

Académie de LYON

BTS CIEL **Brevet de technicien supérieur cybersécurité, informatique et réseaux, électronique**
Option Informatique et Réseaux

Session 2025

Lycée ORT Lyon



Epreuve E62 – Projet technique

VITINI

Candidats:
Goncalves Ethan,
El Anziz Ben Said Aounou,
Molinari Bastien,
Derrag Rayane

Sommaire

Table des matières

1	Le besoin.....	5
1.1	Cahier des charges	5
1.1.1	Viticulture	5
1.1.2	Vinicuture	5
	Constraintes techniques et économiques	5
	Ressources disponibles	5
2	Analyse fonctionnelle.....	7
3	Conception générale	15
	Architecture de la solution.....	15
4	Candidat : Bastien	17
5	Candidat: El Anziz Ben Said Aounou.....	27
5.1	Étape – Tâche 1 : Analyse et personnalisation de la base de données	28
5.1.1	Objectifs	28
5.1.2	Eléments de réalisation.....	28
5.1.3	Bilan.....	29
5.2	Etape – Tache 2 : API Capteur	29
5.2.1	Objectifs	29
5.2.2	Eléments de réalisation.....	30
5.2.3	Bilan.....	32
5.3	Tache 3 : Gestion des relevés.....	33
5.3.1	Objectifs	33
5.3.2	Eléments de réalisation.....	33
5.3.3	Bilan.....	34
5.4	Tâche 4 : Gestion des utilisateurs et sécurité	34
5.4.1	Objectifs	34
5.4.2	Eléments de réalisation.....	34
5.4.3	Bilan.....	35
5.5	Tâche 5 : Gestion des sauvegardes	36
5.5.1	Objectifs	36
5.5.2	Eléments de réalisation.....	36
5.5.3	Bilan.....	37

5.6 Tâche 6 : Gestion des alertes :	37
5.6.1 Objectifs	37
5.6.2 Eléments de réalisation.....	37
5.6.3 Bilan.....	38
6 Candidat : Rayane.....	40
6.1 Etape – Tache 1	41
6.1.1 Objectifs	41
6.1.2 Eléments de réalisation.....	42
6.1.3 Bilan.....	43
6.2 Etape – Tache 2	43
6.2.1 Objectifs	43
6.2.2 Eléments de réalisations	43
6.2.3 Bilan.....	47
6.3 Cahier de suivi	48
7 Candidat : Goncalves Ethan.....	49
7.1 Etape 1 – Le panneau solaire	50
7.1.1 Contraintes.....	50
7.1.2 Choix de l'appareil.....	50
7.1.3 Bilan.....	51
7.2 Etape 2– Installation du panneau solaire.....	52
7.2.1 Objectifs	52
7.2.2 Eléments de réalisation.....	52
7.2.3 Bilan.....	53
7.3 Etape 3 – Transmettre les valeurs.....	54
7.3.1 Objectifs	54
7.3.2 Eléments de réalisation.....	54
7.3.3 Bilan.....	55
7.4 Etape 4 – Installation des programmes sur la Raspberry Pi 5.....	56
7.4.1 Objectifs	56
7.4.2 Eléments de réalisation.....	56
7.4.3 Bilan.....	58
7.5 Etape 5– Migration du code vers la Raspberry Pi 5	59
7.5.1 Objectifs	59

7.5.2	Eléments de réalisation	59
7.5.3	Bilan	60
7.6	Etape 6– Teste du pluviomètre sur la Raspberry Pi 5	61
7.6.1	Objectifs	61
7.6.2	Eléments de réalisation	61
7.6.3	Bilan	63
7.7	Etape 8– Envoi vers la base de données	64
7.7.1	Objectifs	64
7.7.2	Eléments de réalisation	64
7.7.3	Bilan	65
7.8	Etape 9 – Lancement du code a l'allumage de la Raspberry pi5	66
7.8.1	Objectifs	66
7.8.2	Eléments de réalisation	66
7.8.3	Bilan	67
8	Intégration.....	68
8.1	Partie Web (Aounou et Bastien)	68
8.2	Envoi des données via (Aounou et Ethan)	68
9	Annexes	69
9.1	Source utiliser.....	69

1 Le besoin

Le projet VITINI vise à développer un système de surveillance pour améliorer les processus de viticulture et de viniculture au Domaine Miolanne, une exploitation viticole biologique située au pied du Massif du Sancy.

1.1 Cahier des charges

1.1.1 Viticulture

Température :

Relevés de température et tableaux de bord de suivi.

Alerte en cas de gel pour permettre à l'exploitant de prendre des mesures (pulvérisation d'eau, tours antigel, etc.).

Pluviométrie :

Relevés de pluviométrie et tableaux de bord de suivi.

Alerte à partir d'un seuil paramétrable.

Pression atmosphérique :

Relevés de pression atmosphérique et tableaux de bord de suivi.

Alerte en cas d'inversion de la variation de pression.

1.1.2 Vinicuture

Fermentation :

Surveillance de la fermentation des cuves (température, masse volumique, pH, taux de CO₂).

Surveillance plusieurs fois par jour pendant au moins un mois et demi.

Stockage des bouteilles :

Surveillance de la température, du taux d'humidité et du niveau de luminosité moyen sur 24h.

Contraintes techniques et économiques

Budget maximal de 1000 €.

Surveillance extérieure sans branchement au secteur.

Utilisation de moyens de communication classiques (Wi-Fi) pour la cave.

Accès aux tableaux de bord via smartphone ou ordinateur de bureau.

Ressources disponibles

Système didactique did@LoraWan pour tests locaux.

EndDevice Lora, systèmes Raspberry, dispositifs Arduino, divers capteurs (baromètre, température, humidité).

Recherche d'une alimentation solaire pour le EndDevice et de capteurs pour la fermentation et la pluviométrie.

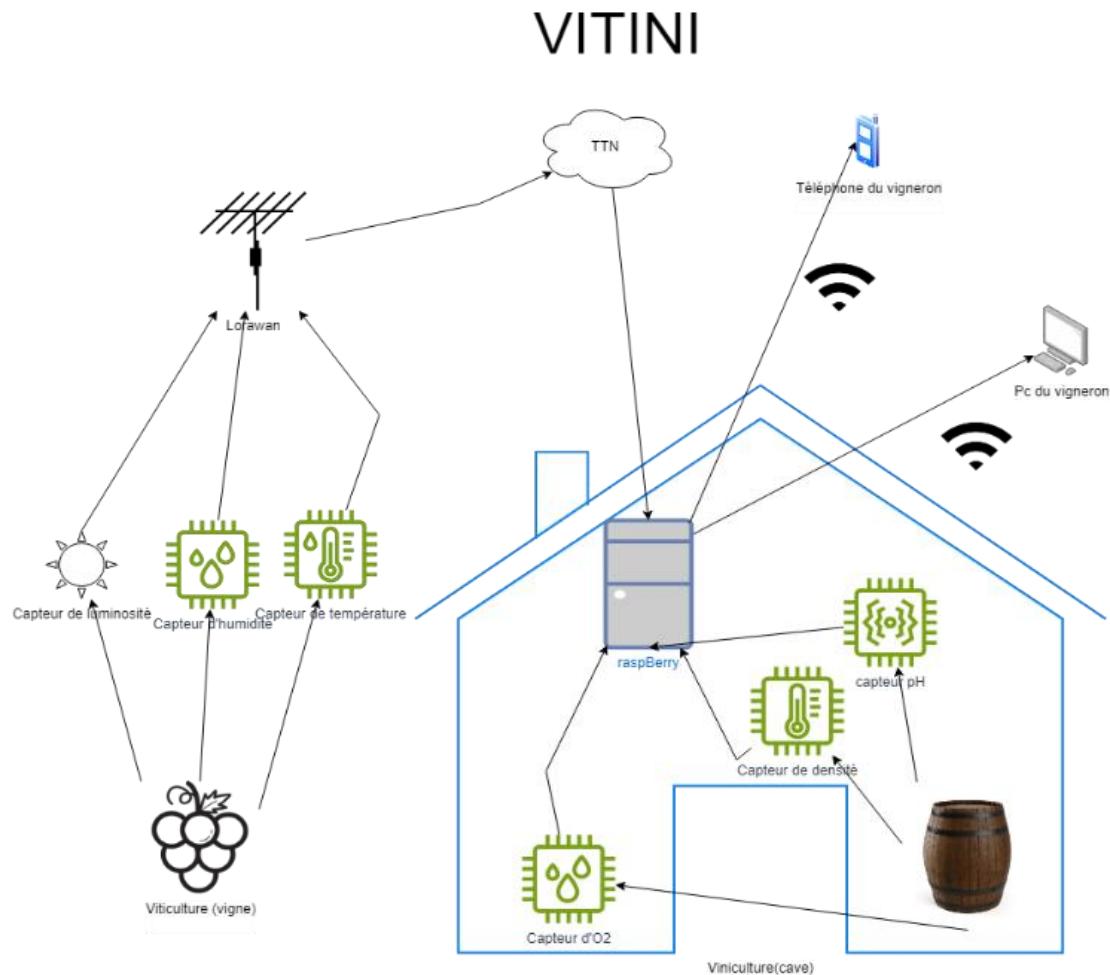


Figure: Architecture du projet

Ce plan montre visuellement à quoi va ressembler l'installation. On a deux séparations distinctes avec une partie à gauche qui est la vigne avec des capteurs qui envoient au les données au LoRaWan et on a à droite la cave qui contient des capteurs qui sont brancher à la Raspberry. Tous ces données sont stockées dans la Raspberry et sont accessible via son pc en se connectant a internet

2 Analyse fonctionnelle

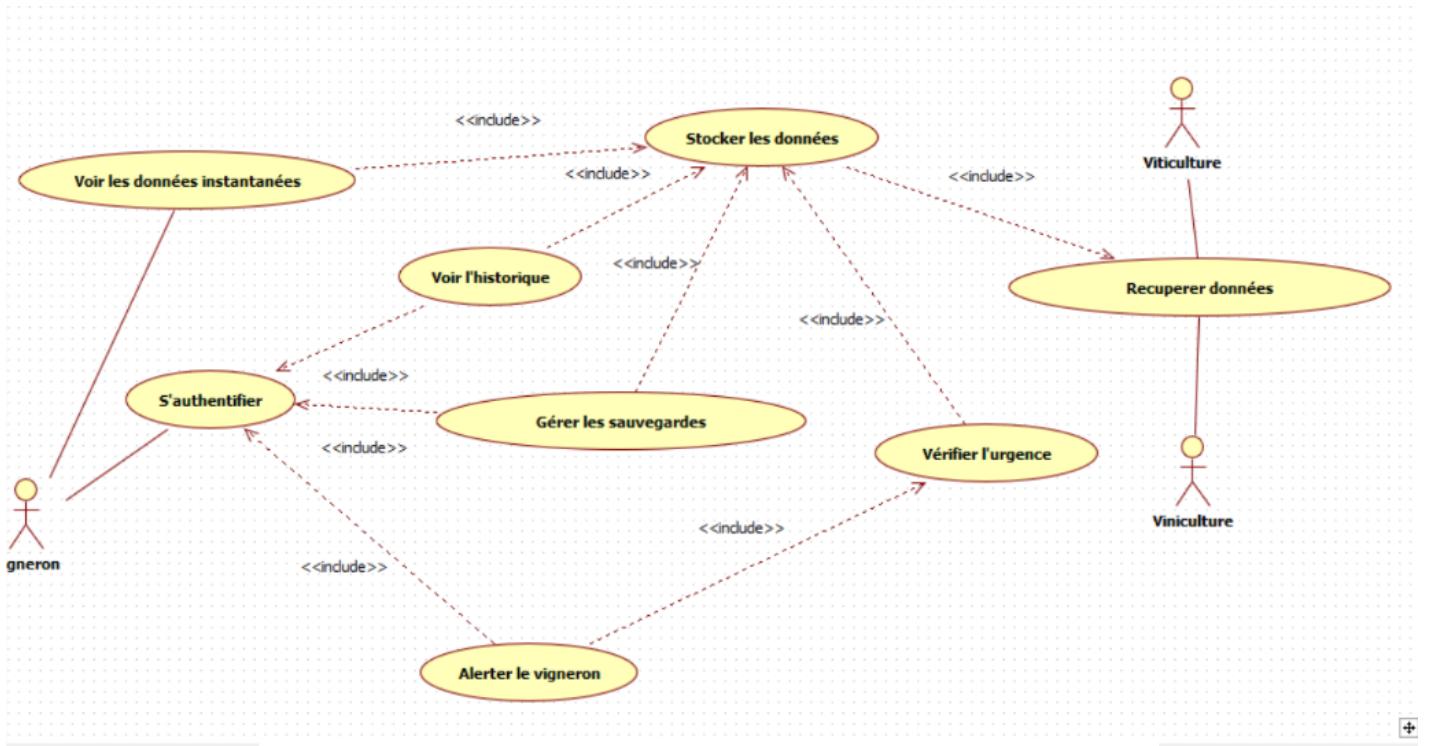


Figure: use case

Ce diagramme de cas d'utilisation illustre comment un vigneron peut utiliser un système pour gérer les données de viticulture et de viniculture. Le vigneron, acteur principal, s'authentifie pour accéder aux fonctionnalités du système. Il peut consulter des données en temps réel et l'historique des récoltes et processus de vinification. Le système permet de stocker et gérer les données, tout en vérifiant les situations d'urgence nécessitant une intervention rapide. Des alertes sont envoyées au vigneron pour l'informer des conditions anormales.

