Sorbonne Universite : Faculte des Sciences et Ingenierie

Rapport de DAR

BetCoin

Jordan JEUDY
Justin RUDAT
Michael TA

Master 2eme Annee
STL 2018-2019

Sommaire

1 Présentation de l'application **3**

1.1 Qu'est-ce que BetCoin ? 3

2 Manuel d'utilisation **3**

2.1 Inscription 3

2.2 Login et Logout 4

2.2.1 Token 5

2.3 Compte utilisateur 5

2.4 Modification des informations du compte 6

2.5 Onglet glissant 7

2.5.1 Liste des salons de pari actifs 7

2.5.2 Page d'un salon de pari 8

2.5.3 Graphiques statiques 9

2.5.4 Paris associés au compte 10

3 Architecture **11**

3.1 Bootstrap 12

3.2 Base de données 12

3.2.1 PostgreSQL 12

3.2.2 MongoDB 13

3.3 Client 14

3.4 Serveur 15

3.4.1 bd 15

3.4.2 filtres 15

3.4.3 services 16

3.4.4 servlet 17

4 Choix Techniques Et Difficultés **18**

4.1 Hébergeur du serveur 18

4.2 Hébergeur du client 18

4.3 Bases de données 18

4.4 CRON 19

4.4.1 Problèmes 19

4.5 Communication et collaboration 19
4.6 API 20
4.6.1 Rest Countries 20
4.6.2 Gravatar 20
4.6.3 Crypto Compare 20
4.7 Angular 20
4.8 Devise des pronostiques 20
4.9 Connection lorsqu'une session est déjà ouverte 21
4.10 Recupération des résultats 21
4.11 Désinscription 21
4.12 Vérification Client et Serveur 21

5 Extensions possibles **22**
5.1 Ajout de publicité 22
5.2 Ajout de notification 22

1 Présentation de l'application

1.1 Qu'est-ce que BetCoin ?

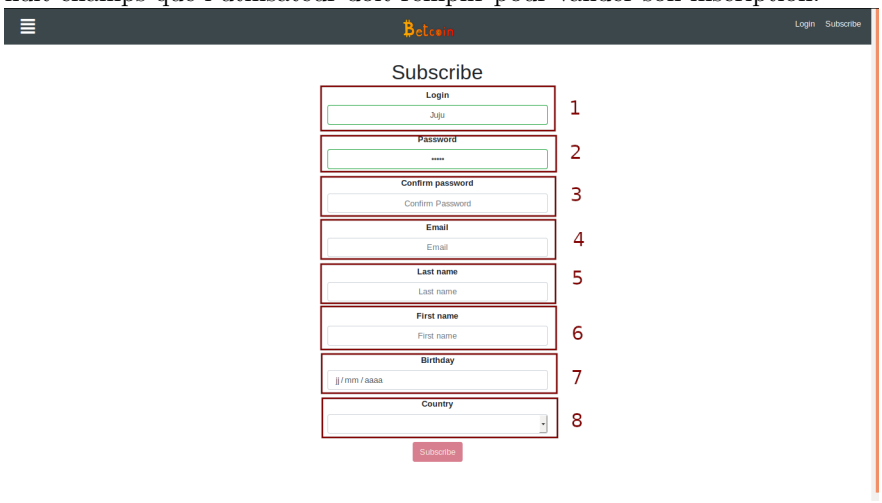
L'application "BetCoin" permet aux utilisateurs de réaliser des paris avec de la monnaie fictive (des **BetCoins**) sur le cours des cryptomonnaies. Ainsi, nous pouvons parier sur différentes cryptomonnaies tel que le **BitCoin**, l'**Ethereum**, l'**Ethereum Classic**, le **LiteCoin**, l'**EOS**, le **Bitcoin Cash**, le **XRP**, le **ZCash**, le **NEO** et même le **Dash**. Il existe deux types de paris, nous pouvons soit parier sur la valeur exact qu'aura une cryptomonnaies ou bien parier sur son évolution (Si son prix augmentera ou descendra).

2 Manuel d'utilisation

Nous allons, dans cette section, présenter un manuel succinct pour les utilisateurs, reprennant les diverses fonctionnalités proposées.

2.1 Inscription

Dans un premier temps, sur ce type de site, vient l'inscription. On y trouve huit champs que l'utilisateur doit remplir pour valider son inscription.



1. **Login :** Il s'agit de l'identifiant de connexion. Ce dernier sera unique et l'inscription ne pourra se faire s'il est déjà pris par un autre compte.
2. **Password :** C'est le mot de passe associé au compte pour la connexion.
3. **Password confirmation :** On souhaite que l'utilisateur tape une seconde fois son mot de passe afin de le confirmer et d'éviter une faute de frappe.

