Green Flag

Document de conception

Application web de rencontres

Bettina-Sarah Janesch - 1649508

Vincent Fournier - 1259008

Projet synthèse 420-C61-IN

2024-09-25

Introduction

Rappel du projet

Notre projet Green Flag est une application web de rencontres qui souhaite rehausser l'expérience utilisateur en améliorant l'algorithme de suggestion de matchs. Notre approche repose sur un système de retour d'expérience (feedback) qui redonne le pouvoir aux utilisateurs en leur permettant de signaler les personnes manquant d'engagement. Cela améliore le tri des profils pour tous, qui contribue à affiner les suggestions proposées par l'application.

Présentation

Ce document vise à faire un survol de tous les technologies utilisées dans notre projet, ainsi que de l'aspect technique et de sa structure. Il nous permettra d'avoir une vision plus concrète et clair du projet et par ce fait facilitera le développement de ce dernier. Il nous évitera également ainsi les déboires de réécrire le même code à plusieurs par manque de précision des problèmes à résoudre.

Éléments de conception

1. Infrastructure de développement

Notre projet sera déployé sur le web. Nous programmerons principalement en **TypeScript** et **Python,** avec la base de données en **PostgreSQL**. Les Framework logiciels utilisés seront React dans le frontend, et Flask dans le backend. Dans la base de données on utilisera PL/pgSQL pour faire des vues, fonctions, trigger etc. qui vont accélérer les processus internes et augmenter la performance.

Nous avons choisi TypeScript pour plusieurs points forts. Le typage statique aide à trouver les erreurs plus rapidement et renforce des retours stricts dans les fonctions, des instances de classe d'un type spécifique, qui aide l'auto-documentation du code. Ça aide à réduire les bogues aussi et sert à augmenter la robustesse de l'application.

Ensuite, ça permet d'élaborer des interfaces et types spécifiques qui assurent que les composantes sont utilisées de la bonne façon – pour des grandes applications ça augmente la consistance et clarté du code.

Nous avons aussi choisi Python comme notre langage backend car il est assez répandu dans l'industrie et possède une syntaxe simple. Sa simplicité et sa collection de Frameworks et librairies nous permettra un développement plus rapide de notre backend. Cependant, en creusant les caractéristiques plutôt 'pytonesques' permet d'avoir une architecture élégante du code.

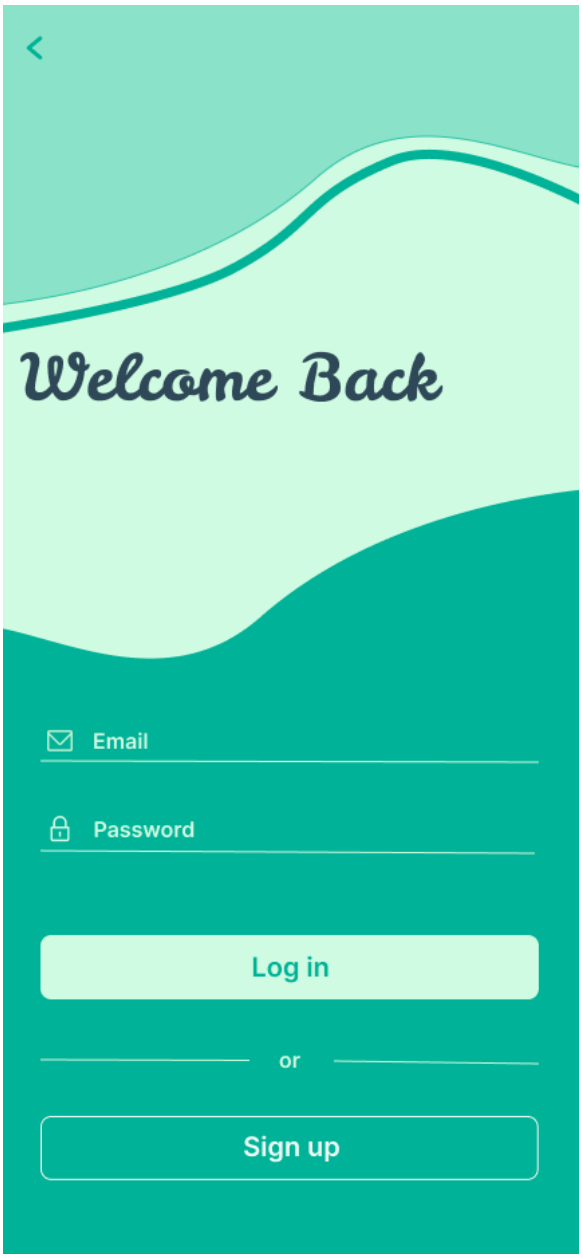
Sachant qu'il s'agit d'une grande partie de notre projet, la base de données sera en PostgreSQL parce que PostgreSQL est un des bases de données la plus mature, avec beaucoup de soutien de l'industrie et de la communauté SQL.

Ce qui concerne des librairies, les choix sont encore assez ouverts et on s'adaptera selon les besoins de l'application et de notre stade de développement. A date on compte utiliser ce qui suit:

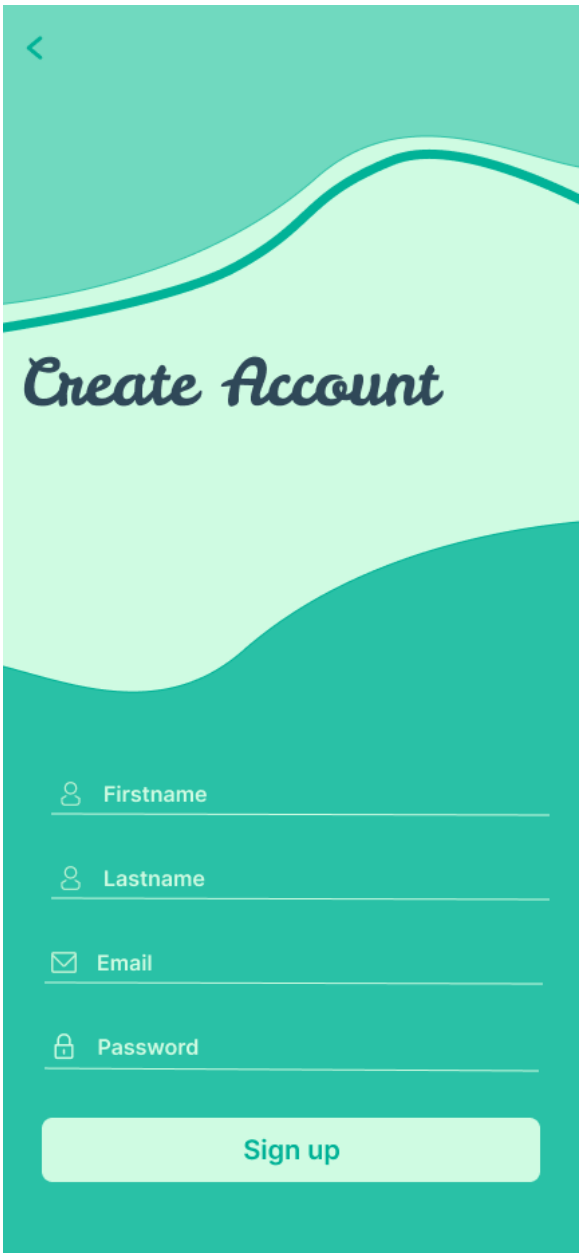
- swipeable: librairie de React pour implémenter des mouvements type ‘swiping’ qui imiteront les mouvements naturels du téléphone. Faut spécifier que le plan c’est de faire une application PWA (progressive web app) mais qui sera aussi accessible via le browser.
- API de géolocalisation en React pour extraire la longitude et la latitude de l’usager pour nos calculs.
- Flask pour le serveur HTTP dans le backend
- Jsonify pour recevoir le JSON du frontend et le transformer en structure native de Python (et l’inverse)
- Websocket IO pour le Chat instantané par websocket
- Sci-kit pour exploiter des algorithmes de clustering
- Psycopg pour intégrer le backend avec la base de données Postgres

