DOSSIER DE PROJET

Nicholas ROSSETTE-CAZEL

SOMMAIRE

Liste des compétences du référentiel couvertes par le projet :

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique
- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d’une application web ou web mobile

RÉSUMÉ DE PROJET

Suite à mon stage chez Gerwin Software, j'ai participé à l'élaboration d'un projet web pour leurs applications mobiles Marketluck (jeux de loterie gratuit).

Ce projet répondait à deux besoins majeurs, d'une part pour l'équipe marketing de pouvoir gérer plus facilement les lots gagnés par les joueurs, et d'autre part au joueurs de pouvoir suivre l'évolution de leurs gains plus en détails.

Cette application web responsive est accessible depuis une page “gains” des applications natives sous forme de webview. Elle est accessible depuis toutes les applications Marketluck

La webview permet à l'utilisateur de prendre connaissance des documents à fournir relatif au traitement des lots gagnés ainsi que d'avoir une vue globale de l'état de ses lots (documents manquants, états du processus de traitement)
Au besoin il peut upload de nouveaux documents.

Enfin côté marketing on accède à une vue générale des gains non traité avec une possibilité de filtrer par etats et utilisateur.
Pour un gain donné il est possible de vérifier la validité des documents fourni et de procéder à son traitement / envoi au gagnant.

PRÉSENTATION DE L'ENTREPRISE

GERWIN SOFTWARE

Situé à Aix la Duranne (zone d'activité en proximité d'Aix en Provence) Gerwin Software est une petite entreprise de jeux vidéo sur Mobile.
https://www.gerwinsoftware.com

Les applications développées par Gerwin Software sont essentiellement des jeux de Quizz en ligne, leur dernière en date "Quiz Panic" se base sur le nombre de joueurs contrairement aux autres.
Quiz Panic joue sur ce nombre, en montrant les décisions des autres joueurs pour vous faire douter, certaines réponses sont piégés et vous empêche de changer votre choix etc …
On a aussi "Quiz Rush" qui se base sur une course, chaques mauvaise réponse vous fait trébucher et vous ralentissez.
Gerwin Software essaye d'innover tout en restant dans un domaine qu'il maîtrise, les jeux de quiz.

Depuis quelques années, Gerwin Software travail en étroite collaboration avec MarketLuck une filiale de GiFi, ils ont développé ensemble une application de jeux de loterie gratuite "Bravoloto" cette application est publié sous le nom MarketLuck et non Gerwin Software.
MarketLuck est composé essentiellement de commerciaux, Gerwin Software quant à eux comprennent tous les développeurs sur le projet.

Lors de mon stage en Février 2019, Gerwin Software sous le nom de MarketLuck a publié leur nouvelle version de loterie intitulé "Animoloto", c'est une loterie animée sous forme d'une course de chevaux mais avec des animaux.

Un peu comme pour les jeux de Quiz, Gerwin Software / MarketLuck essayent de sortir régulièrement de nouveau jeux de loterie en innovant sur certains aspects.

Les derniers jeux développés par Gerwin Software sont développés sur l'Engine Unity.

L'équipe GERWIN

Gerwin Software est une petite entreprise de seulement 4 employés, lors de mon stage il y avait en plus 2 développeurs en CDD de 6 mois sur Unity.
En tout on était 6 dans les locaux Gerwin, car Yann leur développeur back-end travaille à distance.

Pierre Germain : Directeur de l'entreprise, développeur de métier
Ronan Joncour : Manager d'équipe et de projet (Agile, Scrum)
Pierre Martin : Lead développeur sur Unity
Yannick Winling : Développeur back-end

SPÉCIFICATIONS FONCTIONNELLES DU PROJET

VIPRIZE

Suite à la sortie de Bravoloto, l'équipe de MarketLuck a commencé à perdre beaucoup de temps à traiter les lots d'argents et d'objets physiques.