4. **Email :** On demande un email qui servira, entre autres, à l'ajout d'une icône utilisateur dans le profil via gravatar, que nous présenterons plus tard. Cet email devra correspondre au format classique d'un email, à savoir l'adresse locale suivit du séparateur '@' et enfin de l'adresse du serveur :
jsmith@example.com

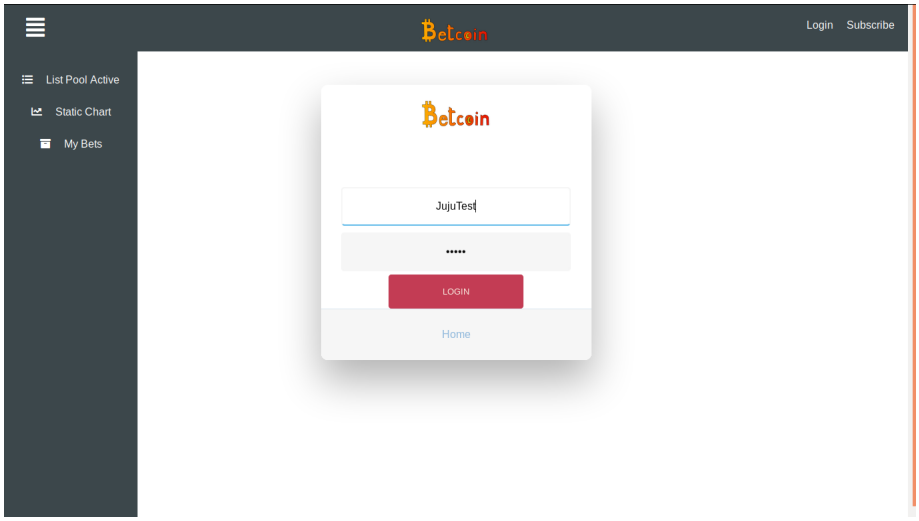
5. **Last Name :** On demande à l'utilisateur de renseigner son nom de famille.

6. **First Name :** On demande à l'utilisateur de renseigner son prénom.

7. **Birthday :** On demande à l'utilisateur de renseigner sa date de naissance.

8. **Country :** On permet à l'utilisateur de renseigner son pays parmis une liste pré-définie, nous récupérons cette liste grâce à l'API **REST country**.

2.2 Login et Logout

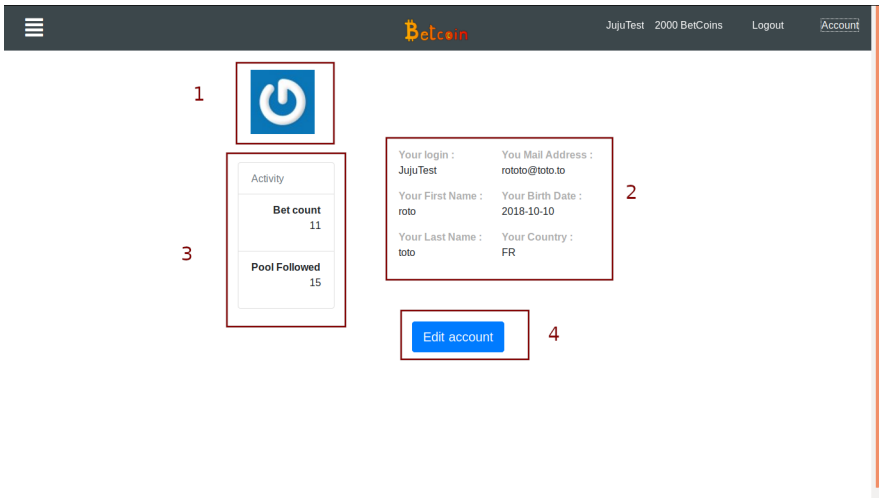


Suite à l'inscription, notre nouvel utilisateur est redirigé vers une page de connexion. On y retrouve donc les champs correspondant au login et au mot de passe associé à son compte.
Il ne pourra pas poursuivre la navigation vers d'autres pages du site sans être inscrit et sans s'être connecté, hormis la page des salons actif et celle des graphiques statique.
Une fois connecté, l'utilisateur sera redirigé vers une liste des salons de pari actifs.
L'option 'logout' viendra remplacer le 'login' dans le navigateur du haut une fois l'utilisateur connecté.

2.2.1 Token

Pour accéder aux services nécessitant d'être connecté, l'utilisateur en se connectant va récupérer un token qui va être stocké localement et passé en paramètre des requêtes à notre API REST. Le serveur va ensuite vérifier que le token est bien valide, pour accepter ou refuser l'accès au service.

2.3 Compte utilisateur



1. **Image de profil :** Il s'agit d'une image de profil du compte qui pourra être modifiée par l'utilisateur avec son compte gravatar.

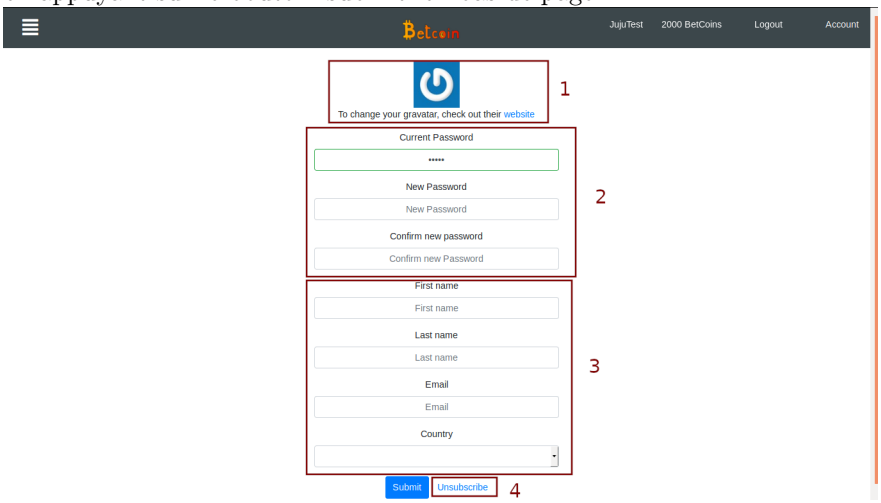
2. **Affichage des informations du compte :** Dans ce panneau, on retrouve toutes les informations associées au compte tel que le nom d'utilisateur, le prénom et nom, l'adresse email, la date de naissance et son pays.

