Rapport projet application web : N7WS

Réalisé par :

- BESSEL Théo L
- TRUONG Nell L
- SOYER Benjamin A
- FALLOT Ysabel A

Introduction

Ce projet s'inspire de l'application AWS (Amazon Web Services) et a pour objectif de permettre le lancement à distance de services sur des machines de l'ENSSEIHT via des scripts. En complément, l'application permet d'obtenir des informations système (CPU, RAM) sur ces machines.

Fonctionnement

Pages et fonctionnalités

Pages publiques

- **Page d'accueil** : permet d'accéder à la page de connexion.
- **Page de login** : permet à l'utilisateur de se connecter pour accéder aux pages sécurisées. Elle contient également un lien vers la page d'inscription.
- **Page d'inscription (register)** : permet de créer un compte utilisateur. Une fois l'inscription effectuée, l'utilisateur est automatiquement redirigé vers la page de connexion.

Pages sécurisées

Toutes les pages sécurisées partagent une structure commune (voir section *Détails sur le front*), permettant l'accès à un menu de navigation, l'affichage du nom et prénom de l'utilisateur connecté, et la possibilité de se déconnecter (ce qui redirige vers la page de login).

- **Page dashboard** : page d'accueil après connexion. Elle affiche toutes les machines

regroupées par salle. Pour chaque machine, les composants (processeur, carte graphique) sont listés. Lorsqu'une machine est sélectionnée, des graphiques temps réel d'utilisation CPU et RAM sont affichés.

- **Page scripts** : permet d'ajouter un script bash via un formulaire (nom + chemin sur la machine cible) et d'afficher la liste des scripts existants.
- **Page services** : permet d'ajouter un service via un formulaire (nom, port, scripts associés) et de consulter la liste des services définis ainsi que les scripts liés à chacun d'eux.

Cas d'utilisation

Cas 1 — Encadrant de TP

Un encadrant souhaite lancer un serveur web sur chaque machine d'une salle avant le début d'une séance de TP. Plutôt que de le faire manuellement, il utilise N7WS pour automatiser cette tâche.

Il crée un script bash pour le lancement du serveur, le lie à un service, puis déclenche ce service depuis l'interface graphique. Chaque machine exécutera alors le script associé.

Cas 2 — Élève en recherche de ressources

Un élève souhaite effectuer des calculs sur une machine peu utilisée. Il consulte la liste des machines et leurs charges CPU/RAM en temps réel pour choisir la plus disponible et ainsi maximiser les performances de ses calculs.

