Maxime DAGNIÈRE

Présentation différée du projet de stage avec la formation Aformac. Du 5 février 2018 au 23 mars 2018 dans l’entreprise WS312

Création d’une messagerie instantanée

WS312 Agrandir les possibles

**Sommaire :**

* Abstract
* Résumé
* Présentation de l’entreprise
* Liste des compétences du référentiel qui sont couvertes par le projet
* Spécifications fonctionnelles
  + Contexte et objectif du projet
  + Contraintes
  + Fonctionnalités
* Spécifications techniques
  + Langages et modules utilisés
  + Description de la base de donnée
* Réalisation
  + Création salon de discussion
  + Mise en place de discussion privé
* Conclusion

**Abstract :**

WS est une entreprise qui crée ses propres logiciels. L’un d’entre eux Wospi est un ERP (Enterprise Resource Planning) modulaire qui permet aux utilisateurs d’ajouter des modules optionnels à leurs sites. Ces derniers sont déployés dans plusieurs configurations.

Actuellement, il manque au logiciel un module supplémentaire qui permettrait au site de proposer un service client intégré pour les utilisateurs finaux.

La création de ce module est mon objectif de stage. Le projet fonctionne en deux parties:

-Pour l’utilisateur l’appel du script permet de générer un environnement t’chat de discussion (pop-up de messagerie permettant d’interagir avec les opérateurs).

-Pour les opérateurs cela permet d’accéder à l’ensemble des interfaces de gestion du module: Dashboard, interface de modération, gestion des profils opérateurs et l’administration des comptes modérateurs pour le ‘super-administrateur’ (validation, suppression des comptes).

Dans le cadre de mon intervention durant le stage, j’ai travaillé dans l’objectif de rendre le module totalement autonome (création et gestion des comptes opérateurs) pour une utilisation future.

Langages et technologies utilisés:

-JavaScript.

-En front, j’ai eu la possibilité de m’appuyer sur les librairies JQuery et Bootstrap.

-Pour le back, mise en place d’un serveur nodeJS auquel j’ai ajouté les Framework express et socket.io pour la liaison entre le back et le front.

**Résumé :**

WS est une entreprise qui crée ses propres logiciels. L’un d’entre eux Wospi est un ERP (Enterprise Resource Planning) modulaire qui permet aux utilisateurs d’ajouter des modules optionnels à leurs sites. Ces derniers sont déployés dans plusieurs configurations.

Actuellement, il manque au logiciel un module supplémentaire qui permettrait au site de proposer un service client intégré pour les utilisateurs finaux.

La création de ce module est mon objectif de stage. Le projet fonctionne en deux parties:

-Pour l’utilisateur l’appel du script permet de générer un environnement t’chat de discussion (pop-up de messagerie permettant d’interagir avec les opérateurs).

-Pour les opérateurs cela permet d’accéder à l’ensemble des interfaces de gestion du module: Dashboard, interface de modération, gestion des profils opérateurs et l’administration des comptes modérateurs pour le ‘super-administrateur’ (validation, suppression des comptes).

Dans le cadre de mon intervention durant le stage, j’ai travaillé dans l’objectif de rendre le module totalement autonome (création et gestion des comptes opérateurs) pour une utilisation future.

Langages et technologies utilisés:

-JavaScript.

-En front, j’ai eu la possibilité de m’appuyer sur les librairies JQuery et Bootstrap.

-Pour le back, mise en place d’un serveur nodeJS auquel j’ai ajouté les Framework express et socket.io pour la liaison entre le back et le front.

**Présentation de l’entreprise :**

L’entreprise WS312 qui a pour slogan « Agrandir les possibles » a été fondé le 01/01/0000 par Baptiste et Carole Moreau. Elle est spécialisée dans le développement web.

**Listes des compétences de référentiel qui sont couvertes par le projet :**

• Mettre en place une base de données

• Développer des pages web en lien avec une base de données

• Maquetter une application

•Développer une interface utilisateur

• Utiliser l’anglais dans son activité professionnelle en informatique

Ayant travaillé sur le modèle de données No SQL et en back-end le langage JavaScript durant le stage, je n’ai pas eu l’occasion de validé les compétences SQL et PHP.

Mais celle-ci ont déjà été acquises durant ma formation à travers de nombreux projets. C’est pour cette raison que mon maître de stage m’a confié ce projet totalement différent de tout ce que j’avais rencontré jusqu’à présent.

Mais si vous me le demandiez lors de mon oral, je pourrais vous décrire mes autres projets qui pourraient valider ces compétences.

Merci de votre compréhension.

**Spécifications fonctionnelles :**

**1. Contexte et objectif du projet :**

WS312 est une entreprise qui crée ses propres logiciels. L’un d’entre eux Wospi est un ERP (Enterprise Resource Planning) modulaire qui permet aux utilisateurs d’ajouter des modules optionnels à leurs sites.

Le but est d’ajouter à l’ERP un module permettant aux utilisateurs finaux de pouvoir contacté le service client du site par un service de messagerie instantanée.

**2. Contraintes :**

• L’application doit être en temps réelle.

• Elle doit pouvoir être totalement indépendante du site.

**3. Fonctionnalités :**

*En commun :*

• Messagerie instantanée

*Utilisateur :*

• Possibilité de « noter » son opérateur.

*Opérateur :*

• Gérer ses nouvelles conversations (lorsque qu’un utilisateur créer une conversation celle-ci est en attente d’opérateur).

• Mettre à jour les informations du client si celui les donnent.

• Ajouter des commentaires sur la conversation.

• Transférer la conversation à un opérateur au choix parmi ceux en ligne.

• Définir son statut (En ligne, break, hors ligne).

• Créer son compte opérateur (Il sera en attente de validation)

• Modifier son profil (email, nom, mot de passe, avatar)

Administrateur :

• Gestion des comptes opérateurs (création, suppression).

• Validation des comptes en attente.