Afin de simplifier le cycle de gestion des lots des jeux Marketluck, le besoin s'est posée de créer un nouveau service proposant :

- au gagnant de prendre connaissance des pièces à fournir depuis l'application (RIB/IBAN, pièce d'identité)
- au gagnant de fournir les pièces nécessaire depuis l'application
- au gagnant de suivre l'état du traitement de son lot depuis l'application
- au gestionnaire de suivre la liste synthétique des virements et commandes à réaliser et de mettre à jour leur état depuis une interface d'administration
- au gestionnaire de consulter l'état des lots d'un joueur précis

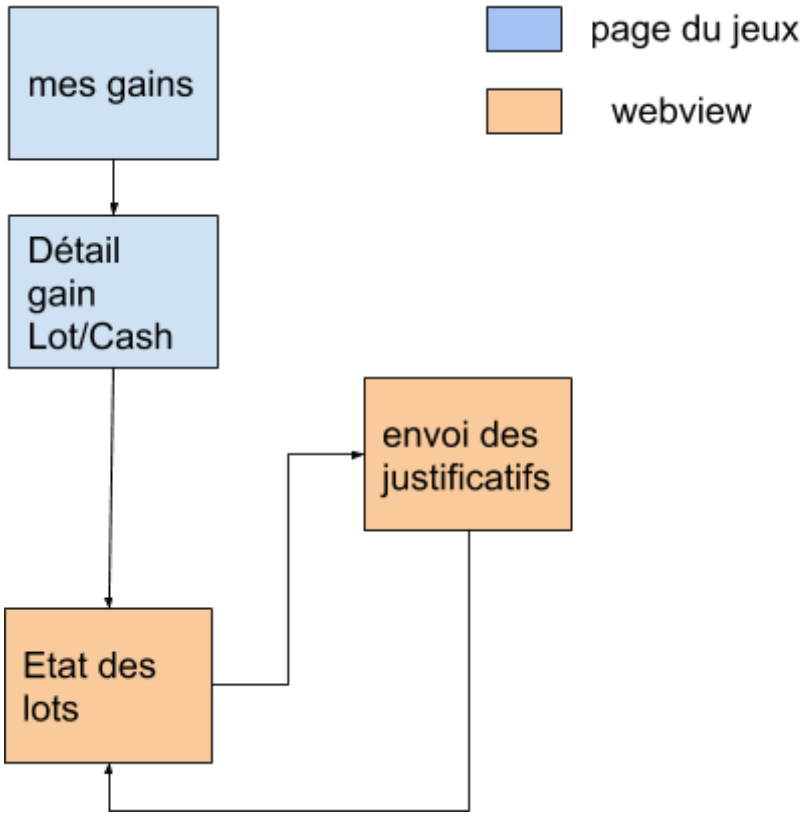
Pour répondre à ce besoin il a été décidé,

Côté utilisateur :
Créer une application web responsive permettant à l'utilisateur de soumettre les pièces justificatives et faire le suivi depuis l'application. Cette application doit être disponible depuis la page "mes gains" sous la forme d'une WebView.
Cette WebView est utilisable par toutes les application MarketLuck.

Côté admin :
Intégrer au site web d'administrateur déjà existant les pages pour la gestion de lots. Les pages doivent permettrent de consulter et mettre à jour les lots en cours, de façon globale avec des filtres d'états, et également pour un joueur donné.

Pages côté client

Page flow



Page envoi des justificatifs

Les justificatifs demandés sont une photo de la carte nationale d'identité, et un RIB comportant un numéro IBAN.
La page doit guider l'utilisateur pour l'envoi des documents et l'informer sur la complétion de son dossier.
Des messages d'informations doivent lui indiquer comment rectifier ses erreurs.
Les documents peuvent être transmis en envoyant des images ou PDF, ou en prenant une photo depuis son mobile.
A l'issue de la saisie en cas de succès on affiche la page d'état des lots.

La page comporte un guide pour indiquer les documents à fournir, et peut indiquer les documents restant à fournir si l'équipe marketing a estimé qu'il manquait un document.