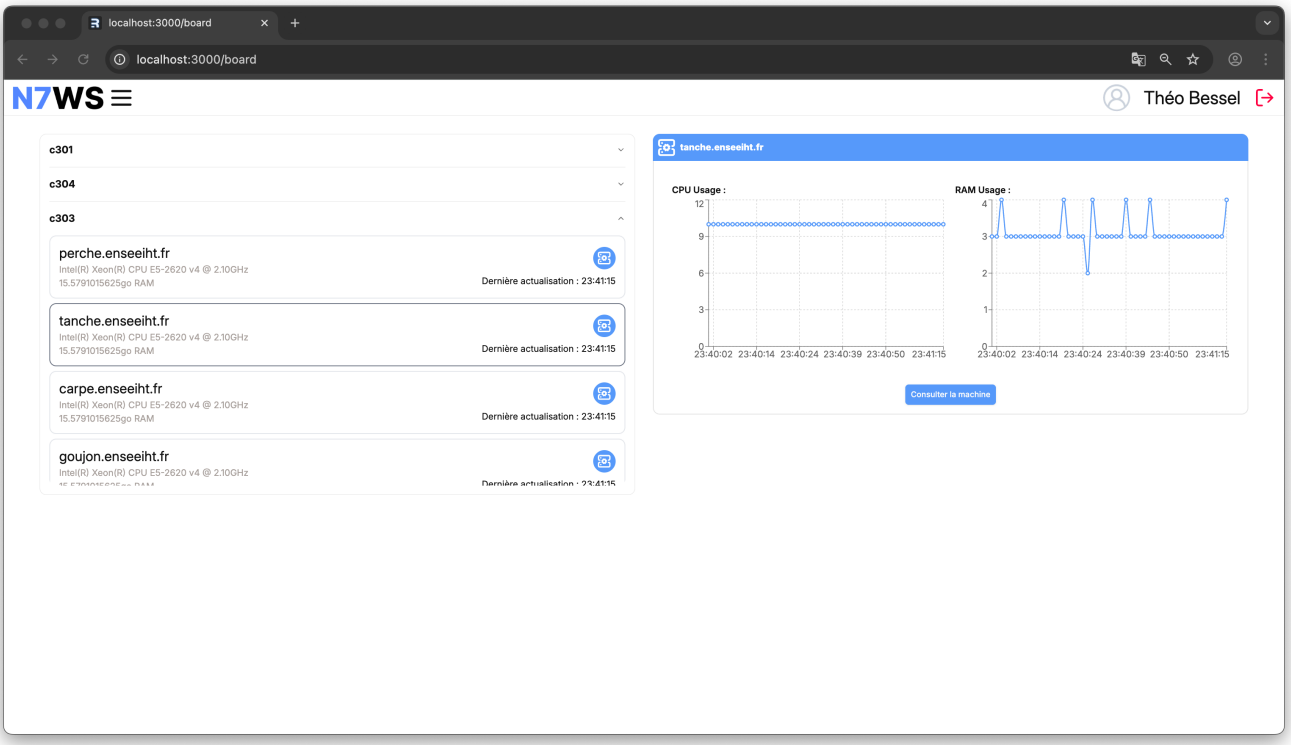
Détails techniques

Stack utilisée

Frontend

Le frontend est développé avec **React**, en utilisant **Remix**, un framework concurrent de Next.js. La bibliothèque **Zod** est utilisée pour la validation de schémas en TypeScript, garantissant que les données envoyées aux APIs sont conformes aux attentes (par exemple, éviter l'envoi de données de mauvais type via un formulaire).

Les graphiques d'utilisation CPU/RAM sont affichés en temps réel grâce à une **WebSocket**.