3. **Activité du compte :** Dans ce panneau, on retrouve l'activité du compte qui correspond au nombre de paris réaliser ainsi qu'au nombre de salon auquel ce même compte s'est inscrit.

4. **Bouton de modification :** Nous retrouvons en dessous de toute ces informations un bouton redirigeant l'utilisateur vers une page où il pourra modifier les informations liées à son compte.

2.4 Modification des informations du compte

Les modifications qui seront entrées dans ce formulaire seront prise en compte en appuyant sur le bouton 'submit' en bas de page.



1. **Image de profile via Gravatar :** Nous utilisons gravatar pour la photo de profil, qui est un site permettant d'associer une image à une adresse mail. Nous proposons ici une redirection vers leur site et ainsi, une fois leur choix fait en utilisant gravatar, cette image de profil sera automatiquement modifiée.

2. **Modification du mot de passe :** Pour modifier le mot de passe, nous demandons tout d'abord à l'utilisateur de saisir son ancien mot de passe, puis comme lors de l'inscription, de saisir le nouveau mot de passe ainsi qu'une confirmation de ce dernier.

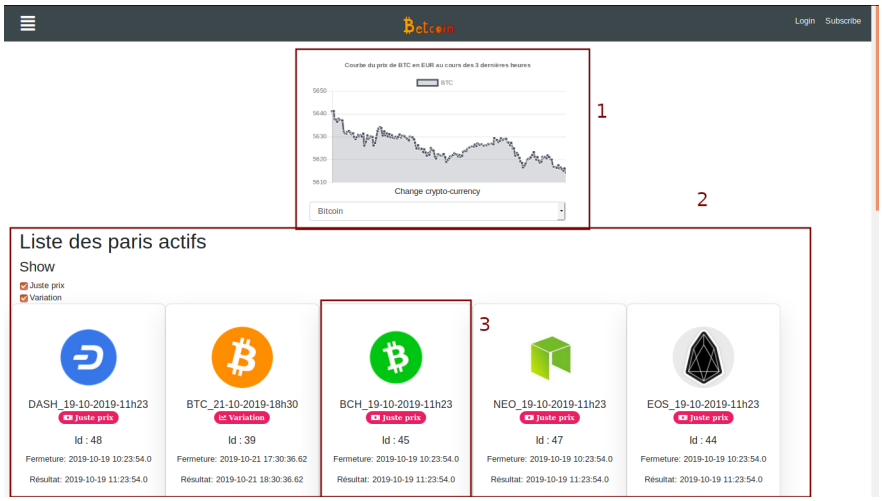
3. **Informations générales :** Nous laissons la possibilité à nos utilisateurs de modifier leur nom, prénom ainsi que leur email et leur pays. Nous avons décidé de ne pas leur donner la possibilité de modifier la date de naissance.

4. **Désinscription :** En appuyant sur ce lien, l'utilisateur sera redirigé vers une page où il devra entrer son mot de passe afin de valider sa demande de désinscription. Son compte sera alors bloqué. Nous avons fait ce choix afin qu'un autre utilisateur ne puisse pas récupérer son identifiant.

2.5 Onglet glissant

Nous allons maintenant énumérer les fonctionnalités principales de notre application qui sont regrouppées dans un onglet glissant sur la gauche de l'écran. L'onglet peut se fermer ou s'ouvrir en cliquant sur l'icone des quatre barres verticales en haut à gauche de l'écran.

2.5.1 Liste des salons de pari actifs



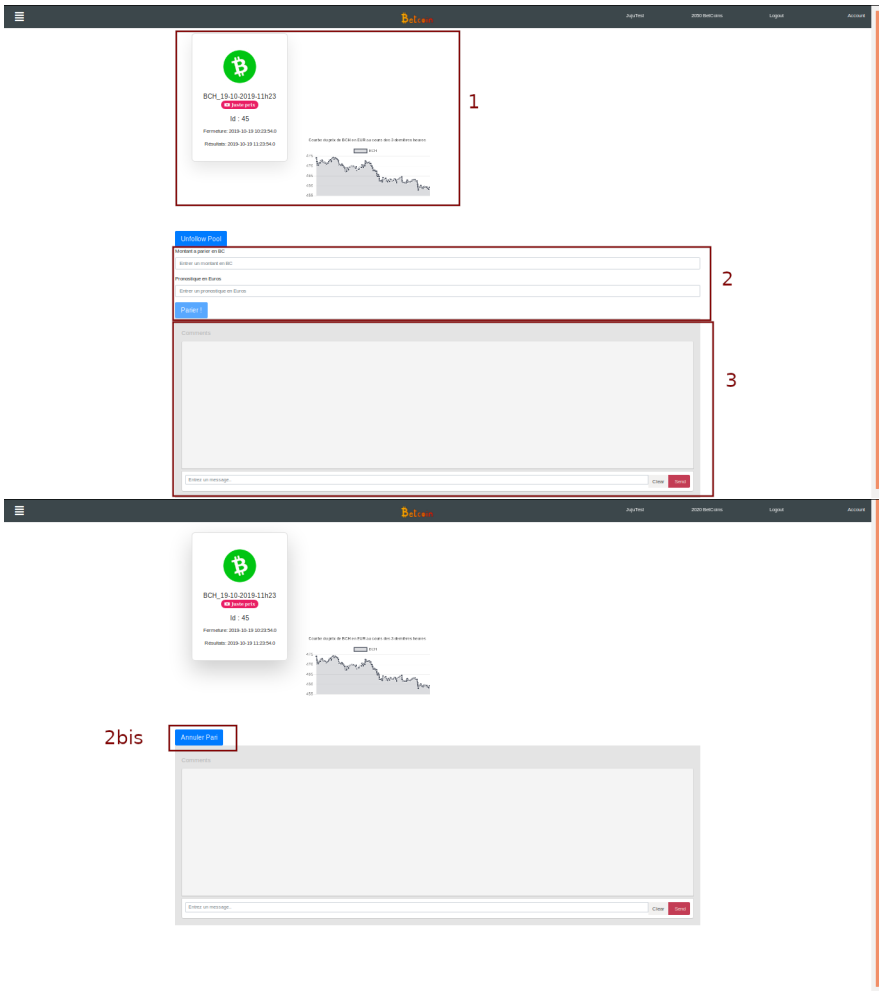
Au cœur de notre application nous retrouvons donc les paris.

