

SAVECORP



SAVECORP

SAVECORP

Projet Transversal

Administration Infra système et réseaux

Hugo FERRY, Ian LOGEAS, Xavier MERLE, Thomas LEBAS

Table des matières

Introduction :	2
Organisation	3
Infrastructure	4
Le serveur de mail	4
Le serveur web	6
Le serveur de base de données	7
Sauvegarde client :	Erreur ! Signet non défini.
Clé SSH	Erreur ! Signet non défini.
Rsync	Erreur ! Signet non défini.
Scripting Bash, notifications et interface web	8
Attribution fichier à un utilisateur	9
Script stat	10
Script Archive	11
Script de délai de la sauvegarde	11
Notifications	12
Lexique	Erreur ! Signet non défini.
Piste d'amélioration et futures améliorations	13
Conclusion	14
Annexe	14



Introduction :

Pour ce projet, toute l'architecture réseaux se trouve dans les locaux de la Savecorp, ainsi tout fonctionne par **WAN** et vous n'avez donc besoin d'aucun matériel si ce n'est d'un ordinateur et d'une connexion internet afin d'accéder à votre serveur de sauvegarde.

Ainsi, nous avons commencé à élaborer une infrastructure réseau et matérielle.

Afin d'accéder à notre serveur de sauvegarde en toute sécurité et à distance, nous avons mis en place un **firewall** Pfsense, qui permettra d'instaurer les règles de sécurité et de sécuriser l'accès aux machines.

De plus, nous vous proposons une inter-compatibilité entre Windows et linux, en effet cela marchera pour les deux systèmes d'exploitation. Cette mesure nous semblait importante car dans une entreprise il n'y a jamais que du Windows ou que du linux.

Afin de répondre aux besoins mis en place, voici un petit récapitulatif des machines utilisées pour ce projet :

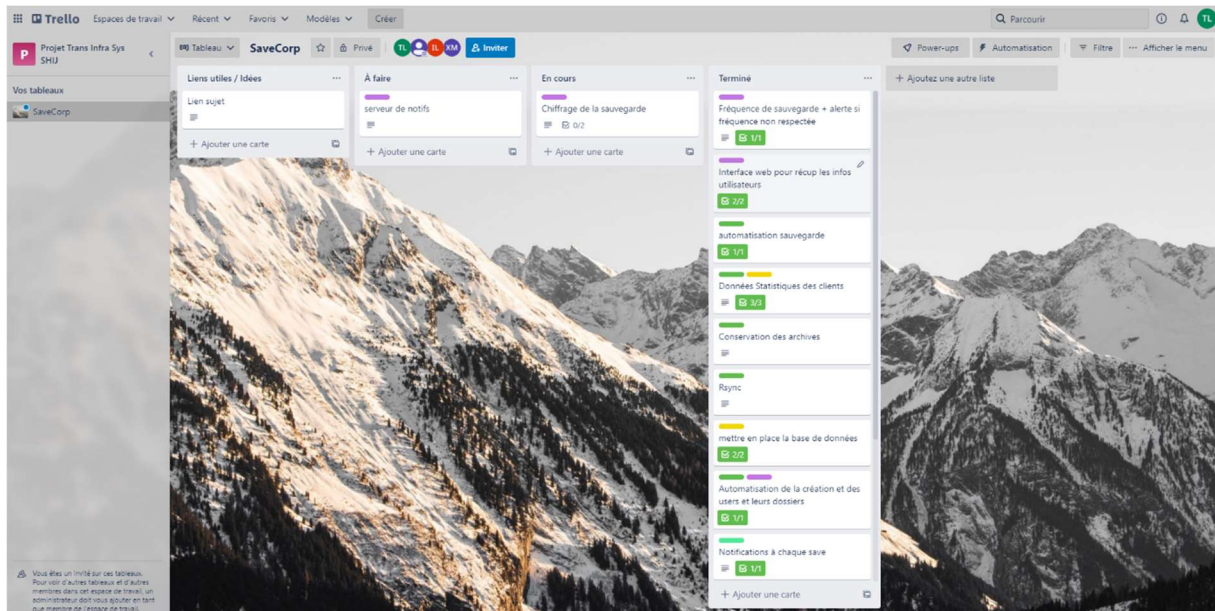
- 1 firewall Pfsense
- 1 serveur de sauvegarde utilisant Ubuntu 20.04 LTS
- 1 machine cliente utilisant Ubuntu 21.10 édition desktop
- 1 machine cliente Windows 10

Pour avancer pas à pas dans notre architecture, nous commencerons par vous expliquer la mise en place de notre organisation. Ensuite, nous verrons les différents composants de notre architecture et enfin nous verrons les scripts ainsi que leur automatisation.



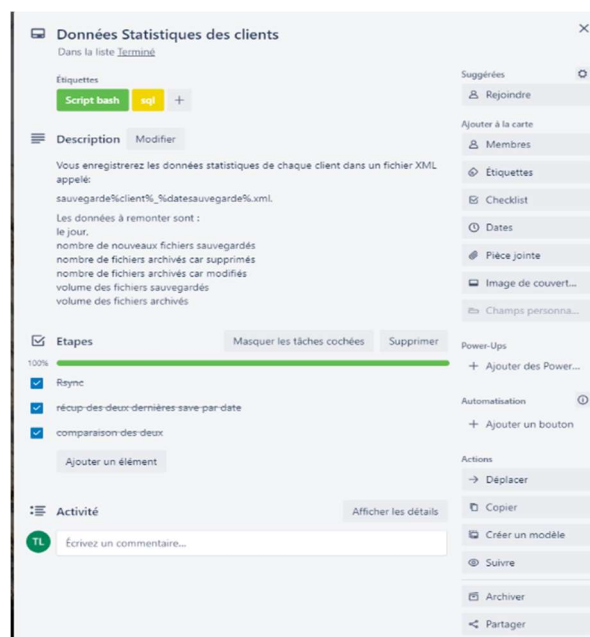
Organisation

Afin de bien s'organiser et répartir les différentes tâches nous avons dû utiliser différents outils. Tout d'abord nous avons mis en place un trello disponible à tous les membres du groupe.



Dans ce trello, nous retrouvons plusieurs informations. Tout d'abord nous avons quatre colonnes, une pour les liens importants, une autre pour les tâches à faire, une pour les tâches en cours et enfin une pour les tâches terminées.

Chaque tâche a été définie par des étiquettes afin de savoir rapidement dans quel domaine elle s'inscrivait, comme « sql » ou bien encore « php ». Enfin, on retrouve certaines tâches avec une checklist car certaines tâches étaient complexes et nécessités des étapes pour voir l'avancé de ces tâches. Pour les moyens de communication nous avons principalement utiliser Discord afin de stocker certains fichiers sur notre serveur et faire nos réunions de travail.