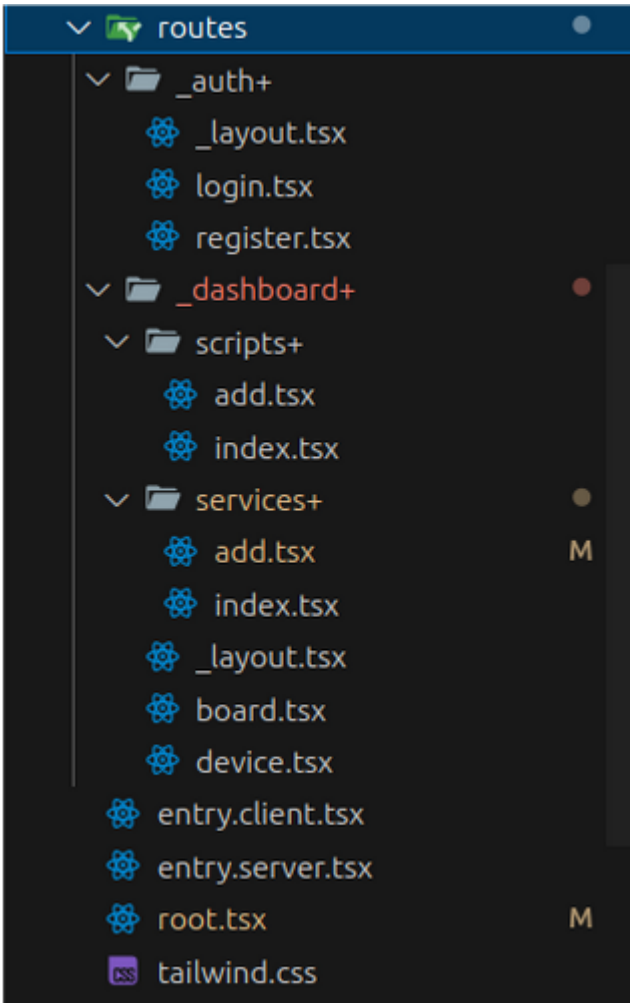


Backend

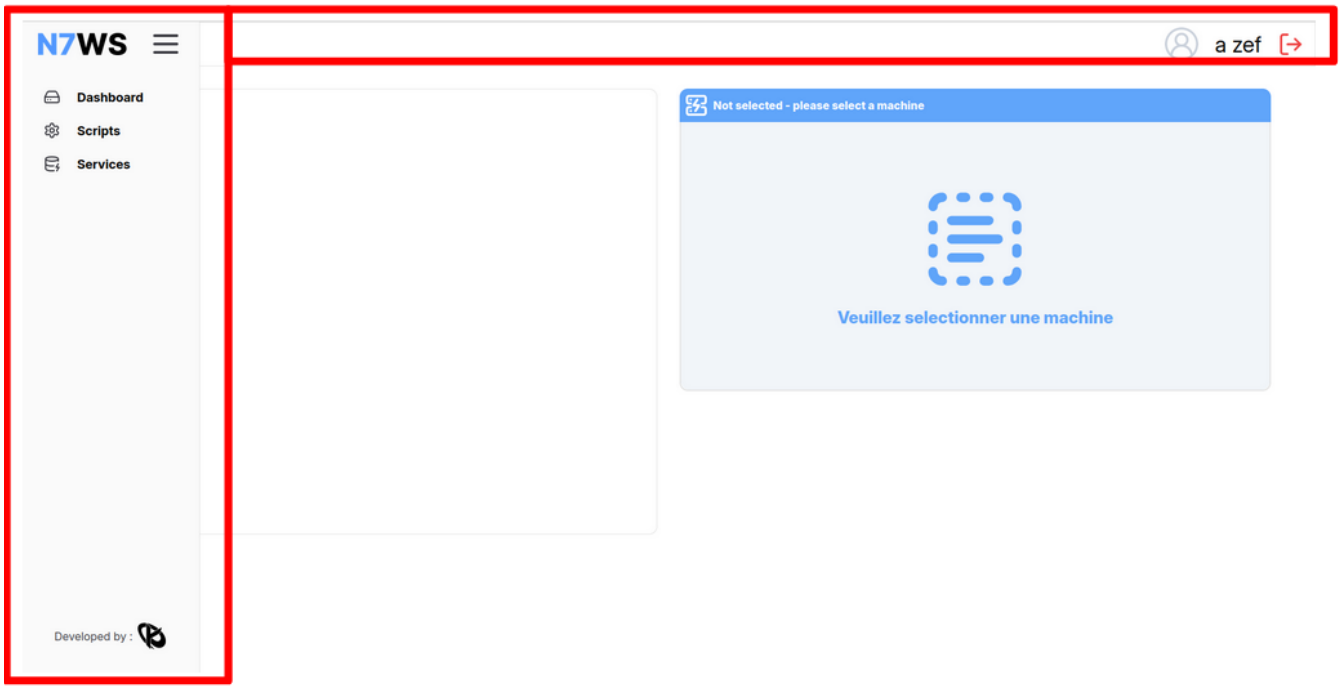
Le backend repose sur **Spring Boot** et **Spring Security** pour la gestion des tokens JWT et l'accès sécurisé aux routes protégées (voir section *Spring Security et JWT*).

Détails sur le front

Grâce à Remix, le routage est simple : un fichier TypeScript correspond à une route.



Chaque fichier correspond à une page décrite précédemment (cf. Pages et fonctionnalités), à l'exception des fichiers `layout`. Ces derniers permettent de factoriser le code commun à toutes les pages sécurisées :



Le contenu encadré en rouge est partagé entre toutes les pages sécurisées. Cela permet d'éviter la redondance de code HTML.

Détails sur le back et ses composants

Le backend suit une **architecture hexagonale**, séparant la logique métier des interfaces (API, base de données) de manière plus poussée que dans un MVC classique, facilitant le découplage.

