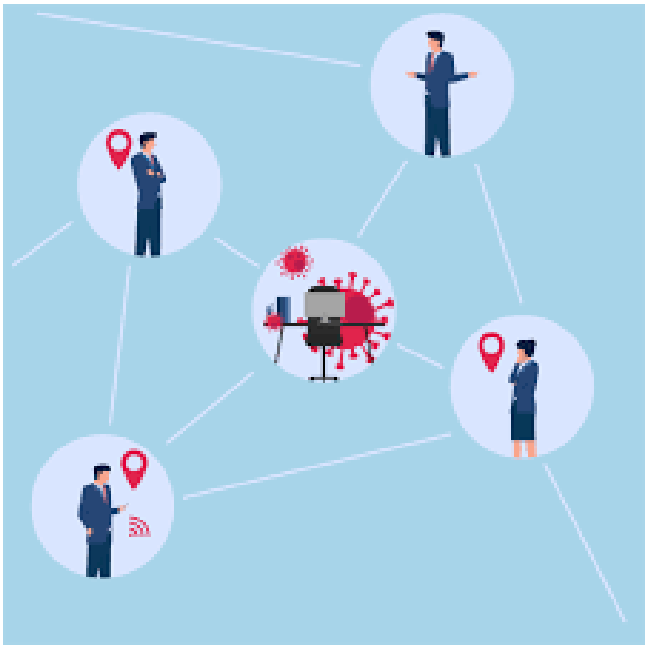
SORBONNE UNIVERSITES

Système Chat



Rapport de TX

13.09.2021

—

Dylan CORNELIE
Quoc Gia Cat TRAN

Table des matières

Objectif

L'objectif de cette TX est de développer une application de chat qui permettra d'améliorer l'interactivité des réunions entre plusieurs équipes travaillant à distance.

Lors d'une réunion, un ou plusieurs des participants de la réunion occupent le rôle de coach. Un coach a la possibilité pendant la réunion d'envoyer des documents ou des messages par le biais de l'application de chat à l'ensemble des participants de la réunion. Le coach a aussi la possibilité d'envoyer des messages privés aux participants afin de leur rappeler les règles de bienséances d'une réunion mais aussi les féliciter s'ils sont actifs pendant la réunion et que leurs remarques sont pertinentes.

Cahier des charges

Avant de commencer le développement, nous avons décidé de rédiger un cahier des charges afin de s'assurer que notre application réponde à toutes les fonctionnalités attendues.

Concernant l'authentification et la création de compte, l'utilisateur a la possibilité de créer un compte en renseignant un e-mail, un login qui doivent être unique, un mot de passe, son prénom et son nom. Pour s'authentifier, l'utilisateur doit renseigner son e-mail et son mot de passe.

À propos des meetings, sur la page d'accueil l'utilisateur a une vue globale sur tous les meetings auxquels il participe. S'il le souhaite, il peut avoir plus d'informations sur ce meeting en cliquant sur le bouton du meeting, donnant notamment accès à la liste des participants du meeting, laissant la possibilité de quitter le meeting, ajouter des participant ou en retirer dans le cas où l'utilisateur est propriétaire du meeting. Si le propriétaire d'un meeting souhaite quitter son meeting, cela entraînera la suppression de tout le meeting.

Sur la page d'accueil, l'utilisateur a la possibilité de créer un nouveau meeting, rejoindre un meeting et aussi se rendre sur la page de gestion de son compte.

Sur la page de gestion du compte, l'utilisateur a la possibilité de se déconnecter, souscrire aux push notifications, modifier sa photo de profil ainsi que son nom, son prénom et son mot de passe.

Sur la page de chat, dans le cas où l'utilisateur est coach, il a la possibilité d'envoyer des messages privés ainsi que des messages à destination de tous les participants du chat. Les utilisateurs qui ne sont pas coachs n'ont pas la possibilité d'envoyer des messages. Les coachs, ainsi que le propriétaire du meeting peuvent désigner d'autres coachs, retirer le statut de coach et exclure un participant du meeting.

Les messages peuvent contenir du texte avec des liens, ils peuvent aussi contenir un unique document, si c'est une photo celle-ci sera affichée dans la bulle de message, si c'est un document une icône sera affichée afin de proposer à l'utilisateur de télécharger la pièce jointe.

