

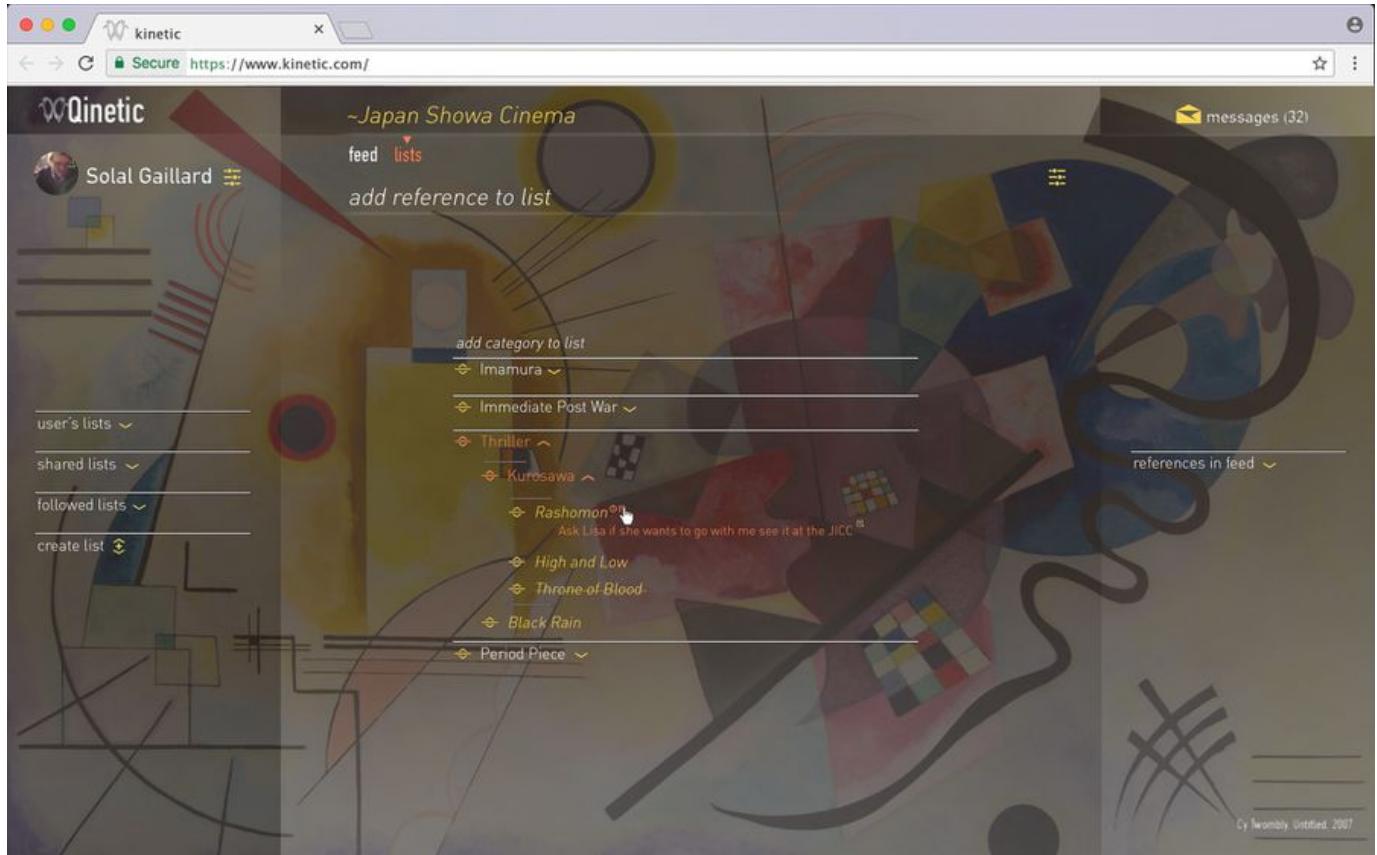
 **kinetic**

documentation technique

& utilisateur

1. Qinetic	2
1.1 Design	5
1.1.1 UX/UI Éléments Généraux	6
1.2 Architecture	6
1.2.1 Front end - Etats de l'application & Composants react.js	6
1.2.1.1 Routes de React Router & Composants associés	6
1.2.1.1.1 /	7
1.2.1.1.2 /home	14
1.2.1.1.3 /settings	19
1.2.1.1.4 /search?list={search-term}	20
1.2.1.1.5 /list/{list-id}/feed	20
1.2.1.1.6 /list/{list-id}/settings	23
1.2.1.1.7 /list/{list-id}/by-category	25
1.2.1.1.8 /list/{list-id}/by-scheduling/list	25
1.2.1.1.9 /list/{list-id}/by-scheduling/calendar	26
1.2.1.1.10 /list/{list-id}/by-priority	27
1.2.1.1.11 /list/{list-id}/by-rating	27
1.2.1.1.12 /list/{list-id}/by-experiencing	28
1.2.1.1.13 /search?reference={search-term}	28
1.2.1.1.14 /reference/{reference-id}	28
1.2.1.1.15 /search?user={search-user}	28
1.2.1.1.16 /user/{user-id}	28
1.2.1.1.17 /search?hashtag=search-term	28
1.2.1.1.18 /home/inbox	28
1.2.1.1.19 Composants Multi-routes	28
1.2.1.1.20 Composants Agnustics	33
1.2.1.1.21 Super Composants	33
1.2.1.2 Modèle du Store Redux	33
1.2.2 Back end	39
1.2.2.1 API Doc & REST Routes	39
1.2.2.1.1 Account Management API	39
1.2.2.1.2 User Management API	40
1.2.2.1.3 Feed Management API	41
1.2.2.1.4 List Management API	41
1.2.2.1.5 References Management API	41
1.2.2.2 Modèle pour RDBS	41
1.3 DevOps	41
1.4 Algorithmes complexes à implémenter	41

Qinetic



Les buts de l'application

Créer une communauté autour de contenus culturels :

Les utilisateurs doivent pouvoir partager entre eux des références à des contenus culturels (livres, musiques, films, expositions). Ils le feront en écrivant sur des feeds divers qu'ils peuvent juste observer ou bien qu'ils peuvent eux-mêmes administrer.

Organisation des références par liste thématique :

Les références partagées entre les utilisateurs doivent pouvoir être organisées et visualisées selon plusieurs paradigmes. Cela peut être par priorité, par un algorithme qui calcule les notes probables données, par un agenda ou par catégories, ou même par contenu déjà visionné, lu, etc...

Être autosuffisante / Ne pas dépendre d'applications tierces :

Les utilisateurs doivent générer les métadonnées des contenus qu'ils partagent, c'est-à-dire, les descriptions des références. Ces références doivent être uniques. Il faudra donc que le back end s'assure de l'unicité des références et le cas échéant les réunisse ensemble.

La plateforme doit être vivante / être une sorte de forum moderne :

L'idée est de faire circuler des découvertes rapidement et selon un système de proximité de goût des individus. Ici, on ne suit pas ses amis mais ceux qui ont les mêmes centres d'intérêts.

Elle doit offrir la possibilité de recherches sur ce qui a déjà été produit par d'autres utilisateurs :

L'utilisateur doit pouvoir observer les listes des autres, faire des recherches et pourquoi pas poster du contenu dans les listes d'inconnus.

La plateforme est intelligente, elle propose du contenu avec le profil d'un utilisateur fictif, reconnaissable comme étant la plateforme :

La plateforme fait donc des propositions de contenus aux utilisateurs.

La plateforme doit permettre l'interaction aisée entre utilisateurs :

Via le système de posts sur feed, elle oblige à une communication autour de thèmes, c'est-à-dire plus précisément autour des références.

Via le système de messagerie, elle permet une communication directe.

Fonctionnalités

Un utilisateur peut faire une recherche par liste :

En utilisant un méta-caractère, le tilde, l'utilisateur pourra chercher des listes. L'affinité entre utilisateurs, listes et références joueront un rôle important pour la présentation du contenu. Des filtres permettent de préciser la recherche.

Un utilisateur peut faire une recherche par référence :

En cherchant avec un mot-clé, l'utilisateur aura accès aux références qui y correspondent. Des filtres permettent de préciser la recherche.

Les références sont créées par les utilisateurs et soumises à un système de peer-reviews :

Toutes les métadonnées concernant les références doivent être produites par les utilisateurs afin de ne pas dépendre d'un service tiers et d'être agnostique aux catégories thématiques. Ainsi, nous implémenterons un système complexe de peer-reviews. Chaque métadonnée pourra être éditée par un utilisateur. Une fois éditée, elle sera soumise à tous les autres utilisateurs ayant préalablement édité l'entrée. Si plus de 50% d'entre eux acceptent le changement, celui-ci est validé. Si 100% refuse avec entre 3 et 5 utilisateurs, 80% avec plus de 10, l'utilisateur est flaggé. Au bout de cinq flags, l'utilisateur est bloqué pendant une semaine. S'il recommence, 2 semaines et s'il recommence encore, il est interdit de plateforme. La majorité l'emporte si, au bout d'une semaine, tous les utilisateurs ne se sont pas exprimés. Une fois édité, tout changement ultérieur doit attendre la validation pour exister sur la référence. Un utilisateur peut aussi faire un autre choix, celui de faire une suggestion. Si une suggestion obtient la majorité, le droit d'écrire revient à celui qui l'a formulée et le processus de changement repart de plus belle. Lors de la création d'une référence, un email est envoyé à 10 utilisateurs (ils sont choisis par proximité avec l'utilisateur la créant, etc.). Ils peuvent flagger la référence comme invalide ou comme valide. La majorité l'emporte au bout d'une semaine. Ce système fonctionne aussi pour les éditions de contenus en dessous de 3 utilisateurs. On peut imaginer aussi qu'il faut avoir participé préalablement à l'édition de contenu pour créer une référence, qu'il faut avoir un statut privilégié.

Un utilisateur possède un feed général

Sur ce feed, il peut poster des références, et d'autres peuvent aussi y poster. Chaque post doit obligatoirement contenir une référence. Les réponses ne doivent pas forcément en contenir.

Sur un post, il est possible de renvoyer à d'autres utilisateurs avec le caractère spécial @, de lier des listes avec le caractère spécial ~, ou bien des hashtags # liant ainsi une métathématique plus souple.

Un utilisateur possède des listes

L'utilisateur organise des listes thématiques. Il peut souscrire à celles d'autres utilisateurs ou en co-administrer. Ces listes permettent d'organiser le contenu selon un calendrier, une priorité, un jugement de valeur, la consommation déjà effective ou pas du contenu, etc.

Un utilisateur peut suivre et être suivi par d'autres utilisateurs

Un utilisateur peut bloquer tout autre utilisateur qui est intempestif. Il ne le verra alors plus dans ses recherches, ses feeds, ses messages. Ce sera ainsi comme si les deux utilisateurs étaient inexistant l'un pour l'autre.

Suivre ou être suivi permet de dégager une relation avec un autre utilisateur. Cette relation permet de définir qui peut poster dans le feed général et dans le feed des listes qu'un utilisateur possède, ou bien même de définir quels utilisateurs peuvent observer le feed ou les listes d'un tiers.

Chaque liste possède un feed :

L'administrateur décide qui peut poster sur ce feed. Le feed est en tout point similaire au feed personnel de l'utilisateur.

Un utilisateur peut faire une recherche par hashtag avec l'ajout de filtres appropriés (les filtres sont appliqués sur les méta-données des références) :

En cherchant avec un hashtags, l'utilisateur aura accès à tous les posts publics les mentionnant. Des filtres permettent de limiter la recherche.

Un utilisateur se voit offrir une notation sur une référence par l'algorithme :

Cette notation est accessible par rollover sur une référence ou directement sur la page d'une référence.

Un utilisateur peut noter une référence :

Le système de notation reste à définir, s'agit-il d'un système à l'anglo-saxon comme on le voit dans les flats (De A à F) ou une notation décimale. La décision n'a pas encore été prise.

La plateforme possède un système de messagerie :

La messagerie fonctionne par polling requests. Elle s'apparente à n'importe quel système de chat sur une plateforme social.

