|  |
| --- |
| TPI : Messaging  Web-App |



Nicolas Maitre

Table des matières

[1 Analyse 2](#_Toc8799894)

[1.1 Introduction 2](#_Toc8799895)

# Analyse

## Introduction

## Contexte

Ce Projet a été réalisé à la suite d’un premier projet (dans le cadre du PreTPI).

Certaines fonctionnalités avaient déjà été développées, c’est pourquoi certains aspects de ces fonctionnalités ne sont pas approfondis dans ce document.

Les structures déjà présentes étaient :

* Le serveur de distribution de fichiers.
* Le serveur Websocket.
* La base de données MariaDB
* Le Framework JS de génération de pages.

Les fonctionnalités déjà présentes étaient

* Le login (simplifié)
* L’envoi de message

## Utilisateurs

Un utilisateur de l’application est identifié par un pseudo, une adresse email et un mot de passe. L’utilisateur peut prendre part à des groupes de discussion et envoyer des messages. L’aspect d’inscription et de connexion ne sera pas abordé dans ce projet car non demandé dans le cahier des charges.

## Conversations privées

Les conversations privées permettent à deux utilisateurs de s’envoyer des messages entre eux. Une seule conversation privée peut exister entre deux utilisateurs.

## Groupes

Un groupe est créé par un utilisateur. Le groupe doit contenir au minimum 3 personnes. Le groupe permet à ces utilisateurs de converser par message.

Il est possible d’ajouter un titre et une image au groupe.

## Messages

Les messages peuvent être envoyés par les utilisateurs dans les groupes ou dans les conversations privées.

Les messages contiennent du texte mais peuvent également contenir une image ou un lien hypertexte.

Le statut du message (envoyé, distribué, lu) est également affiché mais uniquement dans les conversations privées.

## Story 1 : « Répondre à un message »

Dans une discussion, je veux pouvoir répondre à un message précédemment envoyé.

## Story 2 :

## Story 3 : Ajout d’éléments complexes dans un message¨

L’utilisateur peut ajouter un lien (url) dans un message. Pour ce faire, il lui suffit de l’intégrer au texte de son message.

L’utilisateur peut également ajouter une image dans son message. Pour ce faire, une fenêtre d’ajout d’image est accessible depuis l’interface d’écriture de message.

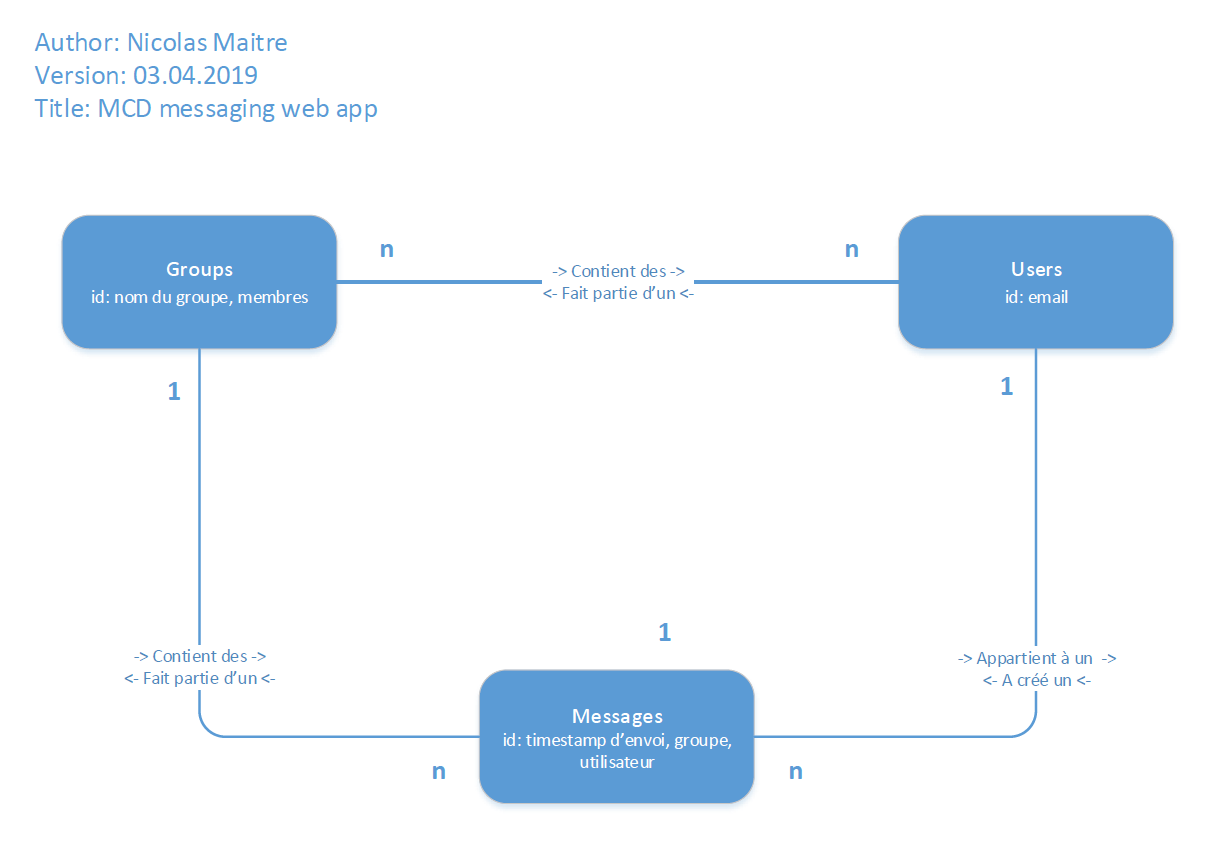
## Changements/précisions dans le cahier des charges