Page Etat des lots

L'utilisateur peut consulter une liste chronologique inverse des lots concernés par cette page, et pour chacun leur état. La liste devrait être scrollable et ne devrait pas nécessiter de pagination vu sa taille restreinte.
Ainsi, on peut trouver pour chaque lot les états suivants :

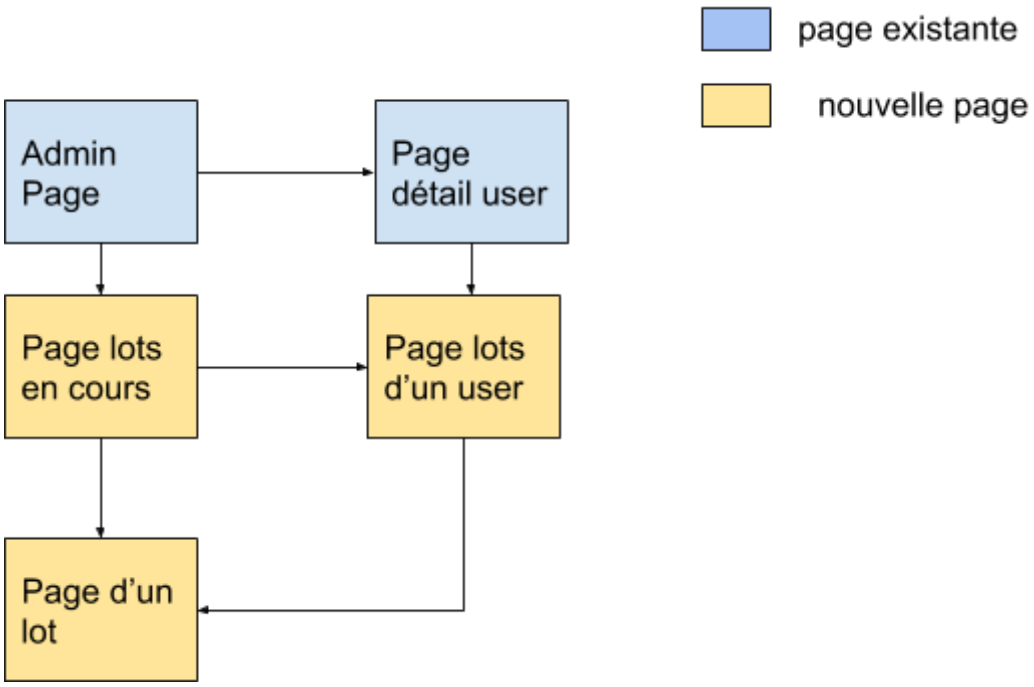
- pour un lot physique :
    - "profil incomplet" si le profil n'est pas saisi ou incomplet, et lien vers la page de profil
    - "documents incomplets", code couleur orange, et lien vers la page de saisie des documents,
    - "en cours de traitement", code couleur bleu, lien d'aide vers la FAQ
    - "lot envoyé", code couleur vert

- pour un gain en Cash
    - "profil incomplet" si le profil n'est pas saisi ou incomplet, et lien vers la page de profil
    - "documents incomplets", code couleur orange, et lien vers la page de saisie des documents
    - "en cours de traitement"
    - "virement effectué"

L'application doit émettre des notifications à l'utilisateur en cas de changement d'état le concernant (docs incomplets, en cours de traitement, traité).

Pages côté admin

Page flow



Page lots en cours

Une table comportant une ligne par lot à traiter, avec dans chaque colonne :

- identifiant du gain (lien cliquable vers le détail du gain)
- date
- identifiant du gagnant (lien cliquable)
- état du traitement
    - non réclamé
    - à valider (si le joueur a transmis les documents)
    - documents incomplets (l'admin a jugé le dossier incomplet et a indiqué les manques)
    - à traiter (l'admin a validé les documents et doit procéder au transfert)
    - traité (indique que le paiement / envoi a été réalisé)

La table comporte des filtres et une pagination permettant de naviguer dans les lots. On peut filtrer par état afin de traiter par exemple tous les dossiers à valider.