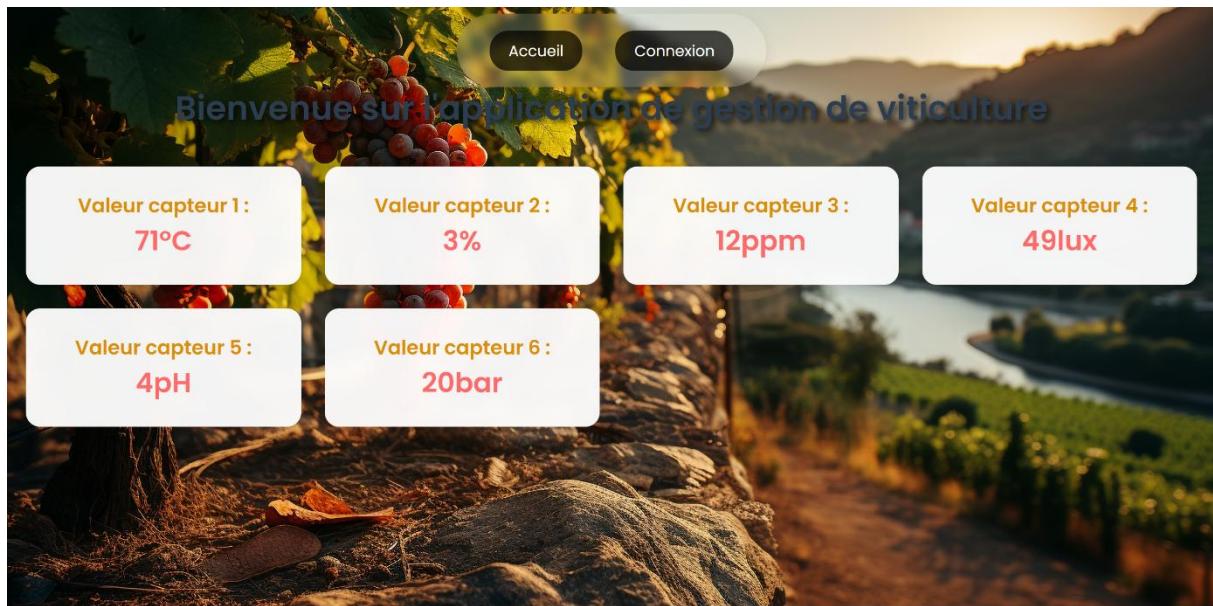


Figure 1: Page d'accueil sans être connecté

Cette page est la première vue de l'application pour un utilisateur non authentifié. Elle permet de présenter le projet et d'orienter l'utilisateur vers les pages de connexion ou d'inscription. L'accès aux autres fonctionnalités est restreint tant que l'utilisateur n'est pas connecté.

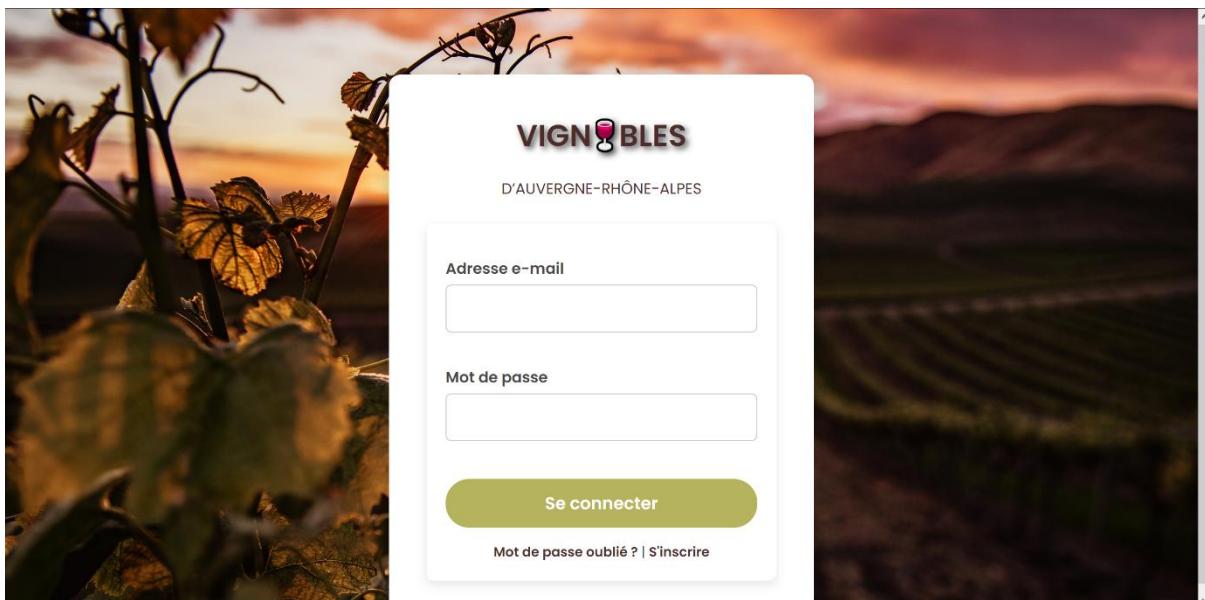


Figure 2: Page pour se connecter avec ses identifiants

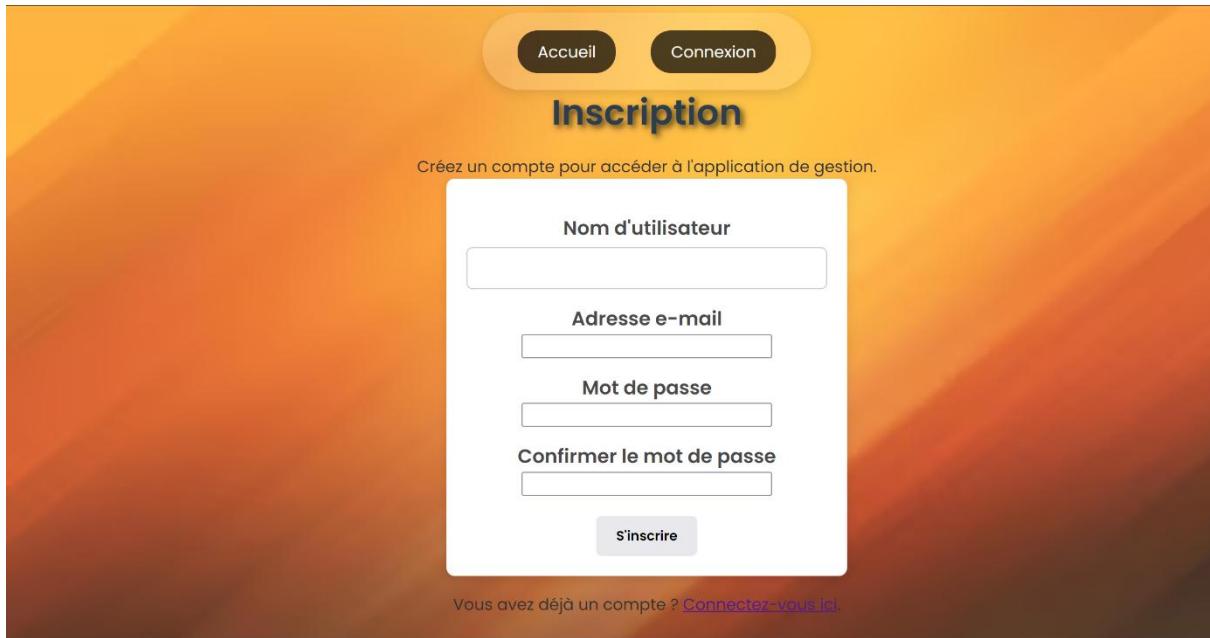


Figure 3: Page pour se créer un compte

Ces deux pages regroupent l'interface de connexion pour les utilisateurs existants et un formulaire d'inscription pour les nouveaux utilisateurs. Elle est reliée à une API qui permet d'enregistrer un compte ou de vérifier les identifiants dans la base de données. C'est une étape obligatoire pour accéder aux fonctionnalités de l'application.

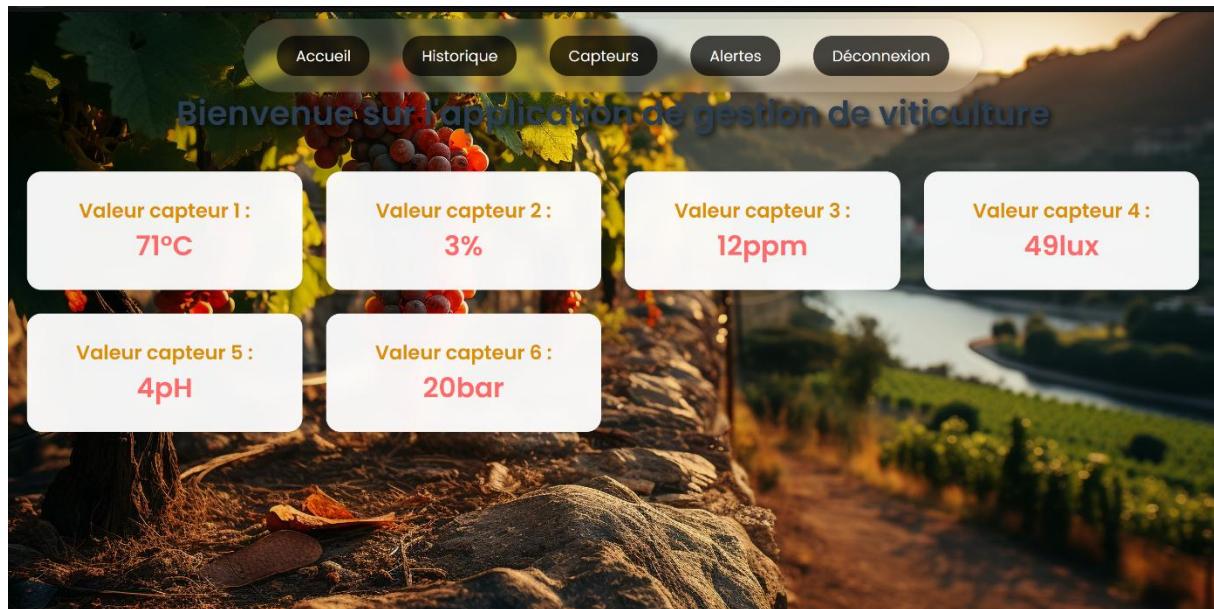


Figure 4: Page d'accueil quand on est connecté

Cette page regroupe l'interface de connexion pour les utilisateurs existants et un formulaire d'inscription pour les nouveaux utilisateurs. Elle est reliée à une API qui permet d'enregistrer un compte ou de vérifier les identifiants dans la base de données. C'est une étape obligatoire pour accéder aux fonctionnalités de l'application.

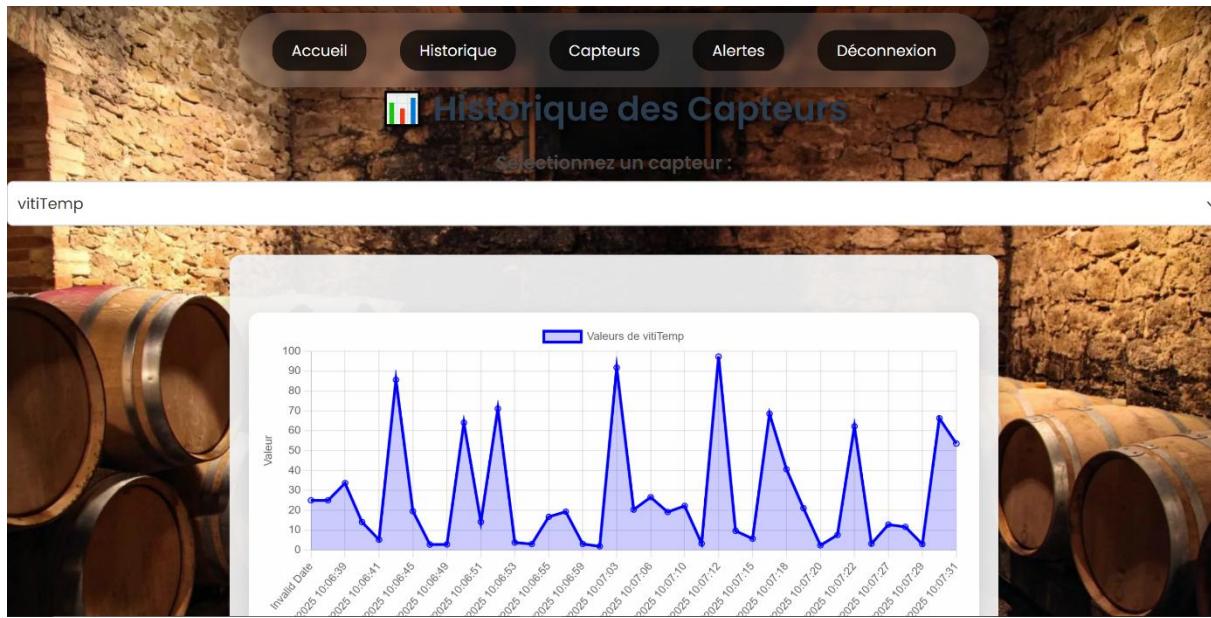


Figure 5: Page des graphiques des données

Cette page donne accès à l'historique des données sous forme de graphique, avec la possibilité de filtrer par période (jour, semaine, mois, année).

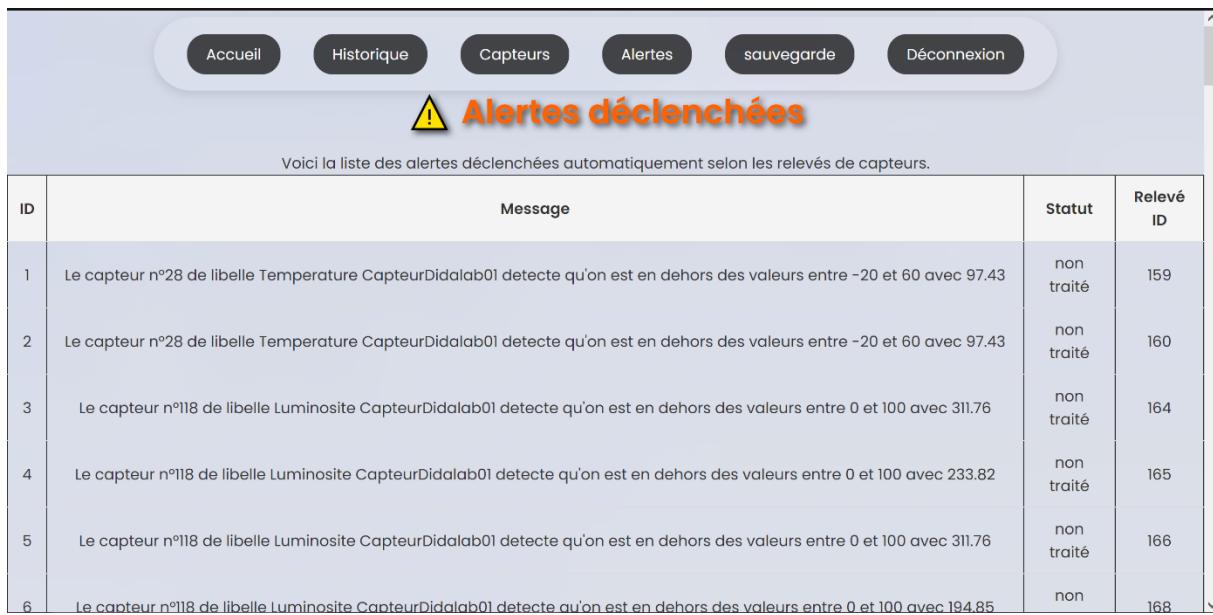
Figure 6: Page de modification des capteurs de la base de données

ID	Libellé	Type	Sueil Minimum	Sueil Maximum	Unité	Localisation	État	Date installation
1	vitiTemp	température	-10	25	°C	Vignes	actif	2025-05-14 15:31:00
2	vitiPluv	pluviométrie	10	100	mm	Vignes	inactif	
3	pression	pression	0	50	bar	viti	actif	2025-05-14 10:34:00
4	fermTemp	température	0	30	°C	Fermentation	actif	
5	fermMV	masse volumique	0	10000	g/L	Fermentation	actif	
6	caveTemp	température	0	30	°C	Cave	actif	
7	cavelumi	luminosité	0	100	lux	Cave	actif	2025-01-13 16:02:59
8	fermPH	ph	0	14	pH	Fermentation	actif	2025-01-13 16:02:59

Figure 7: Page qui liste tous les capteurs de la base de données

Cette page permet à l'utilisateur de consulter et de modifier les caractéristiques des capteurs. Elle est connectée à une API pour récupérer les capteurs et modifier leurs paramètres. La liste complète des capteurs est affichée dans un tableau pour une meilleure lisibilité.

