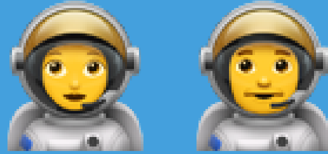


# Scaling a Single Page Application with GraphQL

# #whoami

**Charly POLY**

**Past**

➡️ JobTeaser alumni

➡️ 1 year @ A line

**Now**

# The context

> “ *Première plateforme collaborative de conseil, création et développement pour des projets marketing.*

- 2 products:
  - ACommunity:
    - marketplace
  - The platform:
    - projects
    - chat
    - ACL
    - timeline
    - selection lists

# The starting point

- Redux store for data

```
{

   models: {
      chats: {
         "8660f534-c425-4688-b4a9-d9ab11c6af85": { /* ... */ }
      }
   }
   // ...
}
```

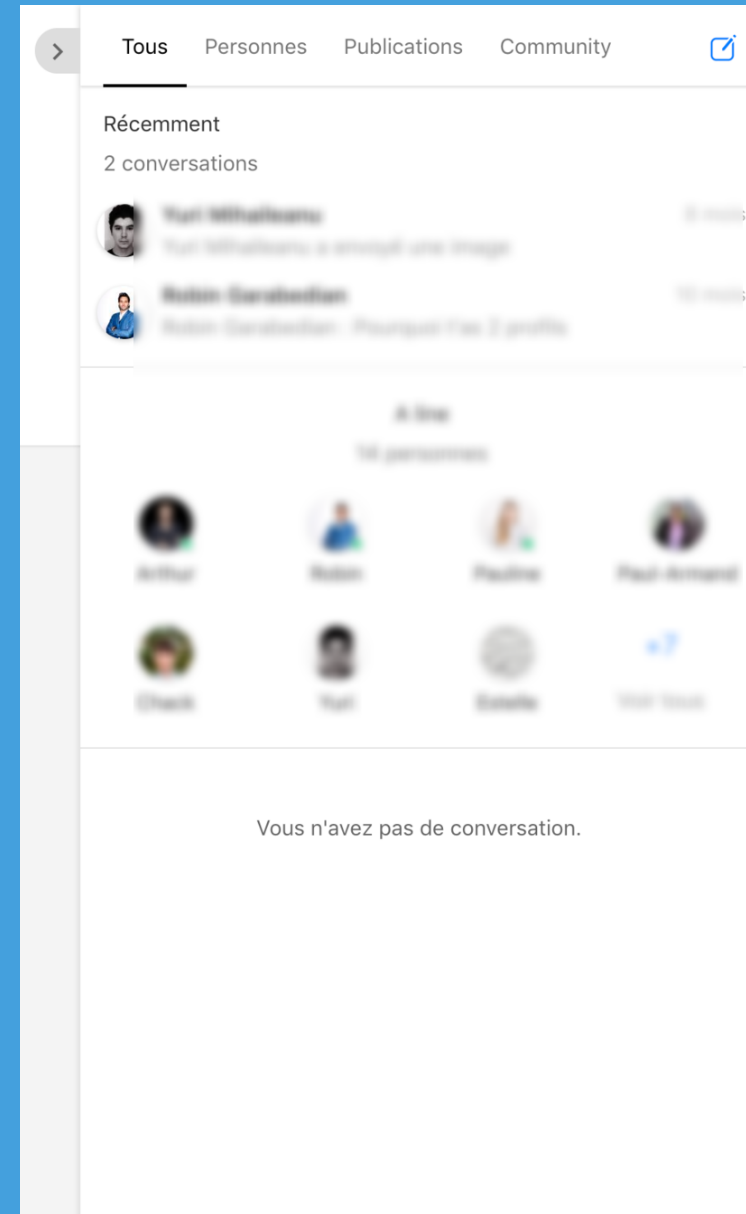
- one redux action per "CRUD":
  - `updateModel, createModel, deleteModel`
- Rest CRUD based service: **HttpService**

