Présentation Architecture Projet

Au Bon Beurre

Dany CORBINEAU, Corentin DUPONT.

# Présentation du projet

**Le problème**

L’entreprise au bon beurre élabore et fabrique des produits laitiers, très sensibles aux bactéries. Des données en temps réelle sont nécessaire pour lancer des alertes en cas de suspicion de contamination. Les mesures techniques sont déjà implémentées, mais la visualisation des paramètres ne se font que sur des pupitres de commandes. Les informations ne sont aucunement centralisées.

**Le besoin**

Le besoin se définit finalement par un affichage temps réelle des données récupérées par les mesures déjà mis en place, qui sont des automates.

**La solution**

Le produit à développer est une interface web dite intranet diffusée sur des moniteurs, affichant en temps réelle les données envoyées toutes les minutes par les automates. Les automates sont regroupés par unités, d’une capacité maximum de 10. Les données s’afficheraient sous forme de graphiques, exportable en PDF. Des alertes seront accessibles sur l’interface web, et la modification des paramètres correspondant aux limites de valeurs sera tracé. Les logs devront à la demande être exportable dans le fichier PDF, à une date précise.

**Techniquement**

Le projet de développement est stocké en ligne sur GitLab. La gestion de version est donc gérée avec Git. Pour correspondre à la demande technique, une destination GitHub a été créée. Le projet est régulièrement mis à jour sur cette destination.

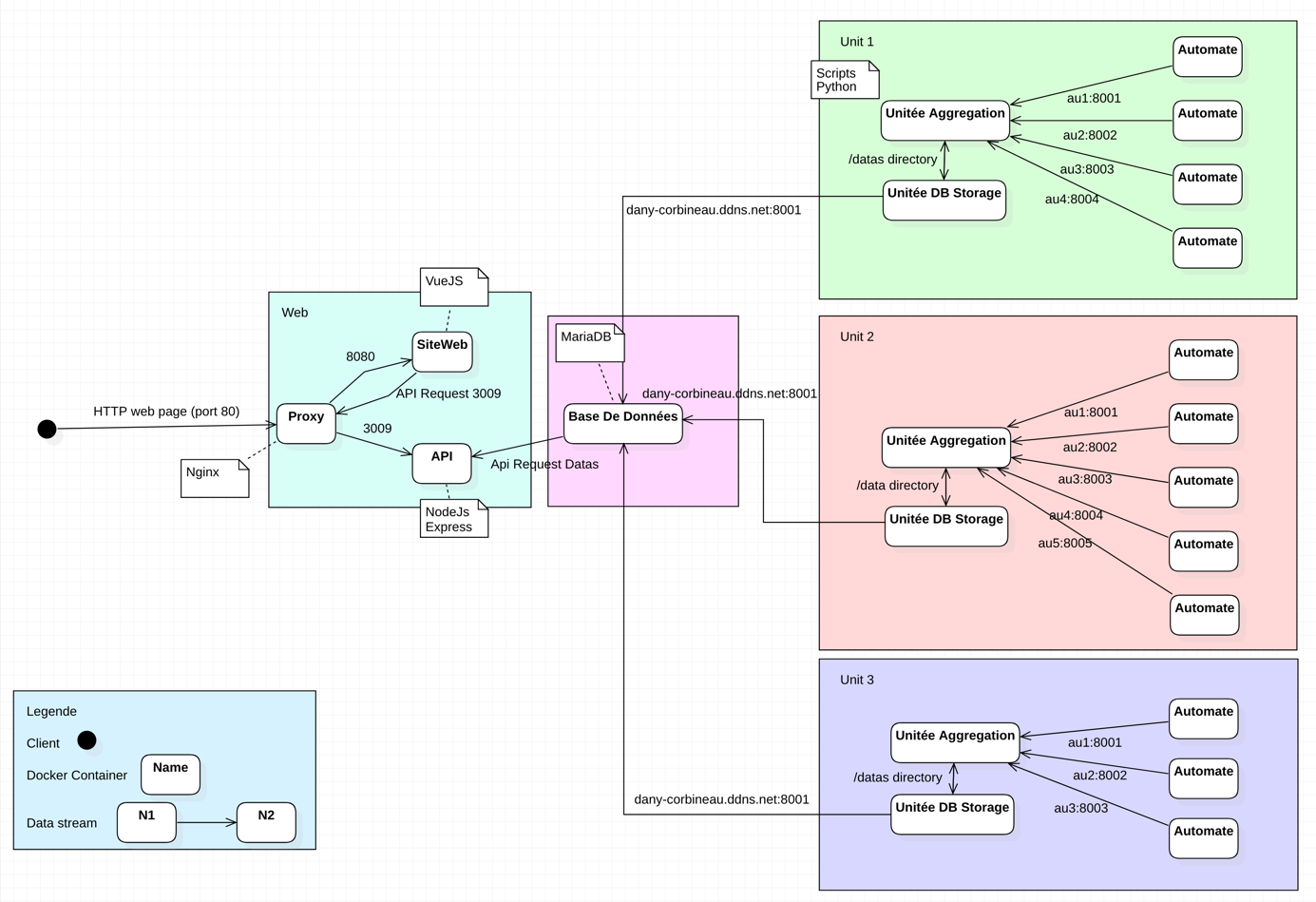
Le client utilise la technologie Vue.js, Framework de développement Front basée sur Node.js. L’affichage des graphiques est géré par la bibliothèque vue-chartjs.