1. **Graphique :** L'utilisateur pourra voir un graphique du cours de la cryptomonnaie qu'il aura sélectionné sur les trois dernières heures.

2. **Liste des salons actifs :** L'utilisateur peut voir ici la liste des salons de paris auxquel il peut accéder afin de parier. En cliquant sur un des panneau de description d'un salon, il sera redirigé vers la page correspondant à ce salon.

3. **Panneau de description d'un salon :** On retrouve sur ce panneau un descriptif du salon de pari. En premier on voit l'icône de la cryptomonaie qui sera sujet des paris de ce salon, ensuite le nom du salon, puis le type qui peut être sois 'Juste prix' sois 'Variation' et l'id de ce salon. En dessous on peut aussi voir la date de fermeture des paris ainsi que la date à laquelle les résultats seront disponibles.

2.5.2 Page d'un salon de pari



Nous voici donc enfin sur la page d'un salon de pari.

1. **Descriptif du salon :** On retrouve dans cette partie la fiche descriptive du salon comme définie dans la section précédente ainsi que le graphique de la cryptomonnaie liée à ce salon sur les trois dernières heures.

2. **Création d'un pari :** Dans cette section de la page, l'utilisateur pourra saisir le montant de BetCoins qu'il souhaite parier ainsi que son pronostique. Ce dernier sera soi un montant précis en euro, ce qui en cas de réussite du pari lui rapportera une grande quantité de BetCoins, dix fois sa mise de départ, soi un choix entre 'montera' et 'descendra' qui correspondra à l'évolution du cours de la cryptomonnaie à la fin de la période de pari, ce qui lui rapportera une somme modérée de BetCoins, sa mise de départ plus la moitié de se montant.
   Une fois un pari réaliser, l'utilisateur aura alors accès à la section 2bis.

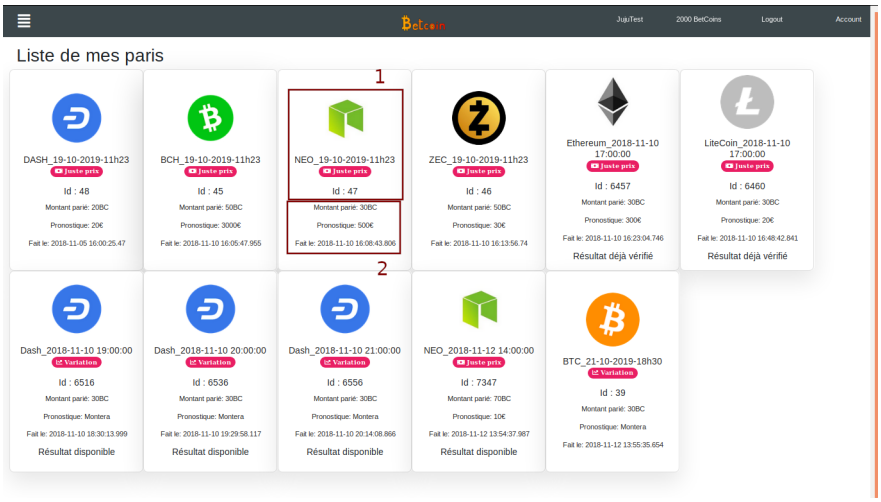
2bis. Annulation d'un pari : L'utilisateur, en cliquant sur ce bouton, pourra annulé son pari tant que les paris n'ont pas encore été fermés sur ce salon.

3. **Espace commentaire :** Les utilisateurs pourront envoyer des messages qui seront liés à ce salon en utilisant l'espace commentaire. L'utilisateur pourra, s'il le souhaite, supprimer les messages qu'il a écrit.

2.5.3 Graphiques statiques

Sur cette page, l'utilisateur pourra accéder aux graphiques de la cryptomonnaie qu'il aura selectionné ou, par défaut, celle de l'Ethereum. Il pourra aussi saisir l'intervalle temporel sur lequel il souhaite voir ce graphique qui sera, par défaut, sur les trois dernière heures.