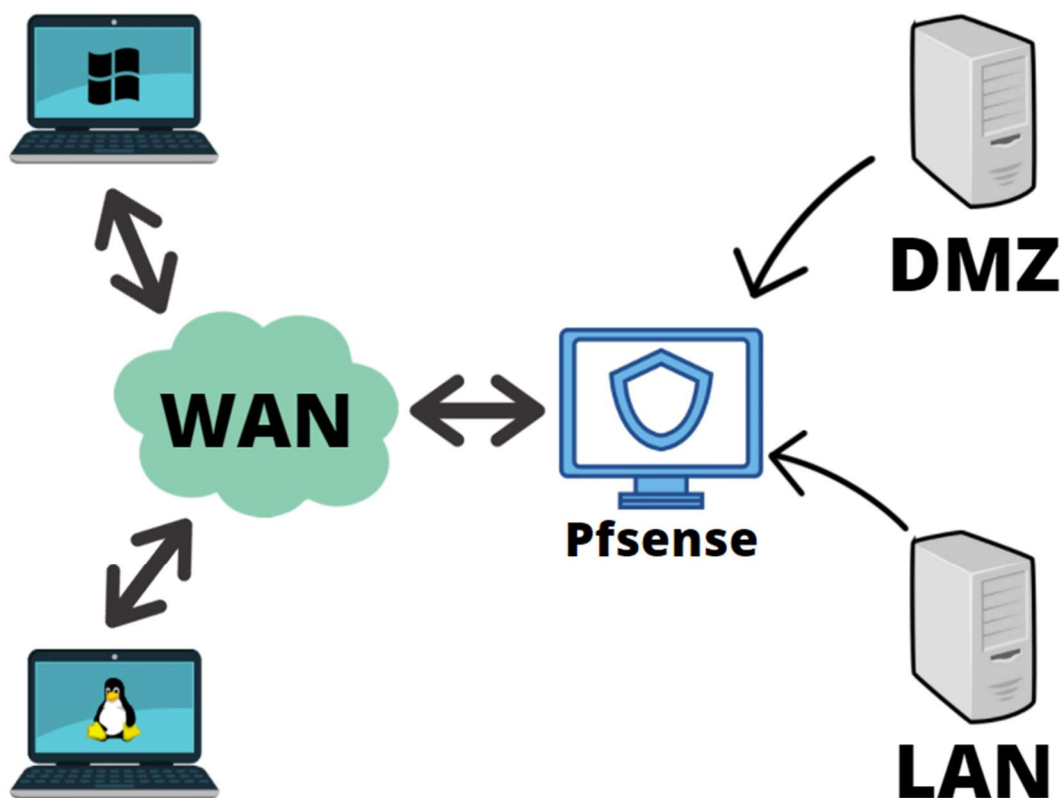


Infrastructure

Notre infrastructure réseau et matérielle s'articule autour d'un **pare-feu** pfsense (c.f partie spécifique), mais aussi composé d'un serveur principal qui va accueillir :

- Un serveur **smtp postfix**
- Un serveur **web apache2**
- Un serveur de **base de données mysql**
- **Une tâche planifiée** exécutant les scripts nécessaires

Tous ces serveurs ont leur utilité propre mais elles se retrouvent toutes employées dans nos scripts. Voici leurs utilisations plus en détails :



Le serveur de mail

Le serveur postfix est paramétré pour envoyer des emails aux clients en cas de problèmes de sauvegarde, par exemple si le délai de sauvegarde n'a pas été respecté. Nous enverrons aussi par mail les rapports de leurs sauvegardes, comme des données statistiques tels que le poids ou encore les dossiers qui ont bien été sauvegardé par exemple. Nous avons aussi prévu de pouvoir recevoir des mails au cas où une personne voudrait nous joindre soit pour modifier ses infos utilisateurs soit pour nous faire une réclamation et poser tout type de questions.



Le serveur web

Le serveur web apache2 lui possède plusieurs fonctionnalités, il nous permet notamment de mettre à disposition une interface web que nous avons créé, celle-ci permet aux utilisateurs de rentrer leurs informations qui nous seront nécessaire de manière simple, rapide, ergonomique, et efficace.

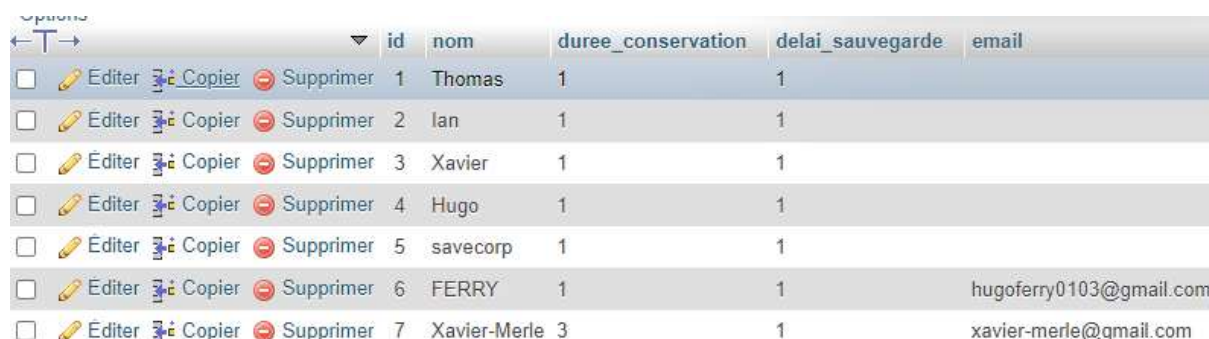
Le serveur apache nous permet aussi d'accéder à l'interface web de notre base de données mysql grâce à l'interface phpmyadmin. Cela nous a offert une meilleure visibilité sur notre base de données tout au long de notre projet.



Le serveur de base de données

Le serveur mysql dont nous venons de vous parler contient les différentes informations des utilisateurs comme le nom, le délai des sauvegardes, la durée de conservation de leurs fichiers ou bien encore leurs adresses mails

Voici un aperçu de notre base de données via phpmyadmin :



				id	nom	duree_conservation	delai_sauvegarde	email
<input type="checkbox"/>	Éditer	Copier	Supprimer	1	Thomas	1	1	
<input type="checkbox"/>	Éditer	Copier	Supprimer	2	Ian	1	1	
<input type="checkbox"/>	Éditer	Copier	Supprimer	3	Xavier	1	1	
<input type="checkbox"/>	Éditer	Copier	Supprimer	4	Hugo	1	1	
<input type="checkbox"/>	Éditer	Copier	Supprimer	5	savecorp	1	1	
<input type="checkbox"/>	Éditer	Copier	Supprimer	6	FERRY	1	1	hugoferry0103@gmail.com
<input type="checkbox"/>	Éditer	Copier	Supprimer	7	Xavier-Merle	3	1	xavier-merle@gmail.com

Nous ferons régulièrement appel à cette bdd pour rendre nos scripts dynamiques et adaptable tout en adoptant une plus grande scalabilité que si nous avions stocké ces informations par d'autres moyens



Nous avons aussi employé l'utilitaire **cron** afin d'exécuter des scripts automatiquement à intervalles réguliers afin d'automatiser des tâches comme la création d'utilisateurs.

Nous allons vous parler plus en détail de ces tâches dans la partie à venir concernant les scripts.



Scripting Bash, notifications et interface web

Dans cette partie, nous allons voir comment cela se déroule précisément lorsqu'un client veut adhérer à nos services.

Nous avons donc décidé d'automatiser un maximum nos services afin que le client n'ait pas à attendre à chaque fois notre retour, comme la création d'utilisateur avec son dossier personnel par exemple.

On commence donc d'abord avec ce que va voir en premier le client, notre formulaire d'inscription. Nous avons mis en place ce formulaire, disponible sur une interface web en php, afin que le client puisse nous donner les informations nécessaires pour nos services.



Information Client

Nom :

Délai de sauvegarde :

Durée de conservation :

Email :

Envoyer

Il suffit donc que le client rentre les informations nécessaires, afin qu'on puisse les stocker et les récupérer via notre base de données.

Les données qu'il rentre sera envoyé à son mail afin qu'il puisse bien vérifier que ses informations soient bonnes. Si elles ne sont pas bonnes, il faudra cependant envoyer un mail à notre équipe pour que nous changions les informations qui sont dans la base de données.

Pour donner suite à cela, l'utilisateur ainsi que son dossier de sauvegarde personnel sera créé via un script (qui s'exécute automatiquement) dans notre serveur de sauvegarde.