Page détail d’un lot

Cette page rappelle le libellé du lot, la date du gain et l’application concernée, l’identifiant du gagnant, son nom et sa photo si disponible.
Elle présente les documents fournis par le joueur avec des previews des images dans la mesure du possible.
Elle présente l’état du traitement en cours avec un code couleur.
Elle permet de modifier l’état du traitement selon l’état actuel.
Selon l’état choisi, on peut avoir à fournir une liste des éléments manquants, par exemple si on choisit “documents incomplets” il faut cocher dans la liste des documents nécessaires les manquants.

SPÉCIFICATIONS TECHNIQUE DU PROJET

VIPRIZE

Spécification technique très importante, il a été décidé d'appeler le projet en développement VIPrize (Very Important Prize) malheureusement VIP étant déjà pris.

Gestion des données
Une des spécification technique imposée était un système de communication unilatéral entre les jeux et l'app VIPrize.
Dès qu'un joueurs gagne un prix les infos sont transmisent à VIPrize sans attendre de retour de l application.

Voir diagramme de séquence **annexe 1**

Les modèles de la DB découle de ces données envoyées ainsi que des templates des prix qui sont sous la forme suivante

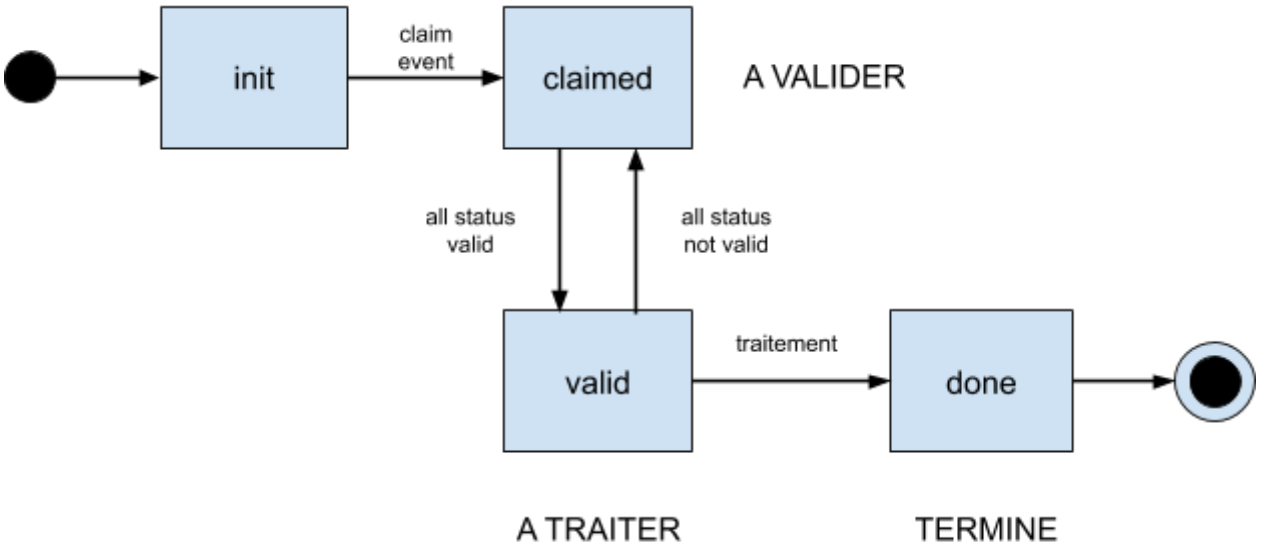
```
{
  "appId": "BL",
  "prizeId": "1",
  "prizeCod": "JACKPOT",
  "currency": "EUR",
  "cat": "cash"
}
```