Pour ajouter des utilisateurs à un meeting, il suffit soit de rentrer leur login si nous les ajoutons en utilisant leur login, soit de flasher leur QR Code. Le QR Code s'affiche lorsque nous cliquons sur le bouton « Join a meeting » sur la page d'accueil. Tous les utilisateurs ont la possibilité d'ajouter des nouveaux participants au meeting.

Réalisation

Maquette

Dans un premier temps, l'objectif était de réaliser une maquette de l'application que nous allons proposer. Cette maquette présente chaque page et fonctionnalité de notre application avant que nous commencions le développement.

L'intérêt de cette maquette est de se lancer dans le développement avec les idées claires pour tout le monde au niveau des attentes et du rendu final. Cela a demandé de faire beaucoup de recherches sur les domaines de l'UI/UX design (User Interface / User Experience).

Afin de réaliser cette maquette, nous avons utilisé l'outil Figma, qui est un outil de design basé sur le web. Les maquettes sont disponibles en annexe.

Il est important de noter qu'au fur et à mesure que nous ajoutions des fonctionnalités durant le projet, nous avons fait évoluer le design de notre application.

Solutions techniques choisies

Client

L'application doit pouvoir s'utiliser sur des plateformes mobiles et notamment Android, nous nous sommes donc dirigés vers les technologies React Native. Cependant après avoir exploré ce framework, nous nous sommes rendus compte que le développement allait être assez lent et compliqué car nous devons réaliser une application complète en partant de zéro.

Nous nous sommes alors penchés sur les Progressive Web App (PWA), qui sont des applications web qui ont le même comportement que les applications mobiles natives. Ce genre d'application peut tout aussi bien être consultée via un navigateur classique qu'être installée. Que ce soit sur tablette, téléphone ou ordinateur et ce peu importe la plateforme. Nous avons donc finalement décidé de réaliser l'application de chat en ReactJS, pour dans un second temps transformer notre web app en PWA.

ReactJS

ReactJS est une bibliothèque JavaScript qui comme React Native est développée par Facebook. Cette bibliothèque permet de créer des applications web assez facilement par l'utilisation de composants et d'états. De plus, de nombreuses librairies existent pour élargir les fonctionnalités de ReactJS comme avec Redux.

Redux

Redux est une bibliothèque JavaScript qui permet de gérer les états d'une application. Pour plus d'informations voici un site qui explique très bien l'utilité de cette librairie : https://dev.to/codebucks/what-is-redux-simply-explained-2ch7 Redux a permis de rendre notre code mieux organisé

Un des problèmes avec Redux, est qu'on ne peut pas dispatcher d'événement asynchrone comme des requêtes à une API, pour remédier à ce problème nous avons utilisé la bibliothèque redux-thunk. Voici un site qui explique l'utilité de cette librairie : https://daveceddia.com/what-is-a-thunk/

React Router

Dans l'optique de gérer la navigation entre plusieurs pages, nous avons utilisé la bibliothèque react-router-dom. Cette bibliothèque permet de gérer le routage dans les applications react en utilisant un routage dynamique, c'est-à-dire que le routage est effectué lorsque l'application est rendu dans le navigateur.

Qrcode react & React qr reader

Pour générer des QR Codes, nous avons utilisé la bibliothèque qrcode react. Cette bibliothèque permet de générer des QR Codes.

Afin de scanner des QR Codes, nous avons utilisé la bibliothèque react-qr-reader.

Workbox

Afin de transformer notre application web en PWA, nous avons utilisé la bibliothèque Workbox de Google et plus précisément le module node workbox-build. Cette bibliothèque permet de mettre en place des PWA qui suivent les recommandations de Google concernant les PWA. Il est important de noter que Google est une des entreprises qui pousse le développement des PWA sur le web.

React icons

Pour la majorité des icônes utilisées dans l'application, nous avons utilisé la bibliothèque react-icons. Cette bibliothèque regroupe plusieurs icônes de pack d'icônes populaires.

React loader spinner

Pour l'affichage des spinner, nous avons utilisé la bibliothèque react-loader-spinner. Cette bibliothèque propose tout un tas de composants spinner faciles d'utilisation.

Html parser

Afin de parser les balises HTML permettant de rendre les liens cliquables dans les bulles de chat, nous avons utilisé la bibliothèque html-react-parser.

Axios

Afin de réaliser les différentes requêtes à l'API, nous avons utilisé la bibliothèque Axios. Axios est une bibliothèque complète très populaire pour réaliser des requêtes vers des serveurs en raison de sa simplicité, Facebook recommande son utilisation avec React.