Attribution fichier à un utilisateur

Chaque dossier de sauvegarde est propre à chacun et n'est accessible que par l'utilisateur ou notre équipe technique.

```
useradd.sh x
useradd.sh
1  #!/bin/bash
2
3  querymax=`mysql -u savecorp -psavecorp -e "USE savecorp; SELECT MAX(id) FROM infos_user"`
4  sql_max=$(echo $querymax | cut -d " " -f 2)
5
6
7  for ((i=1; i <= $sql_max; i++))
8  do
9
10     query=`mysql -u savecorp -psavecorp -e "USE savecorp; SELECT nom FROM infos_user WHERE id = "$i""`
11     result=$(echo $query | cut -d " " -f 2)
12
13     if [ ! -d "/home/$result/" ]
14     then
15         echo "L'utilisateur ${result} a bien été créée, vous pouvez le retrouver dans /home/${result} !"
16         useradd -m $result
17         mkdir /home/$result/saves
18         chmod 402 /home/$result
19     else
20         echo "L'utilisateur ${result} est déjà existant, vous pouvez le retrouver dans /home/${result} !"
21     fi
22     echo $i
23 done
```

Voici tous les dossiers de sauvegarde personnels.

```
savecorp@sauvegarde:~$ ls /home
FERRY Hugo Ian savecorp Thomas Xavier Xavier-Merle
```

Par exemple, l'utilisateur savecorp ne peut pas accéder au dossier du client Thomas.

```
savecorp@sauvegarde:~$ cd /home
savecorp@sauvegarde:/home$ cd Thomas/
-bash: cd: Thomas/: Permission denied
savecorp@sauvegarde:/home$
```



Script stat

Nous avons ensuite un script qui permet de donner des données statistiques entre la nouvelle et la dernière sauvegarde que le client à fait. Nous allons exécuter automatiquement le script via crontab. Le client reçoit ainsi les statistiques par mail, il peut donc voir quels sont les fichiers modifiés, créés, supprimés ainsi que les volumes des fichiers et la date de la sauvegarde. Évidemment, chaque fichier de statistiques ont leur date dans le titre du fichier, le client peut donc rapidement savoir où trouver ses informations.

```
1 #!/bin/bash
2
3 date=$(date -r $2 "+%d-%m-%Y-")
4 fichier=$(date)listing.txt
5 function check
6 {
7     ls "$1" | while read element
8     do
9         if [ -f "$1/$element" ]
10         then
11             if [ ! -f "$DefaultRep${1:nb}/$element" ]
12             then
13                 echo "DELETED / $DefaultRep${1:nb}/$element" >> $fichier
14                 elif [ ! "$(md5sum "$1/$element" | cut -d " " -f 1)" = "$(md5sum "$DefaultRep${1:nb}/$element" | cut -d " " -f 1)" ]
15                 then
16                     echo "CHANGED / $DefaultRep${1:nb}/$element" >> $fichier
17                 else
18                     echo "UNCHANGED / $DefaultRep${1:nb}/$element" >> $fichier
19                 fi
20             elif [ -d "$1/$element" ]
21             then
22                 check "$1/$element"
23             fi
24         done
25     }
26
27 function checkCreate
28 {
29     ls "$1" | while read element
30     do
31         if [ -f "$1/$element" ]
32         then
33             if [ ! -f "$2/$element" ]
34             then
35                 echo "CREATED / $1/$element" >> $fichier
36             fi
37             elif [ -d "$1/$element" ]
38             then
39                 checkCreate "$1/$element" "$2/$element"
40             fi
41         done
42     }
43
44 creationdate=$(date -r $1)
45
46 sizesave=$(du -bch $2 | tail -1 | cut -f1)
47 sizeDelete=$(du -bch $1 | tail -1 | cut -f1)
48
49 fichiersuppr=$(cat $fichier | grep 'DELETED' $fichier | wc -l)
50 fichiersauvegarde=$(cat $fichier | grep 'CREATED' $fichier | wc -l)
51 fichiermodifs=$(cat $fichier | grep 'CHANGED' $fichier | wc -l)
52 fichierinchange=$(cat $fichier | grep 'UNCHANGED' $fichier | wc -l)
53
54 echo -e "Liste des fichiers parcourus : \n\nl y a $fichiersuppr fichiers supprimés\nl y a $fichiersauvegarde fichiers sauvegardé\nl y a $fichiermodif fichiers modifiés\nl y a $fichierinchange fichiers non modifiés\n" > $fichier
55
56 echo -e "La taille totale des fichiers sauvegardé est $sizesave" >> $fichier
57 echo -e "La taille totale des fichiers supprimé est $sizeDelete\n" >> $fichier
58 echo -e "La date de création du dossier est $creationdate\n" >> $fichier
59
60 DefaultRep=$1
61 nb=${#1}
62
63 check $1
64 checkCreate $2 $1
```



Script Archive

Après qu'on ait sauvegardé, nous possédons aussi un autre script qui permet de supprimer les archives de sauvegarde qui dépasse le délai que le client nous a donné via le formulaire. Ainsi, on évite la surcharge de stockage inutile et donne une visibilité plus claire du dossier de sauvegarde du client.

```
Archive.sh X
Archive.sh
1  #!/bin/bash
2
3  #nbUser=`cat /etc/passwd | grep /home | grep 100 | wc -L`
4
5  listeclient=$(ls /home)
6
7  for client in $listeclient
8  do
9
10
11     tempsarchive=`mysql -u savecorp -psavecorp -e "USE savecorp; SELECT duree_conservation FROM infos_user WHERE nom = '$client' " `
12
13     temps=$(echo $tempsarchive | cut -d " " -f 2)
14     echo $client
15     echo $temps
16
17     find /home/$client/saves -mtime +$temps -exec rm -rf {} \;
18
19     #find /path/to/files* -mtime +5 -exec rm {} \;
20
21 done
```

Script de délai de la sauvegarde

Enfin, le dernier script présent dans notre travail sert à savoir s'il y a un client dont sa sauvegarde n'a pas été faite dans le délai qu'il a donné. Il tourne donc en continu sur notre serveur afin qu'on puisse avoir un mail dès qu'il y a un délai non respecté.

```
délai_save.sh X
délai_save.sh
1  #!/bin/bash
2
3  date_ajd=$(date +%s)
4
5  listeclient=$(ls /home/)
6
7  for client in $listeclient
8  do
9
10
11     last_save=$(ls /home/$client | ls -Art | tail -n 1 | date -r /home/$client +%s)
12     intervalle=$((date_ajd - $last_save))
13
14
15     delai_query=`mysql -u savecorp -psavecorp -e "USE savecorp; SELECT delai_sauvegarde FROM infos_user WHERE nom = '$client' " `
16     delai_seconde=$(echo $delai_query | cut -d " " -f 2)
17     seconde_heure=86400
18
19     delai_heure=$((delai_seconde * seconde_heure))
20
21     echo $intervalle
22     echo $delai_heure
23
24     if [ $intervalle -gt $delai_heure ]
25     then
26         echo "Attention ! le client $client n'a pas respecté les délais de sauvegardes" | mail -s "Sauvegarde non effectuées : $client" ian.logeais@gmail.com
27         echo "la dernière sauvegarde de $client n'est pas obsolète"
28     else
29         echo "le fichier de $client reste dans l'intervalle"
30     fi
31
32
33 done
```