Voir DB model **annexe 2**

Les Etats

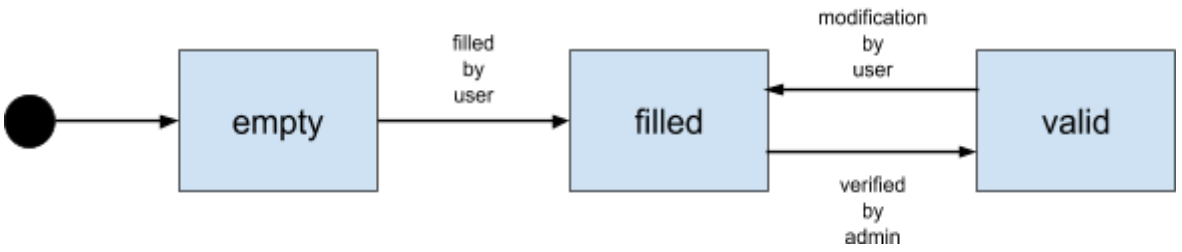
Les états des lots étant déjà gérée par les jeux (pour les lots non argent et non physique : coupon par exemple), l'application VIPrize se devait d'utiliser les même valeurs d'états

ETATS DES LOTS



Quand aux documents et au profil, les etats sont gérés uniquement dans VIPrize il n'y avait donc aucune contrainte de ce coté la.

ETATS DES DOCS / PROFIL



Système de messagerie
Comme expliqué précédemment la communication entre les jeux et VIPrize est à sens unique, les jeux ne doivent pas attendre de recevoir une réponse de la part de VIPrize afin de ne pas bloquer l'envoi de futur données ou surcharger le serveur de jeux.
Par soucis de sécurité et de persistence de données, implémenter un système de messagerie pour cette communication est nécessaire.

A cause du fait qu'un prix doit être réclamé une fois gagné, il y avait un besoin de conserver l'ordre des messages envoyés.

Le système de messagerie RabbitMQ a été choisi pour sa facilité d'implémentation et ses fonctionnalités, notamment le fait de pouvoir choisir si on veut ou non conserver l'ordre des messages.

RabbitMQ peut être utilisé de manière simple, en installant localement 1 node sur l'émetteur et 1 node sur le récepteur.

Le rôle d'une node est de sauvegarder localement les messages jusqu'à ce qu'ils soient envoyé à une autre node ou consommés directement par le serveur.

Voir messagerie simple **annexe 7**

L'installation des nodes ne demande aucune configuration autre que les connexions d'envoi aux autres nodes, toute la configuration se fait localement à l'aide de l'API via le script d'envoi ou de réception.

Il est possible facilement d'agrandir ce réseaux de messagerie ainsi que d'en augmenter la sécurité (pour la persistence des données)

Voir messagerie complexe **annexe 8**

Technologies Utilisées

Coté front - partie Client

Framework : **React** un framework basé sur **Node.js**

Librairies supplémentaires :

react-native-web
https://github.com/necolas/react-native-web
L'intérêt de react-native-web est que le code est compatible à 95 % avec react-native ios/android. Permet aussi d'utiliser des librairies de style react-native qui sont responsive mobile.

react-native-elements
https://react-native-training.github.io/react-native-elements
UI framework pour react-native.

axios
https://github.com/axios/axios#axios
Remplace Fetch API, permet de gérer plus proprement des requêtes http.

iban
https://github.com/arhs/iban.js#readme
Iban validateur, permet aussi de formater un iban de différentes façons.

react-localization
https://github.com/stefalda/react-localization#readme
i18n module pour la traduction des textes en différentes langues.

Coté front - partie Admin

Framework : **React** un framework basé sur **Node.js**

Aucun code n'a été produit durant mon stage concernant la partie Admin.

Coté back - partie Serveur

Framework : **Sails** un framework basé sur **Node.js**
https://sailsjs.com/
Sails permet de créer automatiquement une API REST connecté à n'importe quel types de base de données (par défaut : MySQL, PostgreSQL, MongoDB) une fois que les modèles ont été correctement définis
https://sailsjs.com/documentation/concepts/extending-sails/adapters/available-adapters

Librairies supplémentaires :

sails-generate-scaffold:
Permet de générer des scaffold pour les modèles sails.
A été utilisé pour générer une partie de l'interface admin afin de gérer quelques modèles sans passer directement par la DB.

sails-mysql:
Sails module qui permet de configurer un serveur SQL comme datastore pour Sails.

skipper-s3:
Sails skipper module pour upload des fichier sur amazon s3.
Utilisé pour upload les images des documents requis.

jest.js:
https://jestjs.io/
Utilisé pour créer des tests unitaires, permet de tester des composants React.