**Spécifications techniques :**

**1. Langages et modules utilisés :**

• NodeJS

• Express

• Socket.io

• EJS

• JavaScript

• HTML

• CSS

• Mongo DB

**2. Description de la base de données :**

La base de données est de type No SQL avec Mongo DB, elle compte quatre tables (user, admin, convers, message).

**Description des tables :**

*Table user :*

Champs : Idclient, name, email, phone et date.

Quand un utilisateur se connecte sur le site, une entrée est faite dans la table.

Son identifiant est enregistré côté client sous forme de cookie pour une récupération de ses informations et anciennes conversations.

*Table admin :*

Champs : Idadmin, email, password, name, rôle (pending, moderateur, administrateur), image, date.

La table admin contient tous les opérateurs et administrateurs du service client.

*Table convers :*

Champs : Idconvers, idclient, idadmin, nameadmin, archive (yes, no, break), date, adresseIp, agent, plateform, nombremsg, appréciation.

La table convers contient toutes les conversations existantes. Une conversation est créée dans deux cas de figure :

• Quand le client se connecte pour la première fois au site le cookie idconvers est créé mais ne crée pas d’entrée dans la base de données.

• Quand il se reconnecte le cookie convers est analysé pour voir si une entrée existe déjà dans la table, si c’est le cas on vérifie son archive. Si l’archive correspond à ‘yes’, un nouvel idconvers est créé.

Une conversation devient archivée quand le dernier message échangé date de plus de trois heures et qu’elle ait eu un opérateur. Elle devient de type break après que le client se soit déconnecté plus de trois minutes du site mais sa conversation sera archivée seulement au bout de trois heures.

Sur la conversation sont aussi enregistrées les informations personnelles du client (adresse IP, plateform et navigateur), ces informations sont enregistrées sur la conversation mais supprimées lorsque celle-ci est archivée. Elles sont présentes juste pour permettre à l’opérateur de trouver le problème plus rapidement si le site à des points faibles sur certaines plateforme ou navigateur.

*Table message :*

Champs : Idconvers, auteur, message, type, date.

La table message contient tous les messages échangés, ils se divisent en quatre types :

• Message : comme son nom l’indique les messages de base ont ce type.

• Commentaire : quand un opérateur ajoute un commentaire à une conversation, il est enregistré comme message mais de type comment.

• Log : le type log est en deux parties log et logP pour différencier les logs qui s’affichent ou non au client.

**Réalisation :**

Toutes les codes qui seront décrit dans ce projet, je les ai réalisés de moi-même, à par trois :

- Pour le changement d’avatar d’un opérateur, la fonction de prévisualisation de l’image vient d’internet, je l’ai juste adapté à mon projet.

- Le regex pour valider les emails viens lui aussi d’internet.

- Le regex permettant de lire les cookies, m’a été donné par mon maître de stage.

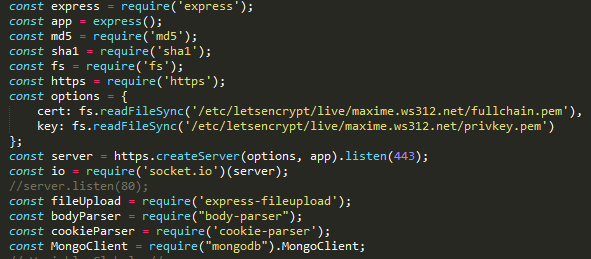
Partie 1 : Début et partie utilisateur

Phase 1 : Mise en place de l’environnement de travail

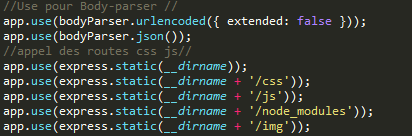
Installation des composants nécessaires sur le pc pour travailler (Mongo DB, No SQL Booster, Sublime Text ainsi que Slack principal moyen de discussion de l’entreprise).

Installation des composants pour le projet (Node JS, Socket.io, Express, EJS).

La partie serveur de NodeJS contient en premier lieu tous les appels aux différents modules :

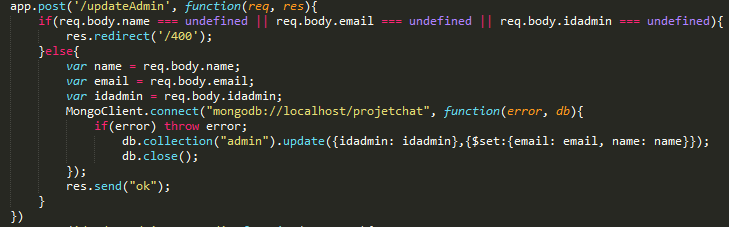


Ensuite viennent les appels use d’Express :



Le routage est fait avec Express :





Description des éléments :

• App pour faire appel à Express.

• Get ou Post pour le paramètre.

• Entre guillemet la route sur cet exemple ‘/’.

• Ensuite on utilise (req, res) en arguments en référence aux éléments du header (response et request).

• Pour récupérer les données qui sont reçues, j’utilise req.body suivi du nom de l’élément reçu.

• Ainsi pour finir on renvoie ce qu’on désire avec res (write, send, end).

Phase 2 : Création de la vue utilisateur :

(Vue de l’utilisateur et maquette consulter les annexes numéros 1 et 2)

Pour faciliter l’intégration du module, la vue utilisateur contient seulement deux éléments :

• Un icon qui servira à l’appel du script.

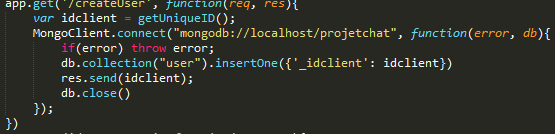
• Une fenêtre modal qui contient le système de messagerie instantanée.