2.5.4 Paris associés au compte

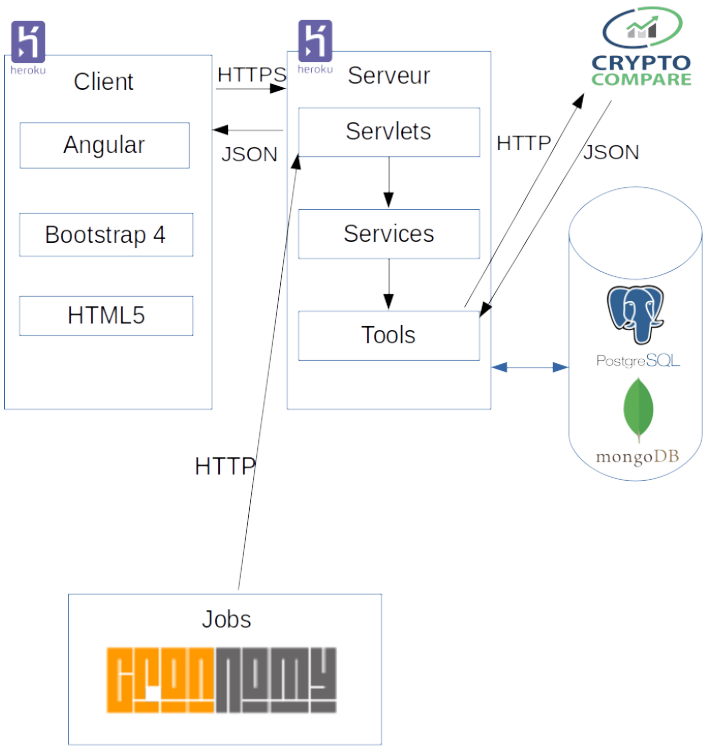


Sur cette page, l'utilisateur pourra voir les paris qu'il a réalisés. De plus c'est en passant par cette page qu'il pourra voir si les résultats sont disponibles. C'est sur la page du salon qu'il pourra récupérer ses gains.

1. **Descriptif du salon :** On retrouve de nouveau le descriptif du salon tel que spécifié précédement.

2. **Informations sur le pari :** Ici l'utilisateur pourra voir le montant en BetCoins qu'il a parié, puis le pronostique qu'il a établi au moment du pari ainsi que la date à laquelle ce pari a été réalisé.
   De plus si la date des résultats est passée, un texte le signalera à l'utilisateur, qui pourra alors cliquer sur le pari en question avec d'être redirigé vers une page où il pourra récupérer ses gains. Cependant si les gains ont déjà été récupérés, ce même texte le signalera aussi.

3 Architecture



Notre application est découpée selon le modèle du schéma ci dessus. Précisons que notre serveur est considéré comme une API **REST**.
Le client de l'application va communiquer avec la partie serveur en utilisant des requêtes HTTPS. Celles ci seront récupérer par les servlets qui utiliseront une sous-couche de service, faisant elle même appel à une couche outil. Cette dernière est la partie du serveur qui communique avec l'API de CryptoCompare ainsi qu'avec les bases de données.
De plus, certaines de nos servlets communique avec un site externe, CRONNOMY, afin de réalisé régulièrement la création de nos salons de pari.
Une fois la réponse à la requête obtenu, le serveur renvoie un json vers notre client afin de transmettre les informations nécessaires pour la poursuite de la navigation sur notre application par les utilisateurs.

3.1 Bootstrap

Bootsrap est une collection d'outils open source utiles à la création (graphisme, animations, interactions, ...) de sites et applications web. Elle contient un ensemble de codes HTML, CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que d'autres composants sous forme de plugins utilisant la bibliothèque JQuery. L'utilisation de **Bootstrap**, nous a facilité toute la partie CSS du projet.

3.2 Base de données

3.2.1 PostgreSQL

Nous avons gardé certaines données dans une base PostgreSQL. Elles correspondent aux données de compte utilisateur ainsi qu'aux données décrivant les salons de pari.
On y retrouve pour les comptes utilisateurs :

iduser : un identifiant numérique unique du compte

login : l'identifiant de connexion unique du compte

password : le mot de passe. Lors de son enregistrement ainsi qu'à chacune de ses utilisations, nous utilisons une fonction de hachage gérer par la base de données, selon l'algorithme **blowfish**.

email : l'email lié au compte

lastname : le nom de l'utilisateur

firstname : le prénom de l'utilisateur

solde : le nombre de BetCoin du compte

islock : un boolean indiquant si le compte a été bloqué

birthday : la date de naissance de l'utilisateur

country : le pays de l'utilisateur

Et pour les salons de pari :

idbetpool : un identifiant numérique unique pour le salon

name : le nom du salon

openningbet : la date d'ouverture des paris du salon

closingbet : la date de fermeture des paris du salon

resultbet : la date où les résultat seront disponible pour ce salon

cryptocurrency : la cryptomonnaie lié au salon

pooltype : un booléen désignant le type du salon, variation si mis à false ou juste prix sinon

openningprice : pour les salons variation, afin de pouvoir faire la comparaison avec le prix à la fermeture du salon

Il est important de noter aussi que la création de salon s'effectue à heure fixe et ne dépend pas de paramètre donné à la servlet "createPool", on a donc ajouté un trigger dans notre base de données PostgreSQL. Ce trigger va simplement garantir de bonne propriétés lors de l'insertion d'un nouveau salon. (fermeture du salon 1h après, result 1H après le close)

3.2.2 MongoDB

En plus de la base de données PostgreSQL, nous avons aussi ajouté une base de données MongoDB.

L-Bet : Une liste des paris par salon où l'on stocke l'id du salon, la valeur de résultat et dans un tableau les différents pari ainsi que les utilisateurs les ayant réalisé.