Xstate:
https://github.com/davidkpiano/xstate#readme
Module pour créer / gérer des state machine.
Utilisé pour les États des lots et des documents.

amqplib:
Module qui permet d'utiliser RabbitMQ en tant que message broker.
http://www.rabbitmq.com/

iban
https://github.com/arhs/iban.js#readme
Iban validateur, permet aussi de formater un iban de différentes façons.

EXTRAITS DE CODES SIGNIFICATIFS

La fonctionnalité la plus importante de l'application côté serveur est le système de communication entre l'application et les jeux.
Dans ce système on a d'une part la messagerie et d'autre part le serveur qui traite le message.

Le serveur traite les messages à l'aide d'un contrôleur. Le fichier du contrôleur se nomme GameEventController.js
Il possède 3 méthodes qui sont chacune rattachées aux 3 events qui proviennent des jeux (annexe 1)

Event USER_WIN voir **annexe 3**

Ce code effectue beaucoup d'action dans la Database (il peuple environ la moitié des tables de la DB), c'est la que le framework Sails entre en jeu.
On peut facilement adopter une syntaxe async / await car Sails le permet, on arrive à un code bien plus lisible.
Sails rajoute aussi une surcouche sur les fonction async pour éviter de devoir utiliser des try / catch, cela donne accès à une méthode intercept (**ligne.51**), qui comme son nom l'indique intercepte les erreurs comme un try / catch mais avec une syntaxe moins lourde.

Mais surtout Sails permet d'utiliser une fonctionnalité de transaction (**l.7)**, cette fonctionnalité annule toutes les opérations faites précédemment dans la database si une des actions échoue. Cela améliore la persistance des données et évite d'avoir des données incomplète qui peuvent potentiellement créer des erreurs plus tard.
Les '.usingConnection(db)' font partie de cette syntaxe de transaction.

Event USER_CLAIM voir **annexe 4 'extrait de code 2'**

User claim est un update simple d'un champ de la Database, ici on perd la syntaxe de transaction, intercept vient se greffer directement sur la requête d'update.
Ce code est intéressant car il fait appel à Xstate qui gère les etats des lots et des documents.
La fonction nextGainStatus (**l.14**) fait appel à une state machine, elle prend pour paramètre l'état initial + un événement pour retourner l'état suivant.

Faire à appel à un système de state machine, permet d'isoler une logique dans un code spécifique, cela est très utile pour éviter les erreurs, effectuer des tests unitaires afin d'avoir un code stable mais aussi permettre à de futurs développeur de comprendre / utiliser la logique des états des gains / docs simplement.

TEST UNITAIRES

Avant mon stage le concept des test unitaires restait un concept très vague pour moi, j'ai donc pris le temps de me documenter sur internet pour pouvoir les mettre en pratique sur ce projet notamment sur la partie du contrôleur qui est point critique du code.

Un point important qui est souvent mal expliqué sur internet est le fait qu'un test unitaire test le code de façon isolé. Le script de test doit être isolé du reste de l'environnement, autrement cela s'appelle un test d'intégration, certe important mais totalement différent.

Afin d'isoler mon test sur mon controleur j'ai donc dû créer des faux modèles qui interagissent un peu de la même façon que les vrai (mocks) dans la mesure du besoin de mon test. Car hors de question de lancer Sails et la DB pour pouvoir faire tourner des test unitaires.

Voir **annexe 5 'extrait de code 4'** mock du model AppUser

Ce code mock les 2 méthodes utilisées dans mon code, findOne et create, les données créées avec la méthode create sont stocker dans un tableau accessible avec la méthode findOne.