Phase 3 : Fonctionnement du script client:

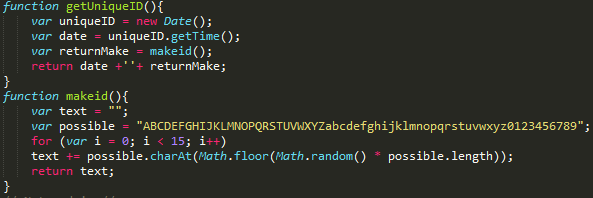
Première partie : création des informations de l’utilisateur

A l’arrivée de l’utilisateur sur le site une première fonction est lancée permettant de savoir si l’utilisateur a un cookie idClient. (Voir annexe)

-S’il n’en possède pas une requête Ajax est instancié permettant de créer un identifiant client et de l’insérer dans la base de données.

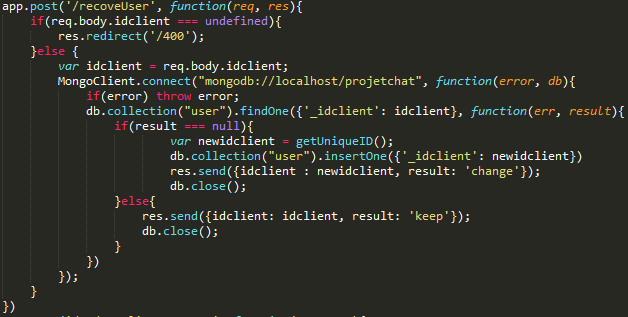


La fonction getUniqueId() permet de créer un identifiant unique pour chaque utilisateur :



Elle se fait en deux partie une fournit le timestamp actuel et l’autre une chaîne de 15 caractères. Les deux additionnés permettent des milliards de combinaison par milliseconde.

-S’il en possède déjà un, on vérifie avec une requête Ajax si l’idClient est déjà présent dans la table. Si c’est le cas, on renvoi ‘keep’ pour dire qu’il peut garder cette identifiant. Dans le cas contraire son identifiant a été supprimé de la base de données et un nouveau lui est attribué. On répond ‘change’ à la requête.



Une fois cette première vérification terminée, on passe à celle de l’idConvers sauf dans le cas où l’utilisateur a été créé suite à une suppression de la table où l’identifiant est créé à la réception du retour d’Ajax.

Pour ce faire la fonction checkConvers est utilisée (Voir annexe). Celle-ci permet de vérifier si le cookie idConvers existe :

-S’il n’existe pas on le créer avec getUniqueId()

-S’il existe on vérifie avec une requête Ajax s’il est présent dans la table et son type d’archive Yes, No



En fonction du retour un nouvelle identifiant est généré et la partie récupération des informations est terminée.

Deuxième partie : Fonctionnement de la messagerie instantanée

Une fois les vérifications terminées, la fonctions goEmit() est instanciée (Voir annexe). Elle a trois partie :

• La première, l’emit de socket.io ‘lastId’ qui émet l’idClient ainsi que l’idConvers.

A la réception de l’emit côté serveur (Voir annexe) :

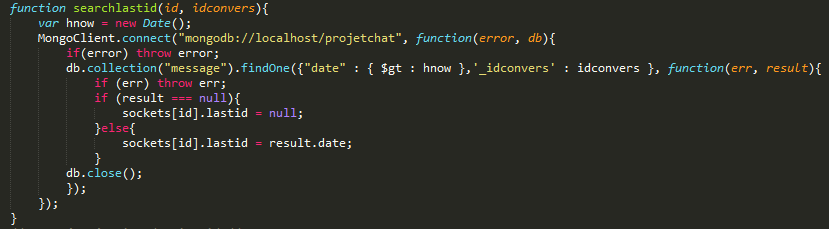
Une première variable globale est renseignée ‘clientOff’, elle est utilisée pour la déconnection d’un utilisateur, si celui quitte le site plus de trois minutes sa conversations seras mis en pause (break).

Une autre variable globale seras renseignée ‘sockets’, permet d’enregistrer toutes les informations du client le temps de sa présence sur le site.

Elle a pour clef l’idClient de l’utilisateur. A chaque fois, qu’un utilisateur charge sa page l’emit est renvoyé, donc si le sockets[id] est déjà défini, on ne le renseigne pas à nouveau.

Seul le socket est renseigné à chaque chargement de page dans sockets[id].session. Car à chaque chargement de page un nouveau socket est généré. Il faut donc enregistrés et supprimés ceux obsolètes.

La fonction searchlastid() est instanciée :



Elle permet de recherchée, les derniers les messages du client. Si ce dernier n’en a pas, on a sockets[id].lastid = null, sinon la variable prend en valeur la date du dernier message.

La fonction addinfos() est lancée (Voir annexe)

Elle permet de recherchée les informations client qui seront renseignée dans la variable sockets[id].

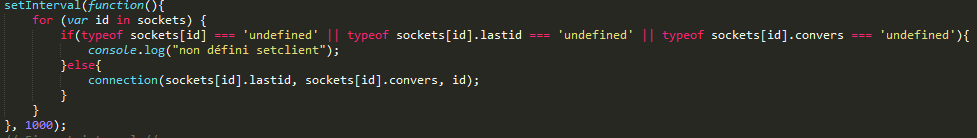
• Le deuxième emit ‘urlclient’ émet le site et la page où se situe le client. Il permet de renseigner le sockets[id].url.

• La dernière action de goEmit() consiste a recherché avec une requête Ajax si la conversation actuel est attribuée à un opérateur. (Voir annexe)

Si la conversation est attribuée, on renvoi, image, le prénom et l’identifiant d’opérateur. Dans le cas contraire on renvoi ‘null’.

En fonction du résultat le header de la fenêtre client renseigne les informations de l’opérateur.

La fonction goEmit() est maintenant terminée. Si des messages sont présent dans la conversation le serveur va transmettre les informations par un emit.



Le setInterval permet de chercher tous les id présent dans sockets et lance la fonction searchMsgClient() pour chaque id trouvé. (Voir annexe)

Dans la fonction searchMsgClient(), en premier lieu on vérifie si la date est null ou défini. En fonction du résultat soit on récupère tous les messages ou seulement ceux ajouter après la date contenue dans sockets[id].lastid. Ce dernier sera à la fin de la fonction renseignée par la date du dernier message. Les messages récupérés sont transmis au client par l’emit ‘push-message’.