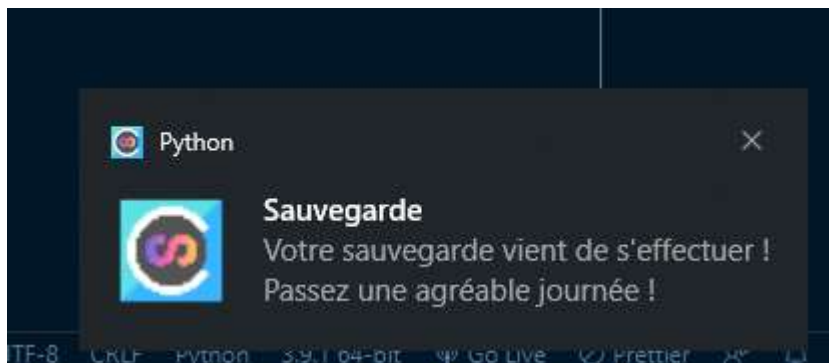


Notifications

Nous avons réalisé un script qui envoie une notification sur l'ordinateur des clients lorsqu'une mise à jour est faite. Pour cela nous avons employés du python grâce à la librairie plyer. Cela nous permet une solution cross-platform qui fonctionne aussi bien sur des systèmes linux que windows.

```
notification.py
1  from plyer import notification
2
3  notification.notify(
4      title = 'Sauvegarde',
5      message = "Votre sauvegarde vient de s'effectuer ! Passez une agréable journée !",
6      app_icon = "savecorp.ico",
7      timeout = 10,
8  )
```

Voici le résultat :



Piste d'amélioration et futures améliorations

Il nous reste donc :

- Créer des utilisateurs et limité leur périmètre d'action grâce à une chroot jail par exemple
- Accès vpn
- Sauvegarde chiffrée et zippée
- Amélioration du formulaire via l'utilisation de regex pour contrôler le format de l'adresse mail ainsi que l'utilisation de chiffre pour les champs durée conservation et délai sauvegarde
- Implémentation de la politique de sauvegarde (incrémentale, full etc...)



Conclusion

L'équipe Savecorp a eu pour mission de rendre accessible et simple d'accès la sauvegarde tout en la rendant automatique afin de perdre le moins de temps possible lors de l'installation des processus. Nous avons pu lier et approfondir nos compétences à la fois en tant qu'administrateur système et développeur afin d'inventer et de créer notre solution de sauvegarde.

Grâce à une équipe soudée et forte d'expérience, de cohésion et de collaboration, nous avons pu mener au mieux ce projet après maintes brainstorming et séances de travail jusqu'au déploiement de la solution.

L'équipe Savecorp vous remercie pour avoir pris le temps d'avoir lu notre rapport et espère vous revoir bientôt.

Annexe

Vous pouvez retrouver tout le code que nous avons créé sur notre github :

<https://github.com/HugoFerry/SaveCorp>



Descriptif de la réalisation professionnelle



Table des matières

Descriptif de la réalisation professionnelle

Table des matières

Coté Serveur

- Conception du projet

- Organisation du code

- Liste API

 - /auth

 - /calls

 - /requests

 - /customers

 - /workers

- Gestion de la sécurité :

Coté Frontend

- Design et différentes pages de l'application

 - Coté utilisateur

 - Coté administrateur

- Point clés de l'application

 - Récupération des données

 - Gestion de l'authentification

 - Gestions des routes

Server Voip Et gestion des appels

- Configuration Asterisk

- Gestion des appels depuis Node

Déploiement

Coté Serveur

Pour le serveur nous avons utilisé Node.js et Express.js et pour la base de donnée nous utilisons MongoDB. Ils vont nous permettre de développer une api qui sera accessible depuis notre front.

Conception du projet

Pour mieux nous représenter le projet nous avons d'abords réalisé un diagramme d'activité.



Puis nous avons créer la base de données avec ces quatre entités .

- Les employés :

```
const workersSchema = schema({
  username: { type: String, required: true, unique: true },
  local: {
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    role: { type: String, default: "basic" },
  },
  avatar: { type: String, default: null },
  number: { type: String, required: true, unique: true },
  state: { type: String, default: "unavailable" },
  lastHangUp: { type: Date, default: null },
});
```

- Les clients :

```
const customerSchema = schema({
  name: { type: String, default: "unknown" },
  number: { type: String, unique: true, required: true },
  email: { type: String, unique: true, default: "unknown" },
  avatar: { type: String, default: null },
});
```

- Les appels :

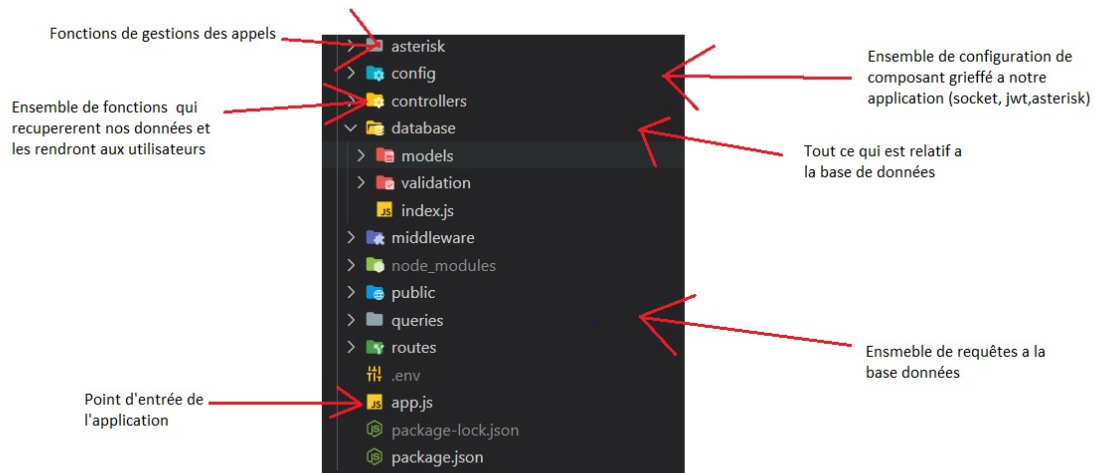
```
const callSchema = schema({
  customer: { type: schema.Types.ObjectId, ref: "customer" },
  number: { type: String, required: true },
  date: { type: Date, default: Date.now },
  time: { type: Number, min: 0, required: true },
  destination: { type: schema.Types.ObjectId, ref: "worker", required: true },
},
);
```

- Les requêtes :

```
const requestSchema = schema({
  message: { type: String, required: true },
  urgencyLevel: { type: String, required: true },
  typeOf: { type: String, required: true },
  deadline: { type: Date, required: true },
  date: { type: Date, default: Date.now },
  done: { type: Boolean, default: false },
  author: { type: schema.Types.ObjectId, ref: "worker", required: true },
  customer: { type: schema.Types.ObjectId, ref: "customer", required: false },
},
  workerDone: { type: schema.Types.ObjectId, ref: "worker" },
);
```

Organisation du code

Le point de départ de notre application est le fichier `app.js`. Lorsqu'elle recevra une requête, elle sera dirigée vers la liste des routes qui la redirigera vers nos controllers qui après éventuelles vérifications pourra questionner notre base de données pour renvoyer une réponse à l'auteur de la requête



Liste API

Voila la liste de l' ensemble des endpoints disponibles sur notre api

/auth

/login

- Description: Connecte l'utilisateur
- Méthode : POST
- Autorisation: Aucune
- Réponse : Renvoi l'utilisateur connecté : {user: user}
- Paramètre: body:{email,password}
- Erreur: Si mauvais identifiants renvoi { message: "Mauvais identifiants" }, sinon renvoi { message: "error" }

/logout

- Description: Déconnecte l'utilisateur
- Méthode : GET
- Autorisation: Authentification d'un utilisateur
- Réponse : Aucune
- Paramètre: Aucun
- Erreur: { message: "error" }