Voir **annexe 4 'extrait de code 3'** test unitaire

Voici une partie du test unitaire du contrôleur utilisant les différents mock de modèles.
Cette exemple de test montre le comportement attendu du contrôleur quand les données envoyées sont correctes, on obtient des données réparties dans 4 tables différentes.

VULNÉRABILITÉS DE SÉCURITÉ

VIPrize étant une WebView intégré directement dans une application mobile, il n'y a pas eu besoin de mettre en place un système de login sécurisé.
L'authentification sur l'application se fera à l'aide d'un Token utilisateur fourni par le jeux.

Les points de sécurités les plus importants pour l'application sont la persistance des données et la disponibilité du service.

On a déjà abordé quelques points sur la persistance des données dans les parties précédentes, avec les transaction, le système de node de la messagerie etc …

Pour ce qui est de la sécurité Sails possède des systèmes de protection intégrée contre la plupart des attaques web, néanmoins en cas de crash de la node de messagerie, il a fallu rajouter du code pour que Sails se reconnecte et relancer la machine :

messagerie => sails => DB

Voir **annexe 5 'extrait de code 5'**

Ce code permet à Sails de relancer le 'worker' qui permet de consommer les messages de la messagerie.
Cette fonction est un peu particulière car elle est dite **récursive**, elle est appelée dans le 'receiver' qui lui même est appelée dans le 'worker' (Voir **annexe 5 'extrait de code 6'**).
Donc elle relance le 'worker' en boucle toutes les X secondes (ici le timer est égal à 10s).

Le danger des fonction récursive est qu'elles peuvent boucler à l'infini et surcharger voir crash un serveur.

Ici pour éviter que la fonction se lance plusieurs fois, on utilise une variable dite **statique** au sein même de la fonction (**l.18**)
Une variable statique est conservé entre chaques appels d'une même fonction.
Donc si il y a déjà eu un setTimeout de lancé, on l'annule au cas où il n'est toujours pas fini pour éviter de créer plusieurs boucles récursives.

Upload de documents
Pour l'upload de documents, il y a 2 types d'input :

- input texte : directement le numéro d'IBAN
- upload de fichier : une photo (carte d'identité, RIB etc)

Il a donc fallu sécuriser ces 2 méthodes.

Pour l'IBAN la librairie npm **iban** donne accès à une fonction qui permet de vérifier si un iban est valide.
Cette fonction est utilisée côté application client pour donner l'information à l'utilisateur si il a rentré un IBAN correcte, mais aussi côté serveur pour vérifier que l'utilisateur n'a pas réussi à envoyer autre chose qu'un IBAN.

Pour ce qui est de l'upload de fichier, les images sont upload directement sur un serveur amazon s3, donc on a pas besoin d'ajouter du code pour la sécurité. La sécurité est gérée directement sur la plateforme du service.

EXEMPLE DE RECHERCHE SUR SITE ANGLOPHONE

Une recherche qui m'a bien servit pour le côté application client a été d'essayer de comprendre le design des applications mobiles.
Les mots utilise lors de cette recherche '**mobile**' '**design**' **'component**' '**application**' donne sur quelque articles de blog, mais le site vraiment intéressant est le site '**material design**' de google.

Le concept de 'material design' a été invité par google en 2014, pour expliquer simplement c'est un ensemble de ligne directrice pour développer de manière consistante des applications ayant une expérience utilisateur similaire.

Dans mon application côté client, j'avais besoin de montrer la liste des gains gagnés par l'utilisateur, la liste des documents etc …
J'avais donc besoin principalement d'afficher des listes d'éléments
La page du site qui m'a le plus ete utile est la pages sur les composant de type 'liste'
https://material.io/design/components/lists.html#

Voici une traduction des parties pertinentes d'information que l'on peut trouver sur cette page, une grande partie du contenue intéressant reste tout de même les images.
Pour voir le résultat donné (voir **annexe 6**)