Dans un premier temps à la réception de l’emit (Voir annexe) on vérifie que les messages reçus correspondent bien à ceux client. Ensuite on chercher à définir si le type est ’message’ ou ‘logP’. S’ils sont de type ‘message’, on définit l’auteur (client ou opérateur). La fonction recoverDataClient est déclenchée (Voir annexe), elle permet de récupérer les informations machines du client et de les enregistrer momentanément dans la table conversation pour les besoins de l’opérateur.

L’envoi de message (Voir annexe), ce fait quand le client appuie sur la touche Entrer depuis le textarea. Le message ne peut être envoyé seulement si le message fait plus de deux caractères et que le premier soir un chiffre, une lettre ou un symbole. Le contenu ainsi que l’idClient et l’idConvers sont transmis au serveur avec une requête Ajax.

Côté serveur (Voir annexe), on vérifie dans un premier temps si l’idConvers se trouve dans la table convers. Trois cas de figure sont possible :

•Il existe et on enregistre juste le message dans la table.

•Il existe mais l’archive est de type ‘break’, on modifie l’archive du message puis on déclenche la fonction searchAdminConvers() (On reviendra à cette fonction dans la partie opérateur).

• Il n’existe pas, dans ce cas-là, on crée une entrée dans la table convers, puis on enregistre un message log précisent que l’utilisateur a créé la conversation.

Côté client en même temps que l’utilisateur écrit son message, des emit sont transmis au serveur, que le client est en train d’écrire, de même quand l’opérateur écrit. A chaque fois les données sont transmises pour avertir que l’interlocuteur est en train d’écrire. (Voir annexe).

Quand un utilisateur crée une conversation, elle n’a pas d’opérateur d’attribué. Mais au moment où elle est affectée, elle reçoit un emit (Voir annexe) qui lui permet de faire le même résultat que l’Ajax recoveUser() mais sans que l’utilisateur n’est à recharger sa page pour voir la modification.

Dernière fonctionnalité du client la possibilité de noter son opérateur (Voir annexe). En fonction du choix soit on enregistre l’icon négatif ou positif. Par une requête Ajax les données sont transmises au serveur qui mets à jour la table convers.

Pour toutes les requêtes Ajax, en cas de retour d’erreur un message s’affiche dans la fenêtre du client précisant, qu’un problème rend indisponible la fonctionnalité.

Partie 2 : Partie opérateur

L’interface opérateur du module est composé de plusieurs pages :

• Page de connexion / inscription.

• Accueil.

• Interface de modération.

• Gestion du profil.

• Administration.

Page de connexion / inscription :

*Phase 1 : Création de la vue:*

Quand un utilisateur arrive sur le site de modération sans arguments de page il est redirigé vers la page login.html.



La page de login contient deux boutons qui font chacun un appel de script différent inscription et connexion.

En fonction de l’appel de script effectué soit le formulaire d’inscription ou celui de connexion apparaît.

*Phase 2 : Fonctionnement du script :*

Partie 1 : Inscription

Il existe deux moyens pour crée un opérateur, depuis la page login ou depuis la page administration (Voir annexe pour la fonction).

Dans un premier temps côté client, on récupère les informations et on les vérifie (l’email à l’aide d’un regex, que les deux password correspondent et font plus de 6 caractères, ainsi que si le rôle est défini). Une fois les étapes précédentes terminé un Ajax envoi toutes les données au serveur.

Côté serveur (Voir annexe), on collecte toutes les informations reçus. Le cryptage du password se fait en trois étapes :

•La première, on créer salt un identifiant unique crée avec getUniqueId().

•La seconde, on crypte séparément le password, l’email et salt avec sha1.

• La dernière, on additionne les trois chaînes créées précédemment puis on crypte le tout à nouveau avec sha1.

Ensuite on créer un idadmin avec getUniqueId() puis on vérifie si l’email renseigné n’existe pas déjà dans la table. Si ce n’est pas le cas, on enregistre le nouvel utilisateur dans la table.

Partie 2 : Connexion

Côté client (Voir annexe), on vérifie que l’email renseigné est valide, puis on fait appel à l’Ajax pour envoyer toutes les données.

Côté serveur (Voir annexe), on vérifie si l’email existe dans la table. Ensuite s’il le password enregistré correspond à celui renseigné par l’utilisateur, on renvoi à la requête l’identifiant de l’opérateur.

A la réception des données du serveur, un cookie est créé puis on redirige l’opérateur vers l’accueil.

Page Accueil de l’interface :

*Phase 1 : Création de la vue*

Pour toutes les vue opérateur, voir annexe.

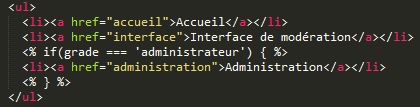
**

La page d’accueil comme toutes les pages de l’interface sont créées avec le générateur de Template EJS. Les variables transmises permettent de :

• Pouvoir afficher le nom de l’opérateur dans la barre de navigation.



• S’il a le grade administrateur l’accès à la partie administration est autorisé.



• Pour la page interface de modération afficher les boutons des conversations en cours de l’opérateur.

La page d’accueil fait office de Dashboard pour l’opérateur, il peut ainsi consulter ses statistiques général (La suite est en cours de création, à terme il pourra consulter ses statistiques par jour, semaine et mois).

Page interface de modération :

*Phase 1 : Création de la vue*

**

L’interface de modération se divise en deux parties :

Vue gauche ou l’on retrouve les éléments suivants :

• Select pour le choix du statut (En ligne, break, hors ligne), par défaut l’opérateur est hors ligne.

• T’chat en attente (Conversation avec aucun opérateur attribué).

• Opérateur (Liste de tous les opérateurs en ligne).