Bet : cette table regroupe les paris avec comme information le login, l'id du salon, un booléen pour savoir si l'utilisateur à récupérer ces gains, le montant du pari, la valeur pariée et la date du pari. Elle nous sert à récupérer les paris d'un utilisateur précis.

Session : ces données regroupent les sessions de connexion des utilisateurs avec le login, le token associé pour cette session et la date de dernière connexion.

L-Message : on stocke ici les messages des salons, avec comme données l'id du salon et un tableau regroupant le login d'un utilisateur ayant posté un message, le texte du message et la date à laquelle celui ci à été posté.

SubscribePool : nous stockons ici pour chaque utilisateur dans un tableau les id des salons auxquels cet utilisateur est inscrit.

3.3 Client

La partie la plus importante de notre client se situe dans le répertoire **app**. C'est ici que l'on va trouver les différents fichier implémentant les services offert coté client. Ces derniers se trouvent à la racine de ce répertoire.
En particulier, on y trouve la définition de certaines classe de données dans des fichier TypeScript, tel que betPool ou encore userProfil. Il y a ensuite le module de routing de notre application, associant aux divers composants des noms de chemin. De plus, tous les services qui seront utilisé par les composants de notre applications sont définis ici dans les fichiers dont le nom se termine par *.service.ts .
Sont définis aussi un décodeur d'URL ainsi que nos constante URL qui seront réutilisées par les composants.
Nous avons aussi ajouté un des composant principal, nommé **app.component** qui correspont à notre navigateur, à l'onglet glissant ainsi qu'au corp de notre site dont on modifiera le contenu en fonction du composant demandé par l'utilisateur. Ces éléments seront présent sur toutes les pages de notre site.
On retrouve aussi dans ce répertoire, à l'intérieur de sous-répertoire, les fichiers définissant nos divers composants qui sont :

bet-pool-detail : un salon de pari

bets : les paris associés au compte

chat : l'espace commentaire des salons de pari

edit-user : la modification des données utilisateur

listpool : la liste des salons de pari

login : la connexion

logout : la déconnexion

price-chart : les graphiques de cryptomonnaie

subscribe : l'inscription

unsubscribe : la désinscription

user-account : le compte utilisateur

3.4 Serveur

Dans cette partie nous allons décrire les différents packages que nous avons réalisé pour la création de la partie serveur de notre application.

3.4.1 bd

Il s'agit de la partie outil de notre serveur. Nous y avons regrouppé toutes les classes auquel font appel nos services. On y retrouve entre autres la plupart de nos requêtes vers les bases de données de notre application.
On y retrouve :

APITools.java : il s'agit de la classe qui va faire les appels vers l'API de CryptoCompare que nous avons décidé d'utiliser.

BetTools.java : c'est la classe qui va s'occuper de réaliser toutes les actions requises en rapport avec les paris.

CryptoEnum.java : qui est une classe d'énumération des cryptomonnaies que notre applications supporte.

Database.java : c'est ici que se fais la gestion de la connexion à la base de données PostgreSQL.

PoolTools.java : où on retrouve les fonctionnalités permettant la gestion des salons de pari.

SessionTools.java : cette classe va gérer les fonctionnalités de connexion, déconnexion et la gestion des sessions d'utilisateur.

UserTools.java : et enfin la classe qui va réaliser les fonctionnalités en rapport avec les utilisateurs.

3.4.2 filtres

CorsFilter.java : Le Cross Origin Ressource Sharing est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant. Pour des raisons de sécurité, les requêtes HTTP multi-origine émises depuis les scripts sont restreintes. En commençant la partie client nous n'étions pas certains d'avoir le client et le serveur hébergés sur le même serveur web (Heroku), nous avons donc créé un filtre qui permet à chaque servlet d'accepter des requêtes HTTP multi-origine.

CronIpFilter.java : Le choix de l'hébergeur Heroku nous a contraint à externaliser une partie métier de notre application. En effet la création de salons de pari est cruciale et elle est réalisée par un cron qui va interagir avec la servlet "createPool". Comme notre serveur est sous forme d'API REST on ne pouvait pas se permettre de laisser cette servlet accessible par n'importe qui. On a donc ajouté un filtre qui agit uniquement sur "createPool" et qui va autoriser l'accès au servlet uniquement pour l'adresse IP de CRONNOMY.

3.4.3 services

Nos classes de services font l'intermédiaire entre nos servlets et les outils décrit précédemment. C'est ici que nous allons créer les JSON de réponse qui seront renvoyés au client et c'est aussi ici que nous allons pouvoir spécifier la nature des erreurs que nous pouvons rencontrer.
Nous y trouvons :

AccountService.java : ce service va permettre de récupérer les données liées à un compte afin de les afficher. Il va aussi gérer la modification de ces données.

APIService.java : ce service va faire les appels vers nos outils pour communiquer avec l'API dynamique et renvoyer les résultats ainsi obtenus.

BetPoolService.java : ce service va réaliser les appels vers les outils de gestion des salons de pari.

BetService.java : ce service va réaliser les appels vers les outils de gestion des paris.

LoginService.java : ce service va s'assurer de la connexion ainsi que de la déconnexion de nos utilisateurs.

ServiceTools.java : ce service va être utilisé par tout les autre. Il comporte des fonctions créant nos JSON de réponse afin de toujours garder le même format.

UserService.java : ce service va prendre en charge toutes les fonctionnalités qui sont en rapport avec les utilisateurs.