# The problem

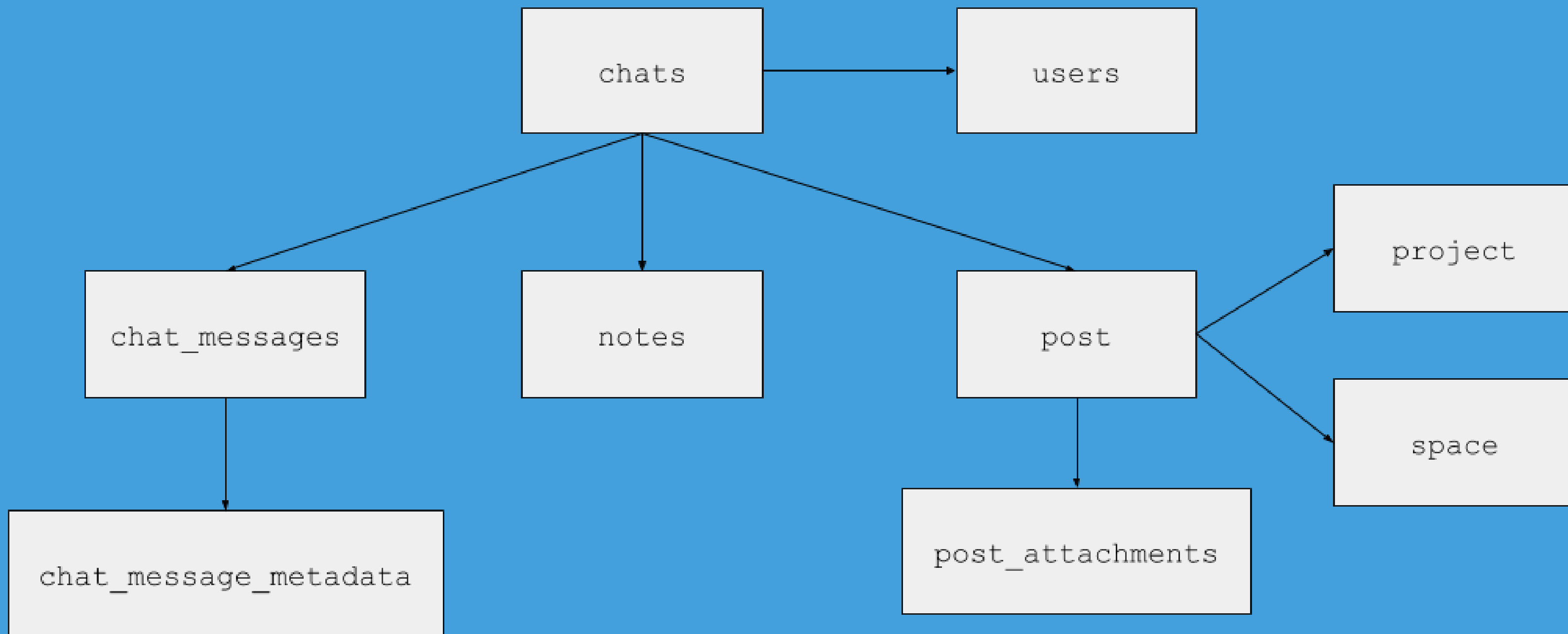
The chat and the timeline components

- **Timeline has posts that:**
  - have different types: media, note, text, links with preview
  - many types of medias: photos, videos
  - theater view (Facebook like)