Sass

Pour le style de l'application web, nous avons préféré utiliser le langage SASS, pour le compiler en CSS nous avons utilisé le compilateur node-sass. SASS nous permet de rendre nos feuilles de style modulaires et plus claires par une syntaxe qui est légèrement modifiée par rapport au CSS classique.

Socket.io

Afin de recevoir les messages des autres utilisateurs en temps réel, nous avons utilisé la bibliothèque socket.io qui repose sur le protocole websocket.

Serveur

Une API sert à exposer localement ou sur le web un catalogue de fonctionnalités au service d'un programme. Le but étant de pouvoir faire communiquer des systèmes avec l'interface pour échanger des données.

Flask

Flask est un cadre de travail (framework) Web pour Python. Ainsi, il fournit des fonctionnalités permettant de construire des applications Web, ce qui inclut la gestion des requêtes HTTP et des canevas de présentation. Nous allons créer une application Flask très simple, à partir de laquelle nous construirons notre API.

Flask-RESTX

Flask-RESTX est une extension pour Flask qui ajoute la prise en charge de la création rapide d'API REST. Flask-RESTX encourage les meilleures pratiques avec une configuration minimale. Il fournit une collection cohérente de décorateurs et d'outils pour décrire votre API et exposer correctement sa documentation (en utilisant Swagger).

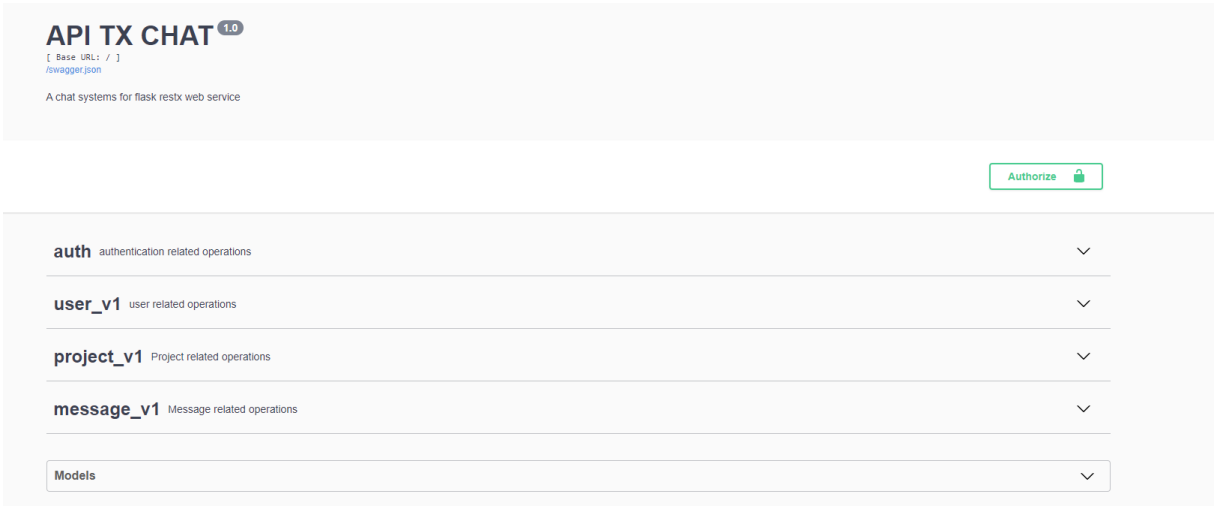


Figure 1. L'interface d'API (en Swagger)

Flask-Cors

Une extension Flask pour gérer le partage des ressources entre origines multiples (CORS), rendant possible l'AJAX d'origine croisée.

Ce package a une philosophie simple : lorsque vous souhaitez activer CORS, vous souhaitez l'activer pour tous les cas d'utilisation sur un domaine. Cela signifie qu'il n'y a pas de manipulation avec différentes en-têtes, méthodes, etc. autorisés.

SQLAlchemy & Flask-Migrate & Psycong2

SQLAlchemy est la boîte à outils Python SQL et le mappeur relationnel objet qui donne aux développeurs d'applications toute la puissance et la flexibilité de SQL.

Flask-Migrate est une extension qui configure Alembic de manière appropriée pour fonctionner avec les applications Flask et Flask-SQLAlchemy. Au niveau des migrations de bases de données proprement dites, tout est géré par Alembic pour obtenir exactement la même fonctionnalité.