/me

- Description: Renvoi l'utilisateur s'il est actuellement connecté
- Méthode: GET
- Autorisation: Aucune
- Réponse: {user: user } s'il est connecté , {user: null} sinon
- Paramètre : Aucun
- Erreur: { message: "error" })

/calls

/

- Description: Récupérer 5 appels et leur nombre total
- Méthode: POST
- Autorisation: Authentification d'un admin
- Réponse : { items: , count: }
- Paramètre: {page, order, sort, search}
- Erreur: { message: "Wrong Request" }

/getcaller

- Description: Récupère à partir d'un numéro le client associé
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse : { customer }
- Paramètre : {number}
- Erreur: { message: "Wrong Request" }

/me

- Description: Récupère 5 de ses appels
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse: { items: , count: }
- Paramètre : { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/calltime

- Description: Récupère le nombre d'appels entre deux dates
- Méthode: POST
- Autorisation: Authentification d'un admin
- Réponse :{ itemsNumber: }
- Paramètre: {start:, end: }
- Erreur: { message: "Wrong Request" }

/worker/:workerid

- Description: Récupère 5 appels d'un employé donné
- Méthode : POST
- Autorisation: Authentification d'un admin
- Réponse :{ items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/customer/:customerId

- Description: Récupère 5 appels d'un client donné
- Méthode : POST
- Autorisation: Authentification d'un admin
- Réponse :{ items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/requests

/

- Description: Récupère 5 requêtes
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse: { items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/worker/worker:id

- Description: Récupère 5 requete d'un employé donné
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse: { items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/customer/:customerId

- Description: Récupère 5 requete d'un client donné
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse: { items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/alert

- Description: Récupère 5 requêtes dont la deadline est dépassé ou finis dans 24 heure et qui n'est pas validé
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse: { items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/requeststimes

- Description: Récupère le nombre de requêtes entre deux dates
- Méthode: POST
- Autorisation: Authentification d'un admin
- Réponse :{ itemsNumber: }
- Paramètre: {start:, end: }
- Erreur: { message: "Wrong Request" }

/:requestId

- Description: Récupère en détails d'une requête donné
- Méthode : GET
- Autorisation: Authentification d'un utilisateur
- Réponse : { item: request }
- Paramètre : Aucun
- Erreur: { item: null }

/delete/:requestId

- Description: Supprime une reqête donné
- Méthode : Delete
- Autorisation: Authentification d'un admin
- Réponse : Aucune
- Paramètre : Aucune
- Erreur:{ message: "Wrong Request" }

/toggle/:requestId

- Description: Invalide ou valide une requête donné
- Autorisation: Authentification d'un utilisateur
- Réponse : {status : status}
- Paramètre : Aucun
- Erreur: { message: "Wrong Request" }

/new

- Description: Création d'une nouvelle requête
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse : (request)
- Paramètre : body: {message,urgencyLevel,typeof, deadline,customerId}
- Erreur:{errors: [{ field: , message: }]}

/customers

/

- Description: Récupère 5 clients
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse: { items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/:customerId

- Description: Récupère en détails d'une requête donné
- Méthode : GET
- Autorisation: Authentification d'un utilisateur
- Réponse : {customer }
- Paramètre : Aucun
- Erreur: { message: "This customer doesnt exist" }

/avatar/:customerId

- Description: Modifie l'avatar d'un client donné
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse : { avatar: pathFile }
- Paramètre : {profile : ["jpeg",png,jpg]} (format : multipart/form-data)
- Erreur: { message: "Wrong Request" }

/new

- Description: Création d'un nouveau client
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse: (customer)
- Paramètre : body: {message,urgencyLevel,typeof, deadline,customerId}
- Erreur: {errors: [{ field: , message: }]}

/delete/:customerId

- Description: Supprime un client donné
- Méthode : Delete
- Autorisation: Authentification d'un admin
- Réponse : Aucune
- Paramètre : Aucun
- Erreur:{ message: "Wrong Request" }

/workers

/

- Description: Récupère 5 employés
- Méthode : POST
- Autorisation: Authentification d'un uadmin
- Réponse: { items: , count: }
- Paramètre: { page, order, sort, search }
- Erreur: { message: "Wrong Request" }

/:workerId

- Description: Récupère en détails un client donné
- Méthode : GET
- Autorisation: Authentification d'un admin
- Réponse : {worker }
- Paramètre : Aucun
- Erreur: { message: "This user doesnt exist" }

/signup

- Description: Inscription d'un utilisateur
- Méthode : POST
- Autorisation: Authentification d'un admin
- Réponse: (user)
- Paramètre : body: { username, email, number, password }
- Erreur: {errors: [{ field: , message: }]}

/update

- Description: Modification du nom ou de l'email d'un employé (on peut seulement se modifier soi même)
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse :(worker)
- Paramètre: body{email,usersame}
- Erreur: {errors: [{ field: , message: }]}

/updatePassword

- Description: Modification du mot de passe d'un employé (on peut seulement se modifier soi même)
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse : Aucune
- Paramètre: body{password}
- Erreur: {errors: [{ field: , message: }]}

/delete/:workerId

- Description: Supprime un employé donné
- Méthode : Delete
- Autorisation: Authentification d'un admin
- Réponse : Aucune
- Paramètre : Aucun
- Erreur:{ message: "Wrong Request" }

/passadmin/:workerId

- Description: Rend un utilisateur administrateur
- Méthode : POST
- Autorisation: Authentification d'un admin
- Réponse : Aucune
- Paramètre : Aucun
- Erreur:{ message: "Wrong Request" }

/passbasic/:workerId

- Description: Rend un utilisateur sans droit d'administrateur
- Méthode : POST
- Autorisation: Authentification d'un admin
- Réponse : Aucune
- Paramètre : Aucun
- Erreur:{ message: "Wrong Request" }

/togglestate

- Description: Alterne état d'un utilisateur entre disponible et indisponible
- Méthode : POST
- Autorisation: Authentification d'un admin
- Réponse : { state: }
- Paramètre : Aucun
- Erreur:{ message: "Wrong Request" }

/avatar

- Description: Modifie l'avatar d'un employé (chaque employé ne peut modifier que sa propre avatar)
- Méthode : POST
- Autorisation: Authentification d'un utilisateur
- Réponse : { avatar: pathFile }
- Paramètre : {profile: ["jpeg","png,jpg"]} (format : multipart/form-data)
- Erreur: { message: "Wrong Request" }

Gestion de la sécurité :

Pour la sécurité de notre application , nous avons choisi d'utiliser des JWT token. Ils vont nous permettre de connaître l'identité des auteurs des requêtes qui arrivent dur l'api , car chacune des requetés d'utilisateur contiendra en cookie contenant le token qu'on pourra analyser .

```
const createJwtToken = ({ user = null, id = null }) => {
  const jwtToken = jwt.sign(
    {
      sub: id || user._id.toString(),
      exp: Math.floor(Date.now() / 1000) + 5,
    },
    secret
  );
  return jwtToken;
};
```

On y stockera l'id de l'utilisateur et la date d'expiration du token avec une possibilité de le renouveler .