3.4.4 servlet

Nous avons divisé les servlets parmi deux groupes. Celle réalisant des requêtes en POST et celle réalisant des requêtes en GET.
Elles sont séparer comme ceci :

POST : **AccountModificationServlet :** permet la modification des données du compte

AddBetServlet : permet l'ajout d'un pari

CancelBetServlet : permet l'annulation d'un pari

ConnectServlet : permet la connexion

DeleteMessageServlet : permet la suppression d'un message

DisconnectServlet : permet la déconnexion

EnterPoolServlet : permet d'entrer dans un salon de pari

GainRetrievalServlet : permet de récupérer les gains sur un pari

MessagePoolServlet : permet d'ajouter un message sur un salon

QuitPoolServlet : permet de se désinscrire d'un salon

SubscribeServlet : permet de s'inscrire sur le site

UnsubscribeServlet : permet de se désinscrire du site

GET : **BetResultAvailableServlet :** permet de savoir si les résultats sont disponible

CreatePoolServlet : permet la création d'un salon de pari

CryptoCompMinServlet : permet de récupérer la valeur d'une cryptomonnaie sur un intervalle de temps défini en minutes.

CryptoCompServlet : permet de récupérer la valeur d'une cryptomonnaie sur un intervalle de temps défini en heure.

GetBetServlet : permet de récupérer les données d'un pari

GetListBetsServlet : permet de récupérer la liste des paris

GetListMessageServlet : permet de récupérer la liste des messages d'un salon

GetListPoolsServlet : permet de récupérer la liste des salons actifs du site

HasBetServlet : permet de vérifier si le compte à fait un pari dans un salon défini

VisualiseAccountServlet : permet de récupérer les données d'un compte

VisualisePoolInfoServlet : permet de récupérer les données d'un salon

C'est dans les servlets que nous faisons appel à la classe **ValidatorHelper** qui est une classe contenant des méthodes renvoyant une exception si les paramètres nécéssaire pour le fonctionnement des servlets n'ont pas été correctement définis.

4 Choix Techniques Et Difficultés

4.1 Hébergeur du serveur

Notre premier choix a été celui de l'hébergeur du serveur, nous avions plusieurs choix possibles que ce soit les Amazon Web Services ou même Google App Engine, cependant on souhaitais avoir un hebergeur totalement gratuit et ne pas avoir une période d'essai. On a donc choisi **Heroku**.
Malheureusement vu la nature de notre site, nous voulions ajouter un moyen aux utilisateurs de gagner des **BetCoins** facilement et gratuitement.

4.2 Hébergeur du client

Nous avions dans un premier temps choisi d'utiliser **GitHub Pages** pour l'hébergement de notre client. Nous avons malheureusement rencontré certains soucis, en effet, **GitHub Pages** n'est pas adapté pour les sites de type **Single Page** imposés par **Angular**, nous avions donc certains soucis au niveau du routage. Nous avons décider de changer notre hébergeur et nous nous sommes retourner vers **Heroku** une nouvelle fois.

4.3 Bases de données

Dans notre application, nous avons deux bases de données, **MongoDB** ainsi que **PostGreSQL**, dans **MongoDB** nous stockons les données qui sont souvent modifiées ainsi que les documents où nous faisons beaucoup d'insertions. Dans **PostgreSQL**, nous stockons les informations beaucoup plus statique.
Nous voulions dans un premier temps nous perfectionner dans la manipulation de ces deux bases de données et en même temps cela nous permet de contourner les limitation imposées par **Heroku**. En effet, nous sommes limité à 10 000 lignes dans **PostgreSQL** et à 496 MB de données dans **MongoDB**. Comme nous possédons un **CRON** insérant régulièrement dans la base de données, nous aurions rapidement atteint la limitation.

4.4 CRON

Nous avons cherché un **CRON** gratuit et nous sommes rapidement tombé sur **https://cronnomy.com/** qui est assez simple d'utilisation.

4.4.1 Problèmes

Lorque le serveur n'est pas suscité, **Heroku** endors le serveur et la première requête qui le réveillera est très longue, cela nous a posé pas mal de soucis. En effet, si la réponse du serveur est trop longue notre CRON échoue. Nous avions pensé a faire une requête au serveur dont le résultat importe peu (Faire un ping au serveur), nous aurions pu appeler le serveur avec une requête **GET** pour le réveiller puis laisser le CRON faire son appel pour créer les différents salons.
Cependant, **cronnomy** possède une large fênetre d'attente pour recevoir la réponse du serveur, assez large pour que le serveur se réveille, ce qui nous a conforté dans le choix de ce **CRON**.

4.5 Communication et collaboration

Dans le cadre du projet, nous avons utilisé plusieurs outils de communication et de collaboration :

Slack : C'est une plateforme d'échange collaboratif, a été notre principal moyen de communication, dans le but d'interragir rapidement en cas de problème ou de s'entraider plus facilement. Cette plateforme étant beaucoup utilisée dans le monde professionel nous avons estimé qu'il s'agissait d'un choix intéressant de communication.

Github : Il s'agit d'une plateforme de développement collaboratif et de versionning. C'est l'approche la plus simple que nous ayons trouvé afin de pouvoir travailler sur diverses parties du code en même temps. De plus cela nous a permis de déceller et de corrigé les erreurs qui ont pu être commises durant la phase de développement plus efficacement.

Google Doc : Nous avons aussi utilisé un Google Doc afin de garder une trace de notre avancement de manière plus explicite qu'avec les commits de Github. De plus cela nous a permis de nous mettre en accord sur certains points tel que le type des données que nous souhaitions garder ainsi que leur utilité. Cela nous as permis aussi d'accorder aux taches à réaliser un ordre d'importance ainsi que leur êtat de développement.

