

Laubier

Martial

BTS SIO SISR

TEAMWORK

1 Rue Jean Carmet, 69800 Saint-Priest



Mettre à disposition des utilisateurs un service informatique

Développement WEB / SSL / FQDN / IP

Table des matières

.....	1
MISE EN PLACE D'UN SERVICE.....	1
1. DEVELOPPEMENT D'UN SITE DE MONITORING DES CERTIFICATS	3
<i>Contexte de l'entreprise :.....</i>	<i>3</i>
<i>L'idée du dashboard de suivi des certificats SSL</i>	<i>4</i>
<i>Fonctionnalités du site de monitoring des certificats SSL</i>	<i>4</i>
<i>Démarche technique (lien annexe)</i>	<i>5</i>
<i>Gestion des problèmes et ajustements</i>	<i>5</i>
<i>Conclusion</i>	<i>6</i>
2. ANNEXE DEVELOPPEMENT D'UN SITE DE MONITORING.....	7
1. <i>Structure du projet React</i>	<i>7</i>
2. <i>Backend : Fichier server.js.....</i>	<i>8</i>
3. <i>Frontend : Composant React</i>	<i>9</i>
4. <i>Fonctionnalité de vérification SSL et affichage visuel</i>	<i>10</i>
5. <i>Gestion des erreurs et des mises à jour</i>	<i>11</i>
6. <i>Conclusion technique du projet</i>	<i>11</i>

Mettre à disposition des utilisateurs un service informatique

1. Développement d'un site de monitoring des certificats

Contexte de l'entreprise :

Chez TeamWork, de nombreux services sont déployés sur des serveurs, qu'il s'agisse de machines AWS ou Azure. Ces services, souvent développés en interne, sont accessibles via des adresses web. Cependant, la gestion de ces adresses, en particulier lorsqu'il s'agit d'y accéder via HTTPS, pose certains défis.

En effet, les services web déployés sur différentes machines peuvent parfois proposer plusieurs services distincts. Par exemple, une même machine peut héberger une instance de **Grafana** sur le port 3000 et un **Wiki interne** sur le port 3001. Pour faciliter l'accès à ces services, l'entreprise utilise des noms de domaines configurés pour rediriger vers l'IP et le port corrects de chaque machine. Pour sécuriser ces connexions et éviter les alertes de sécurité dans les navigateurs des employés, TeamWork utilise des certificats SSL générés via **Certbot**, un outil open-source qui permet de générer des certificats SSL gratuits et auto-signés. Cependant, ces certificats ont une durée de validité limitée (généralement autour des 90 jours), et doivent être renouvelés régulièrement.

Lorsque ces certificats expirent, les employés rencontrent des erreurs de connexion HTTPS, ce qui déclenche un processus **long** et **fastidieux** :

1. L'utilisateur crée un **ticket** pour signaler le problème.
2. Le ticket est transmis à l'équipe **Réseaux & Sécurité**.
3. Un technicien identifie le serveur et le service concernés.
4. Il renouvelle le certificat SSL manuellement.
5. Enfin, il vérifie que tout fonctionne et ferme le ticket.

Ce processus manuel était inefficace et pouvait causer des interruptions de service prolongées, surtout lorsque les certificats arrivaient à expiration sans être renouvelés à temps.

L'idée du dashboard de suivi des certificats SSL

Pour résoudre ce problème, j'ai proposé la création d'un **dashboard simple et intuitif** permettant de suivre l'état des certificats SSL des différents services web de l'entreprise.

Le concept était le suivant :

- Créer une **interface web** accessible à toute l'équipe **Réseaux & Sécurité**, affichant un tableau récapitulatif de tous les noms de domaine utilisés.
- Chaque nom de domaine serait associé à son **FQDN**, son **adresse IP**, le **port** utilisé et le **propriétaire du service**.
- Une **colonne supplémentaire** indiquerait la **date d'expiration** du certificat SSL correspondant.

L'objectif du tableau est de donner une vue d'ensemble des certificats SSL en cours de validité. Ainsi, avant l'expiration d'un certificat, l'équipe peut prendre les mesures nécessaires pour éviter toute interruption de service.