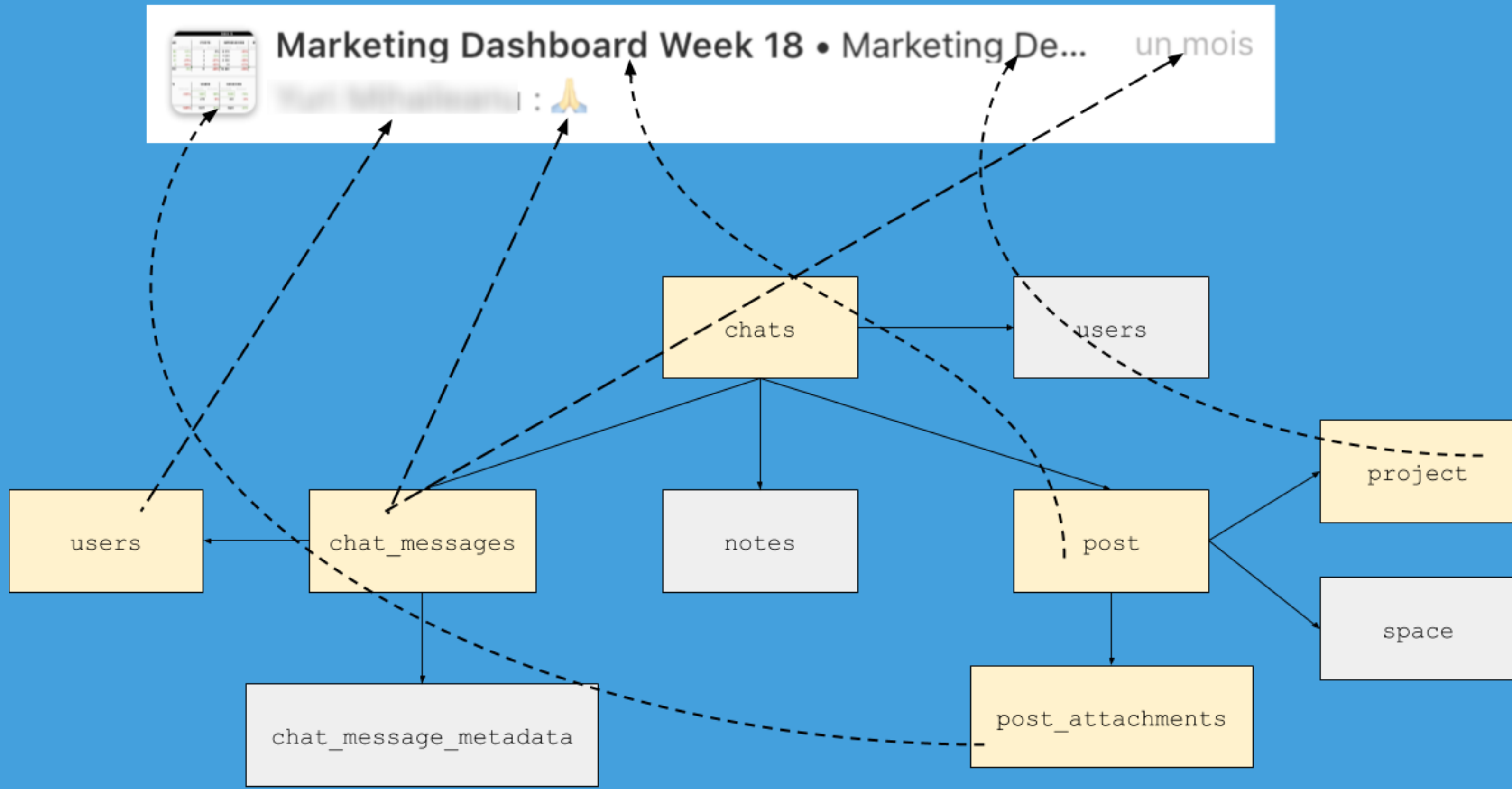
- **Chat are contextual:**
  - people chat (1-1, group chats)
  - post chat