Vue droite qui là encore se divise en deux parties :

Partie gauche ou conversation client :

• Fenêtre contenant tous les messages de la discussion en cours.

• Textarea pour permettre à l’opérateur de répondre au client.

• En bas se trouvent tous les boutons des conversations en cours de l’opérateur.

Partie droite ou action sur la conversation de l’opérateur :

• Un select qui permet de choisir l’action, par défaut ça affiche les informations client.

• Le transfert de conversation permet comme son nom l’indique de transférer l’information à un autre opérateur après avoir choisi la raison (Au choix dans un select) et cliqué sur l’opérateur voulu dans la liste.

• La vue information client permet de :

- Renseigner des informations sur l’utilisateur (Prénom, email, téléphone).

- Ajouter des commentaires.

- Voir le chemin du visiteur à chaque fois qu’il change de page. L’opérateur peut ainsi surveiller s’il va sur les bonnes pages pour résoudre son problème.

- Afficher les statistiques du client (date de la première conversation, nombre de conversations et durée de la conversation actuelle).

- En cliquant sur « nombre de conversations », ça affiche les informations de ces conversations (Opérateur, note, date, nombre de message). En cliquant sur une de ces conversations ça affiche dans une fenêtre modal l’historique des messages de chaque conversations.

- Affichage des informations « machine » du client (adresse IP, plateforme et navigateur).

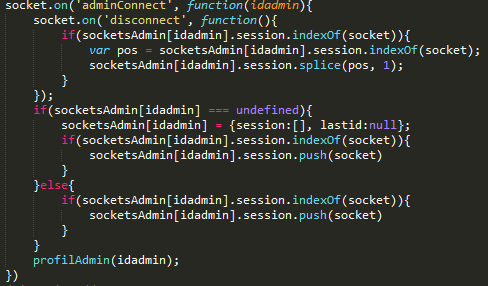
*Phase 2 : Fonctionnement du script*

Partie 1 : Envoi et réceptions des premiers emit

Au chargement de la page l’idadmin contenu dans les cookies est emmagasiné dans la variable ‘name’.

Un première ‘emit’ est lancé ‘adminConnect’ avec l’id de l’admin.

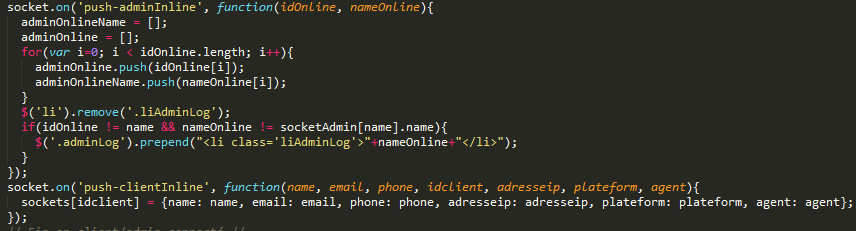


Côté serveur comme pour les clients une variable globale est renseigné socketsAdmin et l’id de l’admin en est la clef.

La fonction profilAdmin() est déclenchée (Voir annexe pour la fonction). Elle permet de récupérer toutes les informations concernant l’opérateur côté client avec emit ‘push-profilAdmin’.

A la réception de ces informations, elles sont enregistrées dans une variable ‘socketsAdmin’.

Un setInterval du serveur cherche toutes les deux secondes les opérateurs et les clients en ligne sur le site (Voir annexe pour la fonction). Il envoi deux emit (push-adminInline et push-clientInline).

Côté client

Push-adminInline permet de remplir deux tableaux l’un avec les prénoms et l’autre des id des opérateurs en ligne. L’autre partie écris sur la page la liste des opérateurs en ligne.

Push-clientInline permet de renseigner la variable sockets[idclient] avec le même principe que celle présente sur le serveur.

Un autre setInterval permet de rechercher les conversations qui ne sont pas attribué (Voir annexe pour la fonction), dans l’emit de cette fonction soit elle contient les nouvelles conversations soit ‘empty.

Côté client (Voir annexe), si la réception contient rien, on supprime tous les boutons existant. Dans le cas contraire on écrit dans la partie ‘chats en attente’, une liste de toutes les conversations non attribuées.

Si un opérateur clique sur une de ces conversations, un emit est envoyé qui exécutera la fonction updateNewClient() (Voir annexe).

Elle effectue quatre actions :

• La première, elle met à jour la conversation en ajoutant dans le champs idadmin, l’id de l’opérateur qui viens de s’attaché la conversation.

• La seconde, elle envoie un emit côté client qui seras décrit dans la partie suivante.

• La troisième, elle enregistre un nouveau message de type logP, précisant que tel opérateur a pris la conversation.

• La dernière, envoie l’emit qui actualise la fenêtre du client en y ajoutant les informations de l’opérateur.

Partie 2 : Création des conversations

Pour afficher la partie conversation de l’opérateur, on doit appeler la fonction createConvers(), on peut l’exécuter de quatre façon :

• La première au chargement de la page, les boutons des conversations sont créés directement depuis les informations transmis avec EJS. Chacun d’eux peut faire appel à la fonction createConvers()

• La seconde, quand on clique sur une des conversations en attente un appel à la fonction se fait.

• La troisième, depuis un bouton conversation crée à la suite d’une attribution de conversation.

• La dernière, depuis un bouton conversation généré après l’emit de la fonction searchAdminConvers() (Voir annexe pour la fonction).

Cette fonction permet quand elle est appelée de rechercher toutes les conversations en cours de l’opérateur et de les envoyer avec un emit.

Sur les boutons conversation, apparait un icon à la réception d’un emit prévenant que le client est actuellement en train d’écrire.

Quand un opérateur reçoit un nouveau message du client, le bouton change de couleur et un ‘badge’ indiquant le nombre de message reçus est visible.

La fonction createConvers(Voir annexe pour la fonction) est lancée avec trois arguments (idClient, idAdmin, idConvers). Elle regroupe plusieurs appel Ajax qui s’effectue sur les éléments crées de manière asynchrone.