2. Interfaces graphique utilisateur

Home Page

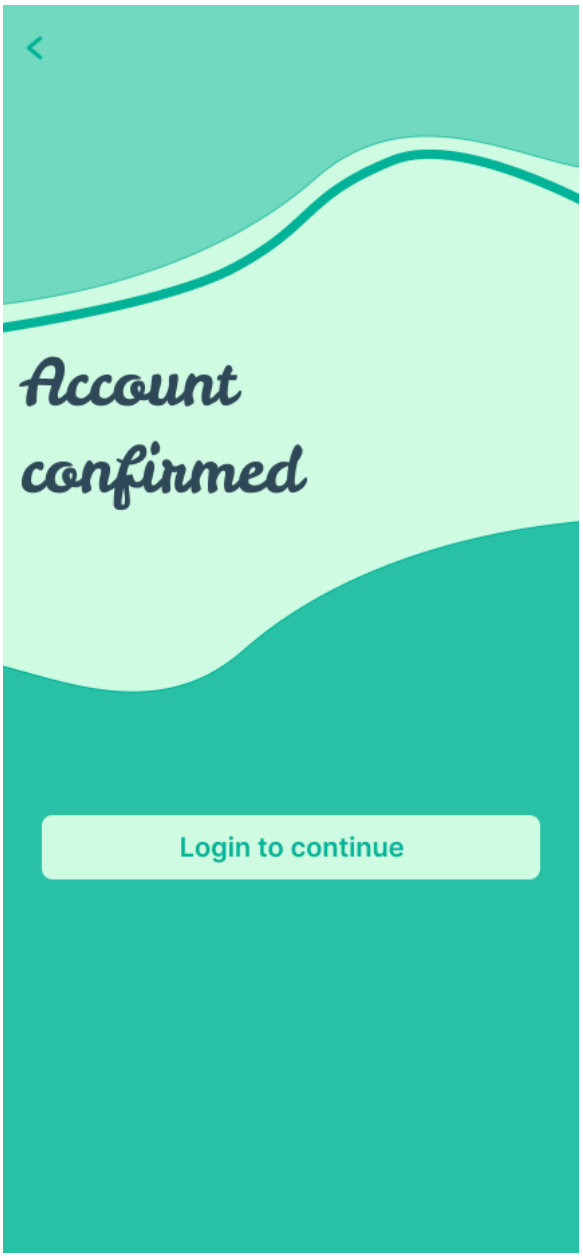


Login Page

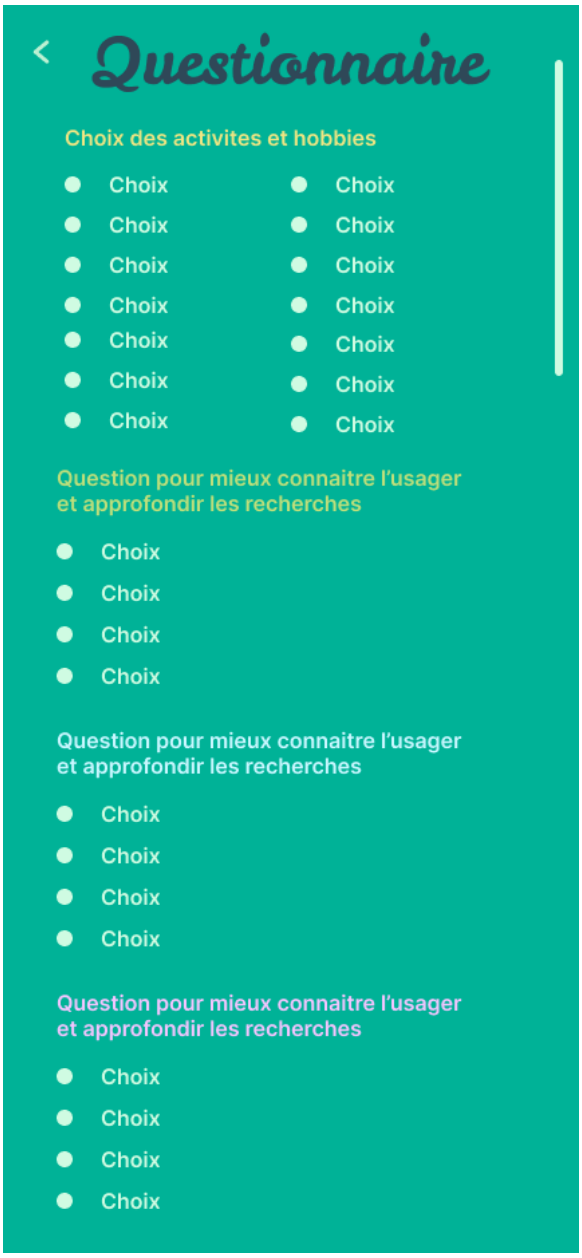


Register Page

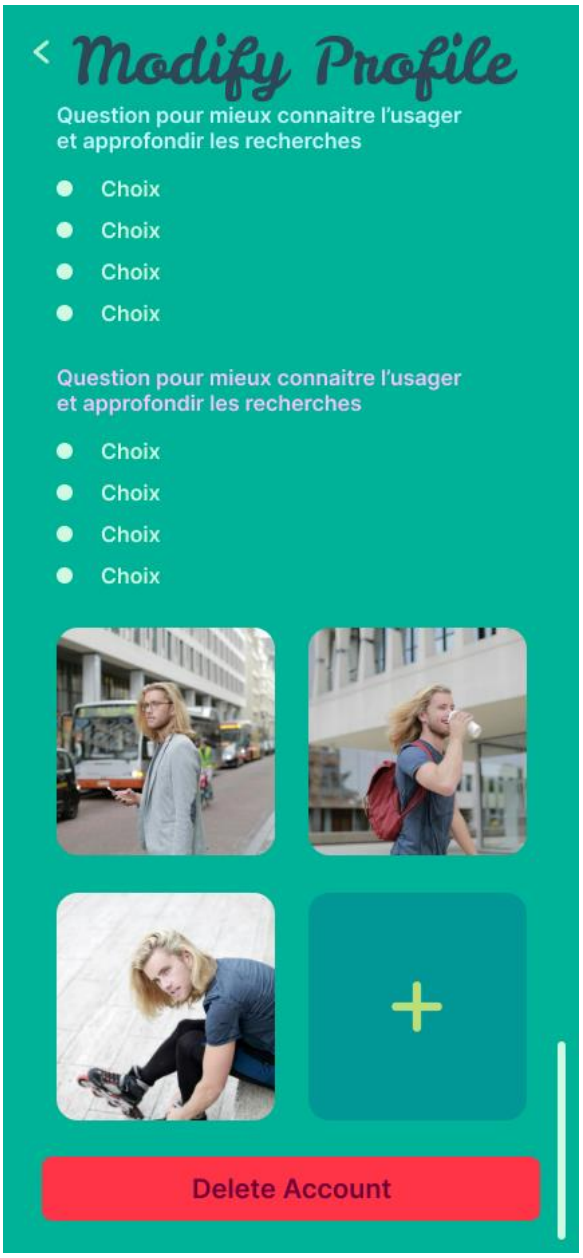
Waiting confirmation Page



Account Confirmed Page

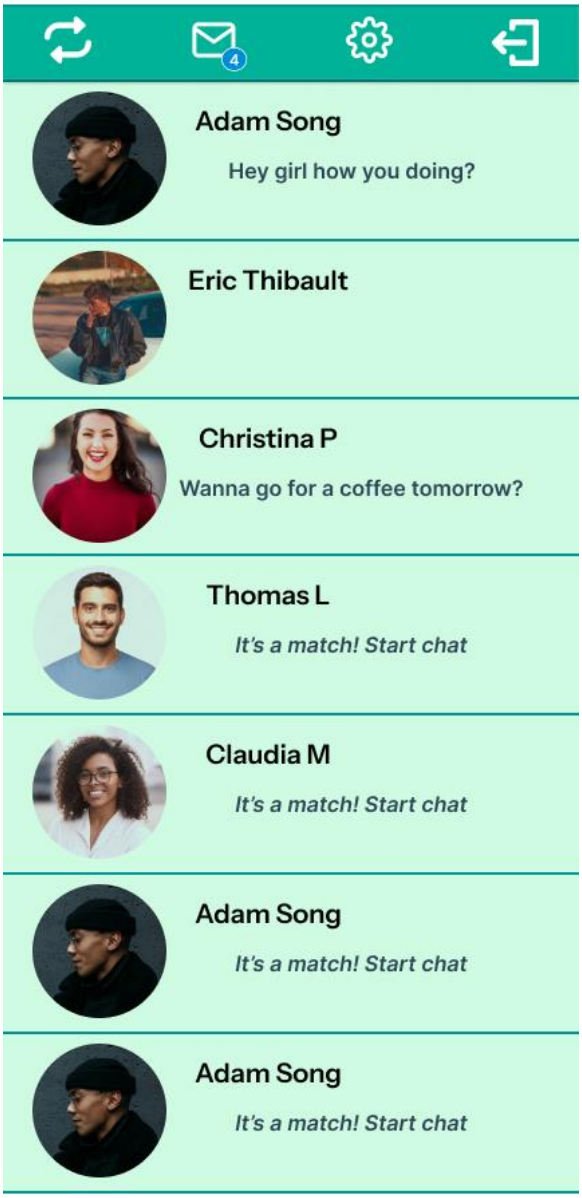


Survey Page

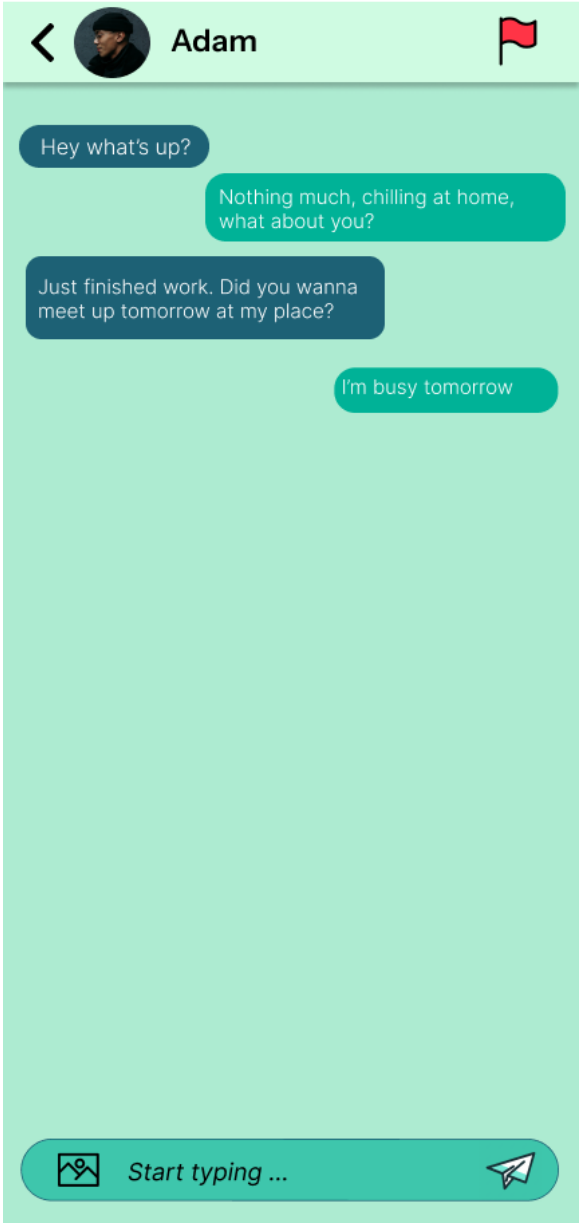


Modify Profile Page

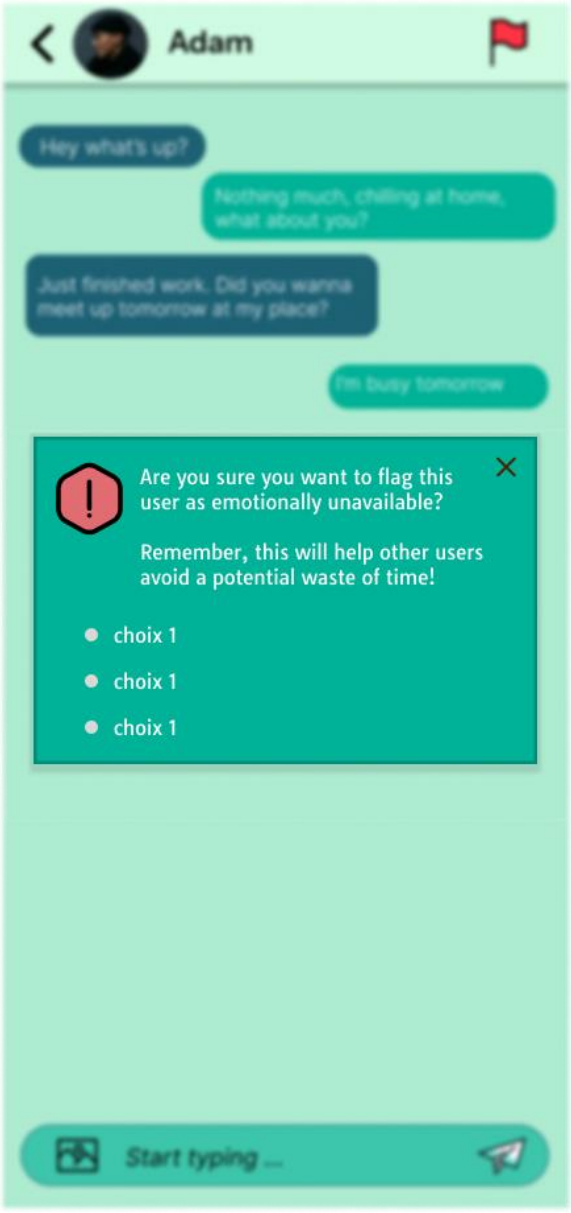
Matching Page



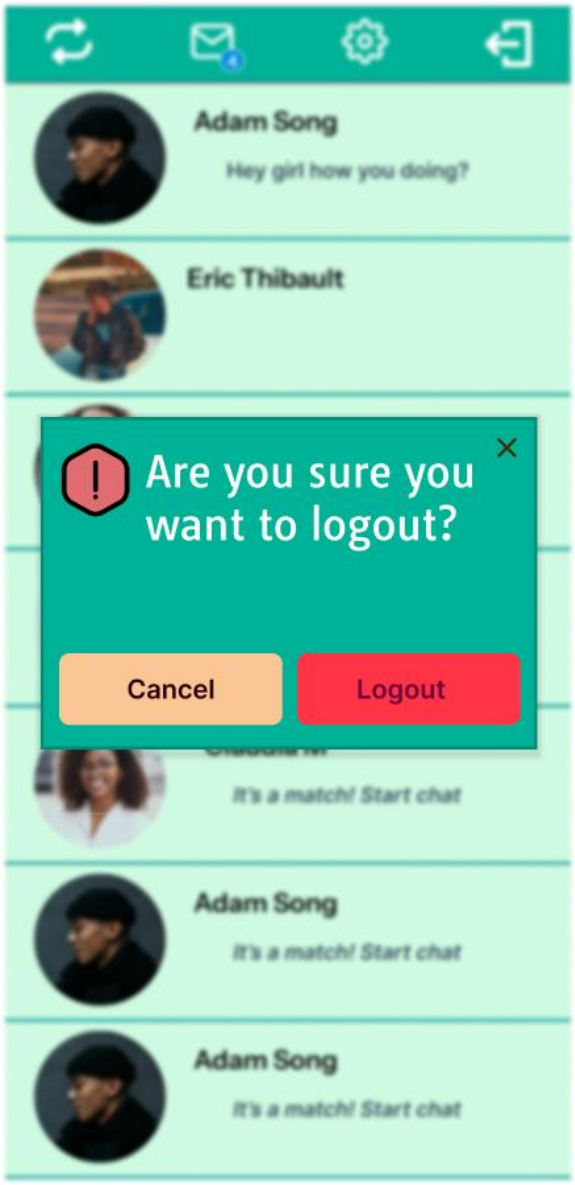
Chatrooms Page



Chat Page



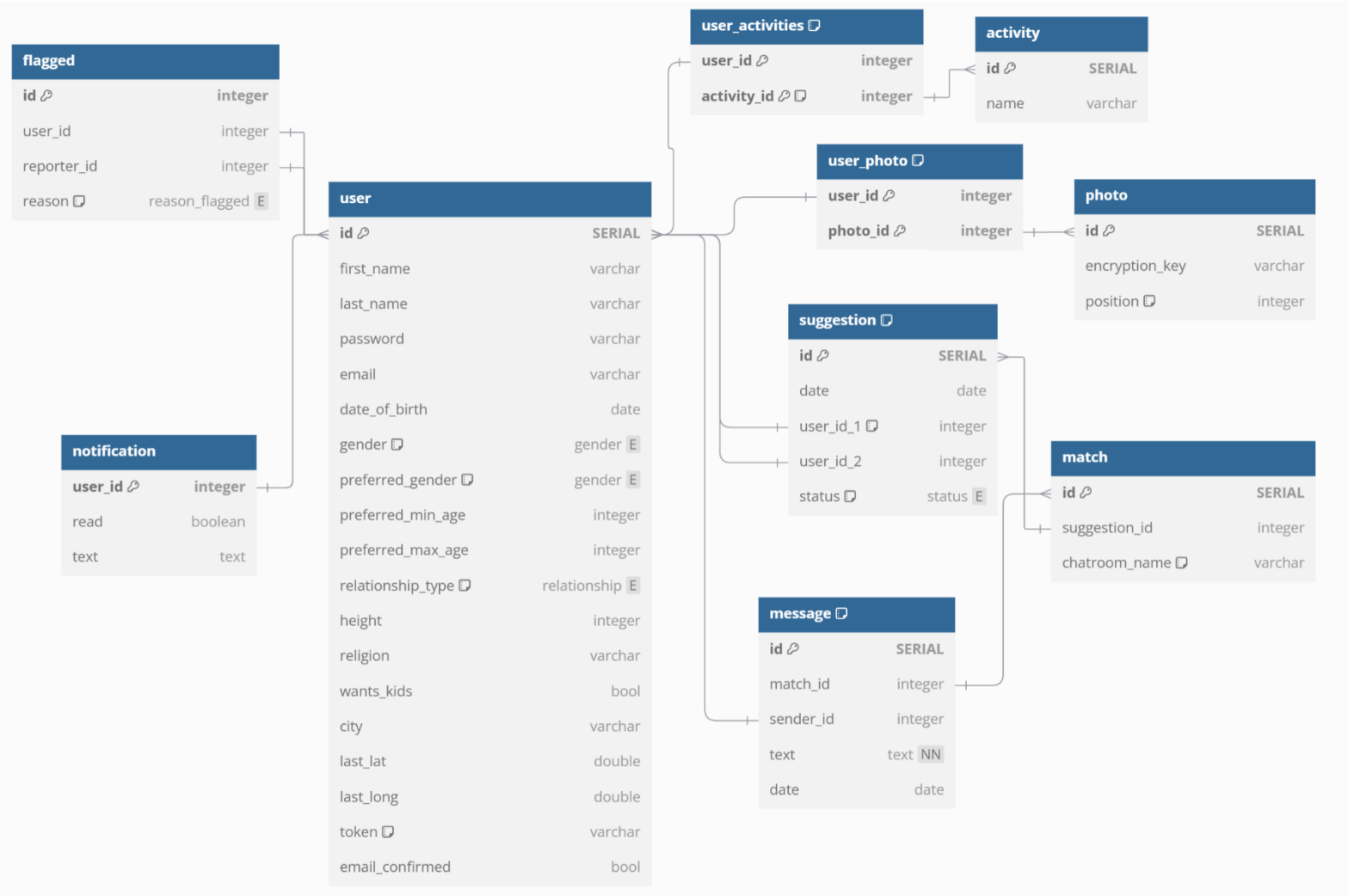
Message Flag



Message de confirmation de Logout

3. Données persistantes

Pour la sauvegarde de nos données persistantes, nous allons user principalement d'une base de données PostgreSQL. Celle-ci gardera l'entièreté de nos données excepter les images uploader par nos usagers. Les fichiers images seront garder dans une base de données Berkeley DB et leur clé d'identification sera garder dans la base de données PostgreSQL. Nous avons choisi PostgreSQL pour plusieurs raisons notamment l'usage de PL/pgSQL qui nous permettra de faire des fonctions propres à nos besoins, le fait qu'il soit ACID et ainsi assure une pérennité à nos données ou encore le fait qu'il supporte des grandes quantités de transactions à la fois. Sachant que nos données seront tous garder à la même place et que des milliers d'utilisateurs devraient pouvoir faire des opérations engendrant des requêtes a notre base de données simultanément, il semble que les qualités précédemment énumérées répondent parfaitement à notre cas particulier.



4. Structures de données

Discharged List – codé

Dans le traitement des suggestions et la communication constante avec la base de données pour la mise a jour de ces suggestions pour chaque usager, il y aura beaucoup d'informations reçues du frontend qu'on voudrait traiter de façon regroupée dans une seule requête SQL. Nous avons 2 contraintes à respecter : une temporale et une de capacité.

Cette structure de données est personnalisée à nos besoins car elle possède une classe abstraite d'observateur définie à l'intérieur qui sera regroupé dans une liste d'observateurs, et eux seront appelés quand l'une de nos limites est atteinte, qui va permettre le traitement de nos suggestions.

Tuple

Cette structure de donnée immuable et peu gourmande en mémoire pourra nous servir à stocker temporairement des attributs de l'usager qui ne changeront pas (tel que nom, prénom, genre, etc.).

Aussi, on utilisera probablement le concept de look up table pour associer les intérêts aux index directement au lieu d'aller avec un dictionnaire.

Liste

Les listes incluant les NumPy arrays seront très utiles dans notre application: premièrement, tout l'algorithme de matching en arrière travaillera avec des np.arrays qui vont accélérer la vitesse de travail surtout parce qu'on aura beaucoup d'usagers avec leurs propres intérêts à regarder.