Figure 8: Page qui permet de gérer les capteurs



The screenshot shows a web interface titled "Alertes déclenchées" (Triggered Alerts). At the top, there are navigation buttons: Accueil, Historique, Capteurs, Alertes, sauvegarde, and Déconnexion. Below the title, a message reads: "Voici la liste des alertes déclenchées automatiquement selon les relevés de capteurs." A table lists six triggered alerts:

ID	Message	Statut	Relevé ID
1	Le capteur n°28 de libelle Temperature CapteurDidalab01 detecte qu'on est en dehors des valeurs entre -20 et 60 avec 97.43	non traité	159
2	Le capteur n°28 de libelle Temperature CapteurDidalab01 detecte qu'on est en dehors des valeurs entre -20 et 60 avec 97.43	non traité	160
3	Le capteur n°118 de libelle Luminosite CapteurDidalab01 detecte qu'on est en dehors des valeurs entre 0 et 100 avec 311.76	non traité	164
4	Le capteur n°118 de libelle Luminosite CapteurDidalab01 detecte qu'on est en dehors des valeurs entre 0 et 100 avec 233.82	non traité	165
5	Le capteur n°118 de libelle Luminosite CapteurDidalab01 detecte qu'on est en dehors des valeurs entre 0 et 100 avec 311.76	non traité	166
6	Le capteur n°118 de libelle Luminosite CapteurDidalab01 detecte qu'on est en dehors des valeurs entre 0 et 100 avec 194.85	non	168

Figure 9: Page qui permet d'afficher toutes les alertes

Cette page sert à afficher les alertes lorsqu'un capteur dépasse les seuils définis. L'utilisateur peut filtrer les alertes par capteur grâce à une liste déroulante. Les alertes sont récupérées depuis une API et présentées sous forme de tableau pour permettre un suivi rapide et clair.

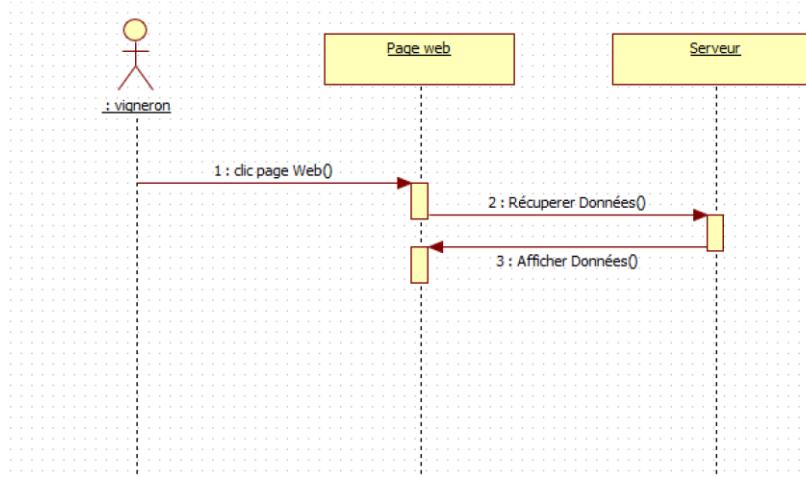


Figure 10 : Diagramme de séquence qui montre l'accès aux données du site

Ce diagramme démontre que le vigneron va sur la page web, le site web récupère les données au serveur et le serveur affiche les données sur le site web.

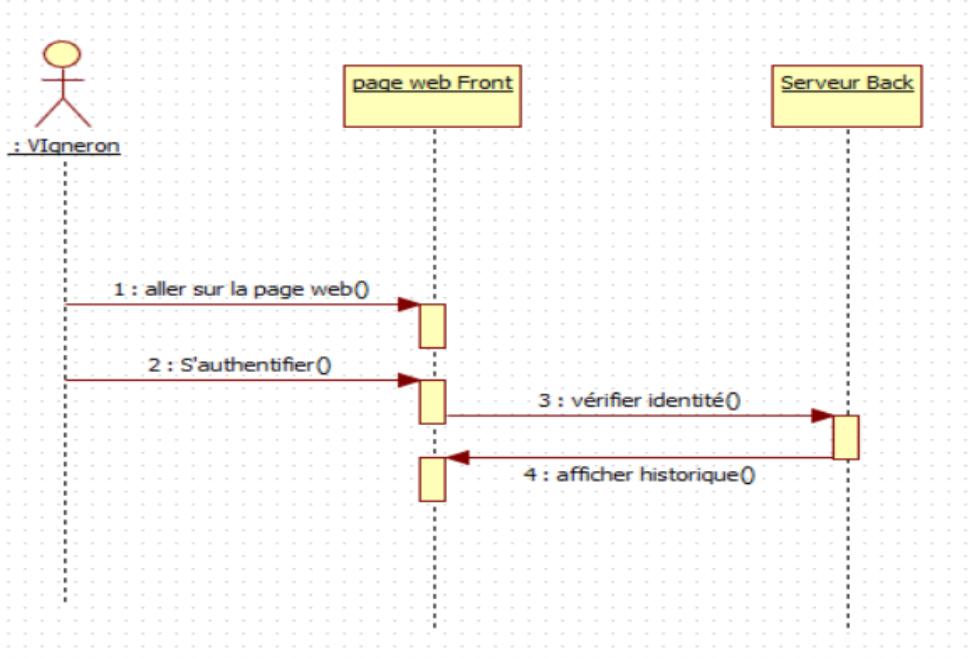


Figure 11: Diagramme de séquence qui montre l'accès avec l'authentification

Ce diagramme démontre le vigneron qui se connecte à la page web, il va s'identifier, le serveur regarde si le compte est bien enregistré et si oui il affiche l'historique sur la page web.

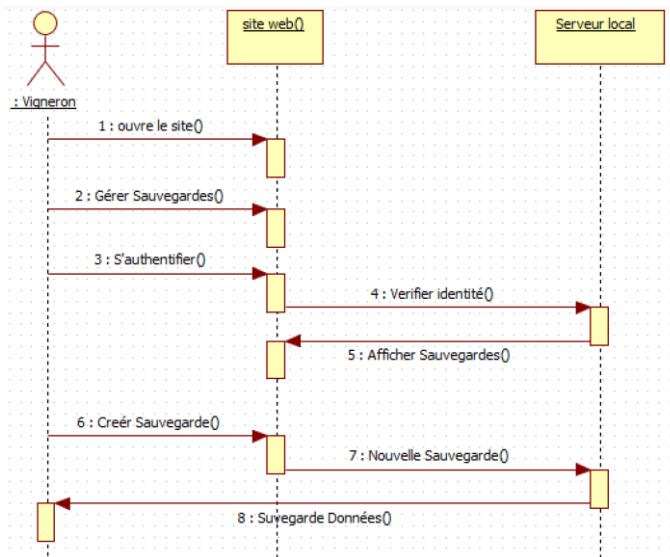


Figure 12: Diagramme de séquence qui montre la sauvegarde des données

Ce diagramme démontre le vigneron qui va sur le site web, veut aller sur l'option sauvegarder les données, il doit s'identifier le serveur vérifie et confirme l'identifiant, il crée une nouvelle sauvegarde et le serveur web télécharge les données sur l'ordinateur du vigneron.

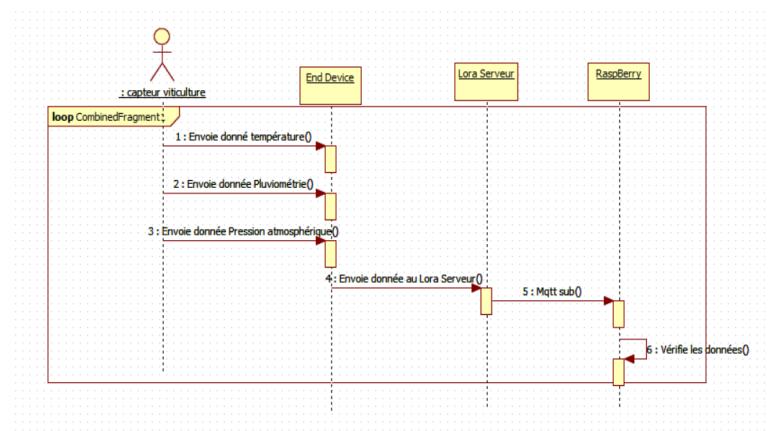


Figure 13 : Diagramme de séquence des capteurs dans la viticulture

Ce diagramme démontre le processus des capteurs de la viticulture où il envoie les différentes données captées par l'end Device, puis il les envoie vers le Lora serveur, qui l'envoie en mqtt sub à la Raspberry qui vérifie les données des capteurs. Ce processus tourne en boucle tous les 1h par exemple.

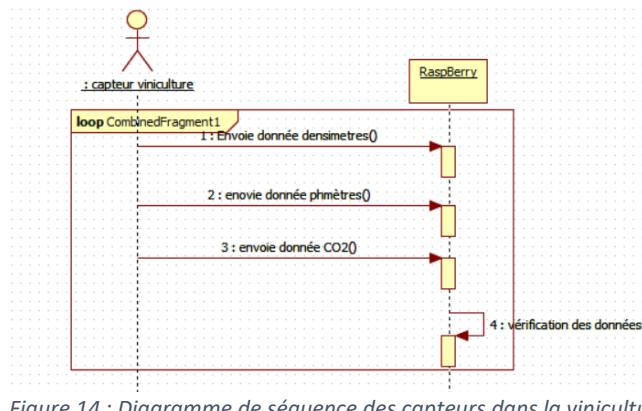


Figure 14 : Diagramme de séquence des capteurs dans la viniculture

Ce diagramme démontre le processus des capteurs de la viniculture où il envoie les différentes données captées par des capteurs branchées directement à la Raspberry qui reçois les données, ces données sont vérifiées. Ce processus tourne en boucle tous les 1h par exemple.

3 Conception générale

Architecture de la solution

Vignes : EndDevice Lora alimenté par panneau solaire, relié aux capteurs.

Cave : Systèmes Raspberry pour relever les conditions ambiantes et les processus de fermentation, avec un serveur web relié à Internet.

Bureau : PC du vigneron pour exploiter les tableaux de bord et stocker les données.

On a besoin de différent appareil que nous n'avons pas déjà à notre disposition. Pour la viticulture on aura besoin d'un panneau solaire qui va être installé sur l'EndDevice ainsi qu'un Pluviomètre

Pour la viniculture on va avoir besoin de différent capteur comme la sonde ph, la sonde de température et la sonde de pression.

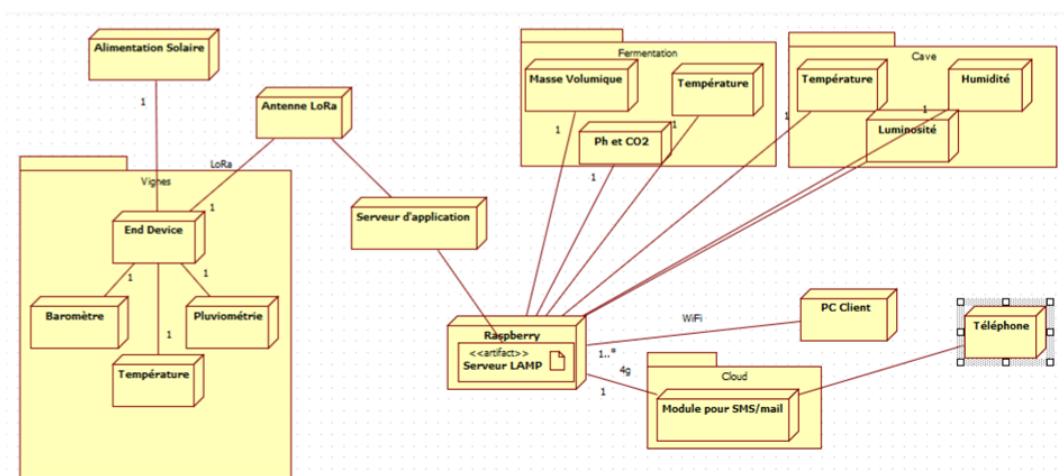


Figure 15: Déploiement version développement

Tâche	Janvier	Février	Mars	Avril	Mai	Juin																
	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8	S 9	S 10	S 11	S 12	S 13	S 14	S 15	S 16	S 17	S 18	S 19	S 20	S 21	S 22
Générale																						
Découverte du sujet																						
Diagramme/schéma																						
Bastien																						
BDD																						
IHM																						
arborescence																						
intégration de l'API																						
Aounou																						
Base de données																						
API Capteur																						
API Relevé																						
API Utilisateur																						
API Sauvegarde																						
API Alerte																						
Intégration																						
Éthan																						
Choisir l'alimentation																						
Transmettre les valeurs																						
Installation des programmes sur la Raspberry Pi 5																						
Migration du code vers la Raspberry Pi 5																						
Teste du pluviomètre sur la Raspberry Pi 5																						
Envoie vers la base de données																						
Mise en place du panneau solaire																						
Lancement du code à l'allumage de la Raspberry pi5																						
Ajout du pluviomètre à l'EndDevice																						
Rayane																						
Choix des capteurs																						
Cablage																						
Pilotage des capteurs																						
Envoie des données																						

Figure 16: Planning général et les différentes tâches des étudiants

Etudiant	Fonctions à développer et tâches à effectuer
E1 : Molinari Bastien	<p>BDD : Conception et réalisation de la base de données Entités et repositories</p> <p>IHM : Structure des pages du site et authentification Tableaux de bord</p>
E2 : El anziz ben said Aounou	<p>Peuplement : Développer les modules d'interfaces permettant de récupérer les mesures et de les enregistrer dans la base de données</p> <p>Alertes : Développer les modules permettant d'alerter le vigneron (mail et/ou sms)</p> <p>Sauvegardes : Mettre en place une solution permettant d'exporter et d'importer sur le PC de bureau toutes les mesures pour une période définie.</p>
E3 : Goncalves Ethan	<p>Sur EndDevice Lora</p> <p>Choix de l'alimentation solaire : Déterminer les caractéristiques du besoin énergétique du End Device Etudier les différentes solutions et choisir le dispositif Installer l'alimentation solaire sur le End Device</p> <p>Pilotage des capteurs : Développer le pilotage des capteurs permettant de mesurer les données atmosphériques de la vigne</p> <p>Remontée des mesures sur le serveur : Développer le module permettant de transmettre les mesures au serveur</p>
E4 : Derrag Rayane	<p>Sur Raspberry</p> <p>Choix des capteurs spécifiques : Faire une étude comparative sur les densimètres, les phpmètres et les capteurs de co2. Selon le tarif, faire un choix.</p> <p>Pilotage des capteurs : Développer le pilotage des capteurs permettant de mesurer les conditions ambiantes de la cave</p> <p>Remontée des mesures sur le serveur : Développer le module permettant de transmettre les mesures au serveur</p>

4 Candidat : Bastien

Dans le cadre du projet VITINI, je suis chargé de concevoir et développer l'ensemble du site web du vigneron dans lequel il pourra visualiser les données des capteurs en temps réel, voir l'historique de celle-ci, ajouter, modifier, supprimer des capteurs, et ajouter des alertes.