Il comprend :

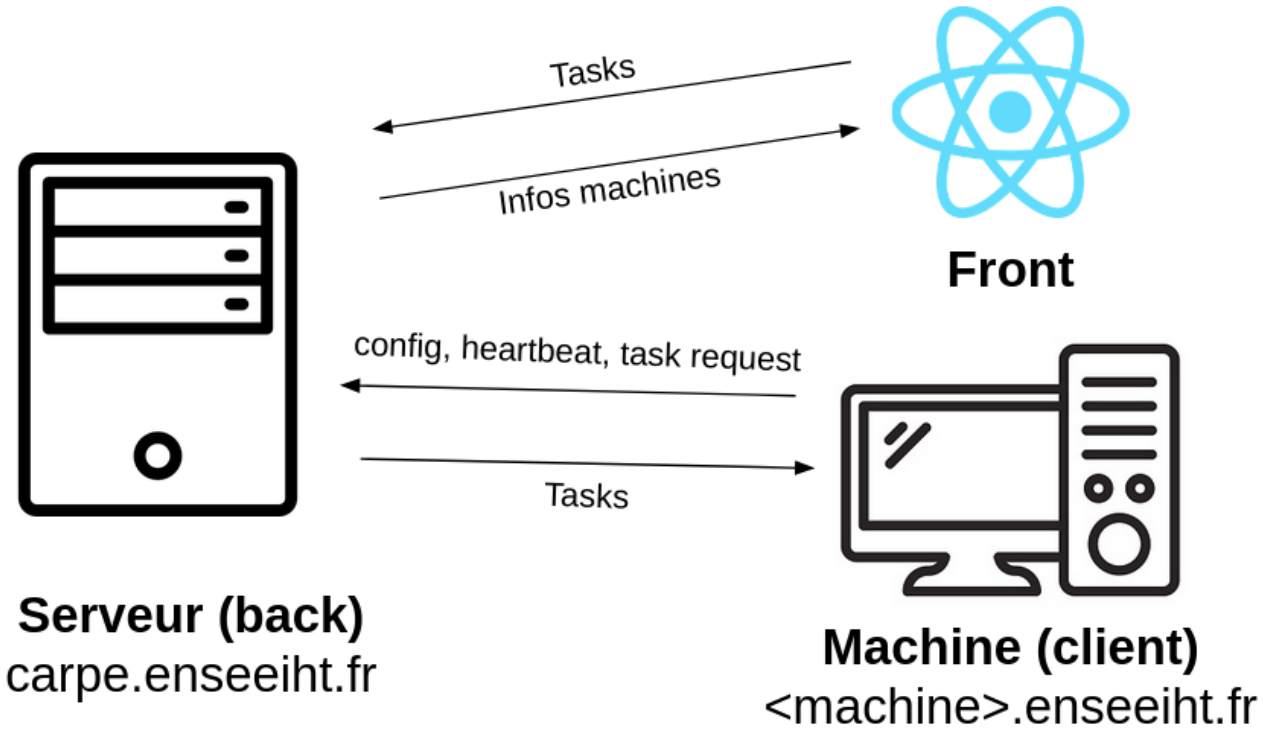
- Une **API REST** pour la création, la suppression et la modification des entités.
- Un **canal WebSocket** pour transmettre en temps réel les données de monitoring.
- Un **client HTTP Java** déployé sur chaque machine, chargé :
    - de transmettre les métriques CPU/RAM et les infos système (nom du CPU, nombre de cœurs, etc.),
    - d'exécuter des services (constitués de scripts), sur demande du serveur.

Détails sur le client et ses interactions

Le **client**, développé en Java, utilise les mêmes objets que le backend (partage de classes). Ces objets sont sérialisés en **JSON** avec la librairie **Jackson** (également utilisée côté Spring Boot).

Au moment de la première connexion, le client envoie sa configuration, enregistrée ensuite en base de données. Il envoie ensuite périodiquement (chaque seconde) les informations d'utilisation CPU/RAM.