La base de données utilise le moteur MariaDB utilisant le système de base de données MySQL. Elle permet de faire la passerelle entre les unités et le client. Une API Node.js avec express.js met à disposition des URLs pour récupérer les données à afficher sur le client.

Des scripts python imitent l’émission de données de la part des automates dans un premier temps. Un autre script, par unité, va récupérer les fichiers de données JSON générés par les premiers scripts pour les regrouper dans un fichier unique. Un troisième et dernier script va s’occuper de lire ces derniers fichiers pour enregistrer les données en base de données via requête MySQL.

On créer une image docker pour chaque partie du système. Tout le projet est encapsulé dans trois système docker compose (web, base de données, unités/automates).

# Architecture Système



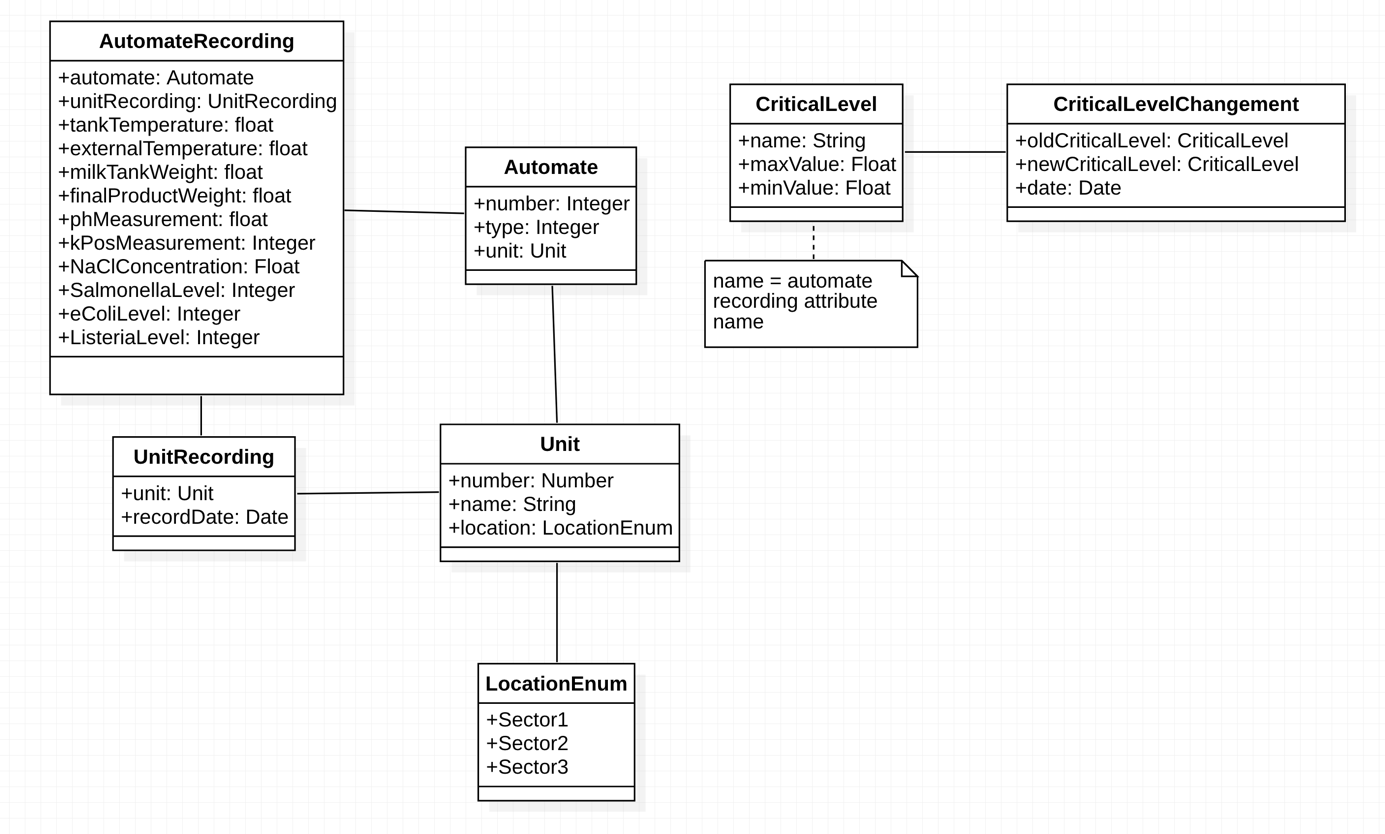
Les unités de production sont encapsulées dans un Docker (un container par unité). Ces unités se connectent à des automates via socket (présent dans un container indépendant). Les données des automates sont enregistrées en fichier dans un dossier /datas. Le dossier est partagé avec un autre container d’unité qui est chargé d’enregistrer les fichiers dans la base de données.