UX/UI



Architecture

Une "Single Page App" :

Le Front end est servi à la première requête, ensuite il récupère les données qui lui sont nécessaires au moyen d'une API REST implémenté sur le serveur au moyen du framework Spring pour Java. On pratique une approche modulaire du développement front end avec le framework react.js, le transpiler babel et le bundler webpack. L'application est complexe et transitionne entre beaucoup d'états différents, nous faisons donc ici le choix de la simplicité et de garder un magasin des données global qui reflète l'état courant. Spring facilite la distribution de l'API grâce à ses méthodes et décorateurs qui transforment objets en JSON et vice-versa. La permanence est assurée par PostgreSQL dont l'interaction se fait par l'object relational mapping framework hibernate.

Stack :

- react.js pour tous les composants front-end.
- Redux pour le store et maintenir les états de l'application côté front-end.
- Modularisation avec babel et webpack côté front-end
- Spring Boot pour servir une RESTful API.
- PostgreSQL pour la permanence et hibernate pour la traduction objets Java vers SQL
- ESLint/Enzyme et Jest pour les tests unitaires du front-end
- *Définir template pour test unitaires de l'API, méthodes et contrôleurs du back-end*
- Umbrello pour la modélisation de classes si nécessaire.

Méthode de travail

Méthode Agile :

Utilisation de Jira et Confluence avec des scrums pour avancer au plus vite.

Définition et pré-documentation des états, routing, store, API et modélisation de la base de données :

On trouvera avant même le développement un "breakdown" de tous les composants du front end et de leurs responsabilités ainsi que les routes empruntées pour maintenir les états côté client (React Router), la permanence des états avec le store global redux ainsi que les routes de l'API serveur avec leur modèles de données. Nous produirons aussi un modèle de la base de données. Ces références seront éditées au fur et à mesure du développement pour représenter les changements dans la représentation de l'architecture en vue des problèmes rencontrés.

Design

Nous réunissons ici toutes les informations relatives au design de l'UI/UX.



Qinetic Final.pdf

UX/UI Eléments Généraux

Nous décrivons ici les éléments du design qui sont généraux et qui dépassent les fonctionnalités des composants

1. Les adresses url des images du slideshow sont récupérés par un appel vers notre API REST. Elles comprennent les paramètres nécessaires pour ajuster un calque d'opacité noir pour permettre la bonne lisibilité du contenu mais aussi les paramètres pour générer le dégradé en mouvement approprié. Lorsqu'un utilisateur est connecté, il a accès à une autre API qu'il lui fournit des images spécifiquement choisies en relation avec le contenu qu'il visionne et son profil.
2. Tous les rollovers apparaissent et disparaissent avec grâce au moyen de petites animations CSS3.

Ceci n'est pas une priorité dans le développement et peut être relegué à bien plus tard.

Related articles

Architecture

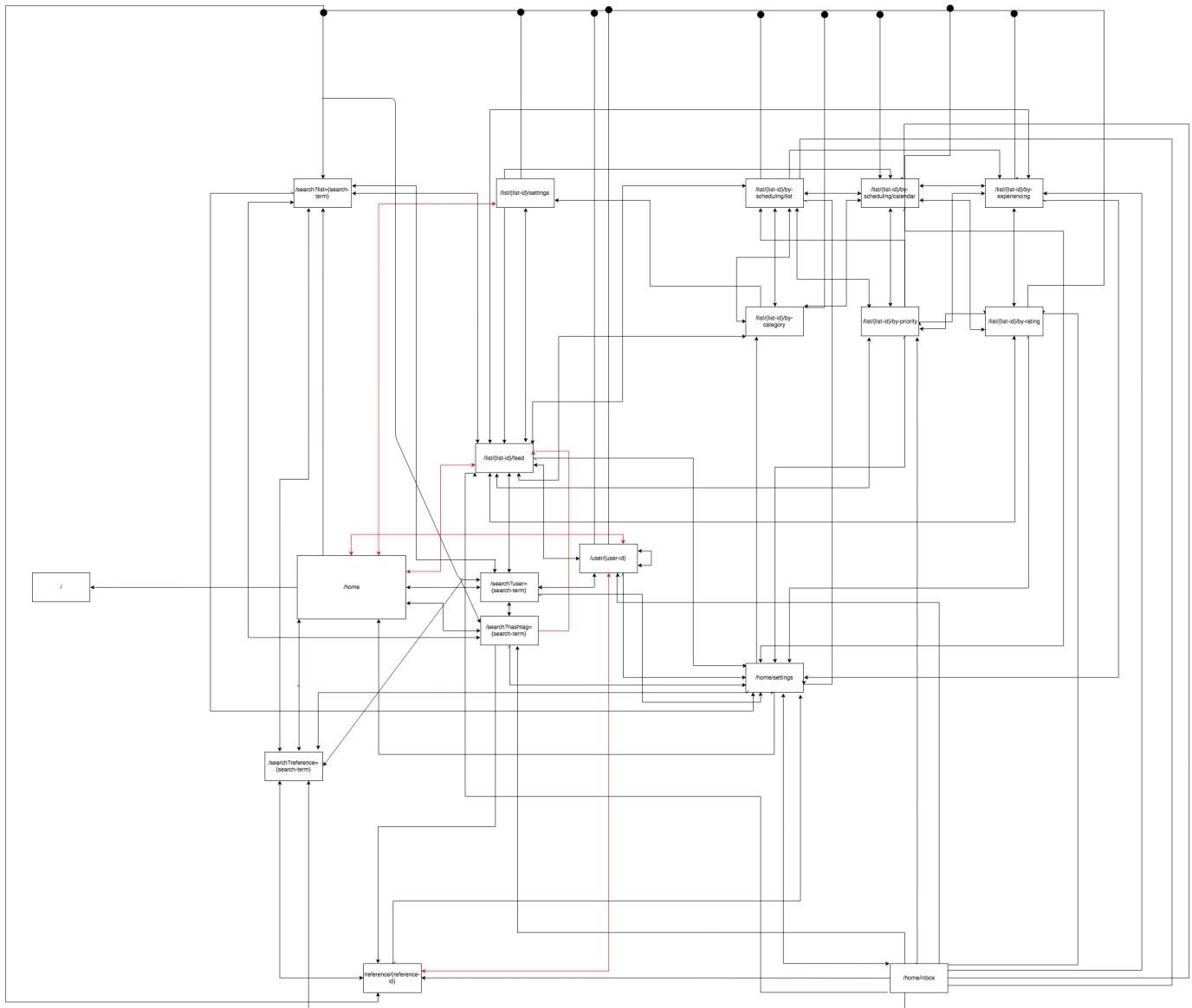
Nous regroupons ici toutes les descriptions concernant les décisions d'architecture du système

Front end - Etats de l'application & Composants react.js

Nous détaillerons ici les routes du routeur react.js pour un maintien de certains états dans la barre URL mais aussi le modèle et les reducers du store de redux pour gérer les différents états de l'application. Nous dresserons aussi une liste des composants react.js avec la possibilité d'une organisation hiérarchique pour la "reusability".

Routes de React Router & Composants associés

Ici, nous décrivons les états de l'application qui méritent un accès par le routeur de react.js, c'est-à-dire auxquels on peut accéder par URL, et qui donc peuvent fonctionner avec les boutons retour et avancé. Le schéma des états ci-dessous montre bien la complexité des interactions et de fait n'a aucune utilité. C'est pourquoi, nous aurons pour chaque composant un schéma individualisé des interactions.



Cette rubrique est en cours de construction et sujette au changement.

Related articles

<https://reacttraining.com/react-router/web/example/basic>

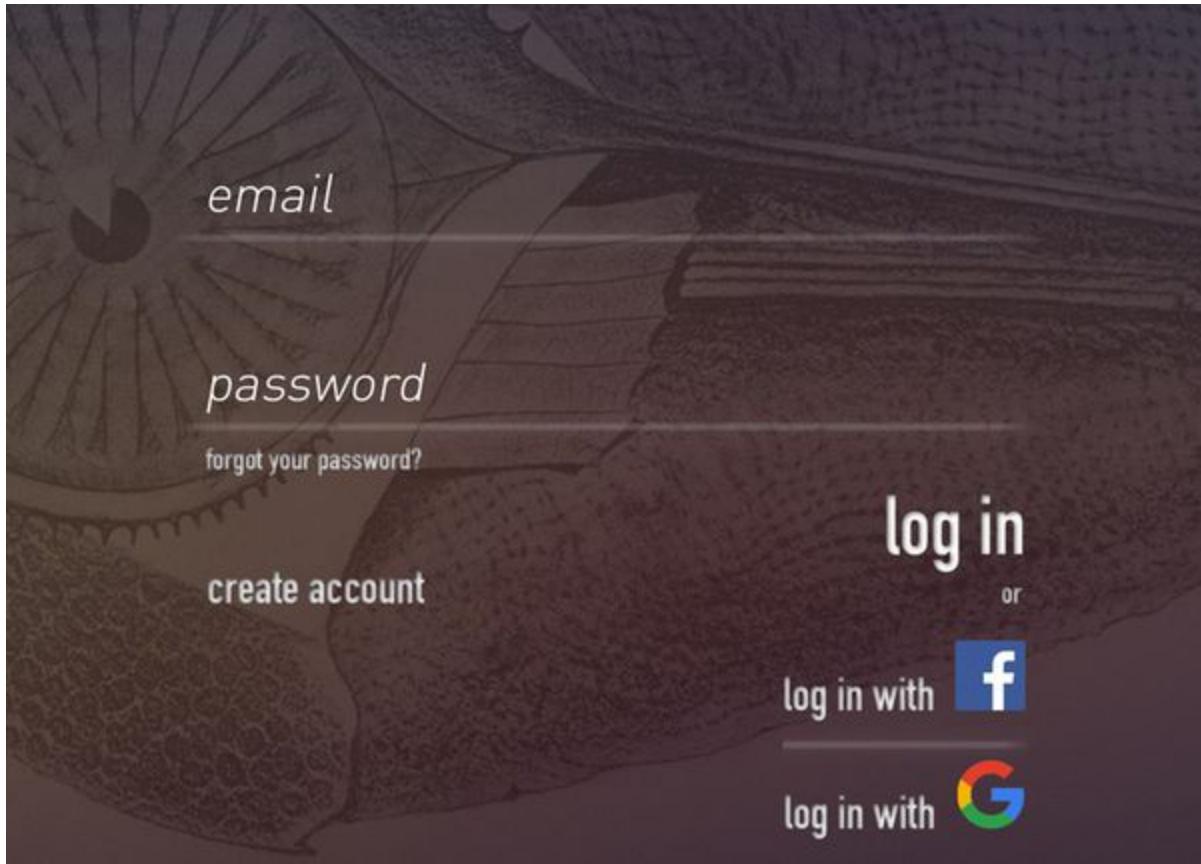
/

Il s'agit de la page principale sur laquelle l'utilisateur atterrit si il n'a pas de token.

Les composants inclus dans cet état sont :

- Login

1 - Composant Login



Description :

Ce composant permet à l'utilisateur de se connecter en utilisant ses identifiants créés pour le site ou bien en utilisant Facebook ou Google. Contient des liens pour récupérer son mot de passe ou créer un compte.