- La conception et gestion de la base de données,
- L'implémentation du frontend (interface utilisateur),
- La création de l'IHM (expérience utilisateur et ergonomie),
- La mise en place des fonctionnalités de gestion (authentification, gestion des utilisateurs, alertes, capteurs, historique, sauvegarde.).
- L'intégration des apis

Bastien												
BDD												
IHM												
arborescence												
intégration de l'API												

Tache	Description	Validée ?
Conception de la base de données	Modélisation des tables et relations nécessaires pour la gestion des capteurs et des utilisateurs.	<input checked="" type="checkbox"/> OK
Développement du frontend	Création des pages d'interface utilisateur (tableau de bord, gestion des capteurs, historique des données).	<input checked="" type="checkbox"/> OK
Tests et corrections	Vérification des fonctionnalités et correction des éventuels bugs.	<input checked="" type="checkbox"/> OK
Affichage des données des capteurs	Récupération et affichage en temps réel des mesures des capteurs via des requêtes API.	<input checked="" type="checkbox"/> OK
Gestion des capteurs	Ajout, modification et suppression des capteurs via l'interface utilisateur.	<input checked="" type="checkbox"/> OK
Mise en place des alertes	Développement du système permettant d'envoyer des alertes lorsque certaines valeurs seuils sont dépassées.	<input checked="" type="checkbox"/> OK
Déploiement	Mise en ligne du site et configuration du serveur.	<input checked="" type="checkbox"/> OK

1. Prise en main de l'environnement de développement

Technologies utilisées : *PHP, MySQL, VS Code, Git*

- Installation des outils nécessaires (*WampServer pour le serveur local, VS Code comme éditeur de code*).
- Configuration de Git pour le suivi des versions du projet.

2. Conception de la base de données

Technologies utilisées : *MySQL, SQL*

- Modélisation de la base de données avec MySQL, en définissant les tables (capteurs, utilisateurs, alertes, relevé, sauvegarde).
 - Création des relations entre les tables pour structurer les informations des capteurs et des utilisateurs.
 - Écriture des requêtes SQL pour insérer, modifier et récupérer les données.
-

3. Développement Frontend et Interface Utilisateur

Technologies utilisées : *HTML, CSS, JavaScript, Chart.js*

- Tableau de bord interactif affichant les données des capteurs.
 - Utilisation de Chart.js pour afficher des graphiques dynamiques des relevés des capteurs.
 - Amélioration de la navigation avec du CSS et JavaScript.
-

4. Système d'alertes et gestion des seuils critiques

Technologies utilisées : *PHP, CSS*

- Développement d'un système d'alertes en cas de dépassement de seuil (*exemple : température trop élevée*).

5. Intégration d'API

Utilisation d'une API REST (*api_capteur.php*) pour :

- Créer un capteur
- Modifier un capteur
- Afficher tous les capteurs

Historique des capteurs :

Création d'une interface graphique utilisant Chart.js permettant de visualiser les valeurs des capteurs sous forme de graphique dynamique.

Les données sont récupérées via l'API *api_releve.php*.

Possibilité de sélectionner un capteur dans une liste déroulante pour afficher son historique en fonction de sa période (jour, semaine, mois, année, tout afficher)

Ajout de valeurs capteurs via *ajouter_releves.php* :

Script PHP automatisé qui simule l'envoi de valeurs réalistes de capteurs à l'API *api_releve.php*.

Chaque capteur génère aléatoirement des données entre un minimum et un maximum, avec un horodatage.

50 relevés envoyés à intervalle régulier via une boucle PHP.

Intégration API – Historique (historique.php) :

Liaison avec l'API api_capteur.php pour récupérer dynamiquement la liste des capteurs.

Liaison avec l'API api_releve.php pour filtrer les données par capteur_id et afficher l'historique correspondant sur un graphique.

Intégration API – Alertes (alertes.php)

Requête GET sur api_alerte.php pour **afficher les alertes actives**.

Système de filtrage par capteur via liste déroulante (liée à api_capteur.php).

Les alertes sont automatiquement **déclenchées par dépassement de seuil** dans les relevés.

Affichage des messages, statuts et capteur lié.

Intégration API – Utilisateurs

Inscription via POST dans api_utilisateur.php.

Connexion via PUT dans api_utilisateur_mdp.php, avec vérification du mot de passe haché.

Gestion CRUD possible des utilisateurs via API.

Intégration API – Sauvegardes (sauvegarde.php)

Requête GET sur api_sauvegarde.php pour **afficher l'historique des sauvegardes**.

Bouton de téléchargement déclenchant un POST via api_utilisation_sauvegarde_post.php, générant une sauvegarde.

Affichage dynamique de la liste avec type, date, utilisateur et nom de fichier.

6. Tests, corrections et optimisation

- Tests unitaires sur les fonctionnalités principales (*ajout de capteurs, affichage des données*).
 - Correction des erreurs et optimisation du code PHP
1. **Objectif global :**

L'objectif principal est de concevoir et développer l'interface utilisateur du site web du vigneron, lui permettant de visualiser les données des capteurs, gérer leur affichage et améliorer l'expérience utilisateur.

Découpage des tâches

I. Conception et intégration de l'IHM

Objectif : Créer une interface intuitive, ergonomique et moderne pour le vigneron.

Sous-tâches :

1. Maquettage du site
2. Intégration du design en HTML/CSS
3. Création de la page se connecter(login.php)
4. Création de la page s'inscrire (register.php)
5. Création de la page de gestion d'accueil (index.php)

6. Création de la page historique (dashboard.php)
7. Création de la page capteurs (capteurs.php)
8. Création de la page alertes (alertes.php)
9. Création de la page sauvegarde (sauvegardes.php)
10. Création de la page deconnexion (logout.php)
11. Stylisation avancée avec CSS et Javascript (*Tableaux interactifs, formulaires ect...*)
12. Ajout d'un sélecteur dynamique pour modifier un capteur facilement (*JavaScript*)
13. Amélioration de l'expérience utilisateur (*formulaire amélioré, graphique tableau de navigation*)

Langages utilisés : PHP, CSS, Javascript

2. Affichage dynamique des données des capteurs

Objectif : Afficher en temps réel les valeurs relevées par les capteurs de manière claire et compréhensible.

Sous-tâches :

2. Intégration des valeurs des capteurs dans un tableau dynamique (*Affichage propre et organisé des données en HTML/CSS*)
3. Création d'un graphique dynamique pour l'historique des capteurs (*Chart.js*)
4. Mise en place du sélecteur de période (jours, semaines, mois, années, tout afficher) (*JavaScript*)
5. Amélioration de l'accessibilité (tailles de texte, navigation amélioré)

Langages utilisés : PHP, CSS, JavaScript (Chart.js)

3. Interaction utilisateur et animations

Objectif : Rendre l'interface interactive et fluide pour améliorer la navigation.

Sous-tâches :

1. Ajout d'effets de transition et d'animations CSS
2. Affichage conditionnel du boutons Modifier (*JavaScript*)

Langages utilisés : PHP, CSS, JavaScript

Bilan :

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Page d'accueil (index.php)	PHP, CSS, JavaScript	Présenter le projet et guider l'utilisateur	Design clair et informatif	<input checked="" type="checkbox"/> OK
Formulaire d'inscription (inscription.php)	PHP, MySQL, CSS	Permettre aux vignerons de s'inscrire	Inscription enregistrée en BDD	<input checked="" type="checkbox"/> OK
Connexion (connexion.php)	PHP, MySQL, CSS	Authentifier l'utilisateur	Connexion sécurisée	<input checked="" type="checkbox"/> OK
Déconnexion (deconnexion.php)	PHP, MySQL	Déconnecter l'utilisateur	Session détruite, redirection	<input checked="" type="checkbox"/> OK
Affichage de L'historique	CSS, JavaScript	Vérifier que les capteurs s'affichent correctement dans un graphique	graphique visible et lisible	<input checked="" type="checkbox"/> OK
Gestion des capteurs (capteurs.php)	PHP, MySQL, JavaScript, CSS	Ajouter, modifier, seuil max, seuil min	Fonctionnalités dynamiques et interactives	<input checked="" type="checkbox"/> OK
Affichage du graphique	Chart.js	Vérifier que le graphique charge bien les données	Courbe correcte	<input checked="" type="checkbox"/> OK
Base de données	MySQL	Structurer les données des capteurs et des alertes	Tables bien organisées, relations fonctionnelles	<input checked="" type="checkbox"/> OK
Sélecteur de capteur	JavaScript	Sélectionner un capteur pour modification	Fonctionne bien, MAJ immédiate du formulaire	<input checked="" type="checkbox"/> OK
Sélecteur de période (jours, semaines, mois, années)	JavaScript	Modifier l'affichage du graphique en fonction de la période	Fonctionne bien avec des données simulées	<input checked="" type="checkbox"/> OK
Page Alertes (alertes.php)	PHP, MySQL, CSS, JS	afficher des alertes	Gestion dynamique des alertes	<input checked="" type="checkbox"/> OK
Intégration API page capteurs.php	PHP, JS – Interaction avec API REST capteurs (GET, POST, PUT, DELETE)	Synchronisation avec la BDD via API	Liste des capteurs affichée, modifiable, suppression possible	<input checked="" type="checkbox"/> OK

Intégration API page historique	PHP, JS – Récupération des données via API releves.php	Graphique alimenté dynamiquement depuis la base	Graphique mis à jour automatiquement selon le capteur choisi	<input checked="" type="checkbox"/> OK
Page d'accueil (index.php)	PHP, CSS, JavaScript, intégration API	Présenter le projet et afficher les valeurs des capteurs en temps réel via api_releve.php	Données des capteurs s'affichent en temps réel dans les cadres "Viticulture" et "Viniculture"	<input checked="" type="checkbox"/> OK
Formulaire d'inscription (register.php)	PHP, MySQL, CSS, API utilisateur (api_utilisateur.php)	Permettre aux vignerons de s'inscrire et enregistrer les données dans la base via API	Inscription enregistrée via requête POST API	<input checked="" type="checkbox"/> OK
Connexion utilisateur (login.php)	PHP, CSS, API de connexion (api_utilisateur_mdp.php)	Authentification via vérification du mot de passe haché	Connexion réussie, redirection sécurisée	
Sauvegardes (sauvegarde.php)	PHP, CSS, JavaScript, API sauvegarde (api_sauvegarde.php)	Afficher les sauvegardes et permettre le téléchargement via API	Liste dynamique affichée et fichier téléchargeable	<input checked="" type="checkbox"/> OK
Script ajouter_releves.php	PHP – Génération automatique de relevés aléatoires pour tests	Relevés insérés en base, exploitables sur le graphique	Relevés insérés avec date, valeur et capteur liés correctement	<input checked="" type="checkbox"/> OK

Tableau des tests unitaires par fonctionnalité :

Nom du test	Technologie utilisée	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Accès restreint aux pages sécurisées	PHP, Sessions	Tester l'accès à dashboard.php sans connexion	dashboard.php sans connexion Redirection vers connexion.php	<input checked="" type="checkbox"/> OK	OUI
Ajout d'un capteur	PHP, MySQL	Ajouter un capteur en BDD	Capteur ajouté et affiché	<input checked="" type="checkbox"/> OK	OUI
Modification d'un capteur	PHP, MySQL	Modifier un capteur existant	Données mises à jour	<input checked="" type="checkbox"/> OK	OUI
Affichage du tableau des capteurs	JavaScript, CSS	Vérifier l'affichage dynamique	Liste bien affichée	<input checked="" type="checkbox"/> OK	OUI
Ajout d'une alerte	PHP, MySQL	Ajouter une alerte	Alerte bien enregistrée dans la BDD	<input checked="" type="checkbox"/> OK	OUI
Suppression d'une alerte	PHP, MySQL	Supprimer une alerte	Alerte supprimée	<input checked="" type="checkbox"/> OK	OUI
Affichage des alertes	PHP, MySQL	Vérifier que les alertes s'affichent bien	Alertes bien affichées	<input checked="" type="checkbox"/> OK	OUI
Mise à jour en temps réel du graphique	Chart.js, JavaScript	Vérifier que les données des capteurs s'actualisent	Courbe mise à jour	<input checked="" type="checkbox"/> OK	OUI
Connexion à la base de données	PHP, MySQL	Vérifier que la connexion à la BDD fonctionne	Connexion réussie	<input checked="" type="checkbox"/> OK	OUI
Inscription d'un utilisateur	PHP, MySQL	Ajouter un nouvel utilisateur dans la BDD	Utilisateur bien ajouté	<input checked="" type="checkbox"/> OK	OUI
Connexion d'un utilisateur	PHP, MySQL	Vérifier l'authentification avec mot de passe	Redirection vers dashboard.php	<input checked="" type="checkbox"/> OK	OUI