Fonctionnalités du site de monitoring des certificats SSL

- **Ajout et modification de services** : Le tableau permet à tout employé responsable d'un service web de **s'ajouter** au tableau en renseignant le **FQDN**, l'**IP**, le **port** et son **nom** en tant que propriétaire du service. Une nouvelle ligne est alors automatiquement créée dans le tableau, et le site se charge de récupérer et d'afficher la date d'expiration du certificat SSL lié.
- **Suivi automatique des certificats SSL** : Le **serveur Node.js** effectue des requêtes périodiques sur chaque domaine pour récupérer la date de validité du certificat SSL. Cette information est ensuite transmise au **front-end** (développé en **React**), qui l'affiche dans le tableau.
- **Indicateurs visuels** : Pour faciliter le suivi, chaque ligne du tableau est colorée en fonction de la **validité du certificat** :
 - **Vert** : Le certificat est valide et sa date d'expiration est supérieure à 14 jours.
 - **Jaune** : Le certificat arrive à expiration dans **moins de 14 jours**, signalant qu'un renouvellement devrait être envisagé.
 - **Rouge** : Le certificat expire dans **moins de 7 jours**, indiquant une **urgence** pour le renouvellement.
- **Mise à jour dynamique** : Chaque fois que la page est rafraîchie, la date de validité des certificats est **mise à jour en temps réel**. Cela permet à l'équipe de s'assurer que les certificats sont toujours à jour sans avoir à interroger manuellement chaque service.

Démarche technique (lien annexe)

Développement du front-end avec React : Le front-end de ce site a été entièrement développé avec **React**. Cette technologie permet de créer une interface utilisateur réactive et dynamique. L'utilisation de **React** a permis de rendre le tableau interactif, avec une mise à jour automatique des informations après chaque rafraîchissement.

Développement du back-end avec Node.js : Le **back-end** est géré par un serveur **Node.js**, qui s'occupe des tâches critiques, comme :

- L'interrogation régulière des serveurs via des requêtes HTTPS pour vérifier la validité des certificats SSL.
- Le traitement des informations reçues (par exemple, extraire la date d'expiration du certificat).
- La transmission des données au front-end sous forme de **JSON**, qui est ensuite interprété et affiché dans le tableau.

Gestion des requêtes et sécurisation : Pour interroger les différents noms de domaine, le serveur **Node.js** utilise des modules comme **https** et **Openssl** afin de récupérer les informations sur les certificats SSL. Ce processus est sécurisé pour garantir que seules les informations nécessaires sont récupérées et transmises.

Déploiement et accessibilité : Le site est déployé sur une **machine virtuelle (VM)** dans le cloud. J'ai configuré un **reverse proxy** pour rendre le site accessible via un nom de domaine interne à l'entreprise. Toutes les communications sont sécurisées en HTTPS, et le site est uniquement accessible via le réseau interne de TeamWork.

Gestion des problèmes et ajustements

Au cours du développement, plusieurs ajustements ont été nécessaires :

- **Problème de format des dates :** Lors de la récupération des certificats SSL, les dates étaient parfois dans des formats différents selon les serveurs. J'ai dû adapter le traitement des dates pour les normaliser dans un format lisible (par exemple, **ISO 8601**), ce qui a impliqué des conversions automatiques au niveau du serveur.
- **Mise en cache des requêtes :** Comme la récupération des certificats pouvait prendre un certain temps, j'ai implémenté un système de **mise en cache** temporaire des résultats pour éviter que les utilisateurs n'aient à attendre un temps de réponse long à chaque rafraîchissement de la page.