Lists **Usage** Lists are a continuous group of text or images. They are composed of items containing primary and supplemental actions, which are represented by icons and text.	**Listes** **Usage** Les listes sont un groupe continue de textes ou d'images. Elles sont composée d'éléments qui contiennent des actions primaires et secondaires, représentées par des icônes et du texte.
Principles **Logical** Lists should be sorted in logical ways that make content easy to scan, such as alphabetical, numerical, chronological, or by user preference. **Actionable** Lists present content in a way that makes it easy to identify a specific item in a collection and act on it. **Consistent** Lists should present icons, text, and actions in a consistent format.	**Principes** **Logique** Les listes doivent être triées de manière logique afin de permettre au contenue d'être analysé facilement, comme par exemple ordre alphabétique, numérique, chronologique, ou préférence d'utilisateur. **Actionable** Le contenu des listes est présenté de telle manière à ce qu'il soit facile d'identifier un objet d'une collection et d'agir dessus. **Consistent** Les listes doivent présenter les icônes, textes et actions dans un format consistent.

Types 1. Single-line list Single-line list items contain a maximum of one line of text. 2. Two-line list Two-line list items contain a maximum of two lines of text. 3. Three-line list Three-line list items contains a maximum of three lines of text.	**Types** 1. Liste à une ligne Les éléments d'une liste à une ligne contiennent au maximum une ligne de texte 2. Liste à deux lignes Les éléments d'une liste à deux ligne contiennent au maximum deux lignes de texte 3. Liste à trois lignes Les éléments d'une liste à trois ligne contiennent au maximum trois lignes de texte
Anatomy Lists are optimized for reading comprehension. A list consists of a single continuous column of subdivisions called rows that contain items of content.	**Anatomie** Les listes sont optimisées pour être compréhensible lors de la lecture. Une liste consiste en une seule colonne continue avec des subdivisions appelées des lignes qui contiennent les éléments du contenu.
Content types Content types can take different forms, depending on the needs of a list. A list control can display information and actions for list items. List items are comprised of three different content types: 1. Supporting visuals 2. Primary text 3. Metadata	**Types de contenu** Les types de contenu peuvent prendre différentes formes, en fonction du besoin d'une liste. Le contrôleur d'une liste peut afficher des informations et des actions pour chaque objet de la liste. Les objets d'une liste sont composés de trois différents types de contenu: 1. Support visuel 2. Texte principal 3. Métadonnées
A list should be easily scannable, and any element of a list can be used to anchor and align list item content. Scannability is improved when elements (such as supporting visual and primary text) are placed in consistent locations across list items.	Une liste devrait être facilement analysable, et chaque objet d'une liste peut être utilisé pour ancrer et aligner le contenu des éléments. L'analyse est améliorée quand les éléments (comme le support visuel et le texte principal) sont placés aux mêmes endroits de manière consistante entre chaque objet de la liste.
Visuals, dividers, and spacing The structure of a list can be clarified using visuals, dividers, and spacing. Do. Improve scannability by anchoring supporting visuals, such as thumbnails, along the row's edge.	**Visuels, diviseurs, et espacements** La structure d'une liste peut être clarifié en utilisant des visuels, diviseurs, et espacements. A faire. Améliore l'analyse en accrochant des support visuel, comme des images miniatures, le long du bord des lignes.

Caution. Placing supporting visuals in the center of the row can make it difficult to see the relationship between primary content and supporting content.	Attention. Placer du support visuel au milieu d'une ligne peut rendre difficile à distinguer le contenu principal du contenu secondaire.
Do. Place a divider between rows with lots of content, such as those with three-line lists. Caution. Distinguish rows by maintaining sufficient space between list items. The primary action takes up the majority of space. Clear hierarchy is created by aligning the most distinguishing content on the left, with the least distinguishing on the right.	A faire. Placer un diviseur entre les lignes qui contiennent beaucoup de contenu, comme des listes à trois lignes. Attention. Distinguer les lignes en gardant suffisamment d'espace entre chaques lignes. L'action primaire prend la majorité de l'espace. Une hiérarchie claire est créé en alignant le contenu le plus voyant sur la gauche, avec le contenue le moins voyant sur la droite.