Au début de la fonction, on ‘nettoie’ la page :

- Suppression de la vue conversation précédente.

- On supprime les éléments qui indiquait un nouveau message du bouton.

- On ajoute au bouton une classe permettant de savoir sur lequel on se trouve.

Ensuite deux blocs sont générés :

- Le premier le bloc conversation contient, la vue de discussion (les messages précèdent et un textarea pour les envoyer.

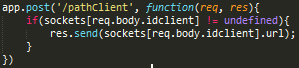
- Le deuxième bloc, va contenir toutes les actions de l’opérateur sur ce client (Information du client et transfert de conversation). Pour générer ce bloc les informations de l’utilisateur sont requise. Elles sont appelées depuis la variable ‘sockets’ créée précédemment.

Une fois les deux blocs construit le début du travail sur les éléments asynchrones commencent. Dans un premier temps quatre requêtes Ajax se lance (Voir annexe) :

• La première permet d’envoyer une requête au serveur pour recevoir tous les messages de la conversation actuel de l’opérateur. De plus elle modifie dans la table, s’ils ont été lu par l’opérateur.

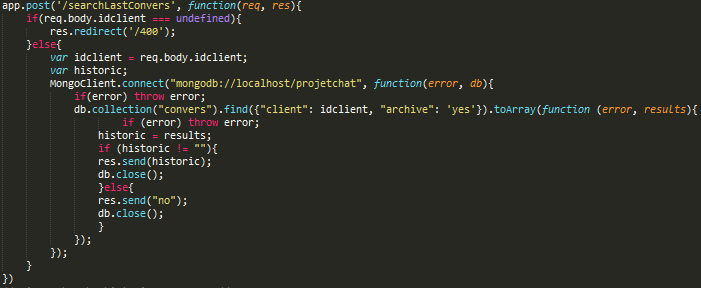


• La seconde, permet de récupérer depuis le serveur les pages sur lequel le client se situe. (Pour le moment en Ajax, mais à terme l’actualisation se feras avec les sockets)



• La troisième (Voir annexe pour la fonction), recherche dans la base les informations conversation (Date de la première conversation, nombre de conversation, durée de la conversation actuel).

• La quatrième, va chercher dans la table convers toutes les conversations qui correspondent au client. Les informations collectées vont permettre de réaliser un tableau de toutes les anciennes conversations. Ce tableau est caché, on peut le faire apparaître ou disparaître en en appuyant sur le ‘nombre de conversation’ de l’Ajax précédent.





Si un opérateur veut voir l’historique des messages pour chaque conversation, un onclick est présent sur le prénom de l’agent.

En appuyant sur l’appel, on appel la fonction tchathistory(), qui fait une requête Ajax.

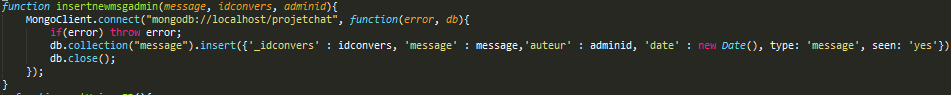


Côté serveur, à la réception des données, on recherche toutes les conversations qui sont archivé, puis une boucle se fait sur la table convers pour trouver tous les messages des différentes conversations du client. Une fois les messages récupéré un emit les envoie côté client.

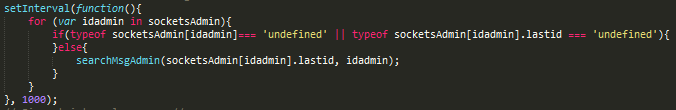
A la réception de l’emit ‘push-history’ (Voir annexe pour le code) on écrit dans la fenêtre générée au retour d’Ajax les informations reçues.

L’envoi des messages de l’opérateur se fait comme pour l’utilisateur en appuyant sur la touche Entrer une fois que son message fait plus de deux caractères (Voir annexe pour la fonction).

Côté serveur, à la différence de l’enregistrement client, la conversation existant déjà on enregistre directement le message exactement pareil sauf pour le champs ‘seen’ qui est ‘yes’ pour celui de l’opérateur.



La réception des nouveaux messages fonctionne elle aussi sur le même principe que ceux du client.



Le setInterval recherche tous les idadmin contenu dans la variable socketsAdmin et lance la fonction searchMsgAdmin() pour chacun des identifiants trouvés.

La fonction commence en vérifiant si lastid est null ou non. Par rapport au résultat la fonction cherche tous les messages des conversations ou seulement ceux reçu après le lastId. Le socketsAdmin[idadmin].lastid prend en valeur a date du dernier message reçu. Ensuite les messages sont envoyés au client avec l’emit ‘push-adminMessage’.

Côté client, à la réception de l’emit deux événements se font :

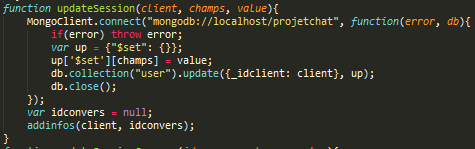
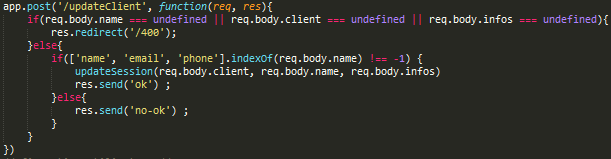
• Le premier, est l’emit cité plus haut qui permet de signaler qu’une conversation a un nouveau message.

• Le deuxième, affiche les nouveaux messages reçus dans la conversation en cours de l’opérateur en fonction du type de message ainsi que s’il est de lui ou du client.

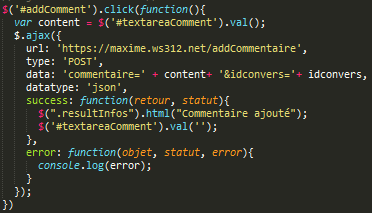
Partie 3 : Information client et commentaire

Maintenant que l’opérateur à une vue globale des informations sauf des informations (Prénom, email, téléphone) si l’utilisateur ne les a pas renseignés ou si c’est sa première visite. Pour cela l’opérateur peut le faire en renseignant les champs et en les appuyant sur Entrer pour lancer la fonction (Voir annexe).