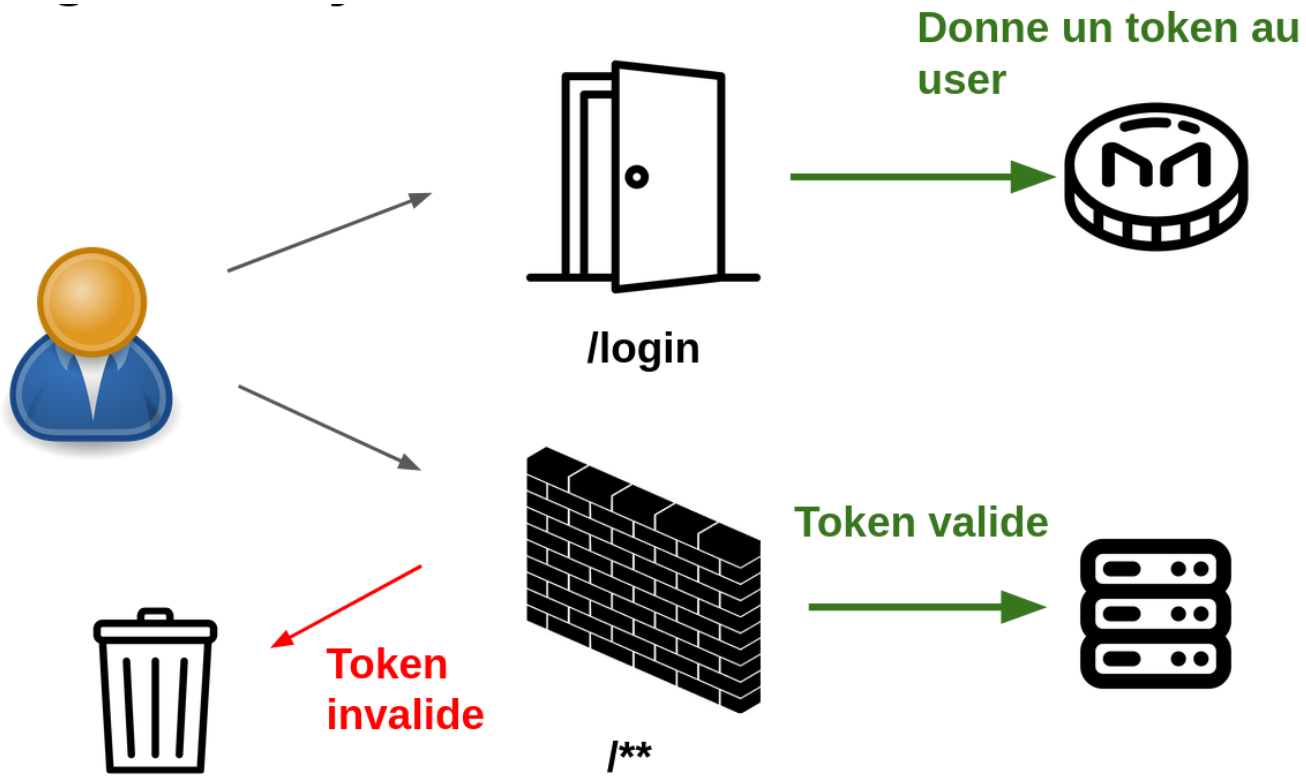
Si des tâches lui sont assignées par le serveur (via une réponse HTTP), il exécute les scripts correspondants en utilisant l'API de gestion des processus de Java.