Est utilisé par les routes du router :

- / si absence de token

Utilise les sous-composants :

- "message d'erreur"

Contient deux champs :

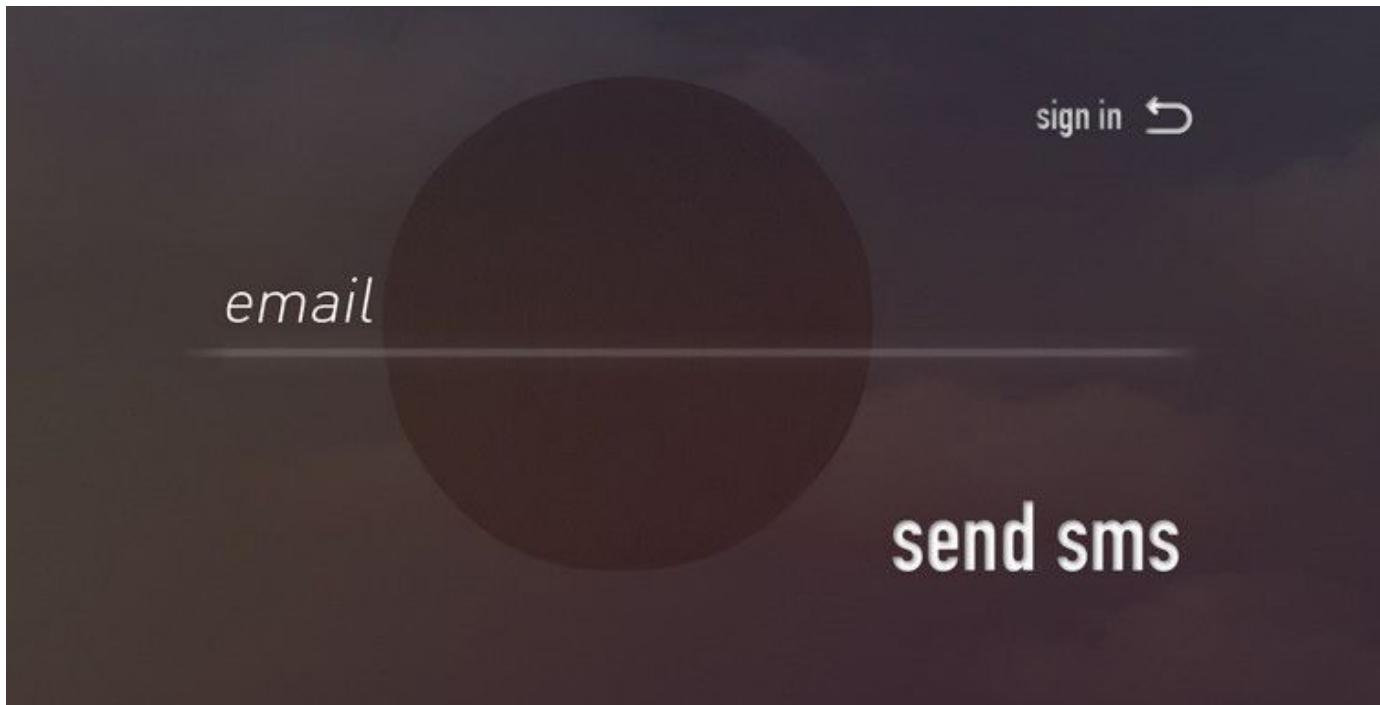
- email
- password

Contient cinq liens hypermedia :

- Login
 - Appel de l'API REST : [/login](#) avec les deux champs
 - Si succès, transition vers route "[Home Feed](#)" avec l'obtention d'un token à placer dans le header http de chaque requête
 - Si échec, chargement du composant "message d'erreur"
- Login with Facebook
 - Open Popup
 - Si succès, transition vers route "[Home Feed](#)"
- Login with Google
 - Open Popup
 - Si succès, transition vers route "[Home Feed](#)"

- Create Account
 - Charge Composant "Create Account"
 - Décharge Composant "Login"
- Forgot Password
 - Charge Composant "Forgot Password"
 - Décharge Composant "Login"

1.1 - Composant Forgot Password / Renseigner Adresse Email



Description :

Ce composant permet à la plateforme d'envoyer à l'utilisateur qui fournit son adresse email un sms afin qu'il puisse récupérer son mot de passe.

Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- "message d'erreur"

Utilise les sub-composants :

- Composant Login (si l'adresse email a déjà été fournie dans le composant Login, Le champ sera prépeuplé)

Contient un champ :

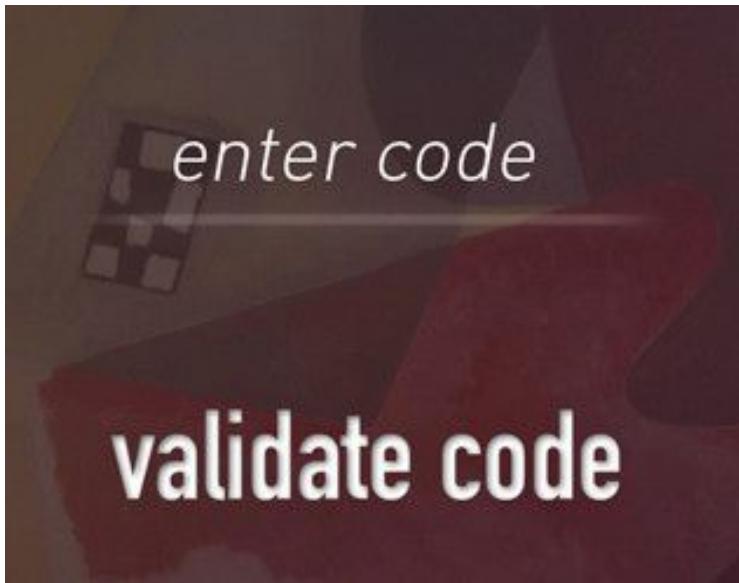
- email

Contient un lien hypermedia :

- send sms
 - Appel de l'API REST : /send-sms avec un champ
 - Si succès, chargement du composant "[Renseigner SMS Code](#)"

- Chargement du composant "Compte bloqué"
- Si échec, chargement du composant "message d'erreur"

1.1.1 & 2.1.1.1 - Composant Renseigner SMS



Description :

Ce composant permet à l'utilisateur de fournir le code sms qu'il a reçu
Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- "message d'erreur"

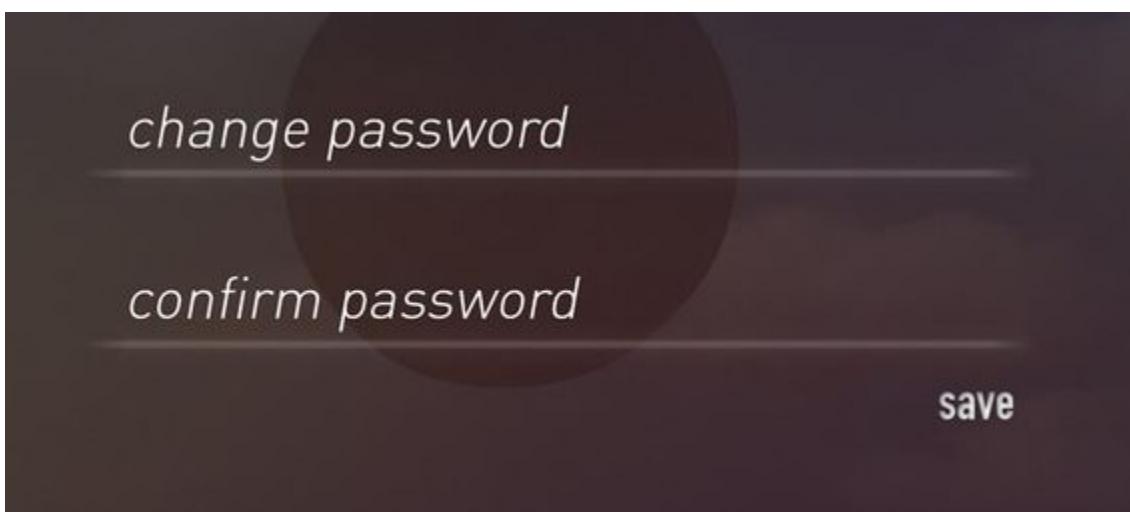
Contient un champ :

- code
- renewPassword

Contient un lien hypermedia :

- validate code
 - Appel de l'API REST : [/sms-verification](#) avec trois champs (+email)
 - Si succès, chargement du composant "Renseigner SMS Code"
 - Si échec, chargement du composant "message d'erreur"

1.1.1.1 Composant Change Password



Description :

Ce composant permet à l'utilisateur de renouveler son mot de passe.

Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- "message d'erreur"

Contient un champ :

- password

Contient un lien hypermedia :

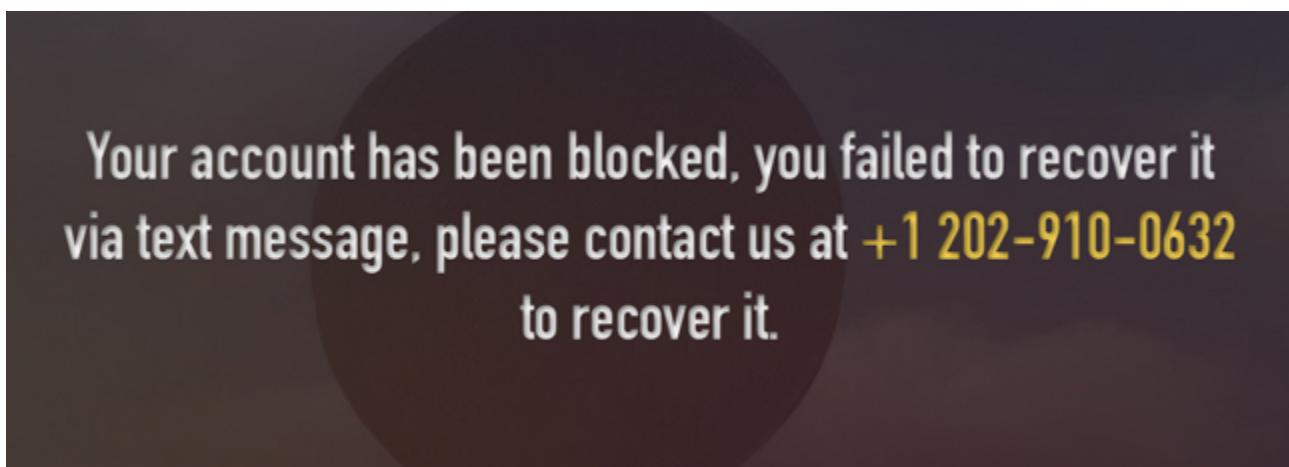
- validate code

- Appel de l'API REST : </user/{user-id}/change-password> avec deux champs (+email et sms-verified in oldPassword field) et avec l'obtention d'un token à placer dans le header http de chaque requête.
 - Si succès, transition vers route "Home Feed"

ou

- Chargement du composant "Compte bloqué"
- Si échec, chargement du composant "message d'erreur"

1.1.2 & 1.1.1.2 - Composant Compte Bloqué



Description :

Après trois tentatives d'envoie de codes par sms, le compte est bloqué pendant une semaine

Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- aucun

1.2 - Composant Create Account

sign in ↗

Solal I *last name*

email

confirm email

password ⚠ password does not match

may ▾ | 15 ▾ | 1993 ▾ ⚠ birthday

⚠ +1 ▾ | *phone number*

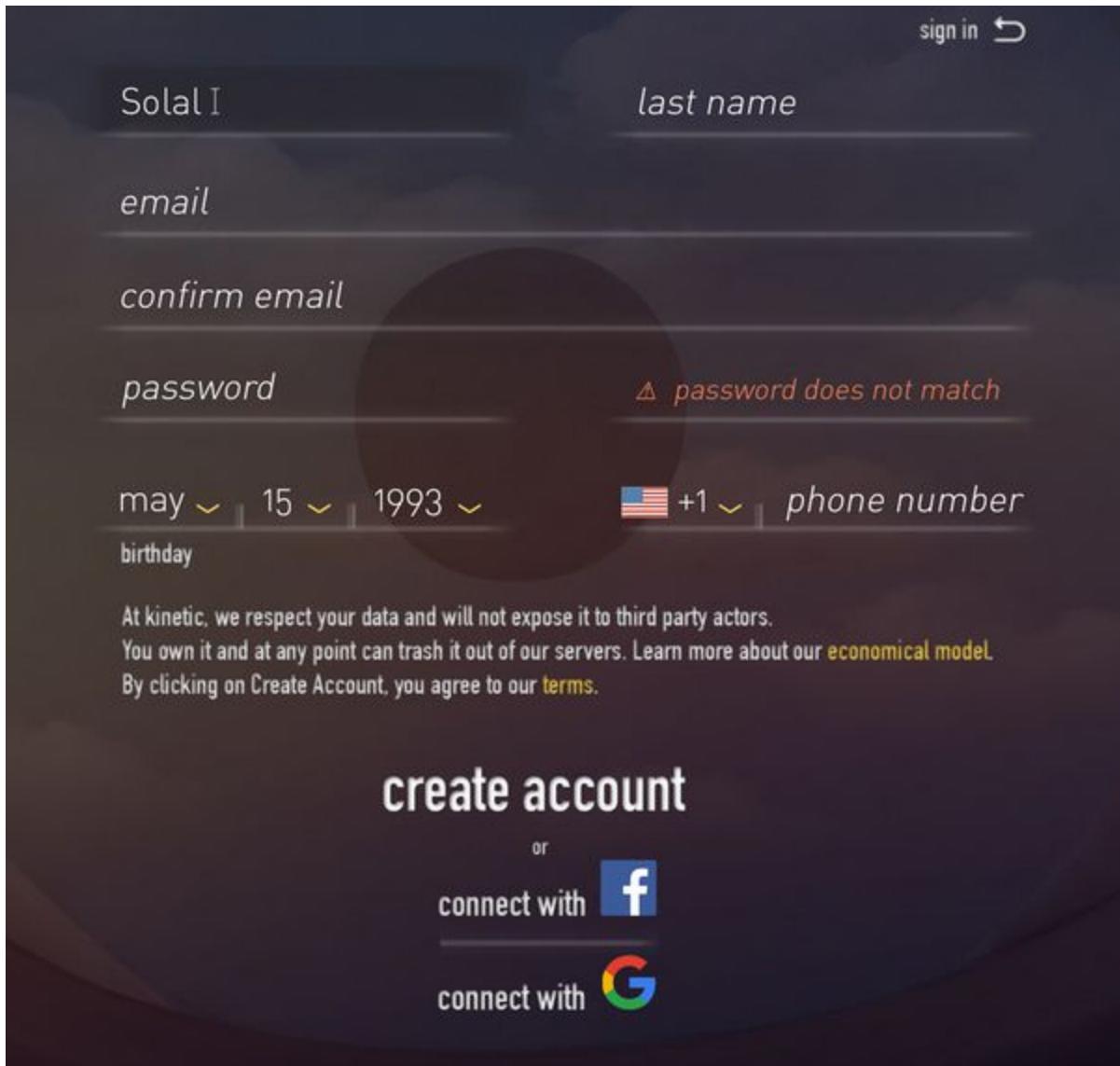
At kinetic, we respect your data and will not expose it to third party actors.
You own it and at any point can trash it out of our servers. Learn more about our economical model.
By clicking on Create Account, you agree to our terms.

create account

or

connect with 

connect with 



Description :

Ce composant permet à l'utilisateur de s'enregistrer auprès de la plateforme ou bien de donner les autorisations à partir de Google ou Facebook pour se connecter à la plateforme.

Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- "message d'erreur"

Contient six champs :

- first name
- last name
- email
- password
- birthday
- cellphone

Contient trois liens hypermedia :

- Login
 - Appel de l'API REST : [/create-account](#) avec les six champs
 - Si succès, chargement du composant "Validate Account"
 - Si échec, chargement du composant "message d'erreur"
- Connect with Facebook
 - Open Popup
 - Si succès, transition vers route "/home"
- Connect with Google
 - Open Popup
 - Si succès, transition vers route "/home"

1.2.1 - Composant Valide Email



Description :

Ce composant attend et fait des polling requests toutes les 30 secondes pour voir si l'email a été vérifié
Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- aucun

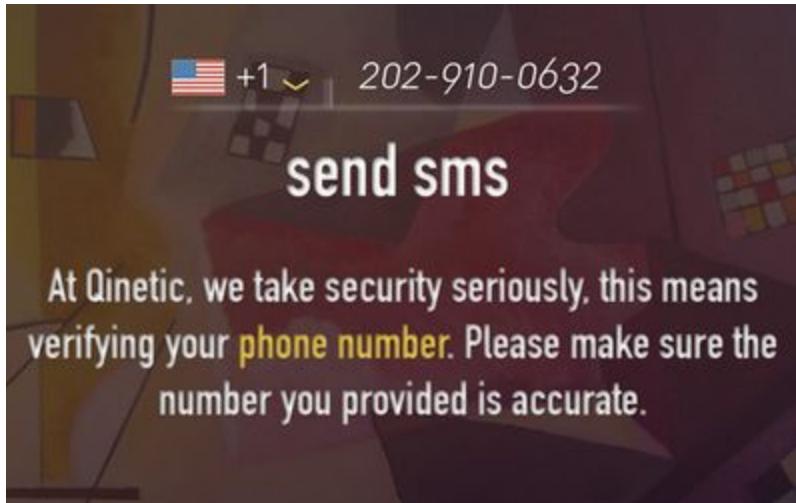
Contient un champ :

- email

Contient un lien hypermedia et une requête invisible :

- resend email
 - Appel de l'API REST : [/email-verification](#) avec le champ
 - Durant appel serveur, disparition du message, usage d'un loader,
 - Si succès, réapparition du même message
 - Si échec, changement du message en message d'erreur
- Polling request
 - Appel de l'API REST : [/email-verified](#) avec le champ
 - Si succès, transition vers route "/home"
 - Si échec, rien

1.2.2 - Composant Validation SMS



Description :

Ce composant permet d'édition le numéro de téléphone entré et d'envoyer un sms
Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- aucun

Contient un champ :

- phone (récupéré du store redux)

Contient un lien hypermédia et une requête invisible :

- send sms
 - Appel de l'API REST : </send-sms> avec le champ
 - Si succès, chargement du composant "Renseigner SMS Code"
 - Si échec, changement du message en message d'erreur

1.2.2.1 Composant Renseigner SMS

Voir [1.1.1 & 2.1.1.1 - Composant Renseigner SMS](#)

/home

Il s'agit de la page principale sur laquelle l'utilisateur atterrit après un login ou vers laquelle on est redirigé si le token d'authentification est toujours valide.

Les composants inclus dans cet état sont :

- Le Profil de l'utilisateur
- La barre de recherche principale
- Le dépliant indiquant les listes
- Le Feed
- Le lien vers la messagerie
- Les dépliants indiquant les relations de l'utilisateur avec les autres

1 - Composant Profile

2 - Dépliant - Lister les listes



Description :

Ce composant permet à l'utilisateur de visualiser trois types de listes, les siennes, les partagées et les suivies. En déroulant une section du dépliant, il verra la possibilité de faire une recherche sur ces listes qui se trouveront filtrées au fur au mesure des lettres tapées. Il permet aussi de créer une nouvelle liste.

Obtient ses données :

- /user (en passant email et token)

Est utilisé par les routes du router :

- [/home](#)

Utilise les sous-composants :

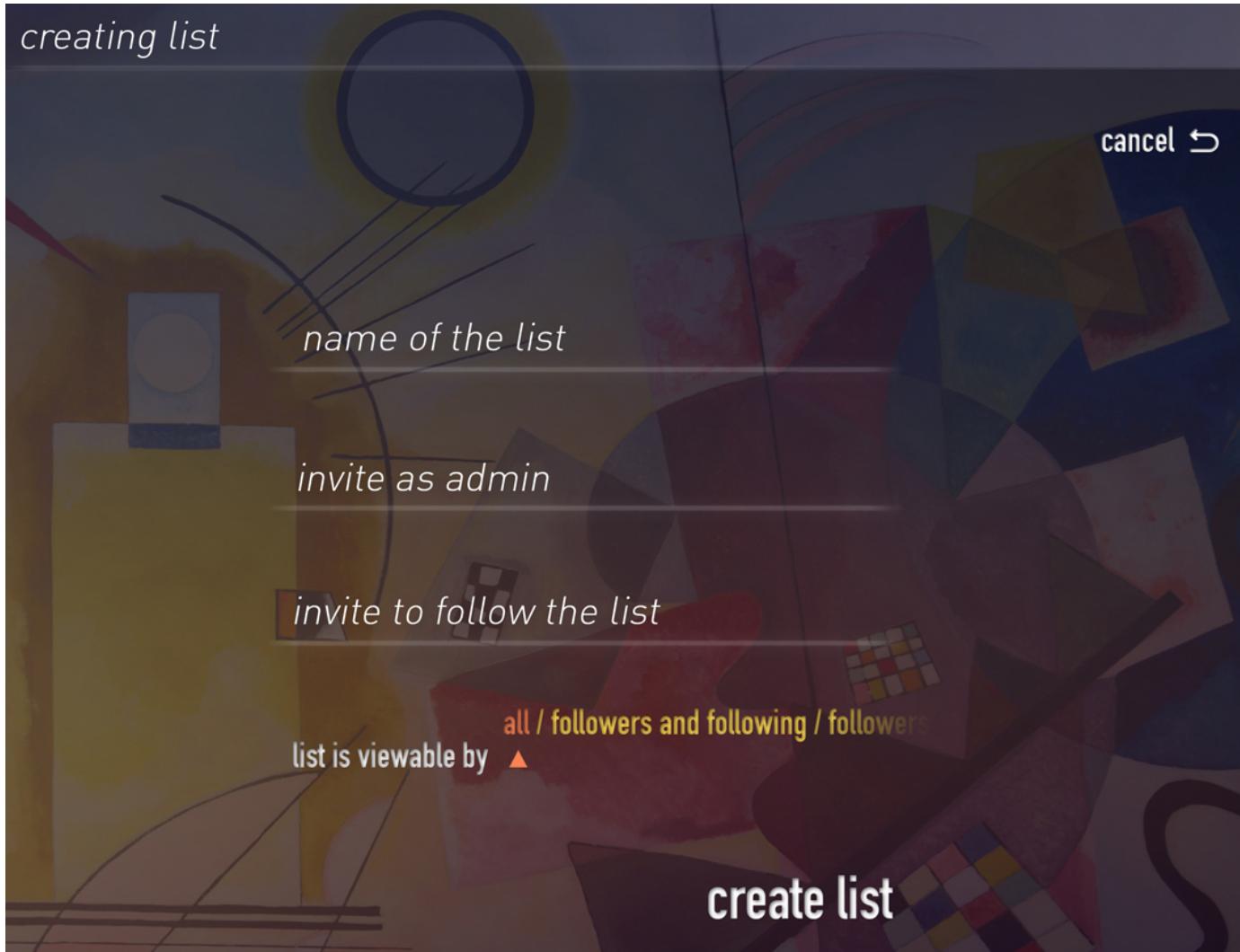
- [Aucun](#)

Contient aucun champ :

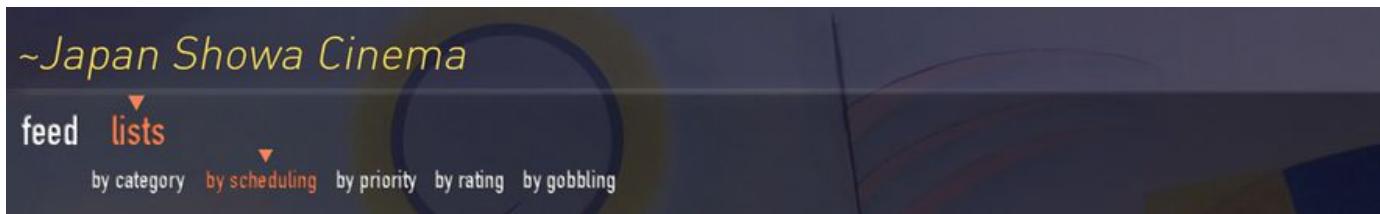
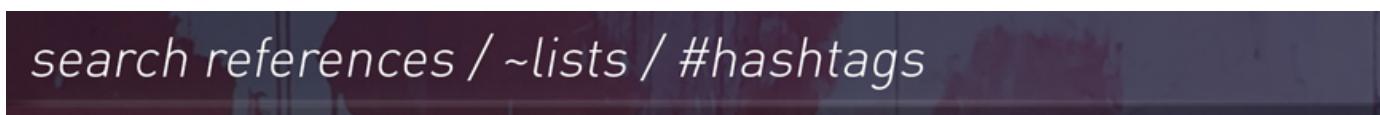
Contient X liens hypermedia :

- {list}
 - Appel de l'API REST : [/load-list/{list-id}](#) avec l'id de la liste et header for authorization
- {list} Settings
 - Charge Composant Paramètres Listes
- Create New Liste
 - Charge Composant Créer Nouvelle Liste

3 - Composant Crée une liste



4 - Barre de Recherche / Barre de titre



5 - Composant du feed

post to your home feed



Lisa Redmond

The Wild Bunch doesn't set out to be liked, it is a harsh eye opening perception of the Western genre, this is the other side of the coin to the millions of Westerns that whoop and holler as the hero gets the girl and rides off into the sunset.