Dictionnaire

Le dictionnaire pourrait être utile à construire des arborescences d'usagers avec leurs informations de base, leurs intérêts, et ainsi de suite à l'intérieur - ça peut devenir intéressant car la clé est un simple string, mais le value sera des tuples, listes etc. Aussi on trouve intéressant l'idée de faire une association de clé avec une fonction dans un dictionnaire – on l'utilisera possiblement dans la classe User.

Set

Dans notre projet, nous avons un système de notification qui dans le backend va faire des requêtes aux secondes afin d'obtenir les dernières notifications. Malgré le fait que nous nous assurions de ne pas recevoir de copie à l'aide d'un statut "pending" dans notre base de données afin de savoir si les notifications ont déjà été envoyés, il est possible que nous recevions une copie d'une notification si la mise à jour du statut n'est pas assez rapide pour qu'il soit fait avant la prochaine requête. Pour parer à ce problème, un set nous permettra d'éviter les copies reçues.

File (Queue)

Vu l'ampleur du nombre de requêtes que l'on compte faire vers la base de données, l'usage d'une File semble évident pour pouvoir palier à un flux important d'entrant sortant. Son fonctionnement de premier arrivé premier sortie est parfait dans notre cas, puisque l'on désire pouvoir ajouter autant de requête à la queue au fur et à mesure qu'ils arrivent et que la plus ancienne requête insérée est toujours la priorité sur les autres.

5. Patrons de conception

Décorateur

Le patron de conception du décorateur est classiquement implémenté comme des objets qui peuvent englober autres objets, sous la condition que l'objet cible et l'objet décorateur suivent la même interface. Ça permet d'avoir une suite de comportements empilés. En Python, nous pouvons accomplir un comportement similaire mais en faisant une fonction englobante, qui servira comme décorateur pour une autre fonction (@fonction_decoratrice). Ça permet à la fonction originale d'exécuter son propre code et garder sa logique interne, mais au même temps exiger un travail supplémentaire par le décorateur qui va influencer le résultat final.

Dans la logique interne d'une bonne application de haute sécurité se trouve une session temporaire ou l'usager a accès à l'application - sous condition qu'il a le droit d'être là, c'est à dire il s'est authentifié avec le bon identifiant et mot de passe associé. Dans ce cas, un jeton JWT (JSON Web Token) sera généré par le backend et stocké dans la base de données, encodé - pendant un temps prédéterminé.
Le processus de vérifier la validité d'un token sera une fonction type décorateur (wrapper - @token_required) qui sera ajouté pour des fonctions internes ou la validité d'un token est importante, par exemple modifier son profile ou se connecter, ou rester connecté après que l'usager a été authentifié.

En conclusion, ça permet une encapsulation du processus de validation des jetons et l'appliquer ou c'est pertinent.

Strategy

Le patron de conception du Strategy sera implémenté dans 2 endroits clés dans notre architecture backend:

1. *Stratégie pour algorithme*

Notre système de suggestions pour le Matching fonctionne à l'aide d'un algorithme qui calcule les usagers à suggérer potentiellement intéressant pour notre utilisateur. Nous avons dans l'intention de pouvoir possiblement faire usage de plusieurs algorithmes de clustering différents et pour faciliter la transition entre ceux-ci l'usage d'un modèle de conception Strategy nous semble tout à fait adapté. Les algorithmes auront tous des méthodes fit, "predict" et "get_clusters" et auront leur propre implémentation de ces méthodes. L'usage du design Stratégie nous permettrait de diversifier les résultats obtenus en usant comme on le désire des différents algorithmes

2. *Stratégie pour simulateur*

Quand nous serons arrivés au moment qu'on a créé des milliers d'usagers, on voudra simuler le comportement de swiper sur des possibles partenaires et faire des matchs. On aimerait simuler la vraie vie, c'est à dire en réalité, une portion des usagers sont extrêmement ciblés dans leurs recherches donc eux vont finir par dire oui au seulement 10% des suggestions offerts (réellement 10%, ou comme condition faut que la personne acceptée partage au moins 4 intérêts en commun avec l'autre, etc.).

Des autres sont plus équilibrés, donc vont accepter plus, par exemple 35% minimum. A l'autre extrême, il y a des usagers qui sont prêts à accepter n'importe qui, dont ils acceptent tous.

Tous ces comportements pourraient représenter des ‘SwipingStrategy’ différentes avec une hiérarchie spécifique des classes.

Observer

L’observer est patron de conception qui nous permet de créer un système d’écoute d’un objet a plusieurs objets abonnés. Lorsque l’état de l’objet sujet change, tous les membres seront notifiés et exécuteront leurs mécanismes respectifs.

Dans le cadre de notre projet, nous allons coder une structure de données personnalisée avec la classe observateur à l’intérieur. Notre gestionnaire de suggestion sera un observateur de cette structure de données et va agir en conséquence si l’état de celle-ci change.

DAO

Le DAO est largement utilisé dans la connexion avec des systèmes de données persistantes. Le concept est d’avoir un pont entre les données persistantes et le programme qui fonctionne peu importe d’où elles viennent. Cela rend flexible les mécaniques de sauvegarde et récupération de données. L’implémentation de nouveau système et leur modification est ainsi faciliter puisque les services qui usent du DAO n’ont pas besoin de savoir quel service donne quelle information et comment.

Dans le cadre de notre projet, l’implémentation du DAO se fera plus comme une abstraction de la connexion avec la base de données. Nous aurons différents DAO qui se chargerons de plusieurs services dont le chat pour la sauvegarde des messages et leur récupération, le compte pour la création et modification du compte et les suggestions pour la sauvegarde et la récupération des suggestions, ainsi que les notifications.

Factory

Le patron de conception créationnel de Factory pourrait être utile pour créer des objets aves comportements différents, mais qui a la base proviennent de la même source. Pour nous c’est pertinent car on va vouloir implémenter des simulations de milliers d’usagers crées tous avec leurs propres préférences qui vont interagir entre eux.

Nous allons avoir un UserFactory, qui peut créer, par exemple, des usagers qui auront une préférence différente de qu’est-ce qu’ils recherchent – selon la relation, ou des préférences d'âge, etc. On aura des User qui auront des préférences larges, ça veut dire qu’ils préfèrent n’importe quel âge, peut-importe si la personne recherchée veut des enfants, bref, ils sont ouverts à tout. Autres User auront une tranche d'âge recherché très étroite (qui va influencer aussi la quantité de suggestions offerts).

Adapter

Quand un usager créera un compte, il sera invité d’accéder à son courriel et cliquer un lien personnalisé qui va confirmer le compte. Le courriel sera envoyé via un Adapteur fait dans notre backend qui utilisera un service tiers pour les courriels – l'adapteur va servir à faciliter l’usage de ce service et l’adapter à notre code – notre backend va communiquer directement avec l’adapter et lui avec le service externe/API.

6. Développement d'une bibliothèque

Middleware

Le middleware est un programme modulaire qui sert de pont entre le frontend et le backend qui traite les requêtes avant qu'elles n'atteignent le serveur ou la réponse avant qu'elle ne soit renvoyée au client. Notre middleware sera codé en Python. Il peut être utilisé pour vérifier la validité des sessions. Cette architecture permet d'assurer une meilleure gestion de la sécurité et s'assurer que des requêtes vers le fond du backend et la base de données ne seront pas traités si la session est expirée.

La classe AuthenticationMiddleware aura la charge de vérifier la validité des sessions en cours : chaque fois qu'un usager se connecte, un jeton JWT (JSON web token) sera généré qui contient des informations pertinentes de l'usager et qui expire dans 1 heure habituellement. Essentiellement, chaque fois que le frontend veut demander quelque chose du backend, le middleware est responsable, comme une barrière, de laisser la requête passer ou non. Si la session est expirée, il redirigera l'utilisateur vers la page de connexion, l'empêchant ainsi d'accéder aux fonctionnalités protégées de l'application sans être authentifié.

Discharged List

Une Discharged List est liste qui se vide selon si elle atteint une taille limite définie ou un laps de temps entre la première insertion et la dernière. Ces deux paramètres pourront être configurés selon l'usage que l'on désire en faire. L'utilité d'une telle structure de données est que lorsque l'on a une quantité importante d'informations ou d'actions que l'on veut traiter, on puisse les traiter en groupe au lieu d'une après l'autre.

Une des caractéristiques intéressantes de cette liste est une classe d'observer définie à l'interne. Ceci est une classe abstraite, avec la fonction interne process(). Toutes les classes qui vont dépendre de l'état de la liste vont hériter d'Observer.

La structure de données aura une liste d'observateurs qui pourront être appelés lorsque la structure atteint une des facteurs-limite : par exemple, on regardera le temps d'inertie, c'est-à-dire combien de temps a passé entre le dernier ajout et l'ajout demandé à l'instant.

Si trop de temps a passé, chaque observateur, à tour de rôle, exécutera sa fonction interne (process()), au travers laquelle elle enverra la liste interne des données. Si toutes les itérations de la fonction process() seront exécutés sans erreur, une confirmation générale permettra a la DichargedList de se vider et faire de la place aux nouvelles demandes du système.

File (Queue)

Une File est une liste chainée qui fonctionne d'une façon particulière. Elle suit la logique premier arrivé premier sortie (FIFO). Elle est utile lorsque l'on veut que la donnée insérée la plus ancienne est prioritaire face aux autres et doit être la première à sortir. La File est souvent utilisé dans le traitement du trafic des requêtes http sur les sites internet.