Côté serveur, on vérifie si les informations sont juste puis on lance la fonction permettant de les enregistrées.



Cette dernière permet deux choses d’enregistrer les nouvelles information du client ainsi que de lancer la fonction addinfos() qui permet de mettre à jour les sockets[idclient].

L’opérateur peut aussi ajouter des commentaires à la conversation. Une fois le message écrit, l’opérateur appuie sur le bouton ajouter pour déclenche l’événement pour ajouter un commentaire : 

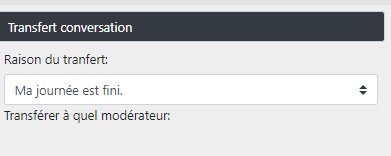


Côté serveur, le commentaire est enregistré dans les messages avec le type ‘comment’ pour être vu seulement de l’opérateur dans la conversation.

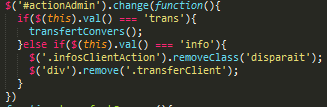
Partie 4 : Transfert de conversation

Si un opérateur pour diverse raison veut transférer sa conversation, il peut le faire parmi les actions du select côté droit.

En interagissant avec le select et choisissant l’option transfert de conversation, l’opérateur voit une nouvelle vue apparaître à la place des informations client.



Cette fonction permettant de changer de vue.



L’opérateur a un select pour choisir la raison du transfert de conversation, pour transférer la conversation il devra cliquer sur le prénom d’un des autres opérateurs en ligne.

La fonction lancée, une requête Ajax est déclenchée, elle transmet les données au serveur et en cas de réponse positive, ‘nettoie’ la page comme lors d’un changement de conversation.

Côté serveur (Voir annexe pour la fonction), on réalise plusieurs actions :

• La première, on modifie le champs ‘idadmin’ de la conversation pour celui du nouveau.

• La seconde, on ajoute dans le champs ‘lastAdmin’ de la conversation l’identifiant du précédent opérateur.

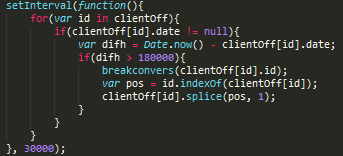
• La troisième, on enregistre un message de type ‘log’ la raison du transfert de conversation.

• La quatrième, on enregistre un message de type ‘logP’ comme quoi la conversation vient d’être transféré pour le client.

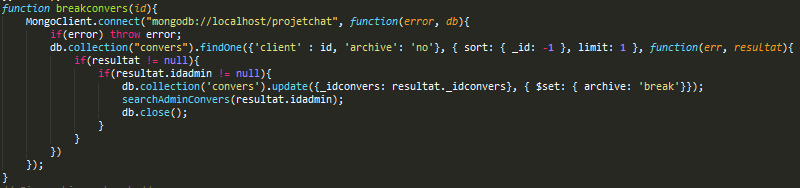
• La dernière, on lance la fonction searchAdminConvers() pour que les deux opérateurs et leurs boutons de conversation mis à jour.

Partie 5 : Fonction du serveur

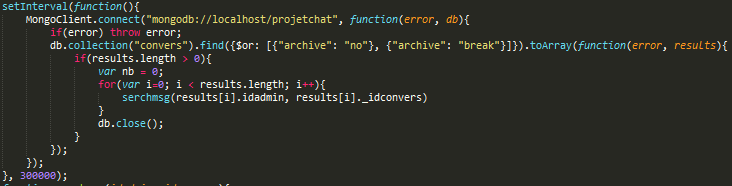
Sur le serveur deux setInterval tourne en permanence pour tenir à jour le statut des conversations :



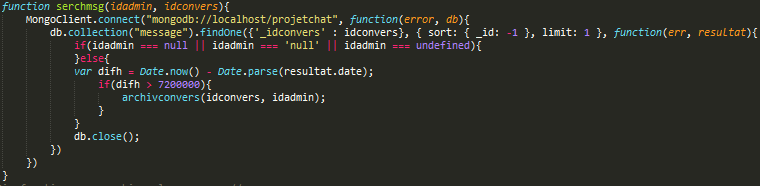
• Le premier, permet de rechercher parmi la liste des clients qui se sont déconnecté depuis plus de trois minutes. S’il en trouve, il déclenche la fonction breakconvers().



Pour chaque identifiant reçu, on recherche s’il avait une conversation en cours. En cas de résultat positif, on met à jour le champs ‘archive’ de la table convers en ‘break’.



• Le deuxième, permet de rechercher parmi toutes les conversations celle dont le champ archive est ‘break’ ou ‘no’. Pour chaque résultat elle déclenchera la fonction searchmsg()



Pour chaque conversation, on vérifie deux chose si un opérateur est attribué et que le dernier message remonte à plus de trois heures. Si les deux conditions sont réunies l’idconvers et l’idadmin sont transmis à la fonction archivconvers().

Cette dernière (Voir annexe) effectue plusieurs actions :

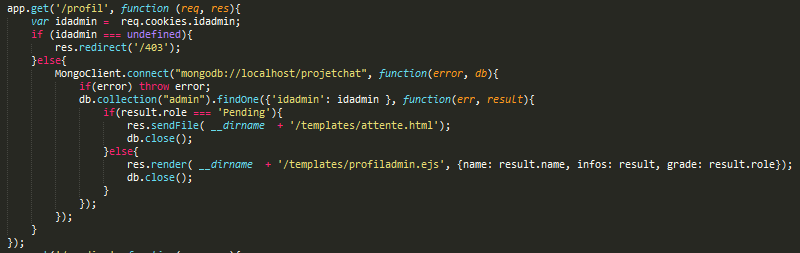
- La première, elle récupère le prénom de l’opérateur.