Et grâce a la fonction suivante on rendra les jwt token utilisables dans tout notre application grâce a des méthodes sur l'objet "req"

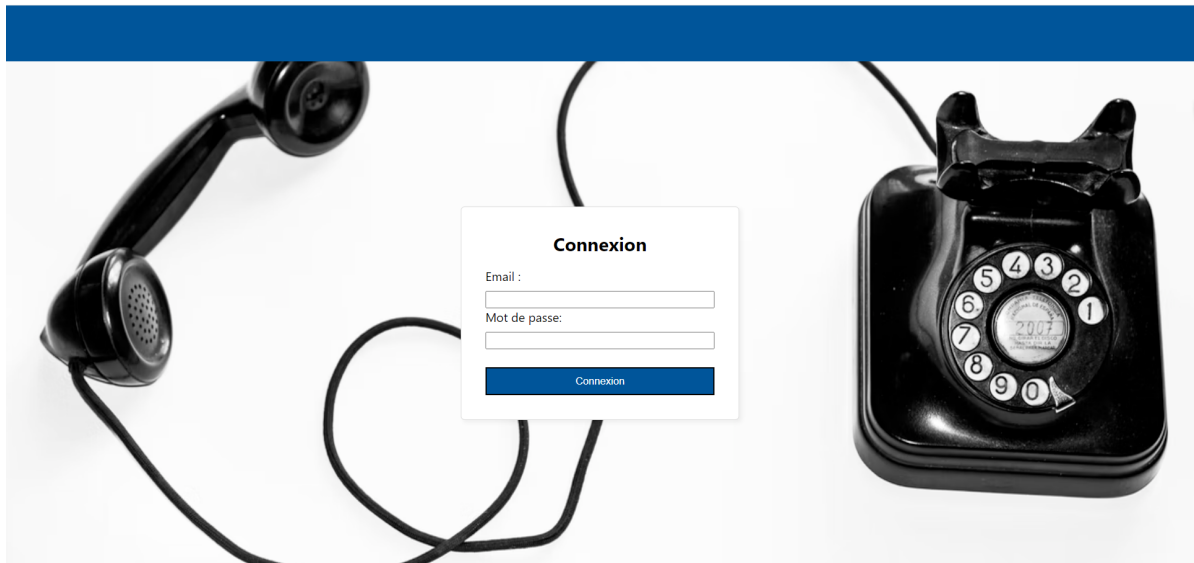
```
const addJwtFeatures = (req, res, next) => {
  req.isAuthenticated = () => !!req.user;
  req.logout = () => res.clearCookie("jwt");
  req.login = (user) => {
    const token = createJwtToken({ user });
    res.cookie("jwt", token);
  };
  next();
};
```

Et donc pour chaque requête en fonction de l'endpoint on pourra effectuer des vérification , comme vérifier si c'est un admin et bloquer la requête ou non en fonction de la réponse :

```
exports.requireAuthAdmin = (req, res, next) => {
  if (req.isAuthenticated()) {
    if (req.user.local.role == "admin") {
      next();
    } else {
      res.status(404).send({ message: "You dont have the rights" });
    }
  } else {
    res.status(404).send({ message: "Your are not logged in" });
  }
};
```

Coté Frontend

Design et différentes pages de l'application

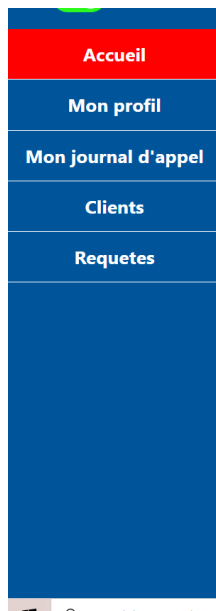


Page de connexion ou est redirigé n'importe quelle utilisateur qui n'est pas connecté

Coté utilisateur



La navbar avec un menu burger , un switch pour se rendre disponible ou non ,un panel administrateur affiché seulement aux administrateurs , et la possibilité de se déconnecter



Menu Pour se rendre dans les différents onglets

Les Requetes en alerte							
<div> <div>Rechercher par email de client</div> <div></div> </div>							
Type	Auteur	Client	Valider	Deadline	Date	Niveau d'urgence	Action
Email	dany@hotmail.com	alptttta25@hotmail.com	✖	il y a environ 1 mois	09/04/2022, 21:07:31		Consultez
Email	dany@hotmail.com	alptttta25@hotmail.com	✖	il y a environ 1 mois	09/04/2022, 21:08:51		Consultez
Call	ali@hotmail.com	alpa25@hotmail.com	✖	il y a environ 4 ans	26/03/2022, 05:08:48		Consultez
Call	ali@hotmail.com	Aucun	✖	il y a environ 4 ans	26/03/2022, 00:15:17		Consultez
Call	dany@hotmail.com	alptttta25@hotmail.com	✖	il y a environ 4 ans	28/03/2022, 14:13:55		Consultez

Home page ou sont affichées les requêtes en alerte (Les requêtes dont la deadline est dépassé et dont la deadline risque d'expirer dans les prochaines 24h et qui n'est pas validé)



pama

dany@hotmail.com

2000

Modifier vos informations

Email

dany@hotmail.com

Username

pama

Modifier

Modifiez votre mot de passe

Mot de passe

Confirmez le mot de passe

Modifier

Mes dernieres requetes

Rechercher par email de client

Ajoutez une nouvelle requete

Page de profil d'un employé

Mon journal d'appel				
<div> <div>recherchez par email de client</div> <div></div> </div>				
Temps	Etat	Email	Numero	Date
00:00	Appel manqué	alpa25@hotmail.com	064347985642789999	2022-04-17T22:29:34.476Z
00:00	Appel manqué	leclientdelamama@hotmail.com	3000	2022-05-28T20:10:50.144Z
01:25	Appel pris	alpa25@hotmail.com	064347985642789999	2022-04-17T12:09:40.229Z
00:41	Appel pris	alpa25@hotmail.com	064347985642789999	2022-04-17T03:00:30.498Z
00:00	Appel manqué	alpa25@hotmail.com	064347985642789999	2022-04-17T22:20:19.782Z

<

1

2

3

4

5

...

8

>

Journal d'appel d'un employé

Les Clients			
<div>Rechercher par l'email du client</div>			<div>Ajoutez un nouveau client</div>
Nom	Email	Numero	Action
client	papasbasah@hotmail.com	12345	Consultez
client	client@hotmail.com	2547863525	Consultez
dany salezh	alpttta25@hotmail.com	06477347985642789999	Consultez
ali salezh	alpa25@hotmail.com	064347985642789999	Consultez
danmamamam alezh	alptttfa25@hotmail.com	064773e47985642789999	Consultez
<div>< 1 2 ></div>			

Tableau des clients enregistrés

Requetes							
<div>Recherchez par email de client</div>							<div>Ajoutez une nouvelle requete</div>
Type	Auteur	Client	Valider	Deadline	Date	Niveau d'urgence	Action
Call	dany@hotmail.com	alpttta25@hotmail.com	✓	il y a environ 2 mois	09/04/2022, 19:32:06	<div></div>	Consultez
Email	dany@hotmail.com	alpttta25@hotmail.com	✗	il y a environ 1 mois	09/04/2022, 21:07:31	<div></div>	Consultez
Call	ali@hotmail.com	alpa25@hotmail.com	✗	il y a environ 4 ans	26/03/2022, 05:08:48	<div></div>	Consultez
Call	ali@hotmail.com	Aucun	✗	il y a environ 4 ans	26/03/2022, 00:15:17	<div></div>	Consultez
Call	dany@hotmail.com	alptttfa25@hotmail.com	✗	il y a environ 4 ans	28/03/2022, 14:13:55	<div></div>	Consultez