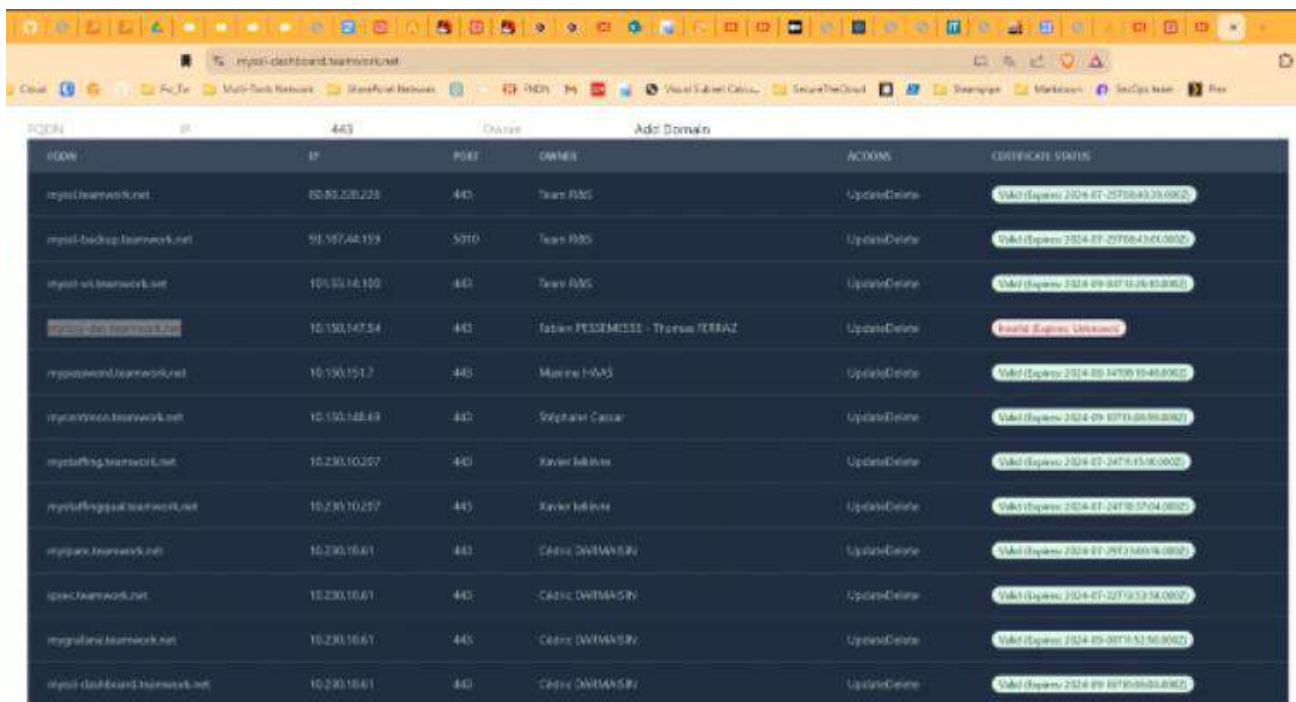
- **Automatisation des mises à jour :** Le serveur **Node.js** interroge régulièrement les certificats sans nécessiter d'intervention manuelle, garantissant que les informations sont toujours à jour.

Conclusion

Ce dashboard de monitoring des certificats SSL a permis à TeamWork de :

- **Optimiser le suivi des certificats SSL** pour éviter les interruptions de service dues à des certificats expirés.
- **Automatiser le processus de vérification** des certificats, réduisant ainsi le nombre de tickets et le temps d'intervention des équipes.
- **Améliorer la visibilité et la gestion des services web** grâce à une interface claire et intuitive.

Le projet a été une réussite, et l'outil est aujourd'hui utilisé régulièrement par l'équipe **Réseaux & Sécurité** pour s'assurer que tous les services critiques sont correctement protégés par des certificats SSL à jour.



The screenshot shows a web browser displaying the 'myssl-dashboard.teamwork.net' interface. The dashboard is a table with columns: FQDN, IP, PORT, OWNER, Addr Domain, ACTIONS, and CERTIFICATE STATUS. It lists various services and their SSL certificate expiration dates.