Psycopg est l'adaptateur de base de données PostgreSQL pour le langage de programmation Python. Ses principales caractéristiques sont l'implémentation complète de la spécification Python DB API 2.0 et la sécurité des threads.

PyJWT

PyJWT est une bibliothèque Python qui vous permet d'encoder et de décoder des jetons Web JSON (JWT). JWT est une norme industrielle ouverte (RFC 7519) permettant de représenter en toute sécurité les réclamations entre deux parties.

Redis Sub/Pub

Redis est un système de messagerie Publish/Subscribe rapide et stable, il peut être utilisé comme plate-forme d'éditeur/abonné. Dans ce modèle, les éditeurs peuvent envoyer des messages à n'importe quel nombre d'abonnés sur une chaîne. Ces messages sont de type fire-and-forget, en ce sens que si un message est publié et qu'aucun abonné n'existe, le message s'évapore et ne peut pas être récupéré.

Flask-SocketIo (Python-socketio)

Flask-SocketIO permet aux applications Flask d'accéder à des communications bidirectionnelles à faible latence entre les clients et le serveur

Flask-Mailman

L'extension Flask-Mail fournit une interface simple pour configurer SMTP avec l’application Flask et pour envoyer des messages à partir de vos vues et scripts.

Architecture de serveur

Notification

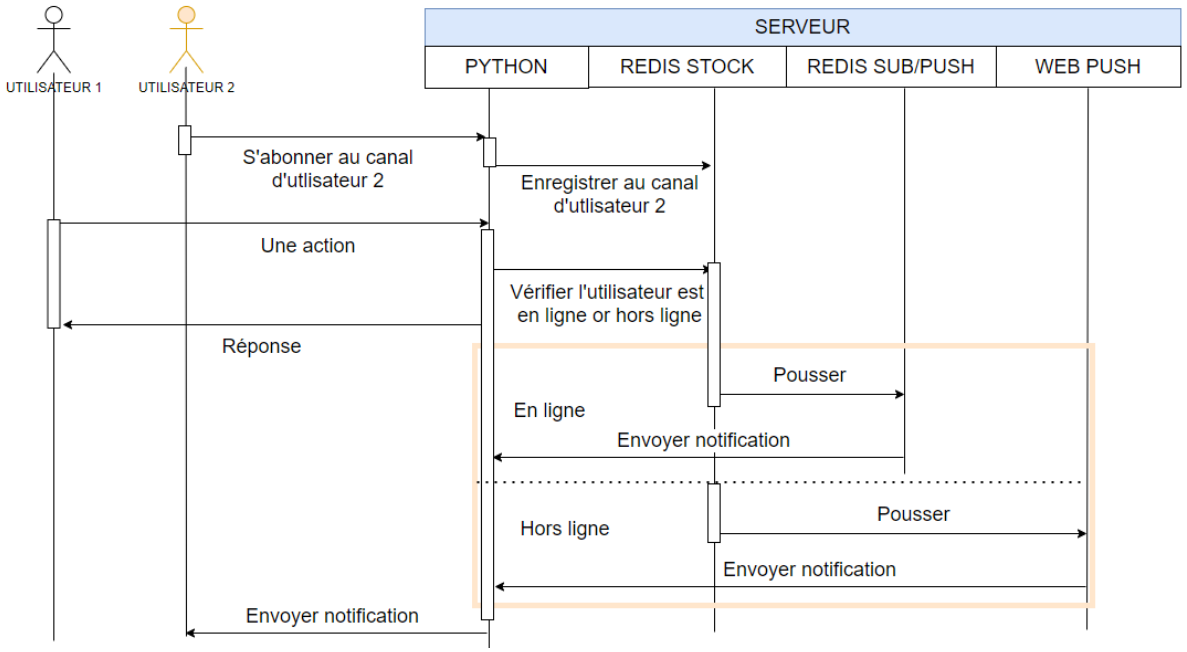


Figure: La diagramme de séquence de notification

Dans notre application, nous implémentons une fonction qui notifie les activités des utilisateurs.

Tout d'abord, tous les utilisateurs doivent s'abonner à son canal sur le serveur. Le système va enregistrer cette chaîne au Redis Stock, pour raccourcir le temps de vérification des données grâce aux données en mémorisant directement sur la RAM.