Déconnexion d'un utilisateur	PHP, Session PHP	Vérifier la destruction de la session	Redirection vers index.php	<input checked="" type="checkbox"/> OK	OUI
Intégration API – capteurs.php	PHP, API REST, JS	Utiliser l'API capteur (GET, POST, PUT, DELETE) via page capteurs.php	Interaction dynamique : création, modif., suppression OK	<input checked="" type="checkbox"/> OK	OUI
Intégration API – historique.php	PHP, JS, Chart.js	Représenter l'historique des valeurs capteurs via l'API des relevés	Graphique dynamique selon le capteur et la période sélectionné	<input checked="" type="checkbox"/> OK	OUI
Ajout de relevés via script (ajouter_releves.php)	PHP	Générer des données de capteurs automatiquement via API	Valeurs insérées dans la base, visibles dans le graphique	<input checked="" type="checkbox"/> OK	OUI
Authentification sécurisée des utilisateurs	PHP, API REST, MySQL, BCRYPT	Lors de l'inscription ou de la connexion, les identifiants sont enregistrés et le mot de passe est chiffré dans la base utilisateur	Utilisateur enregistré avec mot de passe haché, session créée à la connexion	<input checked="" type="checkbox"/> OK	OUI

Retour d'expérience :

Expérience utilisateur fluide (navigation intuitive, design moderne)

Bonne structuration des données (base de données optimisée)

Gestion complète des capteurs et alertes

Affichage dynamique des capteurs et des alertes

Graphiques interactifs et lisibles avec Chart.js

Semaine du	Tache	Résultat	Remarque
Semaine 1	Découverte du sujet	Compréhension du projet VITINI et définition des besoins	Approfondir la documentation technique
Semaine 2-3	Création de la base de données	Tables créées pour stocker les capteurs et utilisateurs	
Semaine 4	Conception de l'arborescence du site	Plan détaillé des pages et navigation validée	Ajustements possibles selon l'évolution
Semaine 5	Développement de la page d'accueil avec menu de navigation	Page responsive avec accès aux principales fonctionnalités	Design à améliorer après retours
Semaine 6	Création de la page de connexion	Formulaire fonctionnel avec vérification des identifiants	
Semaine 7	Développement des pages "Alerte" et "Historique"	Interface utilisateur et affichage des alertes opérationnels	
Semaine 8	Mise en place de la page "Capteurs"	Gestion des capteurs (ajout, modification, suppression)	Problèmes d'affichage corrigés
Semaine 9	Implémentation de la connexion en tant qu'admin	Accès sécurisé avec affichage des options avancées	
Semaine 9	Développement des pages "Déconnexion" et "Inscription"	Fonctionnalités terminées et intégrées	
Semaine 10	Ajout d'un graphique dynamique sur la page historique	Intégration de Chart.js pour visualiser les données	Améliorations possibles pour l'interactivité
Semaine 11 à 15	Gestion des capteurs via une API	Connexion entre la page capteurs.php et l'API pour affichage dynamique	Requêtes GET, POST, PUT et DELETE intégrées avec succès
Semaine 16 à 17	Intégration de l'API pour la page alertes.php	Affichage dynamique des alertes en fonction des relevés stockés	Liaison avec api_alerte.php, filtrage par capteur fonctionnel
Semaine 18	Intégration de l'API pour les utilisateurs (login.php, register.php)	Authentification reliée à la base de données utilisateur, mot de passe chiffré en BCRYPT	Sessions créées, retour API traité correctement
Semaine 19	Affichage de l'historique des capteurs via API	Graphique dynamique généré à partir des données API	Visualisation des valeurs en fonction du capteur sélectionné

Semaine 20	Intégration de l'API dans index.php pour affichage en temps réel des capteurs	Derniers relevés affichés via api_capteur.php	Rendu en direct sur l'interface d'accueil, via fetch JavaScript
------------	---	---	---

5 Candidat: El Anziz Ben Said Aounou

Introduction et présentation générale du rôle au sein du projet

En tant que membre de l'équipe, mon rôle principal dans ce projet a été de développer les fonctionnalités liées à la gestion des données (capteurs, relevés, seuils) et à la création d'une API. J'ai également participé à la conception des diagrammes, à la gestion de la base de données (BDD) et à la mise en place de tests unitaires. Mon travail s'est articulé autour de la création de fonctionnalités en Python et PHP, ainsi que de la gestion des interactions entre l'application et la BDD.

Planning individuel

Voici le planning que j'ai suivi pour mener à bien mes tâches :

Aounou	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22
Base de données	■	■	■	■	■				■						■	■						
API Capteur					■	■	■	■	■	■	■	■				■	■					
API Relevé									■	■	■	■	■			■	■					
API Utilisateur									■							■	■	■	■	■	■	
API Sauvegarde									■										■	■	■	
API Alerte									■							■	■			■	■	
Intégration									■					■	■	■	■	■	■	■	■	

Tableau des tâches

Voici un récapitulatif des tâches que j'ai réalisées :

Tâche	Description	Validée ?
Découverte du projet	Compréhension du sujet et attribution des rôles	OK
Conception des diagrammes	Création des schémas et peuplement de la BDD	OK
Fonction "exporter"	Exporter la table "relevé" en CSV	KO (hors sujet)
Fonction "nouveau relevé"	Découverte de mysql.connector et début du développement	OK
Classe Database	Création d'une classe pour se connecter à la BDD	OK
CRUD des capteurs	Implémentation des capteurs en Entity-Repository	OK
CRUD des relevés	Implémentation des relevés en Entity-Repository	OK
CRUD des utilisateurs	Implémentation des utilisateurs en Entity-Repository	OK
CRUD des sauvegardes	Implémentation des sauvegardes en Entity-Repository	OK
CRUD des alertes	Implémentation des alertes en Entity-Repository	OK
Développement de l'API	Implémentation des méthodes CRUD pour l'API	OK
Finalisation des CRUD API	Finalisation des fonctionnalités	En cours

5.1 Étape – Tâche 1 : Analyse et personnalisation de la base de données

5.1.1 Objectifs

L'objectif de cette tâche était d'analyser et apporter de potentielles améliorations sur la base de données créée par Bastien. Cette base de données devait stocker les données importantes pour les surveillances viticoles et vinicoles, ainsi que pour la web-application avec toutes les relations nécessaires.

5.1.2 Eléments de réalisation

Pour résoudre ce problème, j'ai adopté la solution suivante :

- **Description du matériel :** Pour toute la partie personnelle, j'ai utilisé un ordinateur portable équipé de Windows 11, un serveur local avec WAMP, et le langage Python avec la bibliothèque mysql.connector sur le langage de programmation Visual Studio Code. J'ai utilisé ce matériel pour la programmation durant l'intégralité du projet, je me permettrais donc de ne pas le présenter de nouveau.
- **J'ai suivi les étapes suivantes :**

Au départ, la base de données avait plusieurs tables : alerte, capteur, relevé, utilisateur, sauvegarde ainsi qu'une table seuil pour gérer les limites min/max des capteurs de manière séparée. Les relations entre tables sont explicitement définies par des clés étrangères (ALTER TABLE), assurant l'intégrité des données. Cependant, certains champs comme 'libelle' ou 'unité' sont absents. Cette version était optimisée pour l'implémentation, mais nécessite une vérification des dépendances applicatives en raison des champs manquants.

Ensuite, la base de données a été améliorée. En effet, cette nouvelle version intégrait les champs critiques (min, max, unité) directement dans la table 'capteur' permettant la simplicité avec moins de table à gérer.

Après cela, nous avons ajoutés le champs 'libelle' dans la table capteur afin de plus d'exhaustivité et de faciliter la maintenance.

Enfin, dans un objectif de débogage, tous les accents de la base de données ont été supprimés, ce qui nous donne le résultat suivant :

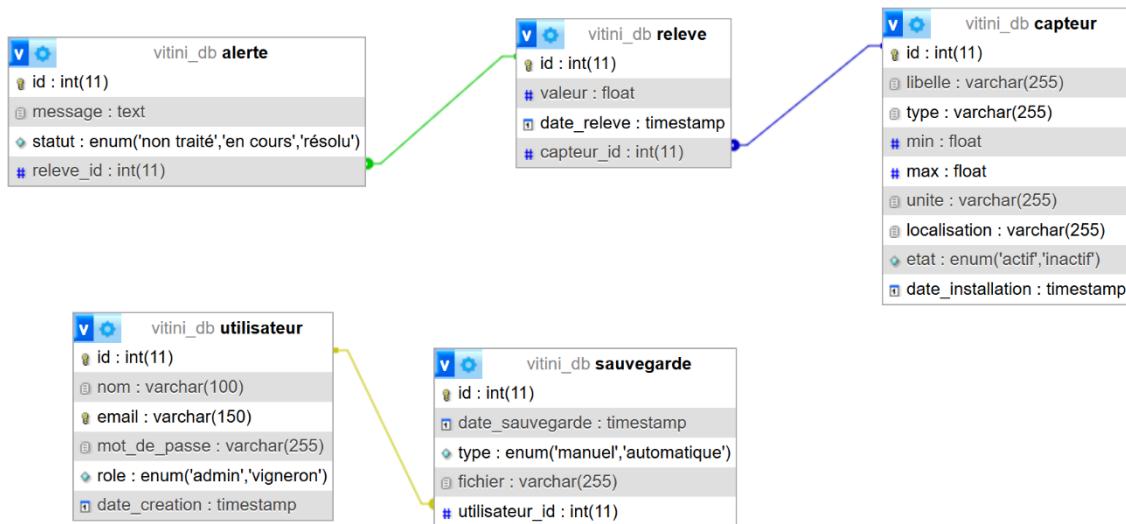


Schéma de la base de données : vitini_db.sql

5.1.3 Bilan

Semaine du	Tâche	Remarque
Semaine 1-3	Création, correction et personnalisation	La base de donnée étant déployée en local, elle était écrite sous un format qui ne correspond pas à celui final

Retour d'expérience (REX) :

Cette tâche nous a permis, à Bastien et moi à commencer le projet et ainsi mieux visualiser ce qu'on devait faire par la suite. C'était également une tâche qui nous a permis de travailler nos capacités à faire du SQL, et à en comprendre les codes, par exemple, le fait de ne pas avoir d'accents.

Ensuite, pour moi l'enjeux était d'importer des données dans la base de données, plus particulièrement des relevés.

5.2 Etape – Tache 2 : API Capteur

5.2.1 Objectifs

Après quelques semaines hors-sujets sur une tâche qui m'a permis de mieux comprendre les missions qui m'ont été données et particulièrement de modéliser mentalement celles-ci. L'objectif de cette tâche était de développer les fonctionnalités CRUD (Create, Read, Update, Delete) pour la gestion des capteurs, puis d'exposer ces fonctionnalités via une API REST. Cela incluait créer un nouveau capteur, lire les informations d'un ou plusieurs capteurs, mettre à jour les données d'un capteur existant, supprimer un capteur de la base de données, exposer ces fonctionnalités via une API pour permettre à d'autres applications d'interagir avec les capteurs.

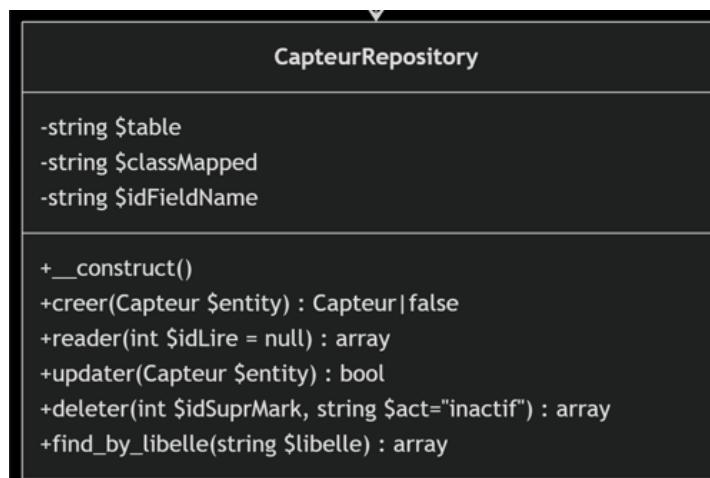
Les résultats attendus étaient une interface PHP et une API REST permettant de gérer les capteurs de manière complète et sécurisée.

5.2.2 Eléments de réalisation

1. Développement du CRUD des capteurs :

J'ai également fait en sorte d'utiliser un fichier config pour se connecter à la base de données. J'ai créé une classe Capteur (Entity) pour représenter un capteur et une classe CapteurRepository pour gérer les interactions avec la base de données.

Les méthodes CRUD (créer, reader, update, delete) ont été implémentées dans le CapteurRepository.



CapteurRepository.php

Les objets Capteur sont hydratés, c'est-à-dire remplis de données, à partir des données JSON reçues dans les requêtes HTTP.

```

Capteur

-id: int
-libelle: string
-type: string
-min: float
-max: float
-unite: string
-localisation: string
-etat: string
-date_installation: datetime
// Getters
// Setters

+__construct(array $data = null)
+hydrate(array $datas = null) : : Capteur
+__toArray() : : array
+jsonSerialize() : : array
+get(string $attribut)
+get_id() : : int
+get_libelle() : : string
+get_type() : : string
+get_min() : : float
+get_max() : : float
+get_unite() : : string
+get_localisation() : : string
+get_etat() : : string
+get_date_installation() : : datetime
+set_id(int $id) : : void
+set_libelle(string $libelle) : : void
+set_type(string $type) : : void
+set_min(float $min) : : void
+set_max(float $max) : : void
+set_unite(string $unite) : : void
+set_localisation(string $localisation) : : void
+set_etat(string $etat) : : void
+set_date_installation(datetime $date_installation) : : void

```

Capteur.php

Comme on peut le voir, les entités sont seulement composées d'attributs qui correspondent aux tables de la base de données ainsi que de setters et de getters pour chacun.

2. Création de l'API REST :

J'ai développé un script PHP (api_capteur.php) pour exposer les fonctionnalités CRUD via une API REST.

L'API supporte les méthodes HTTP suivantes :

- **POST** : Ajouter un nouveau capteur.
- **GET** : Récupérer les informations d'un ou plusieurs capteurs.
- **PUT** : Mettre à jour les données d'un capteur.
- **DELETE** : Supprimer un capteur.

L'API utilise les méthodes du CapteurRepository pour interagir avec la base de données.

Des en-têtes CORS (Cross-Origin Resource Sharing) ont été configurés pour permettre l'accès à l'API depuis n'importe quelle origine (*).