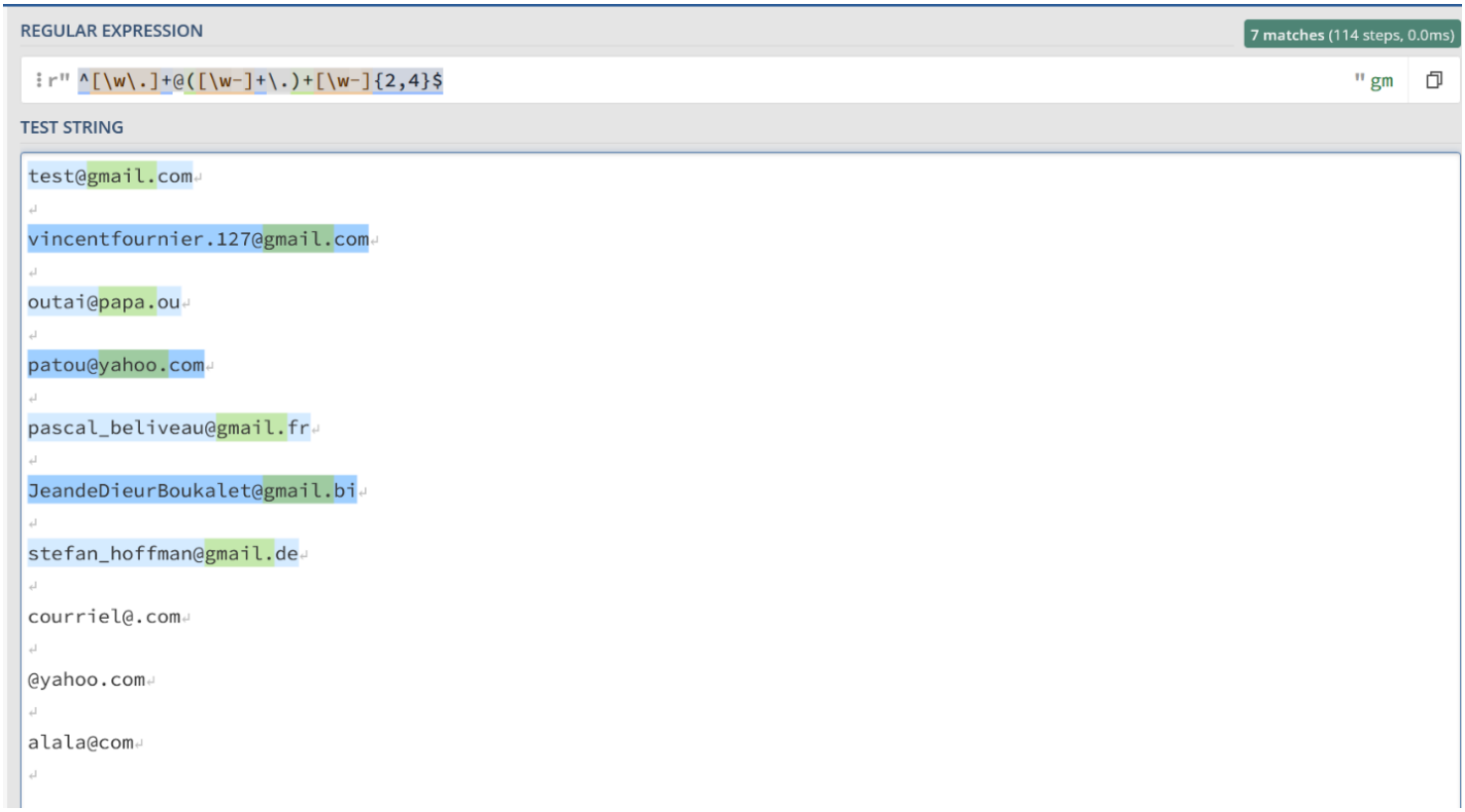
Dans notre projet, elle sera utilisée pour un problème similaire, soit le trafic des requêtes à la base de données. Étant donné que la majorité des opérations dans notre application doit faire usage de la base de données, un bouchon de requêtes va forcément se créer si l'on n'use pas d'une file pour gérer les requêtes une à la fois. Notre file devra être programmé de façon qu'elle puisse être utilisée peu importe la situation.

Elle devra donc comporter des fonctions qui fonctionneront indépendamment des types de données sauvegarder.

7. Expression régulière

Lors de la création d'un compte utilisateur, nous demanderons le courriel de celui-ci comme identifiant et afin de lui envoyer un courriel de vérification. Il nous semble approprié de vérifier le courriel à l'aide d'une expression régulière et ainsi s'assurer qu'il s'agit d'une adresse courriel valide.

L'expression régulière : ^[\w\.]+@([\w-]+\.)+[\w-]{2,4}$



8. Algorithme

Dans le cadre de notre projet, il nous faut suggérer des usagers a d'autres. Pour avoir les meilleures suggestions, il nous faut trouver les utilisateurs ayant des centres d'intérêts communs sans pour autant les étiqueter. L'algorithme qui nous semble le plus propice à répondre à ce problème serait le Meanshift, car ce dernier permet de regrouper des données selon leurs distance euclidienne sans qu'on le dirige.

Son fonctionnement est assez simple. Tout d'abord une zone est définie pour chaque point avec ce dernier en son centre. Puis les points dans cette zone sont utilisés afin de trouver un nouveau centre en trouvant la moyenne pondérée de ceux-ci. Pour finir, l'étape d'avant est répétée jusqu'à ce que le mouvement de ce ou ces points centraux soit minime. Nous finissons par obtenir le centroïde de nos clusters, ainsi de tous les points avec le cluster auquel ils sont attribués.

Aucun changement est nécessaire pour l'adaptation de l'algorithme a notre projet. Ce dernier est parfaitement adapté au problème que l'on tente de résoudre. Cependant, il nous faudra possiblement ajuster les paramètres de l'algorithme ultérieurement afin de palier avec la quantité de plus en plus grandissante d'usager.

9. Mathématique

Nous allons collectionner les informations qui concerne le comportement de nos usagers. Par exemple, lorsqu'un usager veut signaler quelqu'un, il aura besoin de choisir pourquoi. Ces choix seront enregistrés et plus tard analysés avec des statistiques descriptives essentielles tel que moyenne, médiane, variance.

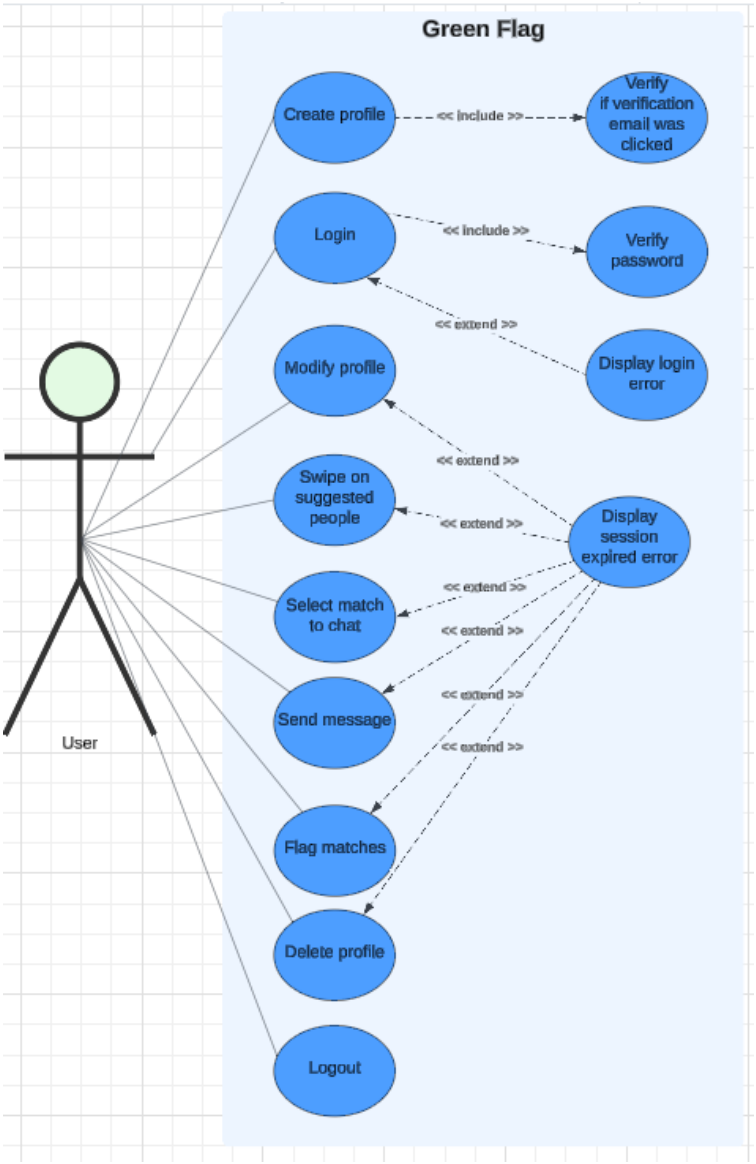
$$\overline{X} = \frac{\sum X}{N} \qquad \sigma^2 = \frac{\Sigma(xi - \bar{x})^2}{N}$$

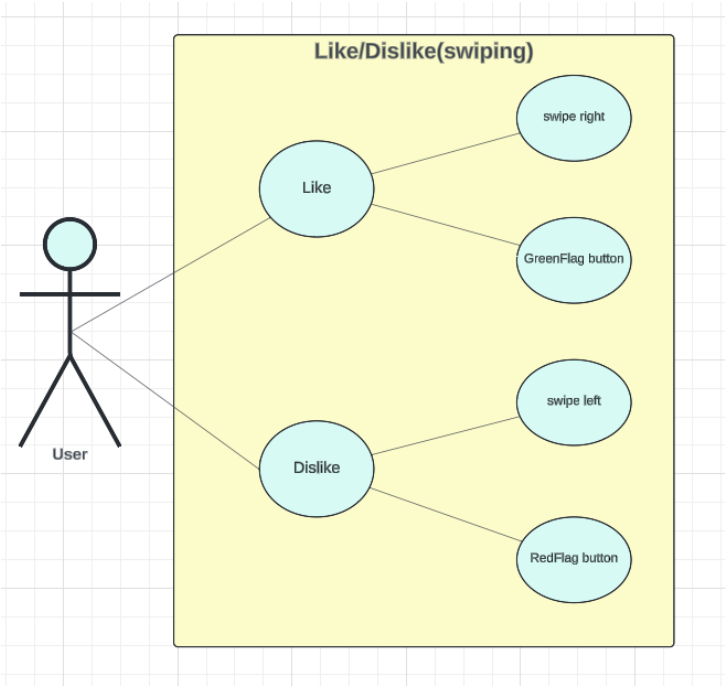
On peut aussi estimer le pourcentage de nos usagers qui se font signalés. De plus, on pourrait analyser les usagers qui ont possiblement un comportement qui démontre la promiscuité, tel qu'envoyer des messages a beaucoup de gens, qui swipent oui sur beaucoup de profiles, en analysant le comportement général et calculant le z-score pour détecter comment eux se démarquent.