Quand un utilisateur envoie une action au serveur (par exemple: création de la salle de discussion), après de cette création, le serveur répond à l'expéditeur. Dans le même temps, le système vérifiera le canal du reste des utilisateurs qui va avoir besoin d'une notification sur Redis Stock. Si ses canaux sont présents dans le Redis, c'est-à-dire qu'ils sont en ligne, le message de notification va être envoyé au Redis Sub/Pub. Sinon, ce message va être envoyé au Web Push. Ensuite, il va répondre au système pour envoyer des notifications à ceux qui en ont besoin.

Socket

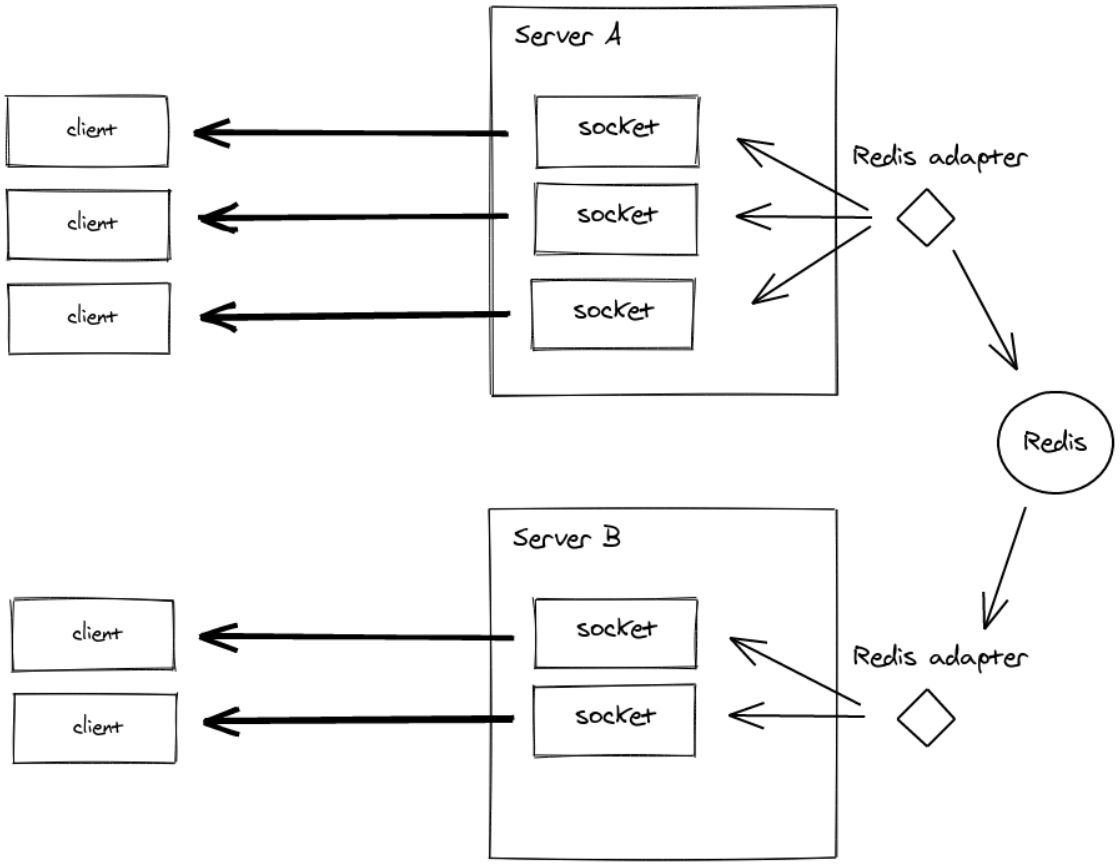


Figure: L'architecture de socketio avec Redis Sub/Pub

Chaque serveur Socket.IO reçoit ce paquet et le diffuse vers sa propre liste de sockets connectés.

Chaque paquet envoyé à plusieurs clients :

- envoyé à tous les clients correspondants connectés au serveur
- publié dans un canal Redis, et reçu par les autres serveurs Socket.IO du cluster

L'adaptateur Redis étend la fonction de diffusion de l'adaptateur en mémoire : le paquet est également publié sur un canal Redis. L'adaptateur en mémoire stocke les relations entre les sockets et les salles.

Organisation du code

Client

Le projet a été créé avec la commande « npx create-react-app » et donc l'organisation du code découle de l'organisation proposée par la commande.