FQDN	IP	PORT	OWNER	Addr Domain	ACTIONS	CERTIFICATE STATUS
myssl.teamwork.net	80.85.235.238	443	Team R&S		Update Certificate	Valid (Expires 2024-07-25T08:39:08Z)
myssl-backup.teamwork.net	91.597.44.159	5010	Team R&S		Update Certificate	Valid (Expires 2024-07-25T08:40:08Z)
myssl-ws.teamwork.net	10.130.16.100	443	Team R&S		Update Certificate	Valid (Expires 2024-09-07T11:26:40Z)
myssl-logs.teamwork.net	10.130.167.54	443	Tabien PESSEMIE - Thomas TERRAZ		Update Certificate	Expired (Expires Unknown)
myssl-wm.teamwork.net	10.130.151.7	443	Marcin F&AS		Update Certificate	Valid (Expires 2024-09-14T09:10:40Z)
myssl-wm.teamwork.net	10.130.148.48	443	Silphane Caste		Update Certificate	Valid (Expires 2024-09-10T10:09:08Z)
myssl-fing.teamwork.net	10.230.10.257	443	Xavier Jelline		Update Certificate	Valid (Expires 2024-07-24T11:15:08Z)
myssl-fing.teamwork.net	10.230.10.257	443	Xavier Jelline		Update Certificate	Valid (Expires 2024-07-24T11:15:08Z)
myssl-sec.teamwork.net	10.230.16.61	443	Chénc DWTM&S/R		Update Certificate	Valid (Expires 2024-07-25T10:09:08Z)
myssl-sec.teamwork.net	10.230.16.61	443	Chénc DWTM&S/R		Update Certificate	Valid (Expires 2024-07-25T10:09:08Z)
myssl-sec.teamwork.net	10.230.16.61	443	Chénc DWTM&S/R		Update Certificate	Valid (Expires 2024-07-25T10:09:08Z)
myssl-sec.teamwork.net	10.230.16.61	443	Chénc DWTM&S/R		Update Certificate	Valid (Expires 2024-07-25T10:09:08Z)

2. Annexe développement d'un site de monitoring

Contexte du projet :

L'objectif de ce projet était de créer un **site web en React** permettant de surveiller les certificats SSL des différents services déployés au sein de l'entreprise TeamWork. Le site devait afficher les **noms de domaine** avec leurs adresses IP et propriétaires, ainsi que la **date d'expiration** des certificats SSL pour chaque service.

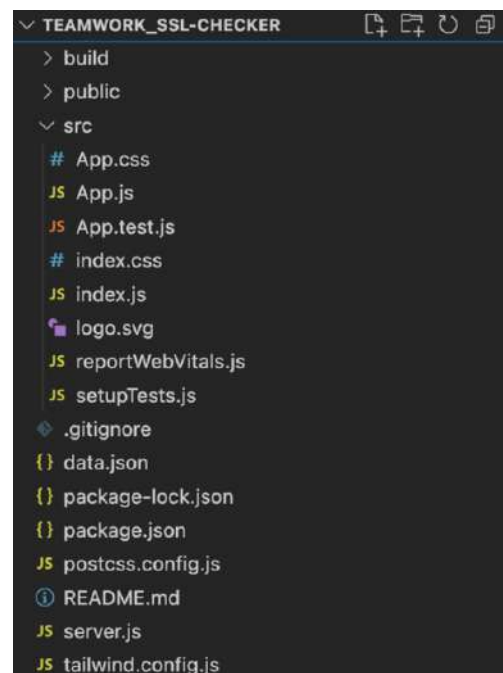
Une alerte visuelle permettait d'identifier facilement les certificats proches de l'expiration, avec une catégorisation :

- **Vert** : Certificat valide
- **Jaune** : Certificat expirant dans moins de 14 jours
- **Rouge** : Certificat expirant dans moins de 7 jours

1. Structure du projet React

Le projet suit une structure classique d'une application **React**, avec une partie **backend** (server.js) en **Node.js** pour gérer les interactions avec les certificats SSL.

Arborescence du projet :



2. Backend : Fichier server.js

Le fichier **server.js** est le cœur du **backend**. Il permet de gérer les requêtes, de surveiller les certificats SSL, et de servir les fichiers React pour l'interface utilisateur.