$$z = \frac{x - \mu}{\sigma}$$

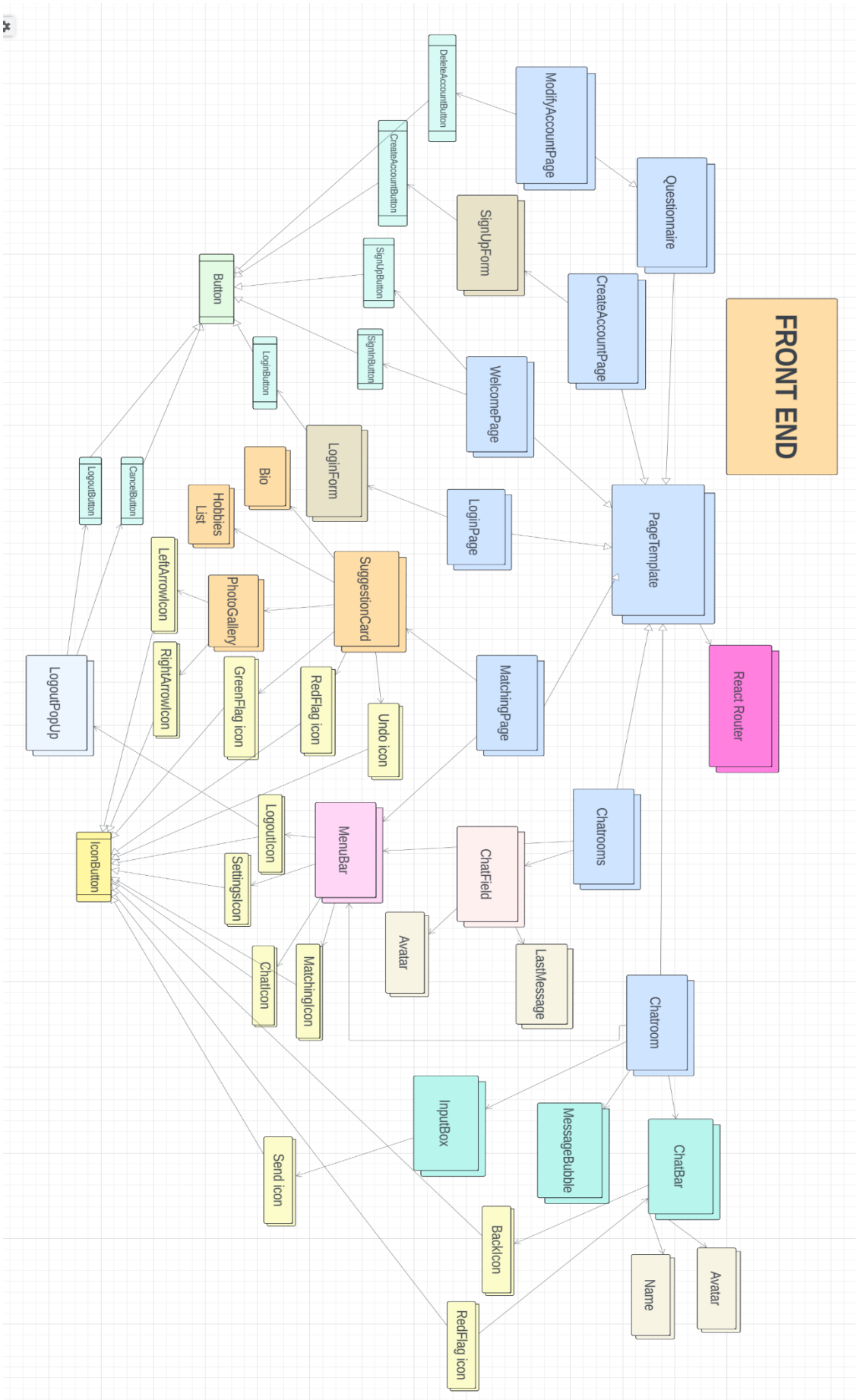
10. Conception UML

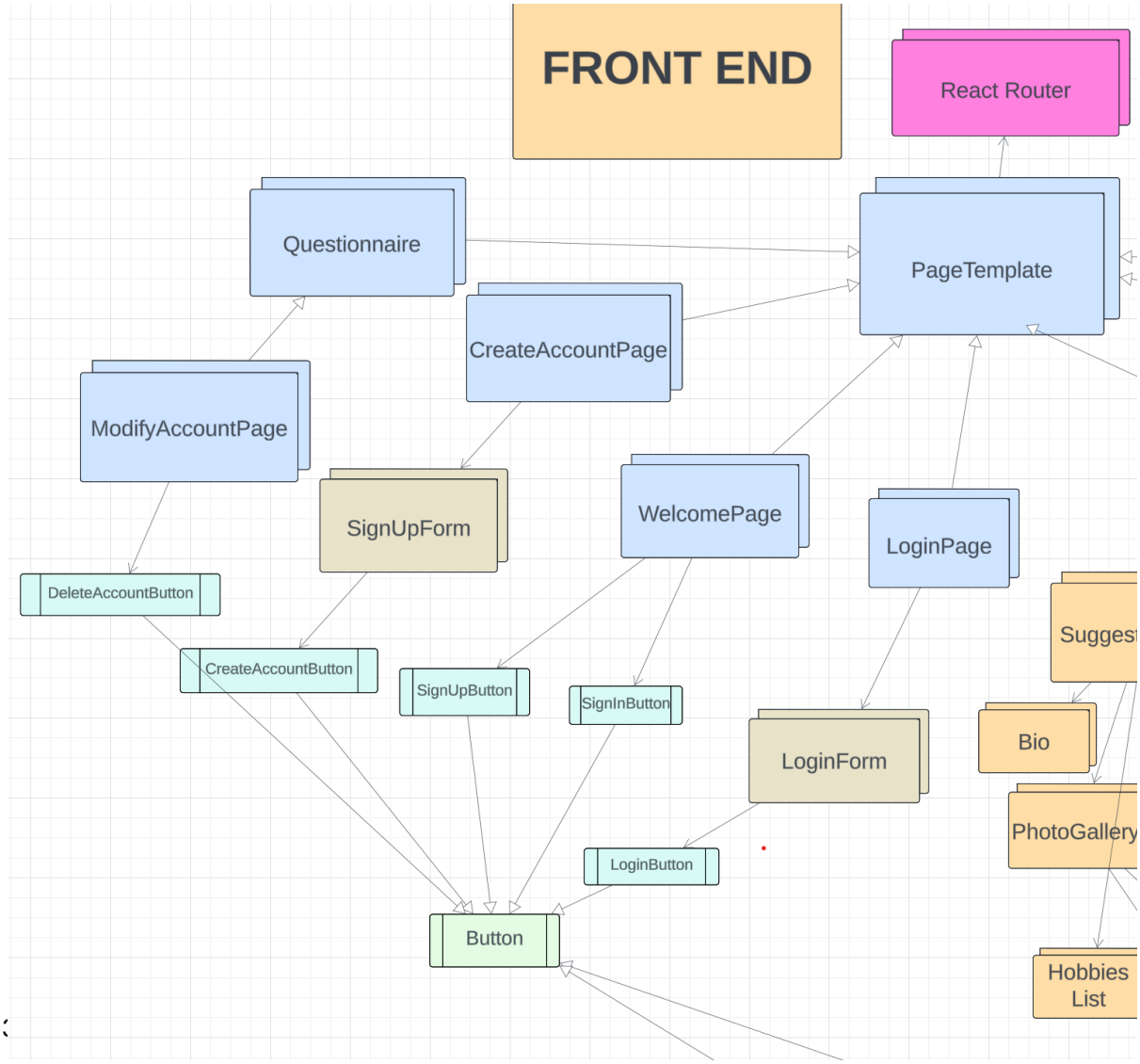
1. Diagrammes de cas d’usage

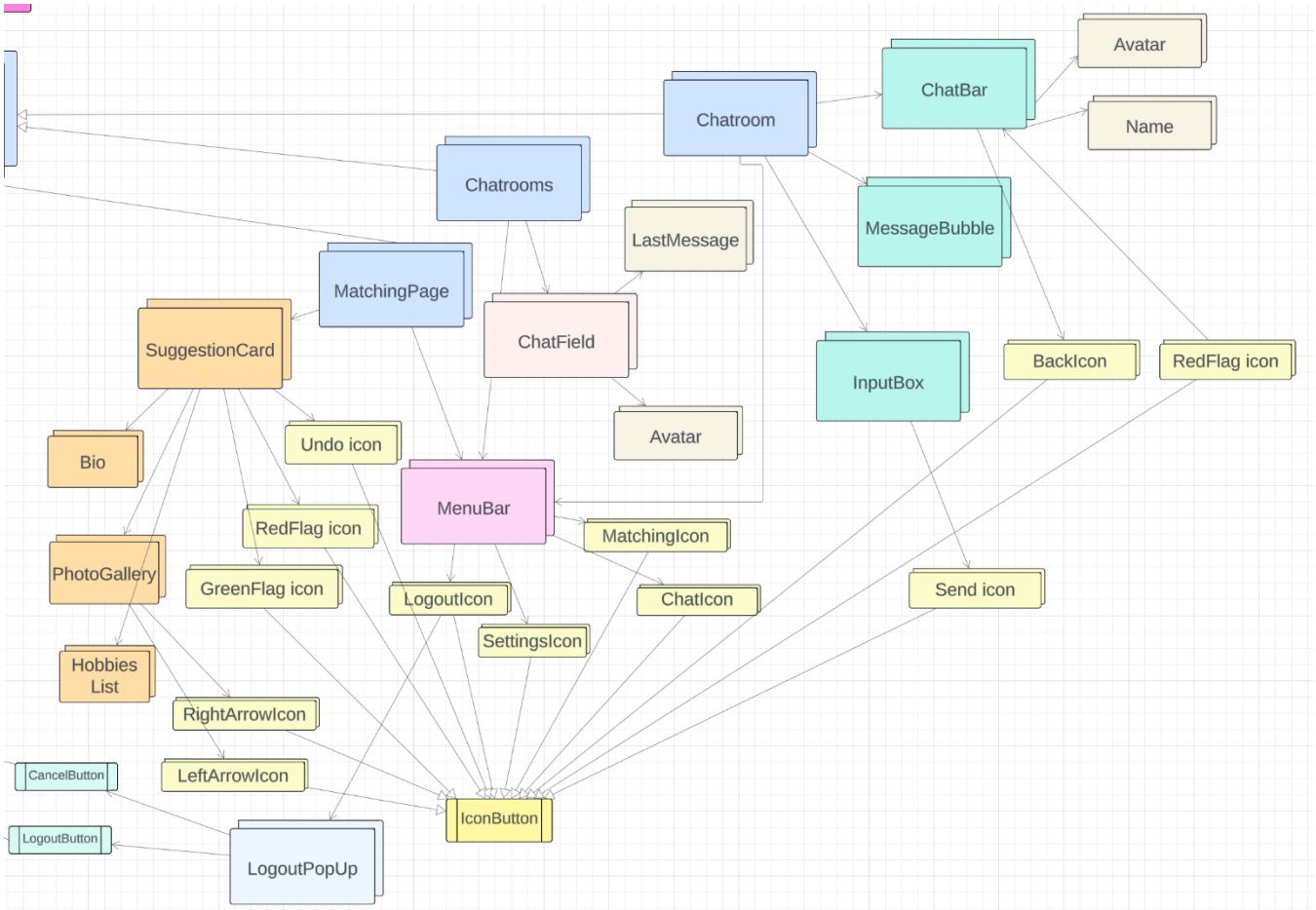




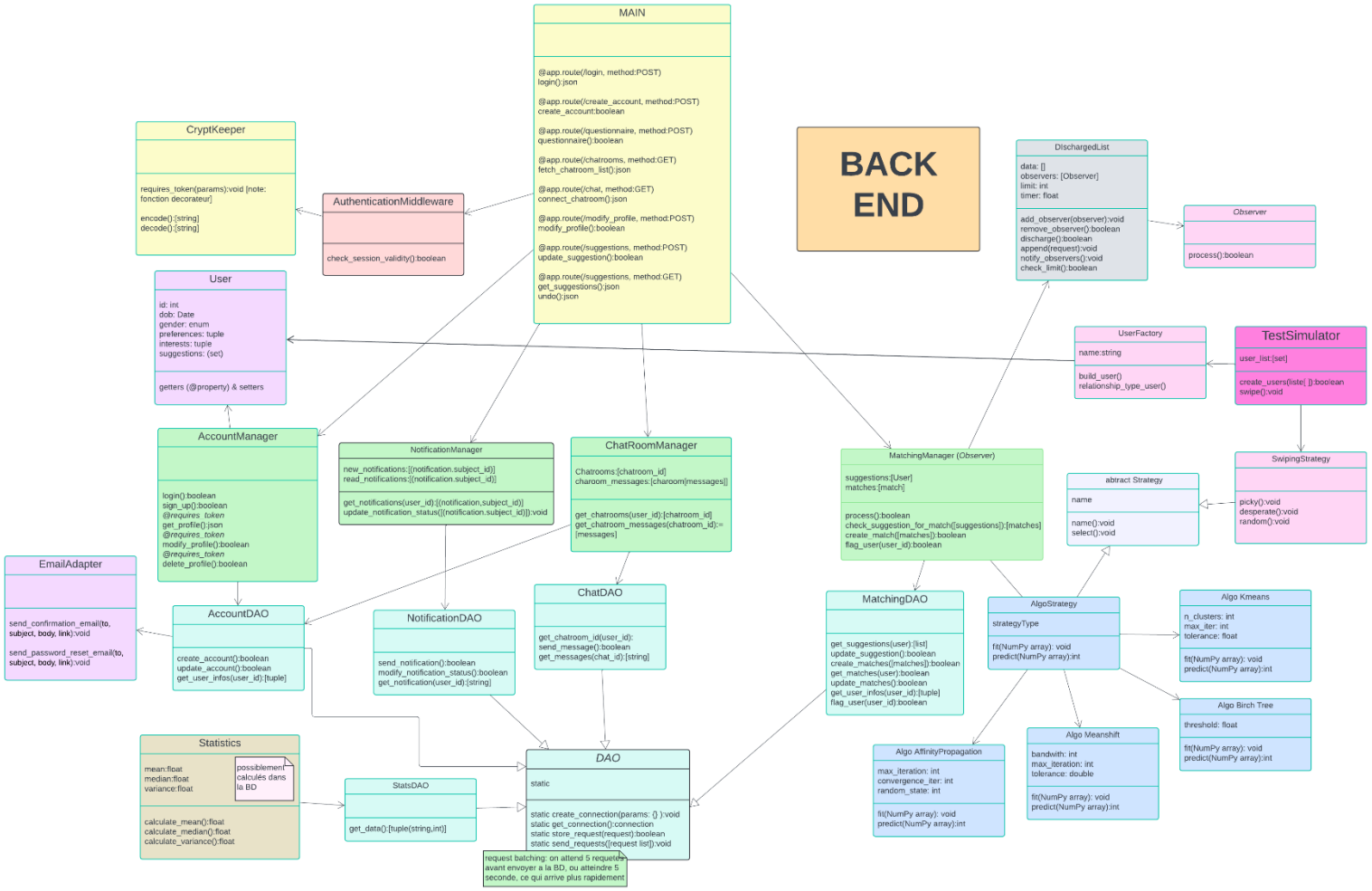
2. Diagrammes de classe