Un reverse proxy nginx permet de gérer un serveur web et de rediriger des flux vers l’API.

L’API se charge de récupérer les données dans la base de données. Via une URL, le client peut récupérer ainsi les données à afficher sur l’interface. Le client va exécuter des requêtes tous les x temps (ici, toutes les minutes).

Le client est une application Vue.js qui est stocké sur le nginx. Elle permet d’afficher sous forme de graphs les données des mesures techniques de la dernière heure, récupérer depuis l’API. Les graphs sont affichés en fonction de deux paramètres, le numéro d’une unité, et le numéro d’un automate. Un graph montre l’évolution d’un paramètre enregistrer par l’automate.

# Architecture de la base de données



La base de données a été construite avec une logique orienté objet. Avec cette façon de structurer, la structure de données est plus pérenne.

# Architecture des dossiers

Tout d’abord, il faut savoir qu’il y a pour chaque dossier, un fichier README.md, qui contient la documentation correspondante.

## Architecture globale

/aubonbeurre

/aubonbeurre-api

/aubonbeurre-dataviz

/nginx

/database

/script

/automate

/unit

## Description

**API (**/aubonbeurre/aubonbeurre-api**)**

Dossier racine de l’API Node.js + express.js.

|  |  |
| --- | --- |
| **Fichiers** | **Description** |
| server.js | Script d'entré pour lancer le serveur. Contient les paramètres de la connexion à la base de données. Définit les routes accessibles par l'extérieur retournant des données formatées de la base de données. |

**Interface Web (**/aubonbeurre/aubonbeurre-dataviz**)**

Regroupe tout ce qui concerne l'application web front vuejs.

|  |  |
| --- | --- |
| **Fichiers / Dossiers** | **Description** |
| src/components | Composants vue |
| src/view | Page web (data-viz + index) |
| Src/router.js | Définition des routes du front web |
| Src/public | Fichiers statiques publique |
| Src/api | Fonctions pour utiliser l’api aubonbeurre-api |

**Scripts (**/script**)**

Permet de gérer la partie création des données, agrégation des données et sauvegarde des données en base de données.

|  |  |
| --- | --- |
| **Fichiers** | **Description** |
| /automate/generator.py | Script d'entrée pour lancer un automate. Créer un serveur socket et génère les données d’un automate. |
| /automate/AutomateRecording.py | Class AutomateRecording pour donner une forme objet aux enregistrements des automates. |
| /automate/run.py | Script de lancement de plusieurs automates simultanément |
| /automate/automate.json | Fichier json de configuration pour permettre au run.py de lancer plusieurs automates |
| /unit/datas | Dossier dans lequel seront stocker les fichiers générés lors de l'agrégation |
| /unit/aggregor.py | Script pour récupérer les données des automates et les enregistrer dans un fichier. |
| /unit/automate.json | Contient les informations pour agréger les données des automates (ce fichier n'est pas obligatoire) |
| /unit/dbStorage.json | Script pour enregistrer les données des fichiers vers la base de données. |
| /unit/DockerFile | Fichier de configuration docker pour générer l'image d'une unité |