↳ Solal Gaillard: Just recently saw it and it is incredible. Reminded me very much of that 
reply to @Lisa Redmond

References - The Wild Bunch ▾



Solal Gaillard

With *Sans Soleil*, Chris Marker skillfully blends image, sound, and voice in a powerful way that I've never experienced before or since. No mere description can begin to convey this film's stunning effect on my intellect and my senses. Not quite a documentary, not quite fiction, Marker's film emerges as a mesmerizing meditation on the meaning of time, space, and memory. "How," he asks, "does one remember thirst?" A film you won't forget.

reply to your post

References - Sans Soleil ▾



kinetic

Netflix is currently showing *Babylon Berlin*, a typical german krimi set in the Weimar Republic. We believe that you would thoroughly enjoy that show.

References - Babylon Berlin ▾

Divisé en plusieurs composants

6 - Dépliant - Relations et affinités entre l'utilisateur et ses pairs

following ▾

followers ▾

blocked users ▾

Description :

Ce composant permet à l'utilisateur de visualiser qui sont les utilisateurs qu'il suit, qui le suivent et ceux qu'il a bloqué. Des interactions sont possibles grâce à un rollover

Obtient ses données :

- [/user/{user-id}](#) (en passant email et token)

Est utilisé par les routes du router :

- [Home Feed](#)

Utilise les sous-composants :

- Aucun

Contient aucun champ :

Contient X liens hypermedia :

- {user}
 - Appel de l'API REST : [/load-user/{user-id}](#) avec l'id (email) de l'utilisateur et id de la cible pour {user} et header for authorization
- rollover {user}
 - Appel de l'API REST : [/user/{user-id}/interact-user/{user-id}](#) avec l'id (email) de l'utilisateur / header for authorization / et action (block, unblock, follow, unfollow)

7 - Composant Messagerie

Belongs to all States past 4

* Polling request

* Go to inbox

Polling request every minute ->

1. HTTP/1.1 GET /kinetik.express/{user-id}/inbox?unread
2. Refresh Component State (Done By React)

Go to inbox ->

1. HTTP/1.1 GET /kinetik.express/{user-id}/inbox
2. Unload all components from current State
3. Load all component from State 15

/settings

Il s'agit de la page qui permet de définir les paramètres de réglages du compte utilisateur.

Les composants inclus dans cet état sont :

- Le Profil de l'utilisateur
- Le titre de la rubrique
- Le dépliant indiquant les listes
- Une photo avatar avec la possibilité de la changer ou de la déplacer dans un cercle
- Deux input pour changer le mot de passe
- Un lien pour supprimer le compte
- Le lien vers la messagerie
- Les dépliants indiquant les relations de l'utilisateur avec les autres

1 - Composant Profile (bis - 1)

2 - Dépliant - Lister les listes (bis 1)

3 - Composant Créer une liste (bis 1)

4 - Titre de la page

5 - Composant Paramètres

6 - Dépliant - Relations et affinités entre l'utilisateur et ses pairs (bis 1)

7 - Composant Messagerie (bis 1)

/search?list={search-term}

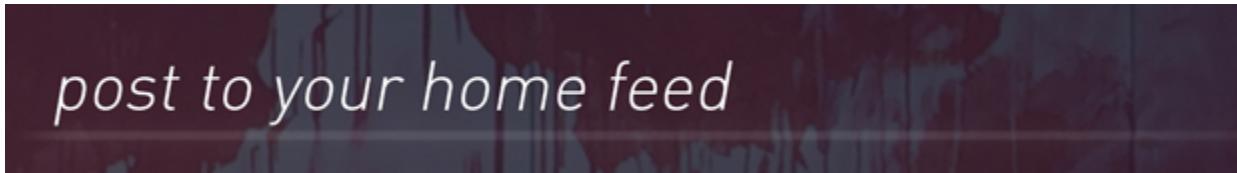
Prends trois paramètres optionnels. filtre, users, shared references.

Filtre s'organise comme ça :

&filter[]something&filter[]somethingElse&filter[]somethingElse&filter[]somethingElse

/list/{list-id}/feed

4.1 - Composant Poster sur le feed



Description :

Ce composant permet à l'utilisateur de poster sur un feed

Obtient ses données :

- aucun

Est utilisé par les routes du router :

- Home Feed
- /list/{list-id}/feed

Utilise les sous-composants :

- "message d'erreur"

Contient un champ :

- Post

Contient un lien hypermedia :

- Post
 - Appel de l'API REST : [/feed/{feed-id}/post](#) avec id du feed / header for authorization et payload

4.2 - Composant du feed



Lisa Redmond

The Wild Bunch doesn't set out to be liked, it is a harsh eye opening perception of the Western genre, this is the other side of the coin to the millions of Westerns that whoop and holler as the hero gets the girl and rides off into the sunset.

↳ **Solal Gaillard**: Just recently saw it and it is incredible. Reminded me very much of that ooo
reply to [@Lisa Redmond](#)

References - *The Wild Bunch* ↴



Solal Gaillard

With *Sans Soleil*, Chris Marker skillfully blends image, sound, and voice in a powerful way that I've never experienced before or since. No mere description can begin to convey this film's stunning effect on my intellect and my senses. Not quite a documentary, not quite fiction, Marker's film emerges as a mesmerizing meditation on the meaning of time, space, and memory. "How," he asks, "does one remember thirst?" A film you won't forget.

reply to your post

References - *Sans Soleil* ↴



kinetic

Netflix is currently showing *Babylon Berlin*, a typical german krimi set in the Weihmar Republic. We believe that you would thoroughly enjoy that show.

References - *Babylon Berlin* ↴

Description :

Ce composant permet à l'utilisateur visualiser un feed. Une requête est envoyée au serveur quand on approche de la fin du scrolling (lazy loading). Ne pas oublier la polling request pour nouveau post

Obtient ses données :

- `/feed/{feed-id}?last={post-id}` avec id du field et dernier id d'un post

Est utilisé par les routes du router :

- Home Feed
- `/list/{list-id}/feed`

Utilise les sous-composants :

- aucun

Contient un champ :

- aucun

Contient X liens hypermedia :

- {user}
 - Appel de l'API REST : </load-user/{user-id}> avec l'id (email) de l'utilisateur et id de la cible pour {user} et header for authorization
- rollover {user}
 - Appel de l'API REST : </user/{user-id}/interact-user/{user-id}> avec l'id (email) de l'utilisateur / header for authorization / et action (block, unblock, follow, unfollow)
- {reference}
 - Appel de l'API REST : </load-reference/{reference-id}> avec l'id (email) de l'utilisateur et id de la cible pour {user} et header for authorization
- rollover {reference}
 - Appel de l'API REST : </user/{user-id}/interact-reference/{reference-id}> avec l'id (email) de l'utilisateur / header for authorization / et action (block, unblock, follow, unfollow)
- {list}
 - Appel de l'API REST : </load-list/{list-id}> avec l'id (email) de l'utilisateur et id de la cible pour {user} et header for authorization
- rollover {list}
 - Appel de l'API REST : </user/{user-id}/interact-list/{list-id}> avec l'id (email) de l'utilisateur / header for authorization / et action (block, unblock, follow, unfollow)

4.2.1 Composant Post

Lisa Redmond

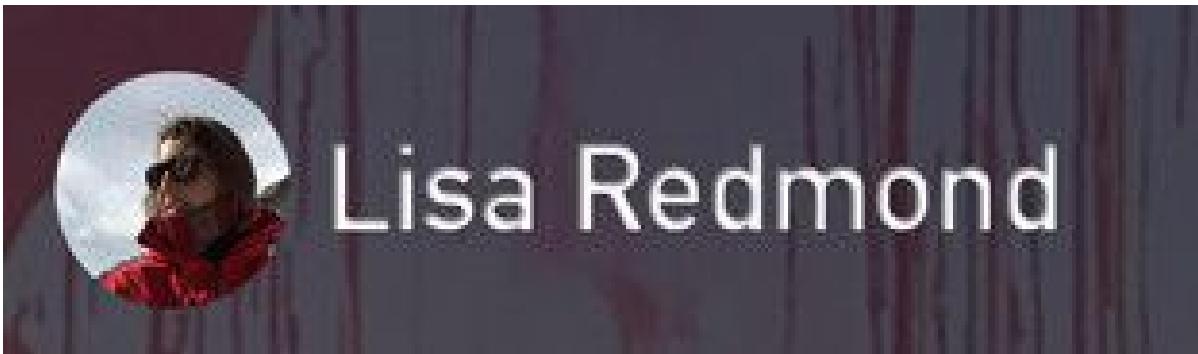
The Wild Bunch doesn't set out to be liked, it is a harsh eye opening perception of the Western genre, this is the other side of the coin to the millions of Westerns that whoop and holler as the hero gets the girl and rides off into the sunset.

↳ Solal Gaillard: Just recently saw it and it is incredible. Reminded me very much of that ooo
reply to @Lisa Redmond

References - *The Wild Bunch* ↴

Nous pouvons découper les composants présentationnels.

4.2.1.1 - Composant Profile



Voir pour l'aspect présentationnel Composant 1 - Profile.

4.2.1.2 - Composant Post Message

The Wild Bunch doesn't set out to be liked, it is a harsh eye opening perception of the Western genre, this is the other side of the coin to the millions of Westerns that whoop and holler as the hero gets the girl and rides off into the sunset.

↳ Solal Gaillard: Just recently saw it and it is incredible. Reminded me very much of that ooo

Description :

Ce composant permet à l'utilisateur visualiser le post. Les réponses sont cachées mais préchargées. Le composant les révèle.

4.2.1.3 Composant Réponse



reply to @Lisa Redmond

Description :

Ce composant permet à l'utilisateur d'écrire en dernier sur le feed.

Utilise les sous-composants :

- "message d'erreur"

Contient un champ :

- réponse

Contient X liens hypermedia :

- send
 - Appel de l'API REST : [/feed/{feed-id}/reply/{post-id}](#)
- Caractère spéciale @:
 - Appel de l'API

AUTRES API ICI POUR FAIRE RECHERCHES INTRA POSTS

4.2.1.4 - Composant References



References - The Wild Bunch ↗

Description :

Ce composant permet à l'utilisateur visualiser toutes les références mentionnées dans le feed. Un clique lui permet d'avoir un accès rapide au contenu.