Les en-têtes Content-Type: application/json et Access-Control-Allow-Methods ont été définis pour supporter les requêtes JSON et les méthodes HTTP.

```
<?php
require '../config/config.php';

// Headers requis
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
```

Les headers dans api_capteur.php

5.2.3 Bilan

Semaine du	Tache	Remarque
Semaine 5	CRUD des capteurs	Implémentation des fonctionnalités CRUD
Semaine 6	Tests unitaires	Validation des fonctionnalités
Semaine 7	Début du développement de l'API	Intégration des CRUD dans l'API
Semaine 8	Déblocage des problèmes de l'API	Résolution des erreurs de format JSON
Semaine 9	Finalisation des CRUD API	Préparation pour la Revue 1
Semaine 12	Débogage API et repository	Retrait des accents et gestion des suppressions avec clés étrangères
	Utilisation des API capteur en Python et PHP	Création de classes et exemples pour l'utilisation des API

Retour d'expérience (REX) : Cette tâche m'a permis de consolider mes compétences en PHP, en gestion de bases de données et en développement d'API REST. Voici un retour détaillé sur cette expérience :

- La séparation des responsabilités entre l'Entity (Capteur) et le Repository (CapteurRepository) a rendu le code plus modulaire et réutilisable.
- L'utilisation de PDO et des requêtes préparées a garanti une protection contre les injections SQL.

- L'API REST a été facile à implémenter grâce au modèle Entity-Repository. Les méthodes du CapteurRepository ont été directement utilisées dans les endpoints de l'API.

Cependant, j'ai fait face à une difficulté qui m'a beaucoup ralenti dans l'implémentation de l'API. Mes méthodes dans CapteurRepository utilisaient toutes des listes de listes. Mais cela empêchait la conversion en JSON. J'ai donc dû d'abord remarquer cette erreur car elle n'empêchait pas la classe CapteurRepository d'être utilisée mais juste l'API. Ensuite, je n'avais plus qu'à modifier le format pour chaque méthode. Enfin, j'ai refait cette tâche pour les autres tables de ma base de données. Il me reste à personnaliser les méthodes et les API en fonction des utilisations que mon groupe voudra en faire.

5.3 Tache 3 : Gestion des relevés

5.3.1 Objectifs

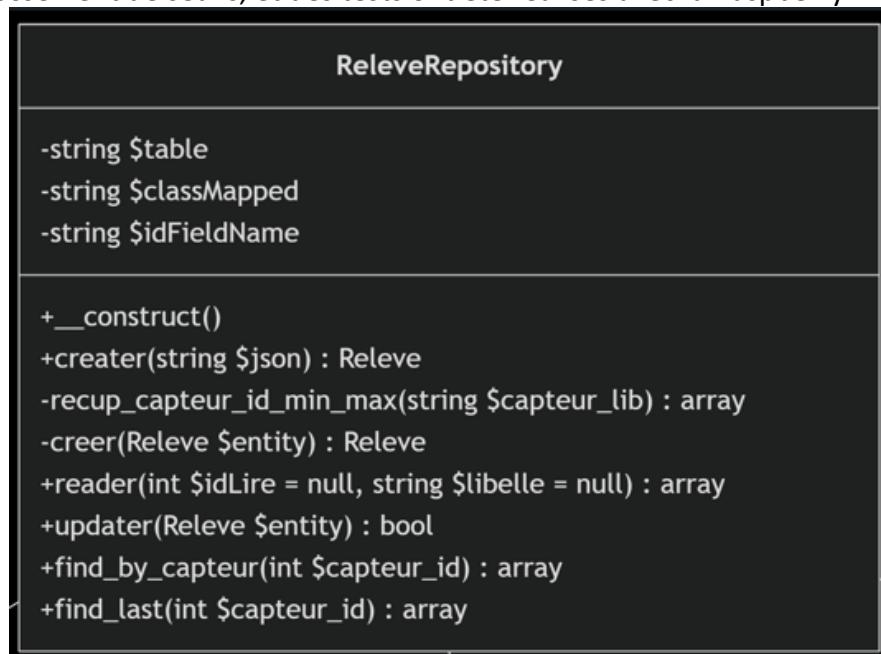
Durant les semaines qui ont suivi, le développement s'est concentré sur l'avancement de l'API, notamment sur la récupération des données des capteurs afin que Ethan et Rayane puisse l'utiliser.

5.3.2 Eléments de réalisation

Tout d'abords, le travail s'est concentré sur la consolidation des fonctionnalités existantes après la Revue 1, la création progressive des end-points de l'API et la validation des fonctionnalités principales lors de la revue.

Des ajustements ont été apportés pour améliorer la cohérence du code.

L'API pour les relevés a été finalisée, avec l'utilisation de la classe d'alertes automatiques en cas de dépassement de seuils, et des tests ont été réalisés avec la Raspberry Pi.



ReleveRepository.php

De plus, certaines méthodes ont été ajoutées dans le but de faciliter l'accès aux données pour Bastien, qui n'est pas obligé de trier lui-même. Par exemple, `find_by_capteur()` qui renvoie les données en fonction du capteur qui correspond à l'entrée. Également, `find_last()` qui renvoie le dernier relevé en fonction du capteur qui correspond à l'entrée.

5.3.3 Bilan

Semaine du	Tâche	Remarque
Semaine 10	Préparation Revue 1	Modification des repository et API pour des utilisations plus précises
Semaine 11	Revue 1	Passage de la revue et validation des fonctionnalités
Semaine 12	Débogage API et repository	Retrait des accents et gestion des suppressions avec clés étrangères
Semaine 13	Finalisation de l'API relevé	Intégration des alertes dans les relevés et correction des erreurs

Retour d'expérience (REX) : Des problèmes de fusion Git ont été résolus, notamment via la gestion du fichier `.gitignore`, et l'intégration des API sur la Raspberry Pi s'est poursuivie avec des corrections ciblées.

5.4 Tâche 4 : Gestion des utilisateurs et sécurité

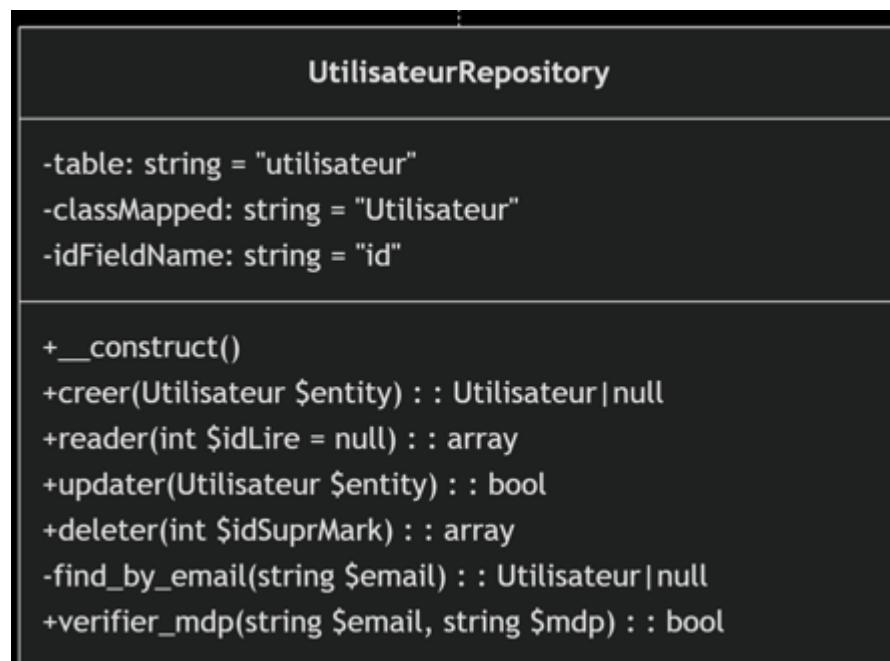
5.4.1 Objectifs

Afin de contrôler, les utilisateurs, ici, sur le site du projet, par Bastien, des end-points dans l'API ont dû être développés.

5.4.2 Eléments de réalisation

Un système complet de gestion des utilisateurs a été développé, couvrant les opérations de création, lecture, mise à jour et suppression (CRUD). L'API utilisateur a été testée et validée, bien que les mots de passe n'aient pas été hashés lors des premiers tests afin de simplifier les vérifications.

Par la suite, le hachage des mots de passe a été mis en place pour renforcer la sécurité.



UtilisateurRepository.php

```

public function verifier_mdp($email, $mdp)
{
    $utilisateur = $this->find_by_email($email);
    if (!$utilisateur || $utilisateur->get_mot_de_passe() === null) {
        return false;
    }

    return password_verify($mdp, $utilisateur->get_mot_de_passe());
}

```

Methode vérifier_mdp

En parallèle, l'intégration finale des API sur la Raspberry Pi a été achevée, avec une attention particulière portée à la correction des erreurs restantes liées aux relevés.

5.4.3 Bilan

Semaine du	Tâche	Remarque
Semaine 18	Développement des utilisateurs (repository et API)	Mise en place des fonctionnalités pour les utilisateurs
Semaine 19	Finalisation des utilisateurs avec mots de passe hashés	Sécurisation des mots de passe

Retour d'expérience (REX) : Grâce à cette tâche j'ai appris que la gestion du chiffrement devait être faite en dernière couche c'est-à-dire dans le Repository ce qui la rendrait plus sécurisée. Sinon, la difficulté principale lors de cette tâche était de retenir les mots de passes.

5.5 Tâche 5 : Gestion des sauvegardes

5.5.1 Objectifs

Un autre système a dû être mis en place, celui des sauvegardes qui permet à l'utilisateur de télécharger directement une sauvegarde de la base de données en format sql. Ceci lui permettrait en cas de perte dans le serveur pour une quelconque raison(piratage, dommages) de récupérer sa base de données telle qu'elle s'était arrêtée.

5.5.2 Eléments de réalisation

Pour ce faire, j'ai utilisé « my_sql_dump » puis les en-têtes de l'API qui permettent à celui qui l'appelle, en l'occurrence l'utilisateur car ce sera sur son poste de recevoir ce fichier.

```
// Execution du dump MySQL
$command = "mysqldump -h localhost -u root vitini_db > " .
escapeshellarg($fichier) . " 2>&1";
exec($command, $output, $returnCode);

if ($returnCode !== 0 || !file_exists($fichier) || filesize($fichier)
== 0) {
    error_log("echec du dump MySQL. Sortie: " . implode("\n",
$output));
    return null;
}
```

Extrait de la méthode creer() de SauvegardeRepository.php

```
case 'POST':
    $sauvegarde = new Sauvegarde($input);
    $result = $sauvegardeRepo->creer($sauvegarde);
    $le_cap = $sauvegarde->__toArray();
    http_response_code(201);

    $bdd_path = $le_cap["fichier"];
    echo $bdd_path;

    if (file_exists($bdd_path)) {
        header('Content-Description: File Transfer');
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment;
filename="'.basename($le_cap["fichier"]).'"');
        header('Expires: 0');
```

```

        header('Cache-Control: must-revalidate');
        header('Pragma: public');
        header('Content-Length: ' . filesize($bdd_path));
        readfile($bdd_path);
        exit;
    } else {
        http_response_code(404);
        echo json_encode(array("message"=>"Fichier non trouve"));
    }
echo json_encode($le_cap);
break;

```

Extrait de api_sauvegarde.php

Enfin, pour faciliter l'intégration, l'utilisation d'un fichier php qui a pour seule fonction d'appeler l'API afin de télécharger le fichier a été retenue afin qu'une simple redirection depuis le site soit faite.

5.5.3 Bilan

Semaine du	Tâche	Remarque
Semaine 18-19	Découvrir les dumps en php	Modification du repository pour créer le dump de la base de données
Semaine 20	Envie du fichier vers le client	Modification du code des end-points de l'API et création d'un fichier php qui télécharge automatiquement le dump

Retour d'expérience (REX) : Ici, la principale difficulté était de réaliser cette tâche directement sur notre serveur web car la commande n'existe pas sur Windows. J'ai également dû comprendre comment faire en sorte que le fichier soit téléchargé chez le client.

5.6 Tâche 6 : Gestion des alertes :

5.6.1 Objectifs

Ensute, ici l'objectif était de faire en sorte que chaque fois que l'on dépasse l'un des seuils maximum et minimum, une alerte soit envoyée par mail. Il faut aussi que l'on puisse voir les alertes sur le site.

5.6.2 Eléments de réalisation

Ensute, le système d'alerte a été déployé, à l'aide de mails et de nouveaux end-points dans l'API, ce qui nécessite également de nouvelles méthodes dans AlerteRepository.php.

```

AlerteRepository

|                                                                                                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-table: string = "alerte"</b><br><b>-classMapped: string = "Alerte"</b><br><b>-idFieldName: string = "id"</b>                                                                                                                                                                   |
| <b>+__construct()</b><br><b>+creer(Alerte \$entity) : : Alerte null</b><br><b>+reader(int \$idLire = null) : : array</b><br><b>+update(Alerte \$entity) : : bool</b><br><b>+deleter(int \$idSupr) : : bool</b><br><b>-envoyer_mail(int \$releve_id, string \$message) : : void</b> |


```

AlerteRepository.php

Afin d'envoyer les mails d'alerte, la solution de PHPMailer a été retenue avec l'utilisation du serveur SMTP de chez gmail.com. Ainsi, j'ai créé une adresse mail, vitiniciel2025@gmail.com. Ceux-ci ont permis l'implémentation d'une classe MailSender composée d'un constructeur et d'une méthode pour envoyer le mail.

Ensuite, une méthode privée a été implémentée dans AlerteRepository.php afin d'envoyer les mails à l'intégralité des utilisateurs de la base de données.

```

class MailSender {
    private $mail;
    private $senderEmail = 'vitiniciel2025@gmail.com';
    private $senderName = 'VITINI';
    private $smtpHost = 'smtp.gmail.com';
    private $smtpUsername = 'vitiniciel2025@gmail.com';
    private $smtpPassword = 'eqs tsoz jjvw lixs';
    private $smtpSecure = PHPMailer::ENCRYPTION_SMTPS; // SSL implicite
    private $smtpPort = 465; // Port standard pour SSL
}

```

Configuration MailSender.php

Ensuite, cette méthode est appelée dans la méthode creer() de la même classe qui est elle-même appelée dans l'API.