Voici un extrait du fichier **server.js** qui montre comment l'application vérifie les certificats SSL d'un domaine et renvoie les informations pertinentes (valide/invalide, date d'expiration, etc.) :

```
const express = require("express");
const sslChecker = require("ssl-checker");
const fs = require("fs");
const path = require("path");
const app = express();

app.use(express.json());

// Fonction pour vérifier les certificats SSL
const checkSsl = async (domain) => {
  try {
    let result = await sslChecker(domain, { method: 'GET', port: 443 });
    if (result.valid && result.validTo) {
      const currentDate = new Date();
      const validUntilDate = new Date(result.validTo);
      const daysRemaining = (validUntilDate - currentDate) / (1000 * 60 * 60 * 24);

      let certificateStatus = 'Valid';
      if (daysRemaining <= 7) certificateStatus = 'Expiring Soon';
      if (daysRemaining <= 0) certificateStatus = 'Invalid';

      return {
        domain,
        hasValidCertificate: certificateStatus !== 'Invalid',
        validUntil: result.validTo,
        certificateStatus
      };
    }
    return { domain, hasValidCertificate: false, validUntil: null, certificateStatus: 'Invalid' };
  } catch (error) {
    console.error(`Error checking SSL for ${domain}:`, error);
    return { domain, hasValidCertificate: false, validUntil: null, certificateStatus: 'Invalid' };
  }
};

// Route pour surveiller un domaine
app.get('/check-ssl', async (req, res) => {
  const domain = req.query.domain;
  if (!domain) return res.status(400).json({ error: 'Domain is required' });

  try {
    const result = await checkSsl(domain);
    res.json(result);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Explication :

Le script effectue une vérification des certificats SSL sur un domaine donné (port 443 par défaut). Si le certificat est valide, il renvoie la date d'expiration et un statut (Valide, Expiring Soon, Invalid). La réponse est envoyée à l'interface utilisateur pour affichage.

3. Frontend : Composant React

Le **frontend** est géré par un composant **React** qui interagit avec le backend pour afficher les informations sur les domaines et leurs certificats. Voici un extrait du fichier **App.js** :

```
function App() {
  const checkCertificate = useCallback((domain, index) => {
    axios.get(`/check-ssl?domain=${domain.fqdn}`)
      .then(response => {
        updateDomainData(index, response.data);
      })
      .catch(error => {
        updateDomainData(index, { hasValidCertificate: false, validUntil: 'Error fetching data', certificateStatus: 'Invalid' });
      });
  }, []);

  useEffect(() => {
    const fetchDomains = () => {
      axios.get('/domains')
        .then(response => {
          setDomains(response.data);
          response.data.forEach((domain, index) => checkCertificate(domain, index));
        })
        .catch(error => console.error('Error fetching domains:', error));
    };
    fetchDomains();
  }, [checkCertificate]);

  return (
    <div>
      {isLoading ? <div>Loading...</div> : (
        <div>
          <input type="text" placeholder="FQDN" value={newDomain.fqdn} onChange={(e) => setNewDomain({ ...newDomain, fqdn: e.target.value })} />
          <input type="text" placeholder="Private IP" value={newDomain.privateIp} onChange={(e) => setNewDomain({ ...newDomain, privateIp: e.target.value })} />
          <input type="text" placeholder="Owner" value={newDomain.owner} onChange={(e) => setNewDomain({ ...newDomain, owner: e.target.value })} />
          <button onClick={handleAddDomain}>Add Domain</button>

          <table>
            <thead>
              <tr>
                <th>FQDN</th>
                <th>Private IP</th>
                <th>Owner</th>
                <th>Certificate Status</th>
              </tr>
            </thead>
            <tbody>
              {domains.map((domain, index) => (
                <tr key={index}>
                  <td>{domain.fqdn}</td>
                  <td>{domain.privateIp}</td>
                  <td>{domain.owner}</td>
                  <td>{domain.certificate.valid} (Expires: {domain.certificate.expires})</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
    </div>
  );
}
```

Explication :

L'interface permet à l'utilisateur d'ajouter un **nouveau domaine**, d'afficher tous les domaines surveillés, et de vérifier automatiquement l'état de leur certificat SSL. Le statut du certificat (valide, expirant bientôt, invalide) est affiché dans un tableau.

4. Fonctionnalité de vérification SSL et affichage visuel

Chaque domaine vérifié reçoit un statut visuel en fonction de l'état de son certificat SSL. Les domaines dont le certificat expire bientôt ou est invalide sont signalés avec des couleurs différentes pour une meilleure visibilité.

Algorithme de vérification des certificats :

Le backend utilise la fonction **sslChecker** pour vérifier la validité du certificat d'un domaine, puis retourne un statut et la date d'expiration. Ces informations sont envoyées au **frontend** pour être affichées.