Utilise les sous-composants :

- aucun

Contient un champ :

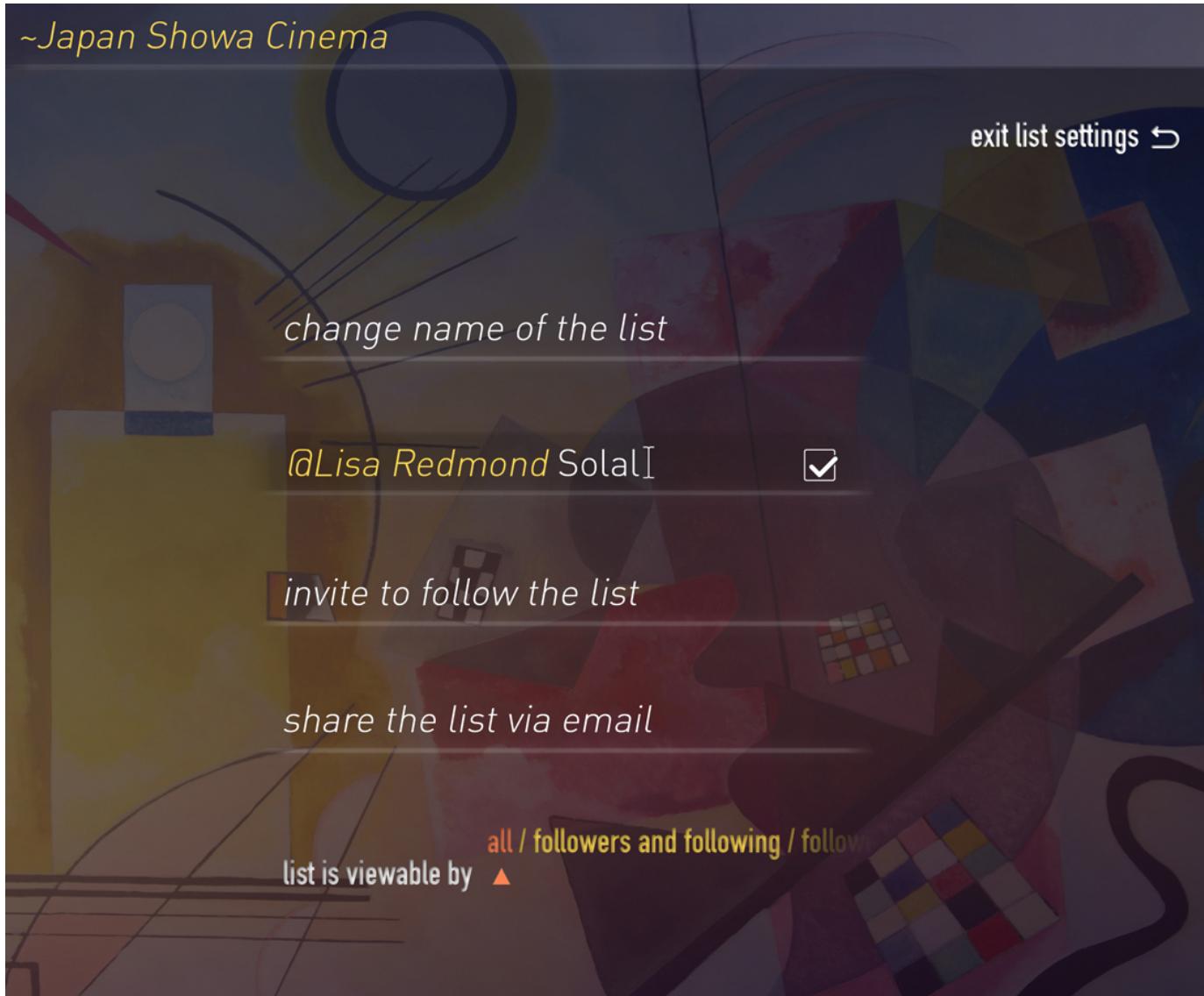
- aucun

Contient X liens hypermedia :

- {reference}
 - Appel de l'API REST : [/load-references/{reference-id}/summary](#) avec l'id (email) de l'utilisateur et id de la cible pour {user} et header for authorization
- rollover {reference}
 - Appel de l'API REST : [/user/{user-id}/interact-reference/{references-id}](#) avec l'id (email) de l'utilisateur / header for authorization / et action (block, unblock, follow, unfollow)
- {reference}
 - Appel de l'API REST : [/load-reference/{reference-id}](#) avec l'id (email) de l'utilisateur et id de la cible pour {user} et header for authorization
- rollover {reference}
 - Appel de l'API REST : [/user/{user-id}/interact-reference/{reference-id}](#) avec l'id (email) de l'utilisateur / header for authorization / et action (block, unblock, follow, unfollow)
- {list}
 - Appel de l'API REST : [/load-list/{list-id}](#) avec l'id (email) de l'utilisateur et id de la cible pour {user} et header for authorization
- rollover {list}
 - Appel de l'API REST : [/user/{user-id}/interact-list/{list-id}](#) avec l'id (email) de l'utilisateur / header for authorization / et action (block, unblock, follow, unfollow)

[/list/{list-id}/settings](#)

6 - Composant Paramètres d'une liste



Description :

Ce composant permet à l'utilisateur de visualiser trois types de listes, les siennes, les partagées et les suivies. En déroulant une section du dépliant, il verra la possibilité de faire une recherche sur ces listes qui se trouveront filtrées au fur au mesure des lettres tapées. Il permet aussi de créer une nouvelle liste.

Obtient ses données :

- /user (en passant email et token)

Est utilisé par les routes du router :

- list/id/settings

Utilise les sous-composants :

- Aucun

Contient aucun champ :

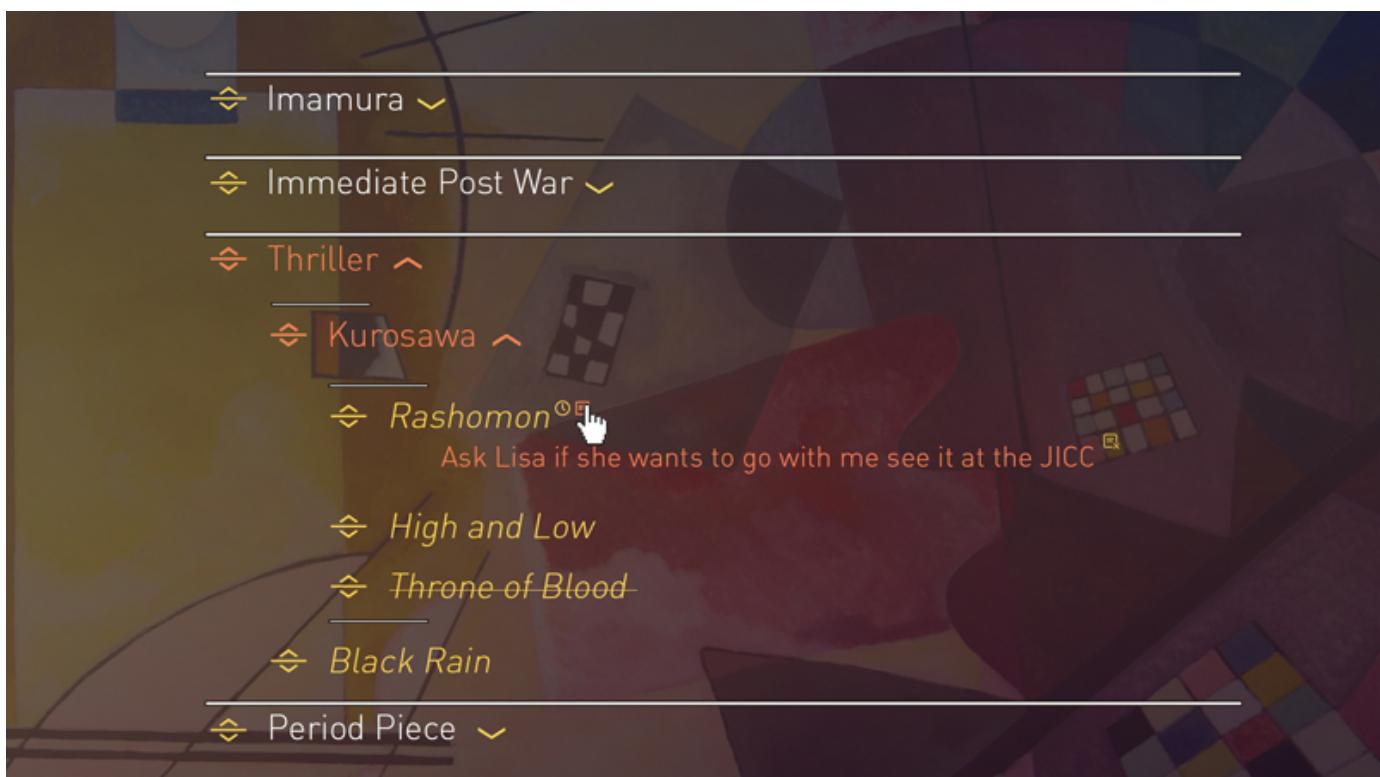
Contient X liens hypermedia :

- {list}
 - Appel de l'API REST : /load-list avec l'id de la liste et header for authorization

- {list} Settings
 - Appel de l'API REST : /list-settings avec l'id de la liste et header for authorization
 - Si succès, message de succès remplace le champ pendant deux secondes et laisse les champs réapparaître.
 - Si échec, charger "message d'erreur"
- Upload new Picture
 - Open file browser, on select :
 - Appel de l'API REST : /update-picture avec le champ picture (+email)
 - Si succès, replace URL dans la fenêtre oval avec la photo avatar
 - Si échec, charger "message d'erreur"
- Align Visual
 - Charge Composant "Aligner l'avatar"

/list/{list-id}/by-category

Composant Catégorie



/list/{list-id}/by-scheduling/list

Composant Calendrier Liste



[switch view to calendar](#)

April 18th 2018

April 19th 2018

⇒ *Rashomon* [®]

April 20th 2018

April 21st 2018

April 22nd 2018

⇒ *The Eel*

April 23rd 2018

April 24th 2018

⇒ *In The Realm of The Senses*

⇒ *High and Low*

April 25th 2018

April 26st 2018

April 27nd 2018

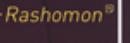
⇒ *Furyo*

</list/{list-id}/by-scheduling/calendar>

Comosant Calendrier

⟨ APRIL 2018 ⟩

switch view to schedule list

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
				⇒ <i>Rashomon</i> 		
			18	19	20	21
⇒ <i>The Eel</i>		⇒ <i>In The Realm of The Senses</i> ⇒ <i>High and Low</i>			⇒ <i>Furyo</i>	
22	23	24	25	26	27	28
29	30					

/list/{list-id}/by-priority

Composant Priorité



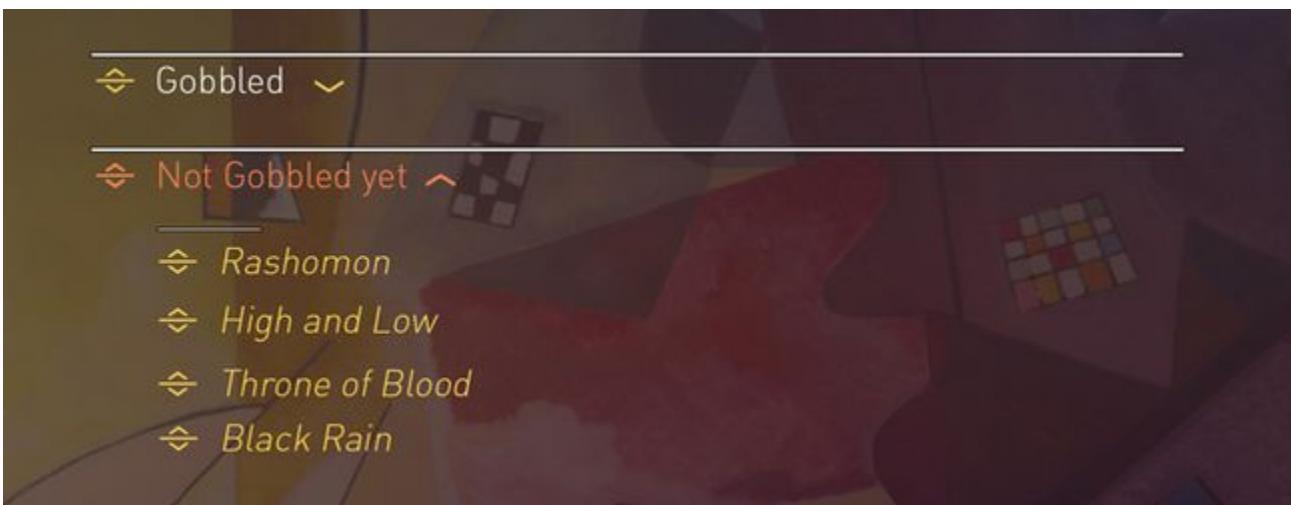
/list/{list-id}/by-rating

Composant Notation



/list/{list-id}/by-experiencing

Composant Experienced yet?