5.6.3 Bilan

Semaine du	Tâche	Remarque
Semaine 20	Gestion des alertes	J'ai d'abord créé un serveur SMTP de test dans le but de l'utiliser avec PHPMailer ce qui était inutile, puis l'utilisation de celui de gmail.com

Retour d'expérience (REX) : J'ai appris l'utilité et l'utilisation d'un serveur SMTP en php, ce qui a été la plus grande difficulté. J'ai perdu du temps en tentant d'en créer un, or j'aurais simplement dû utiliser gmail.

6 Candidat : Rayane

Introduction et présentation générale du rôle au sein du projet

Le projet VITINI vise à développer un système de surveillance intelligent pour améliorer les processus de viticulture et de viniculture au Domaine Miolanne, une exploitation viticole biologique située au pied du Massif du Sancy.

Dans ce cadre, mon rôle est centré sur la surveillance des conditions de vinification en mettant en place un réseau de capteurs permettant de suivre en temps réel les paramètres clés influençant la fermentation et le stockage du vin.

Mes principales missions sont les suivantes :

- Sélectionner des capteurs adaptés pour mesurer des paramètres essentiels (température, humidité, pH, CO₂, luminosité, etc.).
- Tester et configurer ces capteurs pour assurer leur bon fonctionnement avec la Raspberry Pi.
- Développer des scripts en Python pour collecter les données.
- Identifier et résoudre les problèmes techniques liés à l'intégration des capteurs.

Cette surveillance permettra aux vignerons de réagir rapidement aux variations de conditions, garantissant ainsi une meilleure maîtrise du processus de vinification et une amélioration de la qualité du vin produit.

Planning prévisionnel individuel

Tâche	État	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19
Choix des capteurs	Fini																			
Familiarisation avec les capteurs	En cours																			
Codage des capteurs	En cours																			
Envoyer des données																				

Tableau des tâches

Tâche	Description	Validée ?
Recherche des capteurs	Sélection des capteurs adaptés à la surveillance vinicole (température, humidité, pH, CO2, densité, etc.).	OK
Familiarisation avec les capteurs	Étude des protocoles de communication (I2C, 1-Wire, analogique, WiFi) et tests initiaux.	OK
Connexion des capteurs à la Raspberry Pi	Câblage et tests de lecture des capteurs.	OK
Codage des capteurs	Développement de scripts en Python pour récupérer et afficher les valeurs des capteurs.	OK

Après avoir défini mon rôle et listé mes tâches dans le projet, je vais maintenant détailler chaque étape de mon travail.

Dans un premier temps, j'ai recherché et sélectionné les capteurs adaptés à la surveillance vinicole en tenant compte de leur précision, de leur compatibilité avec la Raspberry Pi et de leur facilité d'utilisation.

Ensuite, j'ai commencé à tester les capteurs disponibles pour comprendre leur fonctionnement et voir comment les intégrer au projet.

Par la suite, j'ai essayé de connecter les capteurs à la Raspberry Pi et de récupérer les données avec des scripts Python. Cependant, j'ai rencontré plusieurs problèmes :

J'ai commis des erreurs de câblage et la plaquette d'essai que j'utilisais était défectueuse.

6.1 Etape – Tache 1

L'objectif de cette tâche était d'identifier des capteurs adaptés à la surveillance des conditions vinicoles, en prenant en compte :

- La précision des mesures, essentielle pour garantir un suivi fiable des conditions de fermentation et de stockage.
- La facilité d'intégration avec la Raspberry Pi et les autres composants du projet.
- Le coût, en restant dans les limites budgétaires du projet.

6.1.1 Objectifs

L'objectif de cette étape était de trouver et sélectionner les capteurs les plus adaptés pour surveiller les conditions de vinification. Plusieurs données doivent être mesurées afin de garantir un bon suivi du processus de fermentation et du stockage du vin :

- Température : Elle joue un rôle clé dans la fermentation et doit être surveillée pour éviter des écarts pouvant altérer la qualité du vin.

- Humidité : Un taux d'humidité stable est essentiel pour le stockage des bouteilles afin d'éviter le dessèchement des bouchons et l'oxydation du vin.
- Pression atmosphérique : Utile pour suivre les variations climatiques pouvant affecter la fermentation.
- CO₂ (dioxyde de carbone) : Indicateur direct de l'activité fermentaire, permettant de savoir si la fermentation se déroule correctement.
- pH : Permet de surveiller l'acidité du vin et d'ajuster les paramètres si nécessaire.
- Densité : Indispensable pour suivre l'évolution de la fermentation et déterminer la transformation des sucres en alcool.
- Luminosité : Pour vérifier que les bouteilles sont stockées dans des conditions optimales, à l'abri de la lumière.

Le principal problème était de trouver des capteurs capables de mesurer ces paramètres avec précision et de les intégrer dans notre système basé sur une Raspberry Pi. Plusieurs contraintes devaient être prises en compte :

- Compatibilité avec la Raspberry Pi : Choisir des capteurs utilisant des protocoles standards comme I2C, 1-Wire ou analogique.
- Précision des mesures : Sélectionner des capteurs avec une marge d'erreur faible.
- Facilité d'utilisation : Privilégier des capteurs bien documentés et simples à interfaçer.

6.1.2 Eléments de réalisation

Après une étude des différentes options disponibles, les capteurs suivants ont été sélectionnés :

Grandeur mesurée	Type de signal	Protocole	Référence	Pourquoi ce choix ?
Pression & Température	Digital	I2C	BMP280	Capteur précis, faible consommation et facile à interfaçer avec la Raspberry Pi.
Humidité & Température	Digital	Propriétaire	DHT11	Facile à utiliser et suffisant pour le suivi des conditions de stockage.
CO ₂	Analogique	-	DFRobot SEN0159	Essentiel pour le suivi de la fermentation, permet de surveiller la production de CO ₂ .
Luminosité	Analogique	-	Grove Light Sensor	Permet de s'assurer que les bouteilles sont stockées dans de bonnes conditions d'obscurité.

Grandeur mesurée	Type de signal	Protocole	Référence	Pourquoi ce choix ?
Température liquide	Digital	1-Wire	DS18B20	Sonde étanche adaptée aux mesures dans des liquides, idéale pour la fermentation.
pH	Analogique (avec convertisseur)	-	Gravity: Analog pH Meter Kit	Indispensable pour surveiller l'acidité du vin en fermentation.
Densité	WiFi	Cloud	RAPT Pill ou Brewbrain Float	Permet de suivre l'évolution de la fermentation en temps réel, avec un accès à distance aux données.

6.1.3 Bilan

Retour d'expérience (REX) :

Cette tâche m'a permis de bien démarrer le projet en identifiant clairement les capteurs nécessaires. Elle m'a aidé à mieux comprendre les besoins techniques liés à la vinification et à faire des choix en fonction de la précision, de la compatibilité avec la Raspberry Pi et de la faisabilité technique.

6.2 Etape – Tache 2

6.2.1 Objectifs

L'objectif de cette tâche était de tester les capteurs sélectionnés afin de s'assurer de leur bon fonctionnement et de leur intégration avec la Raspberry Pi.

6.2.2 Eléments de réalisations

- Étude des caractéristiques et des protocoles de communication (I2C, SPI, analogique).
- Connexion des capteurs sur une plaquette d'essai pour effectuer les premiers tests.

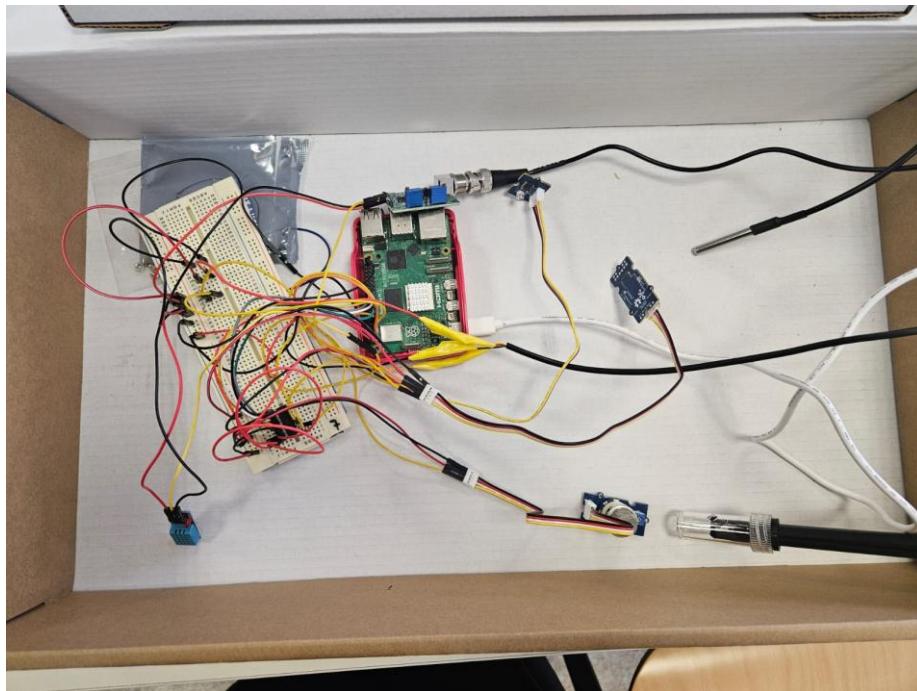


Photo du cablage

- Récupération des valeurs via des scripts Python.

```
#!/usr/bin/env python

from smbus import SMBus
from bmp280 import BMP280
import time
from gpiozero import MCP3008
import adafruit_dht
import board
import glob

# Initialisation du BMP280 (capteur de pression et température)
bus = SMBus(1)
bmp280 = BMP280(i2c_addr=0x77, i2c_dev=bus)

# Initialisation des capteurs analogiques (MQ2, Luminosité, et pH)
mq2 = MCP3008(channel=2) # Canal 0 pour MQ2
lum = MCP3008(channel=0) # Canal 1 pour capteur de luminosité
ph = MCP3008(channel=1) # Canal 2 pour le capteur pH

# Initialisation du capteur DHT11 (sur GPIO 17)
dht11 = adafruit_dht.DHT11(board.D17)

# Initialisation du capteur DS18B20
```

```
ds18b20 = glob.glob("/sys/bus/w1/devices/28*")[0] + "/w1_slave"

# Fonction pour lire la température du DS18B20
def lire_temperature():
    with open(ds18b20, "r") as f:
        lignes = f.readlines()
    while "YES" not in lignes[0]:
        time.sleep(0.2)
    with open(ds18b20, "r") as f:
        lignes = f.readlines()
    return float(lignes[1].split("t=")[-1]) / 1000.0

# Fonction pour calculer le pH à partir de la valeur ADC
def calcul_ph():
    ph_adc_value = ph.value * 1023 # Conversion en 10 bits (0-1023)
    voltage = (ph_adc_value / 1023.0) * 5.0 # Conversion en tension (0-5V)
    print(f"Tension mesurée : {voltage} V") # Afficher la tension mesurée
    ph_value = (voltage - 3.77) * (-5) + 7 # Formule de calibration
    return ph_value

# Fonction pour calculer la luminosité en lux
def calcul_lux():
    Vout = lum.value * 5 # Conversion en tension réelle (0-5V)
    lux = max(0, 2500 / (5 - Vout) - 500) # Formule empirique
    return lux

# Boucle principale
while True:
    try:
        # Lire la température et l'humidité du DHT11
        dht_temperature = dht11.temperature
        dht_humidite = dht11.humidity

        # Lire la température et la pression du BMP280
        temperature = round(bmp280.get_temperature(), 1)
        pression = round(bmp280.get_pressure(), 2)

        # Lire les valeurs des capteurs analogiques
        mq2_valeur = mq2.value * 5 # Conversion en ppm (à vérifier selon
        capteur)

        # Lire la luminosité et la convertir en lux
```

```
lum_valeur = calcul_lux()

# Lire la température du DS18B20
ds18b20_temperature = lire_temperature()

# Calcul du pH à partir du capteur
ph_valeur = calcul_ph()

# Affichage des résultats
print(f"Température DHT11: {dht_temperature}°C, Humidité DHT11:
{dht_humidite}%")
print(f'Température BMP280: {temperature}°C')
print(f'Pression BMP280: {pression} hPa')
print(f'MQ2: {mq2_valeur} ppm')
print(f'Luminosité: {lum_valeur:.2f} lux')
print(f"Température DS18B20: {ds18b20_temperature:.2f}°C")
print(f"Valeur du pH : {ph_valeur:.2f}")
print()

except Exception as e:
    print(f"Erreur: {e}")

# Attendre 1 seconde avant de refaire une lecture
time.sleep(1)
```

Résultat :

```
Luminosité: 0.24 lux
Température DS18B20: 23.75°C
Valeur du pH : 25.84

Tension mesurée : 0.002442598925256334 V
Température DHT11: 24.1°C, Humidité DHT11: 57%
Température BMP280: 23.5°C
Pression BMP280: 995.67 hPa
MQ2: 0.002442598925256334 ppm
Luminosité: 0.24 lux
Température DS18B20: 23.75°C
Valeur du pH : 25.84

Tension mesurée : 0.002442598925256334 V
Température DHT11: 24.1°C, Humidité DHT11: 57%
Température BMP280: 23.5°C
Pression BMP280: 995.68 hPa
MQ2: 0.002442598925256334 ppm
Luminosité: 0.24 lux
Température DS18B20: 23.75°C
Valeur du pH : 25.84

Tension mesurée : 0.002442598925256334 V
Température DHT11: 24.1°C, Humidité DHT11: 57%
Température BMP280: 23.5°C
Pression BMP280: 995.68 hPa
MQ2: 0.002442598925256334 ppm
Luminosité: 0.24 lux
Température DS18B20: 23.75°C
Valeur du pH : 25.84

Tension mesurée : 0.002442598925256334 V
Température DHT11: 24.1°C, Humidité DHT11: 57%
Température BMP280: 23.5°C
Pression BMP280: 995.67 hPa
MQ2: 0.002442598925256334 ppm
Luminosité: 0.24 lux
Température DS18B20: 23.75°C
Valeur du pH : 25.84

^CReceived SIGINT

Traceback (most recent call last):
  File "/home/rasp/capteur2.py", line 88, in <module>
```

Capture d'écran du résultat du script ci-dessus

6.2.3 Bilan

Retour d'expérience (REX) :

Cette tache m'a permis de consolider mes bases en python et en câblage, elle m'a permis de développer mon sens de la réflexion, à la suite de certains problèmes de matériel. Il me reste seulement le capteur en cloud à configurer.