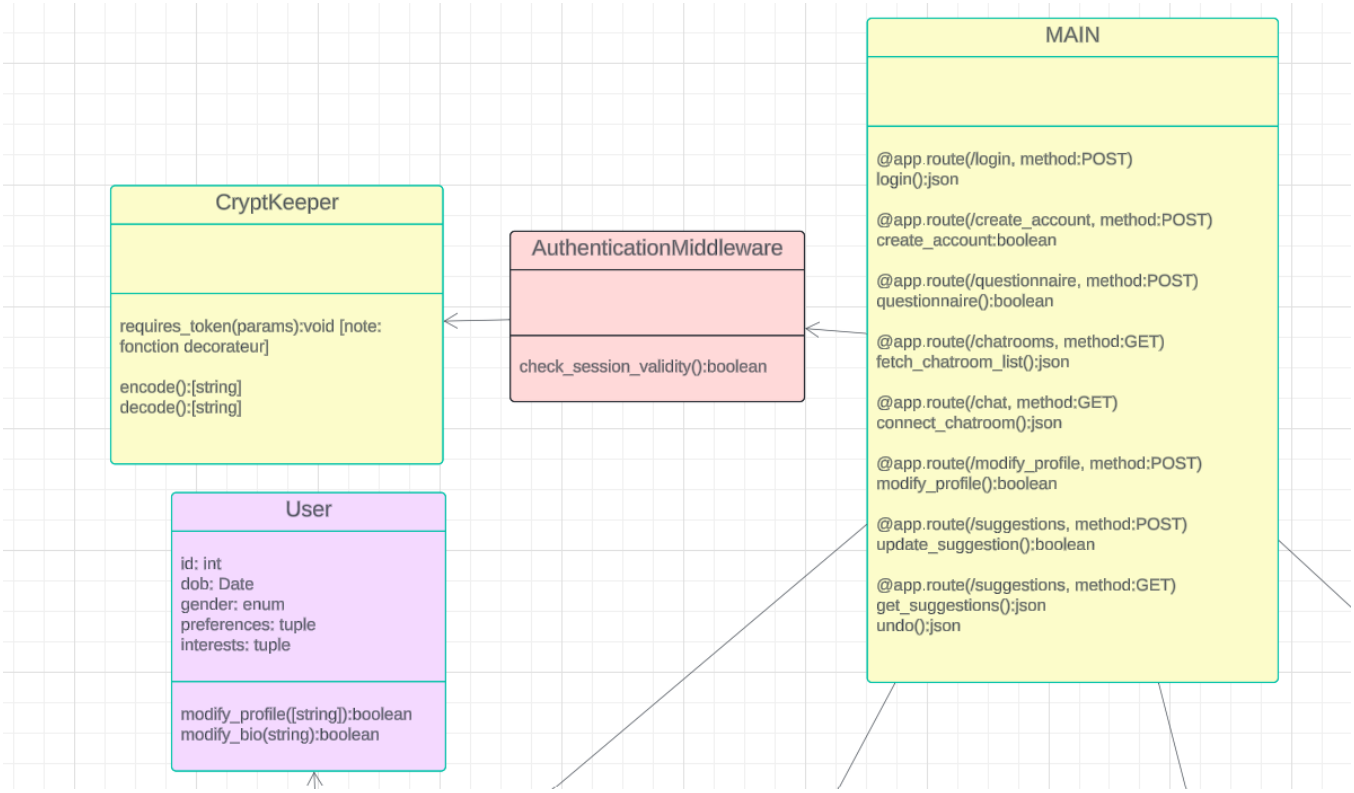

FRONT END
React Router
Questionnaire
PageTemplate
CreateAccountPage
ModifyAccountPage
SignUpForm
WelcomePage
LoginPage
DeleteAccountButton
CreateAccountButton
SignUpButton
SignInButton
LoginForm
Bio
PhotoGallery
LoginButton
Button
Hobbies List