Dans le dossier public, nous retrouvons tous les assets statiques de notre application, à savoir:

- Les logos de notre application aux différents formats
- La bannière utilisée sur certaines pages
- L'image de profil par défaut des utilisateurs
- Une icône utilisée pour accéder à la page « mon compte » sur la page d'accueil.

Toujours dans le dossier public, nous retrouvons « l'index.html » , fichier où tout notre site web réside, le fichier « app.js » qui est inclus dans « l'index.html » permettant d'enregistrer un service worker, obligatoire afin de transformer notre application en PWA et d'accéder au web push. De même pour le fichier manifest.json, il contient tout un tas d'informations sur le comportement de notre PWA. Pour finir le fichier robots.txt est un fichier qui est lu par les robots qui parcourent le web pour leur indiquer à quelles ressources ils peuvent accéder.

Dans le dossier « src » , nous retrouvons tous le code sources de notre application React, ainsi que le style.

- Dans le sous-dossier « actions », nous retrouvons toutes les actions liées à Redux
- Dans le dossier « components » nous retrouvons tous les composants de notre application React.
  - Le dossier « pages » correspond à tous les composants qui représentent les pages de notre application
  - Le dossier « routes » correspond aux composants de routage de notre application
  - Les dossiers « home » et chat correspondent à tous les sous composants des pages du même nom
  - Le dossier « utils » inclus des components généraux, réutilisables dans plusieurs pages différentes
- Dans le dossier « reducers » nous retrouvons tous les reducers de notre application, qui sont ensuite combinés dans le fichier « index.js ». C'est ici aussi des fichiers liés à Redux
- Dans le dossier style, nous retrouvons toutes les feuilles de style au format .scss, l'organisation des feuilles de style est la même que pour les composants. Le fichier index.scss combine toutes les feuilles de style pour avoir la feuille de style finale.
- Dans le dossier « utils », nous retrouvons des fonctions souvent utilisées, elles y sont regroupées afin qu'elles soient centralisées dans un même fichier et facilement modifiées.

Enfin dans le dossier « src », le fichier « App.js » représente notre application au complet et le fichier « index.js » nous permet d'injecter cette application dans notre fichier « index.html » présenté précédemment.

Dans le dossier « client », il existe plusieurs fichiers :

- «.dockerignore», qui permet à Docker d'ignorer les fichiers qui y sont renseignés
- «.gitignore qui permet à git d'ignorer les fichiers qui y sont renseignés
- «Dockerfile» qui permet de créer une image de production de la partie client de notre application afin de la lancer dans un conteneur Docker
- «nginx.conf» qui est un fichier de configuration du serveur web nginx, il est utilisé pour configurer nginx dans le conteneur docker
- «package.json» qui liste les informations concernant notre application, les dépendances, le nom, le numéro de version…
- «README.md» dans lequel une petite notice est écrite afin de lancer notre application
- «sw-build.js» et sw.js qui sont des fichiers qui permettent à workbox de générer les règles de preCache et de cache de notre application.
- «yarn.lock» qui est un fichier propre au gestionnaire de paquet yarn

Serveur

Le structure de «serveur» est organisée comme l'image ci-dessous et on définit des dossiers et plusieurs fichiers.

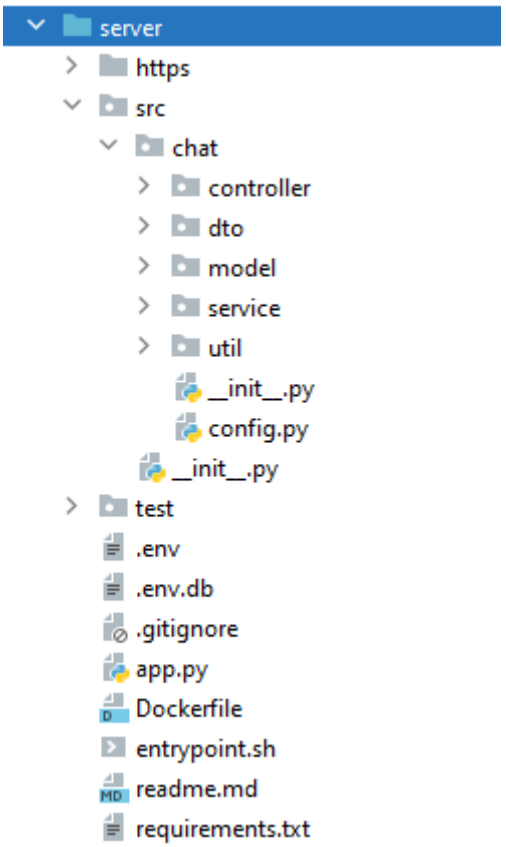


Figure: Structure du «serveur»