- La seconde, elle compte le nombre de message de type ‘message’ que compte la conversation.

- La dernière, elle met à jour la conversation en ajoutant le nombre de message et le prénom de l’opérateur. Elle change l’archive en ’yes’ et supprime les informations machine du client.

Page gestion du profil :

*Phase 1 : Création de la vue*

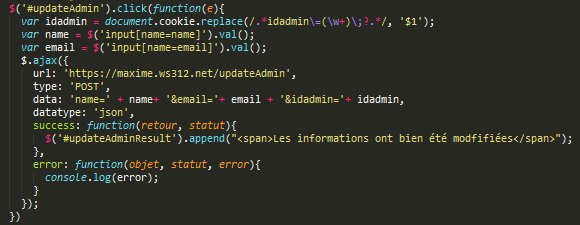
**

La page profil, permet à l’opérateur de modifier les informations de son compte. La modification du profil se fait en trois partie en vue accordéon, l’email et le prénom sont renseigné avec les variable transmis avec EJS.

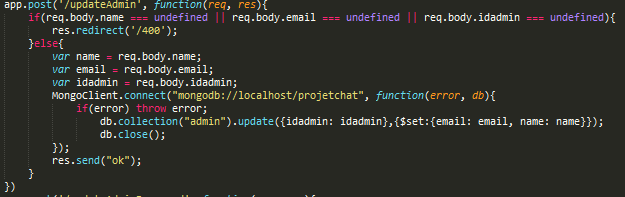
*Phase 2 : Fonctionnement du script*

Première accordéons, email et prénom.

Côté client, on transmet simplement les informations avec une requête Ajax.

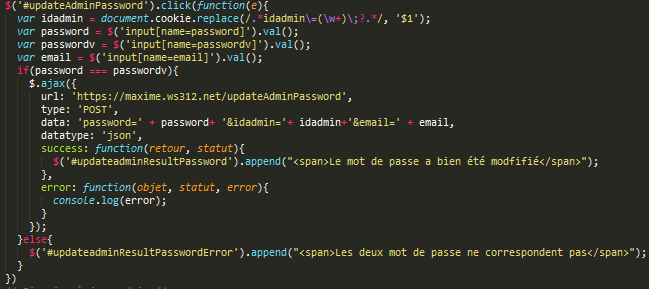


Côté serveur, à la réception des informations on met à jour les informations dans la table ‘admin’.



Deuxième accordéons le mot de passe.

Côté client, on vérifie si les deux password correspondent avant de les envoyer au serveur avec une requête Ajax.



Côté serveur, à la réception des informations on crypte le mot de passe puis on met à jour la table ‘admin’.

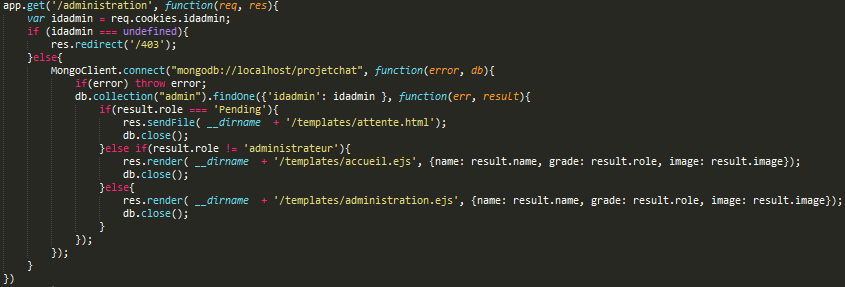
Troisième et dernier accordéons l’avatar.

Côté client, l’opérateur choisi une image de type JPG ou PNG pour son nouvel avatar. Il peut avoir un aperçu avant d’envoyer l’image avec une fonction JavaScript mais ne l’ayant pas créé juste réutilisé.

Côté serveur (Voir annexe), on cherche dans un premier temps le format de l’image s’il n’est pas JPG ou PNG on renvoi une erreur au client. Dans le cas où l’image est valide on l’enregistre dans le dossier avec comme nom de fichier l’identifiant de l’opérateur.

Page administration :

*Phase 1 : Création de la vue*

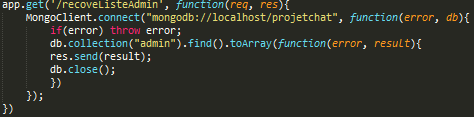
**

La page administration permet au administrateurs de gérer les opérateurs (Création, suppression de compte et modifier le rôle).

*Phase 2 : Fonctionnement du script*

Au chargement de la base un Ajax de type ‘GET’ se lance (Voir annexe) pour récupérer la liste de tous les opérateurs. A leurs réceptions, ils sont mis dans un tableau.

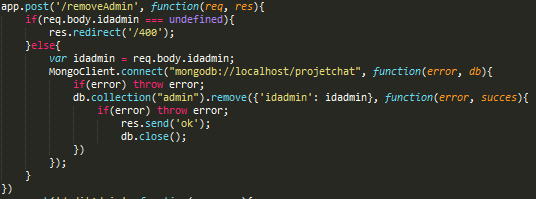
Côté serveur, à la réception de la requête Ajax, on recherche dans la table ‘admin’ la liste de tous les opérateurs avant de les renvoyer.



La création de compte de la partie administrateur est la même que celle de la page login à la différence qu’il peut directement assigner un rôle.

La suppression du compte se fait depuis un icon poubelle sur chaque ligne du tableau. Avant de faire la requête Ajax pour supprimer le compte, une alerte JavaScript se manifeste pour confirmer la décision (Voir annexe pour la fonction).

Côté serveur, à la réception des données on supprime l’entrée qui correspond à l’idadmin qui doit être supprimé.



La mise à jour du rôle de l’opérateur se fait depuis le bouton modifier de la ligne du tableau. En appuyant dessus un select apparaît permettant de choisir le nouveau rôle (Voir annexe pour la fonction).

Côté serveur, à la réception des données on met à jour l’entrée correspondante.