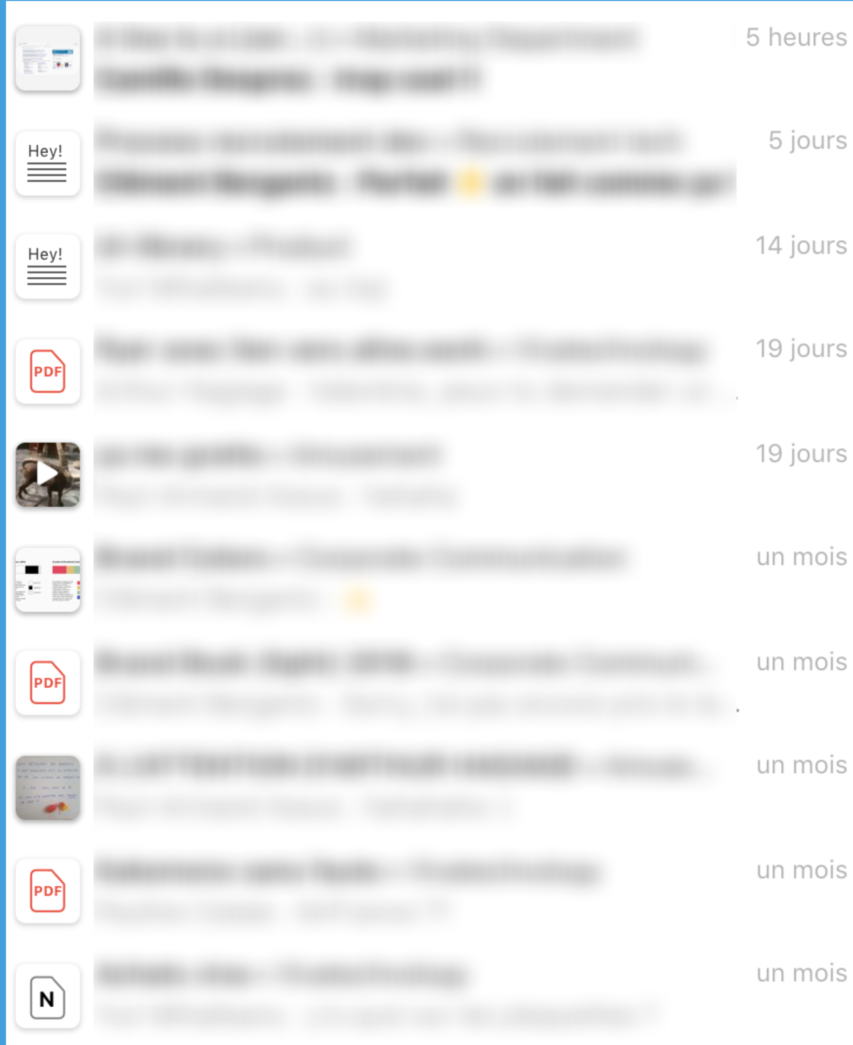
# The chat data



# The chat data



# The chat data




- post with text only
- post with file
- post with video
- post with image
- post with note attached

# The chat REST journey

## Numbers

For a average "list chats" query:

- ➡️ 20-50 chats of all types (without paging)
- ➡️ lot of n+1, n+2 requests per chat
- ➡️ lot of redux store updates
- ➡️ lot of react components re-render 💥💥💥

# The chat REST journey

## Solutions

**paging**
➡️ didn't solved requests issues

**"includes" on API side with n+1 objects included in response**
➡️ do not resolve n+2 queries issue

**preload all chats in a dedicated /preload API endpoint**
➡️ still some perf issue with realtime and updates refetches

# The chat REST journey

## The first working solution

**Hydra**

a custom client side relational cache with transactional redux dispatch

➡️ discover API request based on response data shape
➡️ wait all requests to finish before commit to redux
➡️ on update, ensure redux cache object relations are up-to-date

**Example**: a query on a chat can update a project object in cache

# The chat REST journey

## Hydra

# The chat REST journey

## The first working solution fail

### Users now have average of 80-100 chats

- client cache to many times invalid (too aggressive) 🔥
- API too slow 🔥

# The chat in GraphQL

## GraphQL and Apollo at the rescue

- ➡️ specific chat query with server-side optimisation
- ➡️ no more nesting issue (up to 4 levels easily)
- ➡️ models/data state handled by Apollo using Observables
- ➡️ advanced caching strategies for better UX

# The chat in GraphQL

```
query loadChatsList($offset: Int!, $limit: Int!, $type: String, $space_id: String) {
        chats_count(type: $type, space_id: $space_id)
        chats(offset: $offset, limit: $limit, type: $type) {
            id
            space_id
            last_chat_message_at
            total_chat_messages_count
            is_unread_for_current_user
            last_chat_message {
                id
                message
                created_at
                user {
                    id
                    first_name
                    last_name
                    picture_path
                }
                file_resources {
                    id
                    raw_url
                    cloudinary_url
                    metadata {
                        size
                        name
                        width
                        height
                        type
                    }
                }
                link_previews {
                    id
                    link
                    image_url
                    title
                    description
                }
                notes {
                    id
                    title
                    body
                    created_at
                    updated_at
                }
            }
            user_ids
            users {
                id
                first_name
                last_name
                picture_path
            }
            post {
                id
                created_at
                title
                body
                post_type
                project {
                    id
                    name
                }
                file_resources {
                    id
                    raw_url
                    cloudinary_url
                    metadata {
                        size
                        name
                        width
                        height
                        type
                    }
                }
                link_previews {
                    id
                    link
                    image_url
                    title
                    description
                }
                notes {
                    id
                    title
                    body
                    created_at
                    updated_at
                }
            }
            community_list {
                id
                space_id
                name
                portfolio_ids
                status
                space {
                    id
                    name
                }
            }
        }
    }
```


```
query loadChatsList($offset: Int!, $limit: Int!, $type: String, $space_id: String) {
        chats_count(type: $type, space_id: $space_id)
        chats(offset: $offset, limit: $limit, type: $type) {
            id
            space_id
            last_chat_message_at
            total_chat_messages_count
            is_unread_for_current_user
            last_chat_message {
                id
                message
                created_at
                user {
                    id
                    first_name
                    last_name
                    picture_path
                }
                file_resources {
                    id
                    raw_url
                    cloudinary_url
                    metadata {
                        size
                        name
                        width
                        height
                        type
                    }
                }
                link_previews {
                    id
                    link
                    image_url
                    title
                    description
                }
                notes {
                    id
                    title
                    body
                    created_at
                    updated_at
                }
            }
            user_ids
```