Certains changements ont été apportés au

* La fonctionnalité d’affichage de statut de message n’est accessible uniquement dans les groupes privés.
* Dans la page d’ajout de groupe, la liste d’utilisateurs contient tous les utilisateurs du système. En effet la fonctionnalité « d’amis » a été retirée.
* Le rendu hebdomadaire des documents peut se faire directement dans le repository git au lieu du mail.

## Données

Modèle Conceptuel des données



**Différences par rapport au PréTPI**

* La fonctionnalité d’ajout d’amis ayant été retirée, l’entité « friends » a été également retirée.

# Conception

## Story 1 : Répondre à un message

Pour pouvoir réaliser cette story, certaines fonctionnalités doivent être mise en place auparavant :

* L’affichage d’un groupe.
* L’affichage des messages.
* L’envoi et la réception des messages.

Pour ces tâches, la page client doit accéder à des données sur le serveur, par exemple la liste des groupes ou l’historique des messages. Pour ce faire, il a fallu réaliser l’API du site. *Conception de l’API en section 2.3.*

## Story 2 :

## Story 3 : Ajout d’éléments complexes dans un message

### Ajout d’url

L’ajout d’url n’est pas géré côté serveur. La détection de l’url se fait au moment de l’affichage d’un message.

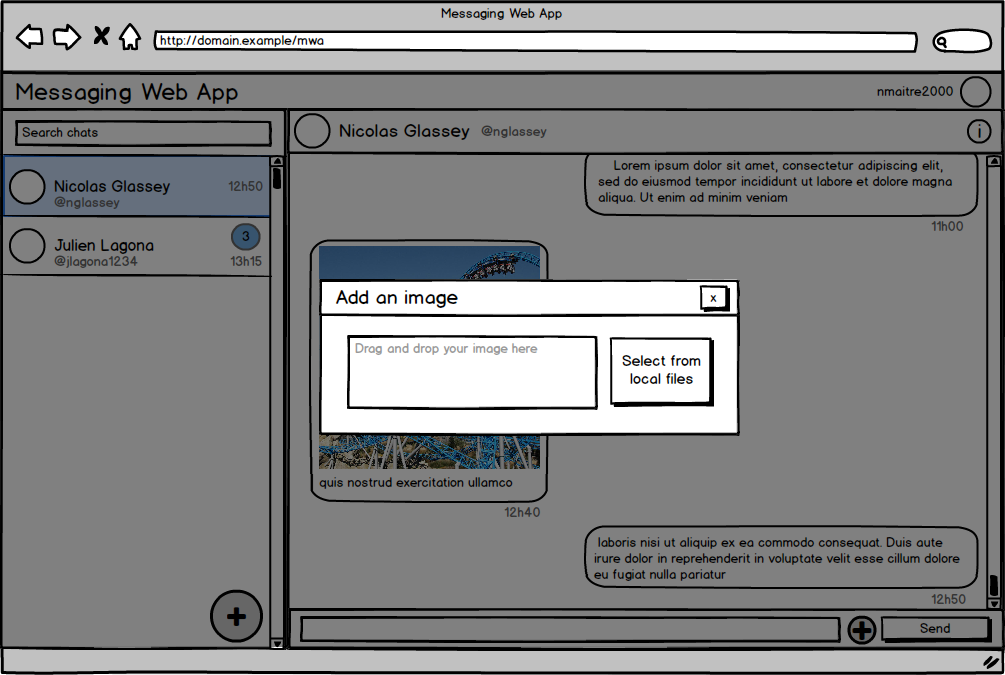
La fonctionnalité est gérée de la même manière dans les commentaires YouTube.