/search?reference={search-term}

Prends paramètres optionnels avec filters et rating

/reference/{reference-id}

/search?user={search-user}

/user/{user-id}

/search?hashtag=search-term

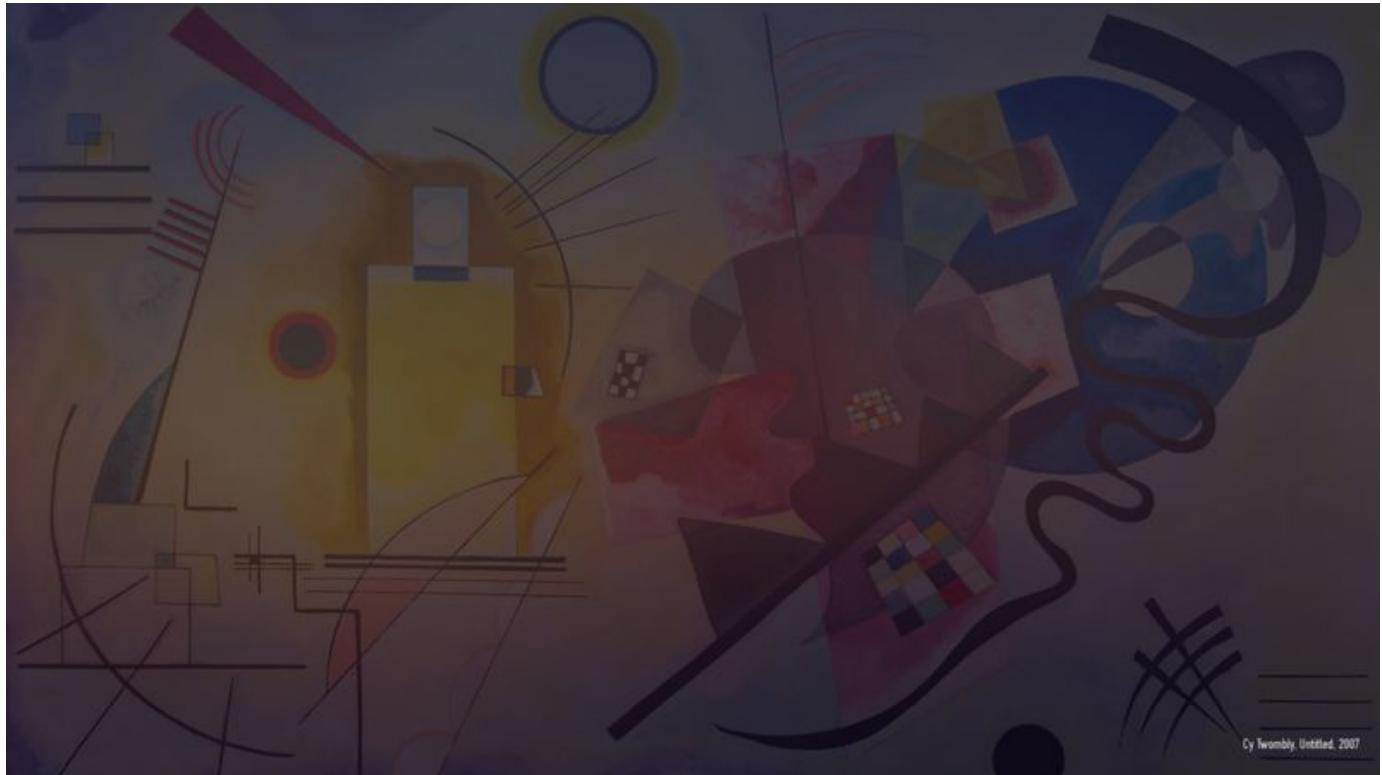
prends filter comme paramètre optionnel

/home/inbox

Composants Multi-routes

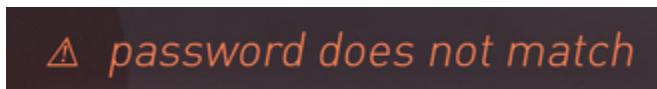
Il s'agit ici de tous les composants qui sont chargés lorsqu'il n'y a pas de token dans le header http ou quand celui-ci est invalide.

X - Composant Background



Cy Twombly, Untitled, 2007

3 - Composant Message d'erreur



Description :

Ce tout petit composant permet à un input de montrer un message d'erreur.

Est utilisé par les composants :

- Composant Login
- Composant Create Account
- Composant Forgot Password
- Composant Renseigner SMS
- Composant Change Password
- Composant Validation SMS

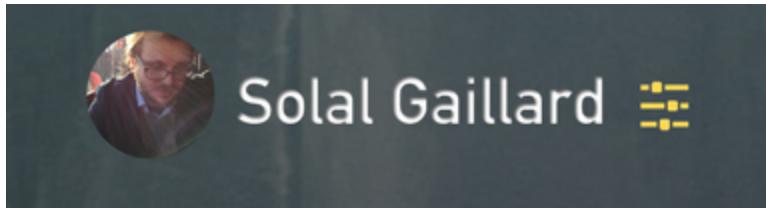
Est utilisé par les routes du router :

- aucune

Utilise les sous-composants :

- Aucun

x1 - Composant Profile



Obtient ses données :

- `/user/{user-id}` (en passant token)

Est utilisé par les routes du router :

- Toutes sauf qinetic.com si aucun token

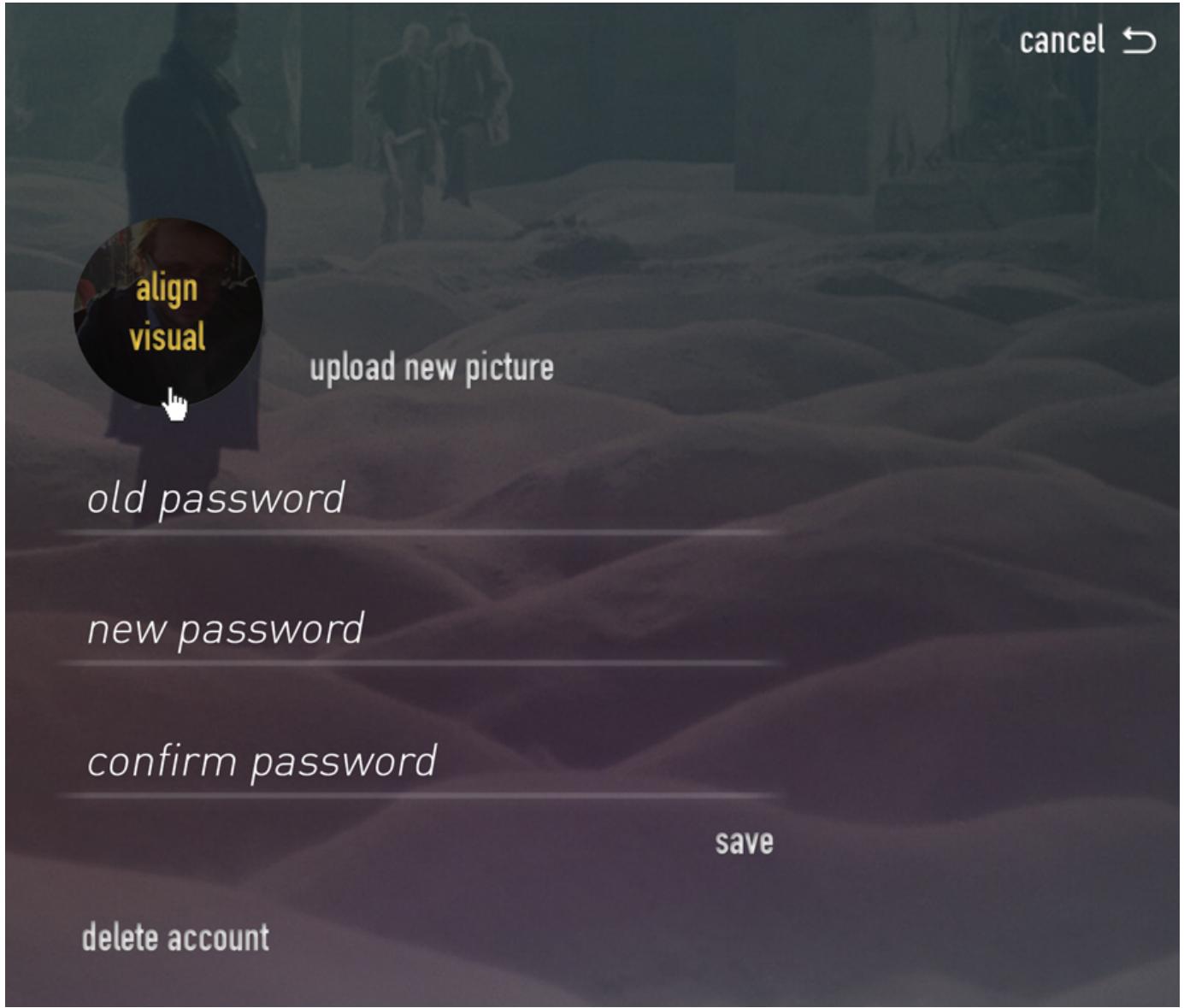
Utilise les sous-composants :

- Aucun

Contient deux liens hypermedia :

- Transition vers route "Home Feed"
- Transition vers route "Personal Settings"

1.1 - Composant Paramètres Profile



Description :

Ce composant permet à l'utilisateur de changer son mot de passe, changer son avatar, de supprimer son compte.

Est utilisé par les routes du router :

- [/settings](#)

Utilise les sous-composants :

- "message d'erreur"

Contient deux champs :

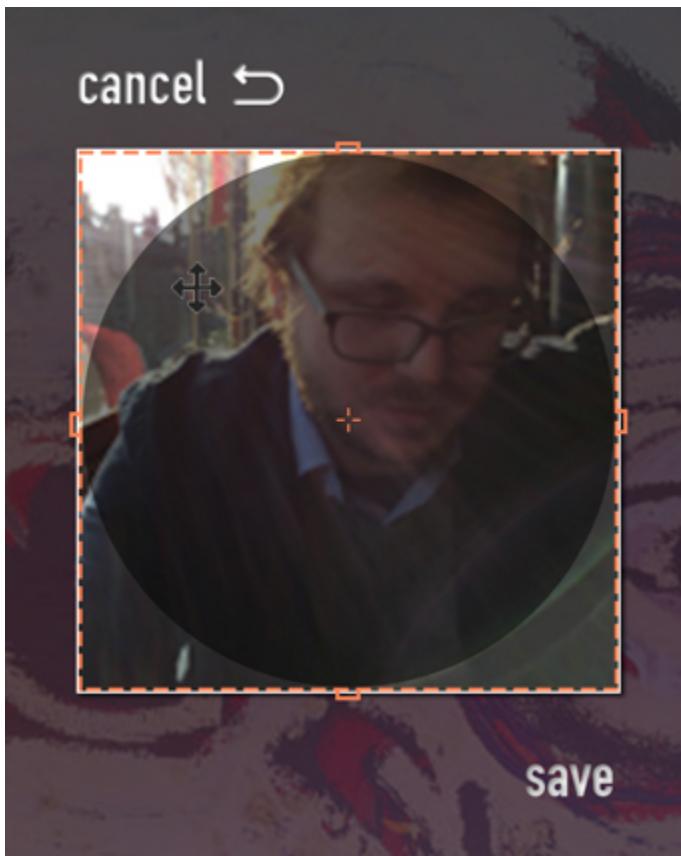
- old password
- new password * 2

Contient cinq liens hypermédia :

- Cancel
 - Transition vers route précédente
- Save

- Appel de l'API REST : </user/{user-id}/change-password> avec les deux champs (+email)
 - Si succès, message de succès remplace le champ pendant deux secondes et laisse les champs réapparaître.
 - Si échec, charger "message d'erreur"
- Upload new Picture
 - Open file browser, on select :
 - Appel de l'API REST : </user/{user-id}/update-picture> avec le champ picture (+email)
 - Si succès, replace URL dans la fenêtre oval avec la photo avatar
 - Si échec, charger "message d'erreur"
- Align Visual
 - Charge Composant "Aligner l'avatar"

1.1.1 - Composant Aligner l'avatar



Description :

Ce composant permet de centrer l'image de l'avatar.