Spring Security et JWT

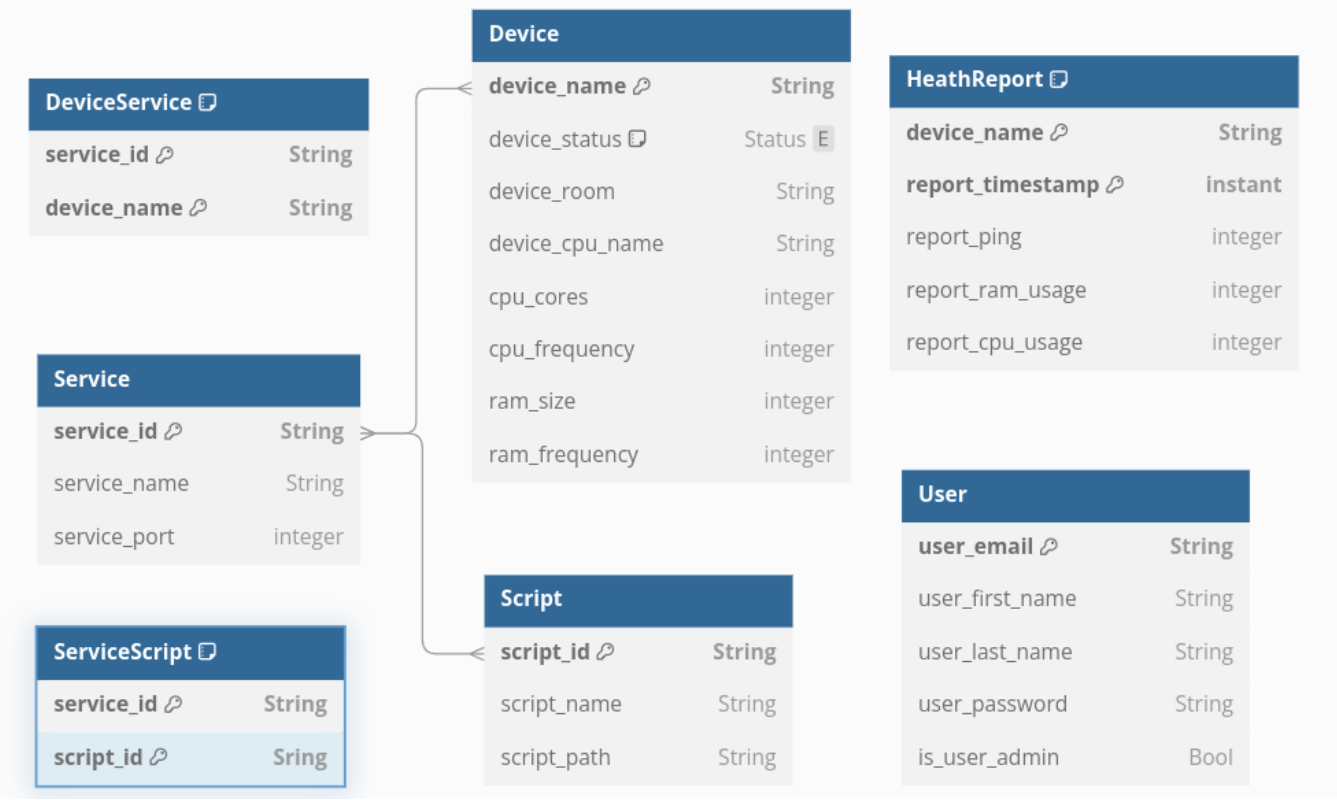
Le mécanisme de sécurité repose sur l'utilisation de **JWT (JSON Web Token)** :

- Lors du login, un token est généré et stocké dans un cookie.
- À chaque tentative d'accès à une page sécurisée, une vérification du token est effectuée.
    - Si le token est valide, l'accès est autorisé.
    - Sinon, une erreur est renvoyée et l'accès est refusé.



Base de données

Le schéma relationnel est le suivant :



La conception suit les principes de **normalisation** vus en cours, afin d'éliminer les redondances et de minimiser le nombre de jointures nécessaires lors des requêtes.

Les tables `DeviceService` et `ServiceScript` sont des tables permettant de gérer les

relations `ManyToMany` avec Spring. Dans notre processus de conception du schéma relationnel nous sommes partis de ces tables, obtenues lors de la normalisation sous 4ème forme normale, pour décider où placer les relations `@ManyToMany` . Même si ces dernières pouvaient également être obtenues de manière naïve, nous avons décidé d'opter pour une approche rigoureuse utilisant le cadre théorique de l'algèbre relationnelle.

Pistes d'amélioration

- Ajouter la possibilité de **transmettre les scripts via des sockets**, ce qui offrirait plus de flexibilité.
- Permettre la **sélection de groupes de machines** pour exécuter un service simultanément.
- Ajouter des fonctionnalités de **monitoring et d'arrêt des services** en cours d'exécution.
- Intégrer une **console SSH dans l'interface** pour exécuter des commandes directement.
- Proposer une **interface administrateur** pour ajouter de nouvelles machines en renseignant leur adresse IP (et leur transmettre directement le script client).