête HTTP." https://developer.mozilla.org/fr/docs/Web/HTTP/Methods (dernière consultation le Mars 30, 2022). 50

[25] "Using HTTP Methods for RESTful Services." https://www.restapitutorial.com/lessons/httpmethods.html (dernière consultation le Mars 30, 2022). 50

[26] "Une API, qu'est-ce que c'est ?." https://www.redhat.com/fr/topics/api/what-are-application-programming-interfaces (dernière consultation le Mars 31, 2022). 51

[27] "API Endpoints - What Are They ? Why Do They Matter ? ?." https://smartbear.com/learn/performance-monitoring/api-endpoints/ (dernière consultation le Mars 31, 2022). 51

[28] "Swagger." https://swagger.io/ (dernière consultation le Mars 31, 2022). 51

[29] "Méthode HTTP HEAD." https://developer.mozilla.org/fr/docs/Web/HTTP/Methods/HEAD (dernière consultation le Mars 31, 2022). 51

[30] "Codes de réponse HTTP." https://developer.mozilla.org/fr/docs/Web/HTTP/Status (dernière consultation le Avril 3, 2022). 52

[31] "MongoDB Supported Languages." https://www.mongodb.com/languages#:~:text=MongoDB%20is%20written%20in%20C,and%20arrange%20them%20in%20collections (dernière consultation le Mars 29, 2022). 55

[32] "Why Use MongoDB ? Advantages & Use Cases." https://studio3t.com/knowledge-base/articles/mongodb-advantages-use-cases/ (dernière consultation le Mars 29, 2022). 55

[33] "How to Use Python with MongoDB." https://www.mongodb.com/languages/python (dernière consultation le Mars 29, 2022). 55

[34] "MongoDB CRUD Operations." https://www.mongodb.com/docs/manual/crud/ (dernière consultation le Mars 29, 2022). 55

[35] "React." https://fr.reactjs.org/ (dernière consultation le Mars 16, 2022). 57

[36] "Most used web frameworks among developers worldwide, as of 2021." https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (dernière consultation le Mars 24, 2022). 57

[37] "React.Component." https://fr.reactjs.org/docs/react-component.html (dernière consultation le Mars 24, 2022). 57

[38] "Constructor." https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes/constructor (dernière consultation le Mars 24, 2022). 57

[39] "Understanding React componentDidMount and how it works." https://linguinecode.com/post/understanding-react-componentdidmount (dernière consultation le Mars 24, 2022). 57

[40] "How to make HTTP Requests with Axios and React.js." https://www.arubacloud.com/tutorial/how-to-make-http-requests-with-axios-and-reactjs.aspx (dernière consultation le Mars 18, 2022). 60

[41] "Client HTTP basé sur les promesses pour navigateur." https://axios-http.com/fr/ (dernière consultation le Mars 18, 2022). 62

[42] "Data visualization with Chart.js : An introduction." https://tobiasahlin.com/blog/introduction-to-chartjs/ (dernière consultation le Mars 16, 2022). 62

[43] "Free Documentation Licence (GNU FDL)." http://www.gnu.org/licenses/fdl.txt (dernière consultation le August 18, 2021). 68

UNI
FR

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Faculté des sciences économiques et sociales
Wirtschafts- und sozialwissenschaftliche Fakultät
Boulevard de Pérolles 90
CH-1700 Fribourg

DECLARATION

Par ma signature, j'atteste avoir rédigé personnellement ce travail écrit et n'avoir utilisé que les sources et moyens autorisés, et mentionné comme telles les citations et paraphrases.

J'ai pris connaissance de la décision du Conseil de Faculté du 09.11.2004 l'autorisant à me retirer le titre conféré sur la base du présent travail dans le cas où ma déclaration ne correspondrait pas à la vérité.

De plus, je déclare que ce travail ou des parties qui le composent, n'ont encore jamais été soumis sous cette forme comme épreuve à valider, conformément à la décision du Conseil de Faculté du 18.11.2013.

Givisiez, le 22 mai 2022

(signature)