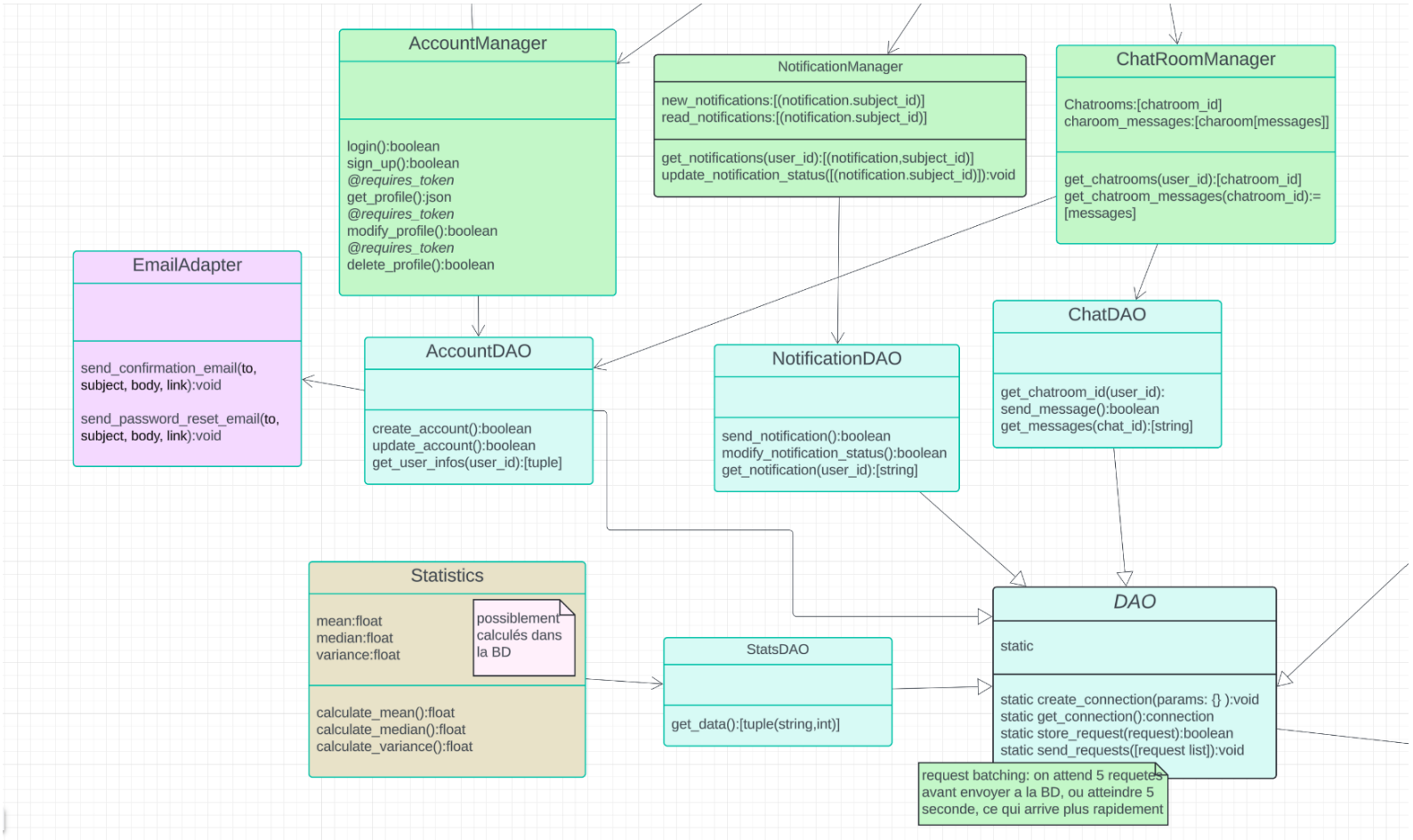

Chatroom
ChatBar
Avatar
Name
Chatrooms
MessageBubble
LastMessage
MatchingPage
SuggestionCard
ChatField
InputBox
BackIcon
RedFlag icon
Bio
Undo icon
Avatar
MenuBar
RedFlag icon
MatchingIcon
PhotoGallery
GreenFlag icon
LogoutIcon
ChatIcon
Send icon
SettingsIcon
Hobbies List
RightArrowIcon
CancelButton
LeftArrowIcon
IconButton
LogoutButton
LogoutPopUp

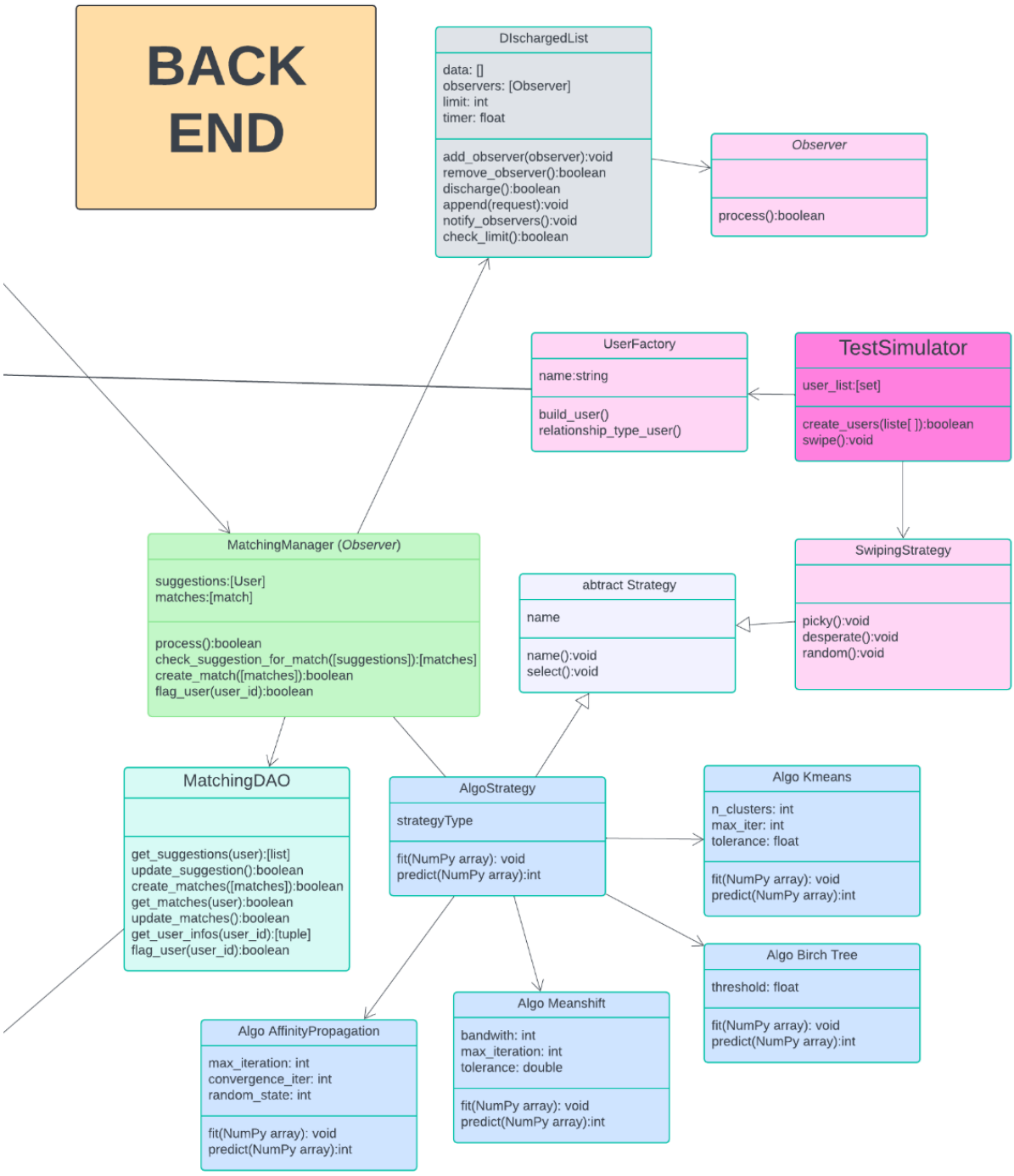
2. Back end vue d'ensemble



3. *Back end – vue détaillé par section*






BACK END
DIschargedList
data: []
observers: [Observer]
limit: int
timer: float
add_observer(observer):void
remove_observer():boolean
discharge():boolean
append(request):void
notify_observers():void
check_limit():boolean
Observer
process():boolean
UserFactory
name:string
build_user()
relationship_type_user()
TestSimulator
user_list:[set]
create_users(liste[ ]):boolean
swipe():void
MatchingManager (Observer)
suggestions:[User]
matches:[match]
process():boolean
check_suggestion_for_match([suggestions]):[matches]
create_match([matches]):boolean
flag_user(user_id):boolean
SwipingStrategy
picky():void
desperate():void
random():void
abtract Strategy
name
name():void
select():void
MatchingDAO
get_suggestions(user):[list]
update_suggestion():boolean
create_matches([matches]):boolean
get_matches(user):boolean
update_matches():boolean
get_user_infos(user_id):[tuple]
flag_user(user_id):boolean
AlgoStrategy
strategyType
fit(NumPy array): void
predict(NumPy array):int
Algo Kmeans
n_clusters: int
max_iter: int
tolerance: float
fit(NumPy array): void
predict(NumPy array):int
Algo Birch Tree
threshold: float
fit(NumPy array): void
predict(NumPy array):int
Algo Meanshift
bandwith: int
max_iteration: int
tolerance: double
fit(NumPy array): void
predict(NumPy array):int
Algo AffinityPropagation
max_iteration: int
convergence_iter: int
random_state: int
fit(NumPy array): void
predict(NumPy array):int

Annexe

Références

https://refactoring.guru/design-patterns/decorator
Information sur le design de conception Décorateur

https://refactoring.guru/design-patterns/factory-method
Information sur le design de conception Factory

https://realpython.com/factory-method-python/
Exemple d'une Factory en python

https://refactoring.guru/design-patterns/observer
Information sur le design de conception Observateur

https://refactoring.guru/design-patterns/strategy
Information sur le design de conception Stratégie

https://scikit-learn.org/stable/modules/clustering.html
Information sur les algorithmes de groupement

https://docs.python.org/3/tutorial/datastructures.html
Information sur les structures de données notamment la file et le set

https://www.geeksforgeeks.org/queue-data-structure/
Information sur les structures de données files