```
const checkSsl = async (domain) => {
  try {
    let result = await sslChecker(domain, { method: 'GET', port: 443 });
    // Logique de vérification et retour des informations
  } catch (error) {
    console.error(`Error checking SSL for ${domain}:`, error);
  }
};
```

PCDN	IP	443	Owner	Aut Domain	ACTIONS	CERTIFICAT STATUS
PCDN	IP	PORT	OWNER			
myssl.teamwork.net	80.80.228.228	443	Team R&D		Update/Delete	Valid (Expires 2024-07-25T08:33:00Z)
myssl-backup.teamwork.net	80.197.44.159	5070	Team R&D		Update/Delete	Valid (Expires 2024-07-25T08:43:00Z)
myssl-us.teamwork.net	104.53.16.182	443	Team R&D		Update/Delete	Valid (Expires 2024-09-04T10:26:10:00Z)
myssl-gcp.teamwork.net	10.150.147.54	443	Fabien PESSEMISSE - Thomas FERRAZ		Update/Delete	Invalid (Expires Unknown)
myssl-eu.teamwork.net	10.150.151.7	443	Maxime HIAS		Update/Delete	Valid (Expires 2024-05-14T09:19:48:00Z)
myssl-nor.teamwork.net	10.150.148.49	443	Dolphane Cassar		Update/Delete	Valid (Expires 2024-09-10T11:05:50:00Z)
myssl-bell.teamwork.net	10.230.10.287	443	Xavier Bellère		Update/Delete	Valid (Expires 2024-07-24T10:15:06:00Z)
myssl-bell-gcp.teamwork.net	10.230.10.287	443	Xavier Bellère		Update/Delete	Valid (Expires 2024-07-24T10:15:06:00Z)
myssl-gcp.teamwork.net	10.230.16.61	443	Cédric DAVEMANS IV		Update/Delete	Valid (Expires 2024-07-29T12:00:16:00Z)
myssl-nor.teamwork.net	10.230.16.61	443	Cédric DAVEMANS IV		Update/Delete	Valid (Expires 2024-07-29T12:00:16:00Z)
myssl-gcp.teamwork.net	10.230.16.61	443	Cédric DAVEMANS IV		Update/Delete	Valid (Expires 2024-07-29T12:00:16:00Z)
myssl-gcp.teamwork.net	10.230.16.61	443	Cédric DAVEMANS IV		Update/Delete	Valid (Expires 2024-07-29T12:00:16:00Z)
myssl-gcp.teamwork.net	10.230.16.61	443	Cédric DAVEMANS IV		Update/Delete	Valid (Expires 2024-07-29T12:00:16:00Z)
myssl-gcp.teamwork.net	10.230.16.61	443	Cédric DAVEMANS IV		Update/Delete	Valid (Expires 2024-07-29T12:00:16:00Z)

5. Gestion des erreurs et des mises à jour

Le site permet également à l'utilisateur de **mettre à jour les informations** (propriétaire du domaine) ou de **supprimer** un domaine du tableau. Chaque opération (ajout, modification, suppression) déclenche une mise à jour côté serveur et une vérification du certificat SSL.

```
const handleUpdate = (index) => {
  const updatedOwner = prompt("Enter new owner name:", domains[index].owner);
  if (updatedOwner !== null) {
    axios.put(`/domains/${index}`, { ...domains[index], owner: updatedOwner });
  }
};

const handleDelete = (index) => {
  axios.delete(`/domains/${index}`)
    .then(() => setDomains(domains.filter((_, i) => i !== index)));
};
```

6. Conclusion technique du projet

Ce projet a permis de mettre en place une solution **simple et efficace** pour surveiller les certificats SSL des services internes de l'entreprise. L'utilisation de **React pour le frontend** et de **Node.js pour le backend** a facilité la gestion des interactions avec les certificats SSL et le stockage des domaines.

Le site est désormais utilisé en interne pour surveiller la validité des certificats et avertir les responsables lorsque des certificats approchent de leur expiration, avec un nom de domaine avec un certificat SSL évidemment.