- Le dossier «https» contient des certificats ssl.
- Le dossier «test» contient des fichiers de test TDD d'application.
- «README.md» a des notices de fonctions de serveur.
- «Dockerfile» qui permet de créer une image de production de la partie serveur dans un conteneur Docker.
- «entrypoint.sh» contient des commandes qui initialisent l'application lorsqu'elle est construite par Docker.
- «app.py» est un fichier pricinpal qui permet de rentrer de notre serveur
- Dans le dossier «src», on à un application «chat», qui est comme un agent d'application, nous permet de trouver tous des codes de système:
    - Le dossier «controller» qui contrôle toutes les extrémités d'API.
    - Le dossier «dto» qui permet de formater l'entrée et la sortie d'un modèle.
    - Le dossier «model» contient des modèles et leur relations.
    - Le dossier «serveur» a des codes de logique de service comme: enregistrer, modifier, supprimer d'objet dans le base de donnée.
    - Le dossier «util» contient tous les utilitaires nécessaires dont nous pourrions avoir besoin dans notre application.

Solution de déploiement

Déploiement serveur

Afin de déployer notre application, il faut dans un premier temps installer docker sur un serveur, par exemple un serveur amazon puis une fois docker installé il faut installer docker-compose. Sur windows ou mac rendez-vous sur le site : https://www.docker.com/products/docker-desktop et lancez l'installation. Sur une machine linux ouvrez un terminal et saisissez le code suivant :

```
sudo apt-get update && sudo apt-get install docker.io && sudo apt-get install docker-compose
```

Une fois docker installé, il faut installer git si le paquet n'est pas déjà installé et cloner le dépôt depuis gitlab, une fois la commande lancée, git vous demandera de vous authentifier auprès de gitlab.

```
sudo apt-get update && sudo apt-get install git && git clone https://gitlab.utc.fr/gidelthi/tx_chat.git
```

Après avoir cloné le dépôt, se déplacer dans le dossier nouvellement créé et lancer la commande docker-compose pour construire les conteneurs puis le lancer.

```
cd tx_chat && docker-compose build --pull --no-cache && docker-compose up -d
```

Sur le serveur s'assurer que les ports 80 et 443 sont exposés pour pouvoir accéder au serveur web. Par défaut, les serveurs linux n'ont pas de pare-feu.

Https et certificat

Afin de profiter de toutes les fonctionnalités de notre application, cette dernière doit utiliser le protocole ssl (https). Par défaut, nous utilisons des certificats auto-signés par notre autorité racine TxChatRCA. Toutes les clés ont été générées à l'aide du logiciel XCA et disponibles dans les sous-dossier « https » des dossiers server et client.

L'adresse du serveur a été configurée pour être www.tx_chat.com dans les certificats, pour que le certificat auto-signé fonctionne, il faut qu'un serveur DNS redirige l'adresse www.tx_chat.com vers l'adresse du serveur sur lequel l'application tourne.

Pour modifier les certificats, il suffit de remplacer tx_chat-certificate.crt et tx_chat.key dans les deux dossiers https présent dans les dossiers client & server par le nouveau certificat et la nouvelle clé. Pour simplifier les choses, assurez-vous de garder les mêmes noms de fichiers. Une fois le certificat et la clé modifiés, modifiez le fichier nginx.conf dans le dossier client et tout en bas du fichier assurez-vous que les deux directives server_name correspondent au nom du site renseigné dans le nouveau certificat du serveur.

Pour que le cadenas https soit vert et donc que le navigateur fasse confiance au certificat présenté par l'application, il faut que le nom entré dans la barre d'url corresponde au nom DNS renseigné dans le certificat, il faudra donc configurer un serveur DNS en conséquence si besoin et soit avoir un certificat signé par une vraie autorité de certification ou alors ajouter l'autorité de certification présente dans les dossiers https ainsi que le certificat dans la liste de confiance du navigateur utilisé. Sur windows, l'ajout de certificat de confiance se fait au niveau du système d'exploitation.