# The chat in GraphQL

## Why Apollo and not Relay?

- ➡️ Very flexible and composable API
- ➡️ Support custom GraphQL schema without "Relay edges"
- ➡️ more complete options on caching strategies
- ➡️ easier migration
- ➡️ possibility to have a local GraphQL schema (state-link)

# The chat in GraphQL

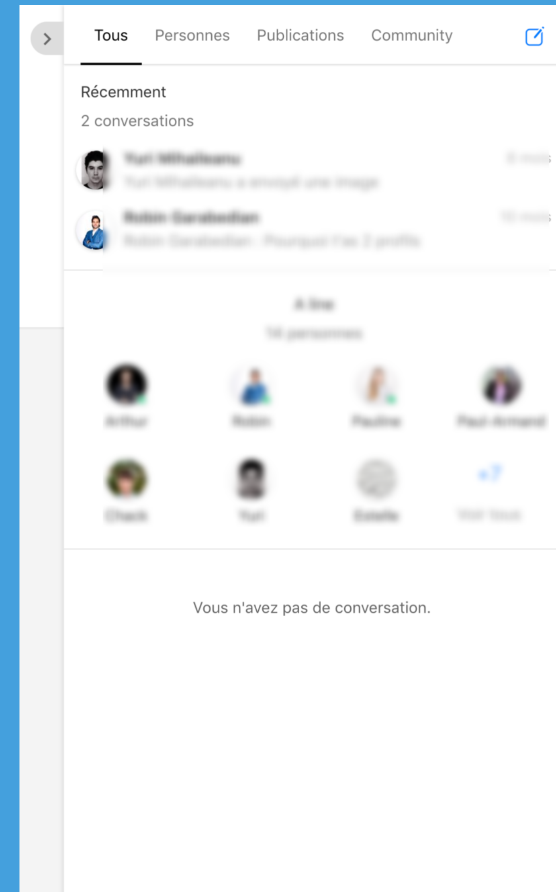
## Apollo: the smooth migration

```
query getMyData($id: String!) {
   chat(id: $id) {
      id
      post_id @export(as: "post_id")
      name
      post @rest(type: "Post", path: "/post/post_id") {
        id
        title
      }
      users {
        name
      }
    }
}
```

# The chat in GraphQL

## Apollo: the caching strategies

- "cache-first" (default)
- "cache-and-network"
- "network-only"
- "cache-only"
- "no-cache"



# The chat in GraphQL

## The Apollo super-powers

- **apollo-codegen**: TypeScript/Flow/Swift types generation
- **Link pattern**: like rake middleware on front side
- **Local state**: a nice alternative to redux

# Going further with GraphQL

## <ApolloForm>

```
import * as React from 'react';
import gql from 'graphql-tag';
import { ApplicationForm } from './forms';

const createTodoMutationDocument = gql`
    mutation createTodo($todo: TodoInputType!) {
        create_todo(todo: $todo) {
            id
        }
    }
`;

const form = p => (
    <ApplicationForm
        title="Todo Form"
        liveValidate={true}
        config={{
            mutation: {
                name: 'create_todo',
                document: createTodoMutationDocument
            }
        }}
        data={{}}
        ui={{}}
     />
);
```

### Todo Form

**There was some errors**

- FormError.create_todo.todo.name.required

todo

name*

completed

Cancel | Save

# Going further with GraphQL

## <ApolloForm>

1. Introspect your API GraphQL Schema
2. Build a JSON-Schema on available types and mutations
3. Create configuration files
4. Automatic form bootstrapping 🎉

# Going further with GraphQL

## <ApolloForm> advantages

- **distinct separation between data and UI:**
  - **Data structure**: what kind of data and validation are exposed
  - **UI structure**: how we want to display this data
- **Your front application is always synced to your API**
- **Easier UI-kit installation and maintainability**

# Conclusion

- GraphQL is useful for rich front-end application
- With Apollo, it can even replace local state management
- With TypeScript/Flow or Swift,
  it allows to keep clients and APIs synced