Est utilisé par les routes du router :

- Change Avatar

Utilise les sous-composants :

- aucun

Contient trois champs :

- x
- y
- z

Contient cinq liens hypermedia :

- Cancel
 - Transition vers route "Personal Settings"

- Save
 - Appel de l'API REST : [/user/{user-id}/update-picture](#) avec les champs x, y, z (+email)
 - Si succès, transition vers route "Personal Settings"
 - Si échec, charger un message d'erreur

X - Composant Logo



3 - Composant Renseigner SMS

Voir 1.1.1 & 2.1.1.1 - Composant Renseigner SMS

4 - Composant Compte Bloqué

Voir 1.1.2 & 1.1.1.2 - Composant Compte Bloqué

Composants Agnustics

Composants qui ne dépendent pas de l'état de l'application

Super Composants

On liste ici les composants qui sont utilisé par plusieurs sub-composants de différentes routes.

Modèle du Store Redux

Nous définissons le modèle d'un store redux

```
{
  "user": {
    "userUUID": "UUID",
    "userToken": "abc",
    "userName": "Paul",
    "userLists": [
      {
        "listUUID": "uuid",
        "name": "Vienna 1910",
        "description": "blah",
        "relation": "admin",
        "settings": {
          "invitedAsAdmin": [
            {
              "userUUID": "UUID",
              "userName": "Jean"
            }
          ],
          "invitedToFollow": [
            {
              "userUUID": "UUID",
              "userName": "Jean"
            }
          ],
          "listViewableBy": "all",
          "feedViewableBy": "all"
        }
      },
      {
        "listUUID": "uuid",
        "name": "Truc",
        "description": "blah",
        "relation": "shared",
        "settings": {
          "invitedAsAdmin": [
            {
              "userUUID": "UUID",
              "userName": "Jean"
            }
          ]
        }
      }
    ]
  }
}
```

```
    }
],
"invitedToFollow": [
{
  "userUUID": "UUID",
  "userName": "Jean"
}
],
"listViewableBy": "all",
"feedViewableBy": "all"
}
}
],
"following": [
{
  "UUID": "uuid",
  "name": "Black"
}
],
"followers": [
{
  "UUID": "uuid",
  "name": "Black"
}
],
"blocked": [
{
  "UUID": "uuid",
  "name": "Black"
}
],
"inboxCount": "32",
"settings": {
"userPicture": "url",
"pictureAlignment": [
"X",
"Y",
"Z"
],
"postToHF": "everyone"
}
```

```
    } ,  
  
    "searchBarTitle": "search references / ~lists / #hashtags",  
  
    "homeFeed": {  
        "post": [],  
        "feed": [  
            {  
                "postUUID": "b",  
                "userUUID": "uuid",  
                "userName": "name",  
                "userPicture": "url",  
                "timestamp": "time",  
                "message": [  
                    [  
                        {  
                            "text": "blah"  
                        },  
                        {  
                            "reference": "blah"  
                        },  
                        {  
                            "text": "blah"  
                        },  
                        {  
                            "list": "blaj"  
                        }  
                    ]  
                ],  
                "reply": []  
            }  
        ]  
    },  
  
    "list": {  
        "description": "",  
        "post": {"canPost": "false", "message": []},  
        "feed": {  
            "canViewFeed": "true",  
            "posts": [  
                {  
                    "postUUID": "b",  
                    "userUUID": "uuid",  
                    "userName": "name",  
                    "userPicture": "url",  
                    "message": [  
                        [  
                            {  
                                "text": "blah"  
                            },  
                            {  
                                "reference": "blah"  
                            },  
                            {  
                                "text": "blah"  
                            },  
                            {  
                                "list": "blaj"  
                            }  
                        ]  
                    ],  
                    "reply": {  
                        "feed": [{  
                            "userUUID": "uuid",  
                            "userName": "name",  
                            "message": [  
                                [  
                                    {  
                                        "text": "blah"  
                                    }  
                                ]  
                            ]  
                        }  
                    }  
                }  
            ]  
        },  
        "reply": {}  
    }  
}
```

```
{
"reference": "blah"
},
{
"text": "blah"
},
{
"list": "blaj"
}
]
]
],
"reply": [ ]
}
}
],
"references": [
{
"referenceName": "bla",
"comments": "trucs",
"category": [ "1", "2", "3" ],
"schedule": "time",
"priority": "1",
"rating": {
"user": "10",
"qinetic": ""
},
"experienced": "false"
}
]

]},

"settings": {
"display": "false",
"invitedAsAdmin": [
{
"userUUID": "UUID",
"userName": "Jean"
}
],
"invitedToFollow": [
{
"userUUID": "UUID",
"userName": "Jean"
}
],
"listViewableBy": "all",
"feedViewableBy": "all"
},
"posters": [ { "userUUID": "uuid", "userName": "name" } ],
"mentionnedReferences": [ "ref1", "ref2" ],
"mentionnedLists": [ { "listUUID": "uuid", "listName": "truc" } ],
"mentionnedHashtags": [ "ff", "fdsf" ]
}

,
"reference": {
"referenceName": "",
"referenceDescription": "",
"referencePicture": "url",
"referenceAlignPicture": { "x": "0", "y": "0", "z": "0" },
"rating": {
"user": "10",
"qinetic": ""
},
"mainTags": [ ],
"secondaryTags": [ ],
"whereToExperience": [ ],
"associatedLists": [ ],
"mentionnedWithHashtags": [ ],
"feed": [
{
"postUUID": "b",
"userUUID": "uuid",
"date": "2023-09-25T14:23:45Z"
}
]
}
}
```

```
"userName": "name",
"userPicture": "url",
"timestamp": "time",
"message": [
[
{
"text": "blah"
},
{
"reference": "blah"
},
{
"text": "blah"
},
{
"list": "blaj"
}
]
]
],
"userEdits": "5",
"editsAwaitingVal": "1",
"accuracyOfReference": "70",

"review": {
"referenceName": {
"edit": "trucinsteadofbidule",
"verdict": "accepted",
"suggestion": [
{
"userUUID": "uuid",
"userName": "Paul",
"suggestion": "",
"upvotes": ""
}
],
"acceptedCount": "",
"disavowedCount": ""
},
"picture": {
"edit": {
"url": "",
"x": "0",
"y": "0",
"z": "0"
},
"verdict": "accepted",
"suggestion": [
{
"userUUID": "uuid",
"userName": "Paul",
"suggestion": "",
"upvotes": ""
}
],
"acceptedCount": "",
"disavowedCount": ""
},
"description": {
"edit": "do the diffing in component",
"verdict": "accepted",
"suggestion": [
{
"userUUID": "uuid",
"userName": "Paul",
"suggestion": "",
"upvotes": ""
}
],
"acceptedCount": "",
"disavowedCount": ""
},
"mainTags": {},
"secondaryTags": {},
"associatedList": {},
"whereToExperience": {}
}
```

```
    },
    },
    "search": {
      "active": "true",
      "searchBarSuggestion": [],
      "resultSearchList": {
        "display": "true",
        "listsCount": "10",
        "referencesCount": "10",
        "sharedReferencesCount": "10",
        "entries": [
          {
            "listUUID": "uuid",
            "listName": "truc",
            "listDescription": "name",
            "referencesCount": "10",
            "categoryFilters": [],
            "isFollower": "false",
            "isFollowing": "true",
            "hasSharedReference": "true"
          }
        ]
      },
      "resultSearchReference": {
        "display": "true",
        "referencesCount": "10",
        "entries": [
          {
            "referenceName": "name",
            "categories": [],
            "listDescription": "name",
            "rating": {
              "user": "10",
              "qinetic": ""
            }
          }
        ]
      },
      "resultSearchUser": {
        "display": "true",
        "usersCount": "10",
        "listsOfUsers": [
          {
            "listUUID": "uuid",
            "listName": "truc",
            "userUUID": "user",
            "userName": "name"
          }
        ],
        "entries": [
          {
            "userUUID": "uuid",
            "userName": "name",
            "userPicture": "url",
            "following": "true",
            "follower": "false",
            "sharedReferences": "true"
          }
        ]
      }
    }
  }
}
```

```
}\n}\n\n
```

<https://jmperezperez.com/high-performance-lazy-loading/>

Back end

API Doc & REST Routes

Nous utiliserons swagger hub pour générer notre documentation.

Account Management API

Tout ce qui a lien avec la création de comptes et l'authentification.

qinetic/login

description :

Permet de récupérer un token en passant un identifiant et un mot de passe. Ce token sera nécessaire pour accéder à toutes les autres routes de l'application.

appelé avec (POST) :

- {
 "email": "value",
 "password": "value"
}

réponse :

- token: aToken

ou

- toValidate: email

ou

- toValidate: phone

ou

- error: error

utilisé par :

- Le composant Login

qinetic/create-account

description :

Permet de récupérer un token en passant un identifiant et un mot de passe. Ce token sera nécessaire pour accéder à toutes les autres routes de l'application.

appelé avec (POST) :

- {
 "first-name": "value",
 "last-name": "value",
 "email": "value",
 "password": "value",
 "birthday": "value",
 "phone": "value",
}

réponse :

- success: success

ou

- error: error

utilisé par :

- Le composant Login

qinetic/email-verification

qinetic/email-verified

qinetic/send-sms

Conditionnel avec soit email, soit téléphone. Ne peut utiliser le téléphone que lors de la première connexion.

qinetic/sms-verification

description :

Vérifie le sms mais attention, deux cas de figure. Peut charger le composant pour remettre à jour le mot de passe ou non

appelé avec (POST) :

- {
 "email": "value",
 "password": "value"
}

réponse :

- token: aToken

ou

- toValidate: email

ou

- toValidate: phone

ou

- error: error

utilisé par :

- Le composant Login

User Management API

Personal Profile API

qinetic/user/{user-id}

qinetic/user/{user-id}/update-picture

qinetic/user/{user-id}/change-password

Trous champs

Old password peut prendre une option spéciale. sms-ver qui vérifiera dans la base de données que la plateforme a été vérifié et gardera l'info pour 5 minutes

Others' Ressources API

qinetic/load-user/{user-id}

qinetic/user/{user-id}/interact-user/{user-id}

Feed Management API

qinetic/feed/{feed-id}

peut prendre /{feed-id}?last={post-id}

Can return empty which tells the front-end not to attempt any fetch anymore

qinetic/feed/{feed-id}/post

qinetic/feed/{feed-id}/reply/{post-id}

List Management API

qinetic/load-list/{list-id}

qinetic/user/{user-id}/interact-list/{list-id}

References Management API

qinetic/load-reference/{reference-id}

qinetic/user/{user-id}/interact-reference/{reference-id}

Modèle pour RDBS

Ici, nous regroupons toutes les décisions concernant la gestion des modèles dans notre base de données PostgreSQL.

- (<https://dzone.com/articles/best-java-orm-frameworks-for-postgresql>)

DevOps

Ici, nous avons toutes les informations liées aux choix DevOps. Ainsi, nous avons pris la décision d'héberger PostgreSQL sur AWS ainsi que la machine virtuel Java sur une instance EC2. En effet, l'hébergement est gratuit pour un an sur la plateforme.

Je maintiens les codes de mon compte AWS et de la base de données sur mon ordi, merci de me les demander si nécessaire. Hibernate est déjà connecté à la base de données.

Voir propriétés de build.gradle pour les dependencies et application.properties pour les paramètres de connexion.

Related articles

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html>

<https://stackoverflow.com/questions/39521152/configure-springboothibernatepostgresql>

Algorithmes complexes à implémenter

Nous listons ici tous les algorithmes non triviaux nécessaires au bon fonctionnement de la plateforme.

Listes des algorithmes

1. Arbre trié pour recherche rapide et suggestion de recherches dans tous les champs de recherches (implémenté avec appel serveur pour recherches sur plateforme, react seulement pour les champs limités sur information déjà contenu dans le store Redux)
2. Validation client et serveur des formes d'enregistrement des utilisateurs
3. Analyse et historigramme des images dans le slideshow d'arrière plan afin de régler le dégradé et son opacité et avoir une lisibilité parfaite d'un texte blanc sur noir.
4. Programmation génétique pour suggestion de contenu par l'IA.
5. Ingénierie de la langue pour faire écrire l'IA

- Lazy loading du feed (<https://jmperezperez.com/high-performance-lazy-loading/>)

Certains de ces algorithmes méritent leurs sections propres car leurs développement n'est pas trivial

Related articles