Tableau des requêtes

Ma nouvelle requete

Message

Veuillez choisir un client

Deadline :

jj/mm/aaaa --:--

Choisissez un type de requête

Choisissez un niveau d'urgence

Valider la requette

Page de création d'une nouvelle requete

Mon nouveau client

Email

Nom

Numero

Valider le nouveau client

Page de création d'un nouveau client

Requete numero : 6251d97357c6ef2208808d8a

dany salezh

alptttta25@hotmail.com

06477347985642789999

pama

dany@hotmail.com

2000

rem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of L

Date de création : 09/04/2022, 21:07:31

Deadline : il y a environ 1 mois

Supprimer

Valider la requette

Page d'une requête

Page client

client

papasbasah@hotmail.com

12345

Ses Requetes

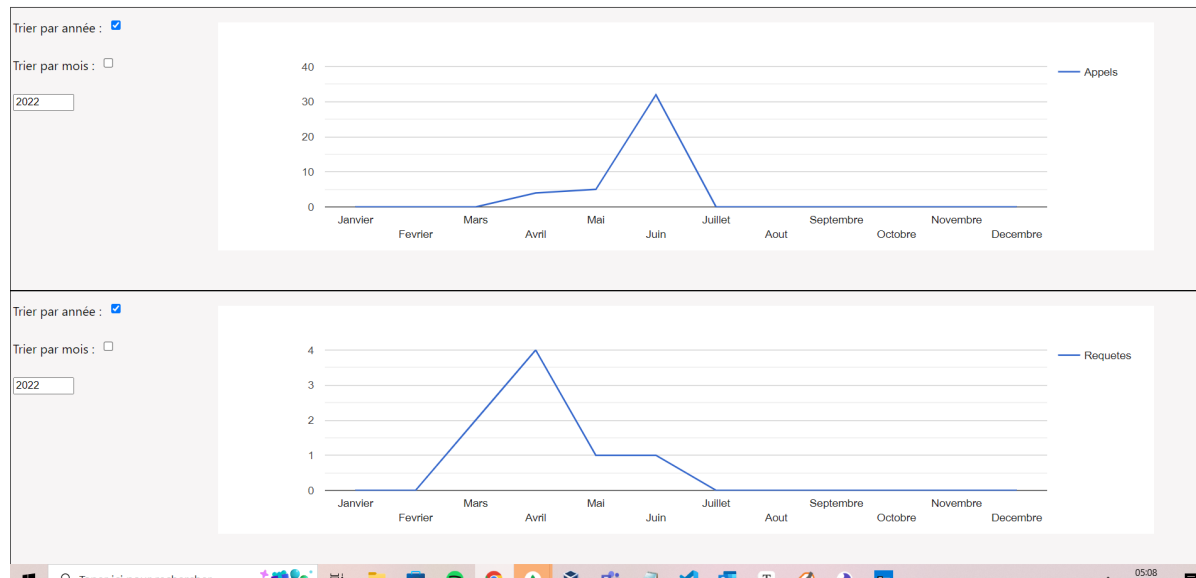
rechercher

Lui rajoutez une nouvelle reuqete

Type	Auteur	Valider	Deadline	Date	Niveau d'urgence	Action
Email	dany@hotmail.com		dans 19 jours	04/06/2022, 23:48:33		Consultez

Page D'un client

Coté administrateur



Home page du panel administrateur avec la possibilité grâce a google charts d'avoir sous forme de stats les appels et les requêtes en fonction de plusieurs critères

WorkerAdmin

Recherchez par email

[Créer un nouveau utilisateur](#)

Email	Role	Disponible	Dernier Appel	Pseudo	Numero	Action
ali@hotmail.com	admin	Indisponible	2022-06-04T20:20:10.183Z	ali	5000	Consultez
dany@hotmail.com	admin	disponible	2022-06-05T23:14:43.672Z	pama	2000	Consultez

Tableau de tous les employés

Tous les appels

Recherchez par email du client

Temps	Etat	Email	Numero	Date
00:00	Appel manqué	alpa25@hotmail.com	064347985642789999	2022-04-17T22:29:34.476Z
00:00	Appel manqué	leclientdelamama@hotmail.com	3000	2022-05-28T20:10:50.144Z
01:25	Appel pris	alpa25@hotmail.com	064347985642789999	2022-04-17T12:09:40.229Z
00:41	Appel pris	alpa25@hotmail.com	064347985642789999	2022-04-17T03:00:30.498Z
00:00	Appel manqué	alpa25@hotmail.com	064347985642789999	2022-04-17T22:20:19.782Z

< 1 2 3 4 5 ... 9 >

Tableau de tous les appels

ali

ali@hotmail.com

5000


admin

Supprimer

Enlevez les droits d'administrateur

Ses dernieres requetes

Rechercher par email de client

Type	Auteur	Client	Valider	Deadline	Date	Niveau d'urgence	Action
Call	ali@hotmail.com	alpa25@hotmail.com		il y a environ 4 ans	26/03/2022, 05:08:48		Consultez

Page de profil d'un employé

Point clés de l'application

Récupération des données

Pour récupérer les données depuis l'api nous allons réutiliser la même fonction :

```
export async function apiFetch(endpoint, options = {}) {
  options = {
    credentials: "include",
    headers: {
      Accept: "application/json",
    },
    ...options,
  };
  if (
    options.body !== null &&
    typeof options.body === "object" &&
    !(options.body instanceof FormData)
  ) {
    options.body = JSON.stringify(options.body);
    options.headers["Content-type"] = "application/json";
  }
  const response = await fetch("http://localhost:4000" + endpoint, options);
  if (response.status === 204) {
    return null;
  }

  const responseData = await response.json();

  if (response.ok) {
    return responseData;
  } else {
    if (responseData.errors) {
      throw new ApiErrors(responseData.errors);
    }
  }
}
```

Puis pour chaque type de données différente on a des parseur différents qui vont nous permettre de les rendre utilisable .

Gestion de l'authentification

Pour pouvoir maîtriser l'authentification et rendre disponible les informations de l'utilisateur dans tout type de composants on va créer un contexte Authcontext dont on pourra faire appel dans toute les parties de l'application. Et on va utiliser aussi le Hook reducer fourni par React pour garder à jour les informations de l'utilisateur à la suite d'une connexion , d'une actualisation ou alors d'une déconnexion.

Gestions des routes

Nous avons déclaré l'ensemble des routes de l'application grâce à différents composants du package "react-router-dom" dans le fichier App.js.

Server Voip Et gestion des appels

Pour le serveur Voip on va utiliser Asterisk qui est un autocommutateur téléphonique privé libre et propriétaire pour systèmes GNU/Linux. On va l'installer sur une machine virtuelle pour l'instant . On utilise asterisk surtout grâce au module ARI qu'il propose . ARI est une RESTAPI qui va nous permettre depuis notre serveur web de contrôler tous les aspects d'un appel .

Configuration Asterisk

Après avoir installé asterisk on doit réaliser quelques modifications comme l'activation du module ARI , ou la configuration du dial plan dans extension.conf :

```
[default]
exten => 1000,1,NoOp()
same => n,Stasis(Callcenter,inbound,PJSIP/worker)
same => n,Hangup()
```

Le dial plan est une suite d'instructions qu'on va appliquer à chaque personne qui appelle notre serveur Asterisk . On va donc rediriger tous les appels vers le numéro 1000 vers Nodejs ,et on pourra y appliquer le traitement que l'on veut .