4.6 API

4.6.1 Rest Countries

Lors de l'inscription, nous demandons a l'utilisateur de renseigner son pays, pour récuperer cette liste de pays nous utilisons l'API **Rest Countries**, nous avons fait le choix de faire un appel à cet API a chaque fois que c'est nécessaire. Nous trouvions qu'il était inutile de la télécharger et de la stocker, en effet nous nous rendons très rarement sur la page de l'inscription une fois que l'on est inscrit.

4.6.2 Gravatar

Nous avons décidé d'utiliser l'API **Gravatar** pour nos images de profil, en effet **Gravatar** est très simple d'utilisation (Il ne suffit que d'avoir un compte Gravatar et de renseigner la même adresse mail que celle du compte Gravatar), et nous permet d'avoir des images de profile sans devoir les stocker nous même dans nos bases de données.

4.6.3 Crypto Compare

Il s'agissait l'une des rares API gratuite pour récupérer les valeurs des crypto monnaies, de plus **Crypto Compare** possède de faibles limitations sur les requêtes ce qui facilite son utilisation.

4.7 Angular

Dans le cadre du projet, nous avons souhaité découvrir un nouvel framework du coté client. Notre choix s'est ainsi tourné vers **Angular**, en effet **Angular** est simple d'utilisation et est populaire, nous trouvions qu'il s'agirait d'un plus pour notre insertion professionnelle. De plus, **Angular** est basé sur des **composants web**, ce qui facilite la répartition des tâches.

4.8 Devise des pronostiques

Dans notre application, nous avons décidé que tout les pronostiques étaient fait en **Euros**, en effet sans ce choix, il aurait été difficile de stocker cette donnée. Nous aurions pu décider de stocker la concatenation du pronostique ainsi que la devise dans **MongoDB** ou nous aurions aussi pu rajouter une colonne dans **MongoDB** pour connaître la devise du pari. Cependant, nous n'avions pas l'ambition que notre application atteigne l'international, nous nous sommes donc concentré sur l'**Euros**.

4.9 Connection lorsqu'une session est déjà ouverte

Nous avions dans un premier temps décidé que lorsque l'on se connecte et qu'une session est déjà ouverte que la personne qui essaye de se connecter était bloquée. Puis, nous nous sommes aperçu que cela posait un problème lorsque l'on ne se déconnectait pas et qu'on vidait son cache (En effet, nous stockons toutes les informations de la session dans le localstorage et donc dans le navigateur du client). Le compte était dans ce cas ci bloqué, et le client ne pouvait le débloquer sans l'aide d'un administrateur.
Pour résoudre ce problème, nous avons rendu la connection prioritaire sur une session ouverte, ainsi même lors du nettoyage de son cache, il pourra toujours se reconnecter et le locastorage sera de nouveau rempli.

4.10 Recupération des résultats

Lorsque l'on souhaite récuperer le résultat d'un pari, nous avons besoin de faire un appel a notre **API** pour recupérer la valeur de la cryptomonnaie. Nous avons décidé qu'il s'agirait du 1er joueur qui vérifie son pari qui fera cet appel puis nous stockerons le résultat dans notre base de données afin d'éviter que les suivants refassent un appel à l'**API**. Nous aurions aussi pu faire un **CRON** qui récupère le résultat dès qu'il est possible. Cependant, cela serait une approche **EAGER** et non **LAZY** comme on le souhaite. En effet, il y aura surement de nombreux salon où il n'y aura aucun parieur (En particulier, celles qui se passent la nuit). De ce fait, récupérer le résultat de tel salon est inutile.

4.11 Désinscription

Lors de la désincription, le compte n'est pas supprimé de nos bases de données, nous avons rajouté un champ "**islock**" pour savoir si un compte est vérouillé ou non. En effet, si nous avions fait le choix de supprimer le compte, il aurait fallu parcourir toutes nos bases de données et suppprimer les informations relatives au compte, tel que les paris qu'il a fait et les messages qu'il a postés. Sans cela, un nouvel arrivant aurait pu reprendre son login et se faire passer pour lui. Nous avons ainsi une nouvelle vérification lors de la connexion pour vérifier que le compte n'est pas bloqué.

4.12 Vérification Client et Serveur

Lors d'une requête au serveur, nous vérifions les données côté client et serveur. En effet, la vérification côté client nous permet d'éviter des appels inutiles au serveur et ainsi avoir un site plus dynamique. La vérification côté serveur permet quant a elle de s'assurer de façon générale de la conformité des requêtes, nous ne pouvons pas faire totalement confiance au client, en effet rien n'empeche un utilisateur de faire son propre client et ainsi déviter toutes les vérifications côté client.

5 Extensions possibles

5.1 Ajout de publicité

Nous pensions ajouter une fonctionnalité qui consistais à regarder des publicités pour gagner des BetCoins, nous avons donc cherché une API pouvant faire un tel travail. Après recherche nous avions trouvé **Google Ads** et **Amazon Ads**. Cependant, ils étaient uniquement utilisable avec les hébergeurs de Google et d'Amazon.

5.2 Ajout de notification

De plus nous avions commencé à ajouter une fonctionnalité de souscription à un salon de pari. Cette dernière aurait permis l'envoi de notifications aux utilisateurs inscrit lors de l'ajout d'un commentaire dans ce salon.