### Ajout d’image

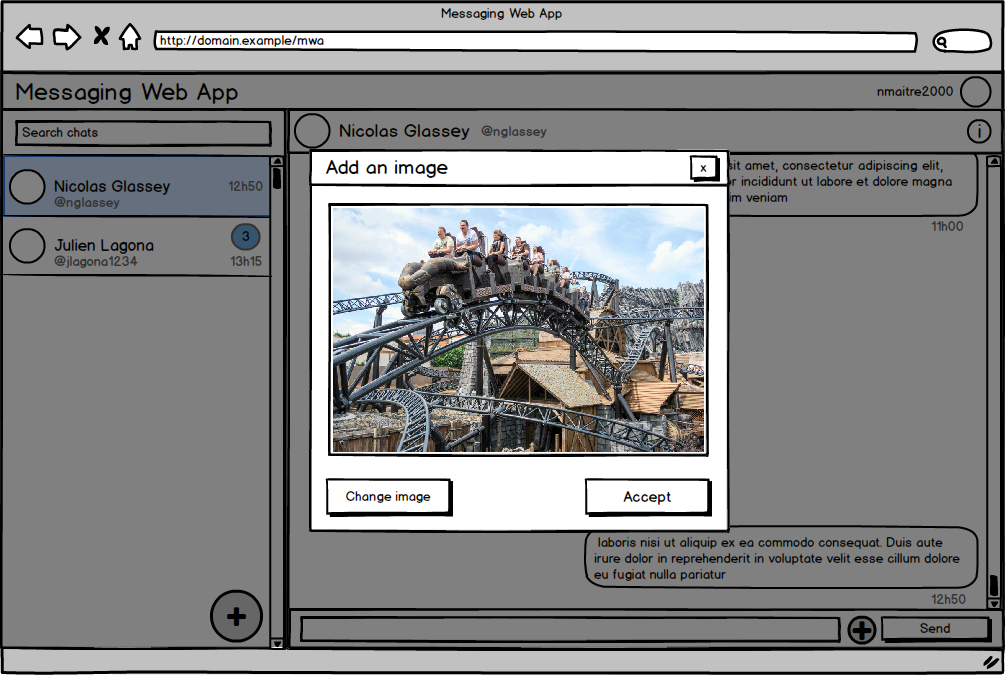
Pour pouvoir réaliser cette fonctionnalité, il faut développer les fonctionnalités suivantes :

* Files API (2.6), pour permettre l’affichage des images dynamiques.
* Upload d’images, pour permettre l’envoi des fichiers image vers le serveur

#### Maquette graphique de l’ajout d’image



Après avoir appuyé sur le bouton C:\Users\Nicolas.MAITRE\AppData\Local\Microsoft\Windows\INetCache\Content.Word\+button.pngà côté de la zone d’écriture de message, une fenêtre va s’ouvrir, demandant la sélection d’un fichier.



Après avoir sélectionné un fichier, la fenêtre de confirmation de l’image va s’afficher.

L’image n’est ajoutée qu’après avoir appuyé sur « Accepter ».

## API

L’api permet à l’interface client d’échanger des données avdec le serveur.

### Technologies

#### Comparaison des technologies utilisables pour la création d’une API

|  |  |  |
| --- | --- | --- |
|  | Avantages | Inconvénients |
| WebSocket | - Connexion unique  - Connexion déjà négociée dans l’application de messagerie.  -Connexion bidirectionnelle. | - *Stateless* (car *stateful* impossible. Donc difficile à tester en dehors du navigateur)  - Plus complexe de mettre en place un flux fonctionnel d’appel. |
| Api http | - Gestion déjà intégrée dans le navigateur.  - Gestion d’erreur standardisée.  - *Stateful* possible (Donc l’utilisation de *Postman* pour les test est possible) | - Connexion renégociée à chaque requête.  - Performance légèrement inférieure[[1]](#footnote-1). |
| Stockage par fichier texte | - Simplicité d’appel. | - Données difficiles à modifier.  - Performances |

Le choix s’est porté sur la création d’une api http. En effet, il me parait important de pouvoir réaliser des tests sur celle-ci en dehors de l’interface graphique du site.

#### Interface client

Le client appelle l’api en XHR (JavaScript). Les appels sont gérés dans la classe JavaScript « ApiManager ».

### Paramètres

Trois paramètres peuvent être passés dans la requête :

* **Action (obligatoire)(url)**  
  L’action définit le type d’action que l’on souhaite réaliser sur le serveur. Par exemple « getMessages » pour récupérer des messages.
* **Paramètres (Obligatoire)(url query)** *Params*  
  Les paramètres sont encodés en JSON et permettent à l’API d’envoyer le bon résultat sur l’endpoint.
* **Options (url query)**  
  Les options définissent le format de données en retour. Par exemple le nombre de données et l’ordre de tri. Ce paramètre est encodé en JSON
* **Authentification (Obligatoire)(Headers)** *Auth*Ce paramètre contient l’identifiant et le *token* de l’utilisateur souhaitant réaliser la requête. Ils permettent de déterminer si l’utilisateur a l’autorisation d’accéder à la ressource. Ce paramètre est également encodé en JSON.
* **Données(body)** *Data*.  
  Ce champ contient des données brutes, par exemple le contenu d’un fichier image. Il peut également contenir des données encodées en JSON.