On va aussi configurer le pjsip.conf qui contiendra ensemble des numéros qui peuvent qui pourront se connecter à notre serveur Voip

Exemple type de l'enregistrement du numéro 2000:

```
[2000](endpoint-basic)
auth=2000
aors=2000

[2000](auth-userpass)
password=2000
username=2000

[2000](aor-single-reg)
```

Gestion des appels depuis Node

Pour gérer le serveur Voip depuis Nodejs on va utiliser le module "ari-client" qui contient une librairie permettant de utiliser l'API d'asterisk plus facilement

```
var ari = require("ari-client");
const System = require("../asterisk/system");
///// all the variables of asterisk
ipAsterisk = process.env.ASTERISKIP;
PortAsterisk = process.env.ASTERISKPORT;
userAsterisk = process.env.ASTERISKUSER;
mdpAsterisk = process.env.ASTERISKPASSWORD;
```

```

ari
    .connect(
        "http://" + ipAsterisk + ":" + PortAsterisk,
        userAsterisk,
        mdpAsterisk
    )
    .then(function (client) {
        var system;

        function onStasisStart(event, channel) {
            system.onStasisStart(event, channel);
        }

        system = new System(client);
        client.on("StasisStart", onStasisStart);

        client.start("Callcenter");
    });

```

Cette fonction va nous permettre dès le lancement du serveur de se connecter a Asterisk et de rester à écoute des redirections des appels de la part d'Asterisk qu'on va pouvoir manipuler

En effet chaque appel sera introduit dans une queue et une salle d'attente qui lui est propre et en arrière plan sera lancé a plusieurs moment de l'application notre fonction nextInTheQueue.

Cette fonction va prendre le plus ancien appel en attente de la queue et en fonctions de certaines conditions on lui appliquera différents processus :

- On trouve un employé disponible dans la base de données et on lui attribue cet appel et on réunit dans le même pont les deux intervenant quand il répond .Au début de l'appel l'employé en base de donnée sera mis comme "occupé" et sera remis disponible en fin d'appel.
- Il y a aucun employés disponible , mais il y en a occupé donc l'appel reste en attente .
- Il n'y a aucun employés disponible ni employés occupés donc on raccrochera a l'appel.
- Un employé est disponible dans la base de donnée , mais son téléphone n'est pas connecté , il sera donc mis indisponible dans la base de données et le client sera remis au début de la queue.

De plus asterisk va nous permettre de rattacher a nos différents objets des écouteurs événement pour pouvoir bien contrôler le flux et gérer tous les cas

Un exemple est qu'on va pouvoir lorsque qu'un client quitte subitement un appel , fermer directement le pont qui le liait a notre employé et raccroché son téléphone.

Implémentation des sockets pour pouvoir notifier visuellement l'utilisateur

Socket.IO est une bibliothèque JavaScript pilotée par les événements pour les applications Web en temps réel. Il permet une communication bidirectionnelle en temps réel entre les clients Web et les serveurs.

Ce qui est idéal pour nous car on a un application front react qui est single page app , et donc qui ne se recharge pas .

Pour configurés nos web sockets depuis nodejs, pour chaque connexion on vérifiera l'authenticité des requêtes grâce a nos jwt et la fonction "ensureAuthenticatedOnSocketHandshake", et après succès on pourra assigner à chaque employé une room personnelle qui se nomme par son id. Cette gestion des room nous permettra vraiment de sélectionner a qui nous communiqueront par web socket .

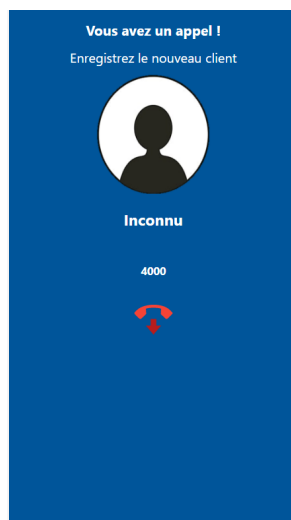
```
let ios;

ios = socketio(server, {
  //authentication
  cors: {
    origin: process.env.ALLOWFRONT,
    methods: ["GET", "POST"],
    credentials: true,
  },
  allowRequest: ensureAuthenticatedOnSocketHandshake,
});
ios.on("connect", (socket) => {
  workerId = socket.request.user._id;

  socket.join(`${workerId}`);
});
```

Et donc du coté front on sera à l'écoute de certains évènement qui provoqueront des réactions en chaines , et du coté serveur on pourra émettre dans les room spécifique grâce a l'id de l'employé qu'on aura récupérer au moment de l'appel.

Un exemple concret est lorsque l'on va appeler le téléphone d'un employé son interface va changer pour lui notifier qu'il a un appel sur son téléphone



Si le numéro est inconnu on proposera à l'employé de l'enregistrer

Les Requetes en alerte

Rechercher par email de client

Type	Auteur	Client	Valider	Deadline	Date	Niveau d'urgence	Action
Email	dany@hotmail.com	alptttfa25@hotmail.com	✖	il y a environ 1 mois	09/04/2022, 21:08:51	●	Consultez
Email	dany@hotmail.com	client@hotmail.com	✖	il y a environ 1 mois	11/04/2022, 00:32:07	●	Consultez
Call	dany@hotmail.com	alptttfa25@hotmail.com	✖	il y a environ 4 ans	28/03/2022, 14:13:55	●	Consultez
Call	ali@hotmail.com	alpa25@hotmail.com	✖	il y a environ 4 ans	26/03/2022, 05:08:48	●	Consultez
Email	dany@hotmail.com	alptttta25@hotmail.com	✖	il y a environ 1 mois	09/04/2022, 21:07:31	●	Consultez

Vous avez un appel !
Consultez la fiche client Lui créer une requette


stanislas

3000


Zoiper5 5.4.12 for Windows 64bit

 **Incoming call**
Anonymous
2000@192.168.17.1@192.168.1.17
DNID: 2000

Si le numéro est déjà enregistré on proposera à l'employé de pouvoir lui créer une requête ou de consulter sa fiche client

Les Requetes en alerte

Rechercher par email de client

Type	Auteur	Client
Email	dany@hotmail.com	alptttt
Email	dany@hotmail.com	client
Call	dany@hotmail.com	alptttt
Call	ali@hotmail.com	alpa25
Email	dany@hotmail.com	alptttt



Anonymous
2000@192.168.17.1@192.168.1.17
00:13

Click here to add a new contact

Vous avez un appel !
Consultez la fiche client Lui créer une requette


stanislas

3000
00:12


Et si on répond un timer sera déclenché en parallèle de l'appel

On pourra même éventuellement raccroché l'appel depuis l'interface web .

Déploiement

Pour le déploiement on a tout d'abord acheter un vps et un nom de domaine : callcenter-ppe.site sur ovh. La première étape fut de s'y connecter grâce a une clé ssh en tant que root puis d'installer installer nginx, Node ,npm et git .

Après avoir importé notre code depuis GitHub et avoir installé nos dépendances sur le côté client et le côté serveur on a configuré nginx

```
server {
    listen 80;
    server_name www.callcenter-ppe.site callcenter-ppe.site;
    location / {
        root /var/www/Callcenter/client/;
        index index.html index.htm;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://51.38.188.152:8800;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

On a ensuite réalisé quelques modifications dans notre code notamment dans les fichiers d'environnement , et on a build notre application react qu'on a placée dans le dossier /var/www/Callcenter

La prochaine étape fut d'utiliser PM2 qui est un gestionnaire de processus pour le runtime JavaScript Node.js et va nous permettre de garder active notre serveur même lorsque l'on ferme notre terminal.

Pour la base de données nous avons utilisé de la déployer sur MongoDB Atlas qui est une base de données cloud entièrement gérée et qui prend en charge toute la complexité du déploiement.

La dernière étape fut configurer les DNS de notre nom de domaine pour les rediriger vers notre application et enfin de configurer un certificat SSL/TLS gratuit grâce à Let's Encrypt's .

Actuellement l'application est en ligne sous <https://callcenter-ppe.site>, mais contrairement a environnement local nous n'avons pas réussi malgré de nombreuses tentative à déployer notre serveur Voip asterisk .Donc le CRUD avec tous les appels d'API fonctionnent parfaitement mais nous ne pouvons pas réaliser d'appel ailleurs qu'en local.