### Format de réponse

La réponse contient les paramètres suivants :

* Les données encodées en JSON (body)
* Le statut http (statut http 200, 404, 500, etc.)

### Exemple de requête de récupération d’informations

L’api permet de récupérer des informations depuis la base de données du serveur.

Dans cet exemple, on va récupérer les 10 derniers messages du groupe référencé par l’id « 0123-1234-2345-3456-4567 ».

Domain.example/api/getMessages ? params = {groupId :’0123-1234-2345-3456-4567’} &options= {start : 0, limit : 20, order : {by : ‘creation\_time’, way : ‘desc’}}

### Exemple de requête d’envoi de fichier

L’api est également utilisée pour l’ajout de fichiers sur le serveur, notamment les images.

Dans cet exemple, la requête permet d’envoyer un fichier sur le serveur.

Domain.example/api/uploadFile ? params = {}}

Le body de la requête contient le contenu du fichier.

La réponse de la requête contient l’id du fichier envoyé.

## WebSocket server

## Files API

La « Files API » permet de distribuer des fichiers dynamiques vers l’interface graphique.

Les fichiers sont stockés à l’extérieur de la structure de dossiers du projet. Par défaut, ils sont stockés dans le dossier « /var/messaging\_web\_app/files ».

L’accès se fait par une requête http et la réponse contient les données brutes.

L’avantage de ce type de retour, est qu’une image peut directement être appelée dans la propriété « src= » d’une balise « <img> » par exemple.

**Exemple de requête**

La requête contient uniquement la référence (UUID) du fichier.

Dans cet exemple, on va récupérer le fichier dont la référence est « feb0daf0-77a4-11e9-8f9e-2a86e4085a59 ».

Domain.example/files/feb0daf0-77a4-11e9-8f9e-2a86e4085a59

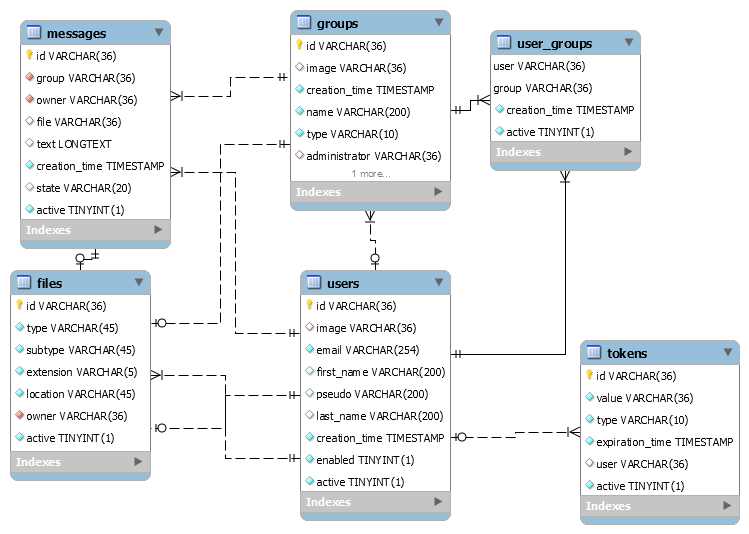
## Base de données

### Technologie

La base de donnée fonctionne sous Maria DB, un *fork* de MySQL. Les fonctionnalités et la syntaxe sont similaires.

Le choix de cette technologie s’est fait pour privilégier le logiciel libre. En effet MySQL appartient à Oracle. Ce moteur de base de données avait également été utilisée lors du PréTPI et a donc été réutilisé.

### Modèle Logique des données



Champs mot de passe

<https://stackoverflow.com/questions/386294/what-is-the-maximum-length-of-a-valid-email-address>

## Contact à l’API

L’API est appelée à l’aide de la « Fetch API ». Malheureusement elle n’est pas encore supportée par tous les navigateurs, comme Safari et IE. Un polyfill existe cependant pour combler cette lacune.

<https://github.com/github/fetch#browser-support>

# Réalisation

## API

### Endpoint UploadFile

* Envoi de fichiers depuis une interface utilisateur
* Request body parsing
* File infos in params

## Story 3 : Ajout d’éléments complexes dans un message

### Ajout d’url

Les urls sont trouvées à l’aide d’un regex. Elles sont ensuites intégrées au texte du message, empêchant ainsi les failles XSS.

# Sources

<https://blog.feathersjs.com/http-vs-websockets-a-performance-comparison-da2533f13a77>

1. <https://blog.feathersjs.com/http-vs-websockets-a-performance-comparison-da2533f13a77> [↑](#footnote-ref-1)