6.3 Cahier de suivi

Semaine du	Tâche	Résultat	Remarque	Validé ?
Semaine 10	Préparation Revue 1	Conforme	Codage des premiers capteurs.	OK
Semaine 11	Revue 1	Conforme	Passage de la revue et validation des fonctionnalités	OK
Semaine 12-13	Codage des capteurs	Conforme	Problèmes de compréhension avec les valeurs reçues.	OK
Semaine 14	Compréhension des valeurs reçues.	Conforme	Compréhension des unités ppm lux et du Ph.	OK
Semaine 15	BTS Blanc	-	Préparation et passage des épreuves blanches	-
Semaine 16-17	Vacances	-	Pause	-
Semaine 18	Codage des capteurs Ph et luminosité.	Conforme	Dernières modifications des conversions.	OK
Semaine 19	Étalonnage du capteur de Ph.	Conforme	Résolution des dernières erreurs et validation de l'intégration	OK

7 Candidat : Goncalves Ethan

Sur le projet de VITINI je suis le chef de projet, je m'occupe de l'organisation des taches, je consulte l'avancée de tout le monde et je réunis chaque partie du projet pour n'en faire plus qu'un. Mon but au sein du projet est de récupérer les données captées par l'end Device depuis la viniculture et d'envoyer ces données sur une Raspberry via LoRaWan qui les transmettra au site internet du vigneron.

Tache	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22
Ethan																						
Choisir l'alimentation		Red																				
Transmettre les valeurs		Yellow																				
Installation des programmes sur la Raspberry Pi 5				Red																		
Migration du code vers la Raspberry Pi 5				Yellow	Yellow	Yellow	Yellow	Yellow														
Teste du pluviomètre sur la Raspberry Pi 5									Red	Red	Red	Red										
Envoie vers la base de donnée									Yellow	Yellow	Yellow	Yellow	Yellow				Yellow	Yellow				
Mise en place du panneau solaire																	Red					
Lancement du code à l'allumage de la Raspberry pi5									Yellow	Yellow	Yellow	Yellow	Yellow				Yellow	Yellow				
Ajout du pluviomètre à l'EndDevice																	Red	Red	Red	Red	Red	

Semaine	Tache	Description	Validée ?
S1	Choisir l'alimentation	Choisir un panneau solaire qui permet d'alimenter l'end Device	OK
S2-S4	Transmettre les valeurs	Récupérer les données sur le PC des capteurs du eid430 en passant par le LoRaWan	OK
S5	Installation des programmes sur la Raspberry Pi 5	Faire les installations nécessaires au bon fonctionnement du code	OK
S6-S8	Migration du code vers la Raspberry Pi 5	Mise en place du code pour qu'il fonctionne sur la Raspberry Pi 5	OK
S10-S12	Teste du pluviomètre sur la Raspberry Pi 5	Test du pluviomètre pour observer son fonctionnement	OK
S12-S17	Envoie vers la base de données	Faire que les capteurs et les données soit envoyé vers la base de données	OK
S18	Mise en place du panneau solaire	Faire que le panneau solaire alimente bien le EndDevice	OK
S19-S20	Lancement du code à l'allumage de la Raspberry pi5	Faire que le code clientmqtt se lance au démarrage de la Raspberry pi5	OK
S20-S22	Ajout du pluviomètre à l'EndDevice	Mise en place du capteur sur le eid430	En cours

7.1 Etape 1 – Le panneau solaire

Pour commencer, ma première tâche est de trouver un panneau solaire pour pouvoir alimenter l'end Device.

7.1.1 Contraintes

Il y a plusieurs contraintes à cela. Il faut tout d'abord que le panneau solaire soit fonctionnel quelles que soient les intempéries, il doit pouvoir fournir l'énergie nécessaire au fonctionnement de l'appareil et ce panneau solaire doit être capable d'être branché en USB. Si toutes ces vérifications sont faites, l'end Device sera fonctionnel.

7.1.2 Choix de l'appareil

Pour choisir le bon panneau solaire, j'ai dû d'abord chercher sur internet ce qui pouvait fournir l'énergie nécessaire au fonctionnement de l'end Device, puis vérifier s'ils pouvaient se brancher en USB et enfin si le panneau solaire était en IP65. J'ai dû dresser une liste où j'ai mis chaque panneau solaire que je trouvais en écrivant en détail ce qu'ils faisaient. Après ça, j'ai fait un grand tri qui consistait à garder et enlever les panneaux solaires qui ne correspondaient pas à la demande exigée.

J'ai donc eu le choix entre le Tapo A200, l'ALLPOWERS 2 pièces 2.5W et l'ECO-WORTHY Kit Panneau Solaire, qui remplissaient toutes les caractéristiques obligatoires. Je suis parti sur le choix du Tapo A200 par rapport aux autres car, comparé aux autres, le panneau solaire de l'ALLPOWERS 2 pièces 2.5W était sur un site où je n'étais pas sûr de la confiance de livraison et de la qualité du panneau solaire qui se nomme iallpowers.fr. Pour le VEVOR Panneau Solaire Portable, son prix était beaucoup trop coûteux et j'ai aussi vu au dernier moment que sa puissance était trop élevée par rapport à ce dont j'avais besoin.



Figure 17 Panneau solaire Tapo A200

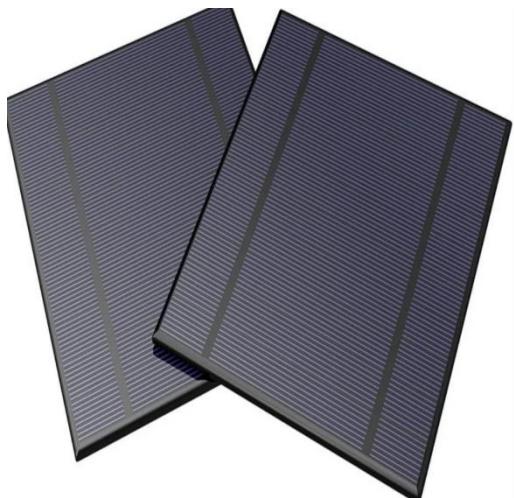


Figure 19 Panneau solaire ALLPOWERS



Figure 18 Panneau solaire VEVOR

7.1.3 Bilan

Le panneau solaire a bien été reçus et prêt à l'utilisation mais son branchement immédiat ne nous est pas utile, on utilisera à la place directement le courant de notre école qui est beaucoup plus pratique à l'immédiat.

Après avoir tout reçus le matériel nécessaire à ma partie du projet, je commence le code qui va permettre de récupérer les données de mon end Device.

7.2 Etape 2– Installation du panneau solaire

Puisque les capteurs sont finalisés et que les capteurs envoie tout à la base de données il ne reste que à installer le panneau solaire pour que le EID 430 000 soit alimenté au milieu des vignes.

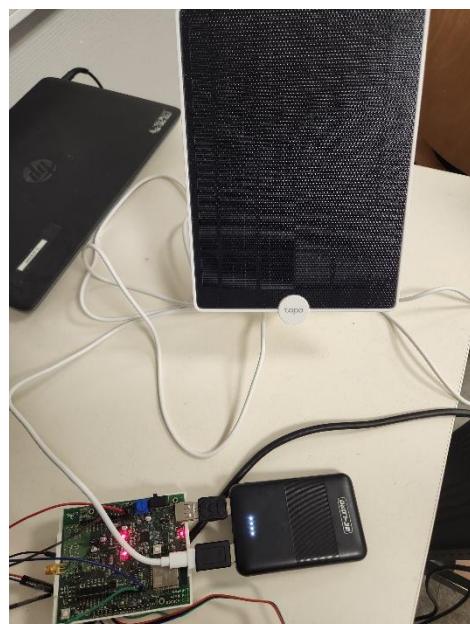
7.2.1 Objectifs

Pour réaliser cela il va falloir brancher le Tapo A200 au EID 430 000 et voir si le panneau solaire alimente assez l'appareil pour qu'il s'allume et fasse tourner les capteurs.

7.2.2 Eléments de réalisation

Pour ce faire, on va devoir acheter des adaptateurs, car l'EndDevice ne peut être branché que via un port USB alors que le panneau solaire lui n'a qu'une sortie micro USB. Après l'ajout d'un adaptateur micro USB femelle et d'un USB male, on va pouvoir voir s'il fournit assez pour alimenter tout l'EndDevice.

On a un autre problème, l'énergie donnée par le panneau solaire varie selon le temps et le panneau solaire n'a pas été fournis avec une batterie ce qui fait si le soleil est couvert, il va certes alimenter assez pour allumer le EID 430 000, mais pas les capteurs qui sont installé dedans. On va devoir ajouter une batterie de secours de téléphone pour que l'on puisse stocker l'énergie et qu'il puisse donc alimenter l'EID 430 000 au complet.



7.2.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Teste si le EID 430 000 s'allume et que les capteurs soit fonctionnel	Je branche le panneau solaire et la batterie de secours et je regarde si les LED sur l'appareil s'allument et je lance le code mqtt.	Les LED sur l'End Device s'allume comme il faut et quand je lance le code je reçois les données des capteurs	Les LED s'allument et je reçois bien les données des capteurs	OK

Grace a l'installation de ce panneau solaire, je finalise donc m'a partis de projet car l'EID 430 000 est alimenté dans le champ, il envoie les données des différents capteurs vers la base de données.

7.3 Etape 3 – Transmettre les valeurs

Par la suite, on va pouvoir commencer le code pour pouvoir récupérer les valeurs de la température, du pluviomètre et de la luminosité.

7.3.1 Objectifs

Pour réaliser ça il faut faire un code qui va devoir vérifier les données reçus de l'end Device, récupérer les données si elles sont valides et afficher les données.

7.3.2 Eléments de réalisation

Pour réaliser ça il faut un LoRaWan, un end Device et un pc pour le code.

Je vais avoir dans mon fichier nommé viniculture qui contient plusieurs fichiers de code tels que :

-clientmqtt.py : qui se connecte au LoRaWan qui récupère et affiche les valeurs

-Device.py : permet de vérifier et convertir les données du pluviomètre.

-luxmetre.py : permet de vérifier et convertir les données de la lumière.

-temperature.py : permet de vérifier et convertir les données de la température.

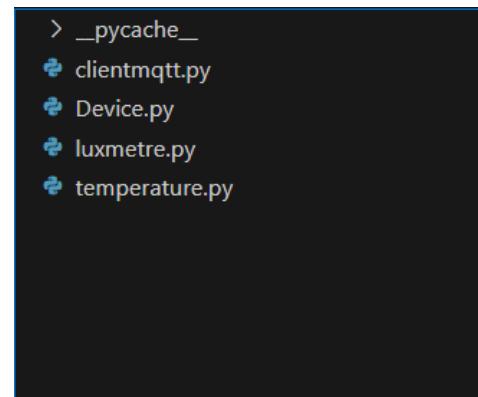


Figure 20 Dossier Viticulture

```
Application: ID= 6 Nom= App_ORT_Lyon
End_Device : Nom= CapteurDidalab01
Capteur     : Port= 3
Luminosité=> 1138120343
Luminosité convertie: 428.68 LUX
Application: ID= 6 Nom= App_ORT_Lyon
End_Device : Nom= CapteurDidalab01
Capteur     : Port= 4
Température => 25.68
```

Figure 21 valeurs des capteurs

Dans cette image on peut voir les résultats du capteur au port 3 qui capte la luminosité avec une valeur de 1138120343 lux mais la donnée n'est pas convertie, c'est la ligne en dessous qui la converti en une donnée lisible qui est 428.68 lux, il y a aussi le capteur 4 qui capte la température et qui donne 25.68°C. Ces données sont récupérées toute les secondes et on enverra ces données sur le site.

7.3.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Teste si on affiche les données récupérer	Je lance le code, il doit réussir à se connecter au LoRaWan et donner les valeurs reçus	Le code réussis à se connecter au LoRaWan et il affiche bien les valeurs des différents capteurs toutes les 1 secondes.	Conforme	OK

Pour faire cette étape du projet, j'ai dû ouvrir un ancien fichier de cours qui se nomme TP LoRaWan v2024.docx qui m'a permis de programmer le code que l'on peut voir en annexe. En suivant d'ancien TP le code la prise en main était plus facile, il fallait juste adapter le TP pour le projet pour arriver à la finalité.

Maintenant qu'on récupère bien les valeurs du capteur sur un ordinateur il faut le mettre sur un Raspberry.

7.4 Etape 4 – Installation des programmes sur la Raspberry Pi 5

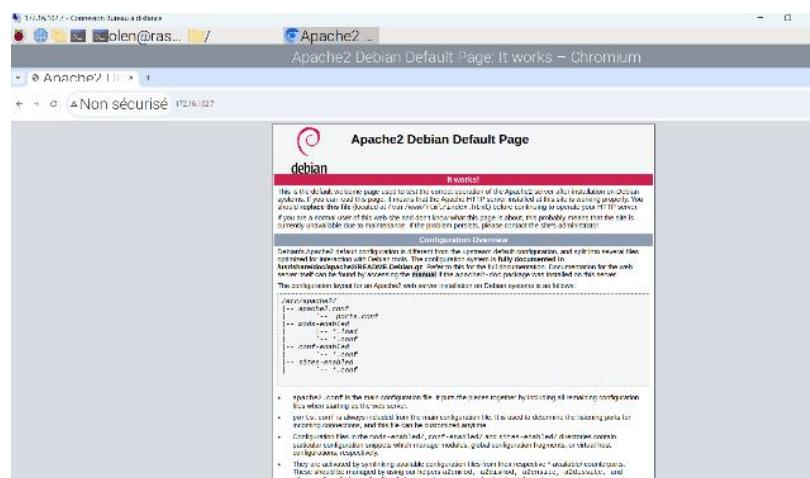
Par la suite, mon objectif va être d'installer les différentes codes et logiciel essentiel au bon fonctionnement de mon code, de celui des autres de mon groupe et enfin de la base de données sur la Raspberry Pi 5

7.4.1 Objectifs

Pour gérer la base de données MariaDB utilisée dans le projet VITINI, j'ai choisi d'installer phpMyAdmin, une interface web intuitive qui permet d'administrer les bases de données MySQL/MariaDB plus facilement. Cette installation se fait en plusieurs étapes, en installant au préalable les composants nécessaires.

7.4.2 Eléments de réalisation

Pour commencer, il faut installer en premier Apache car phpMyAdmin fonctionne avec un navigateur et un serveur web. J'ai choisi Apache car il est compatible avec phpMyAdmin, il est fiable et est facile d'utilisation. Il suffit juste de faire « sudo apt install apache2 » pour l'installer. Après l'avoir installé, j'ai vérifié le bon fonctionnement en ouvrant un navigateur et en entrant l'adresse IP locale de la Raspberry Pi.



Si tout fonctionne, une page « It works! » d'Apache s'affiche.

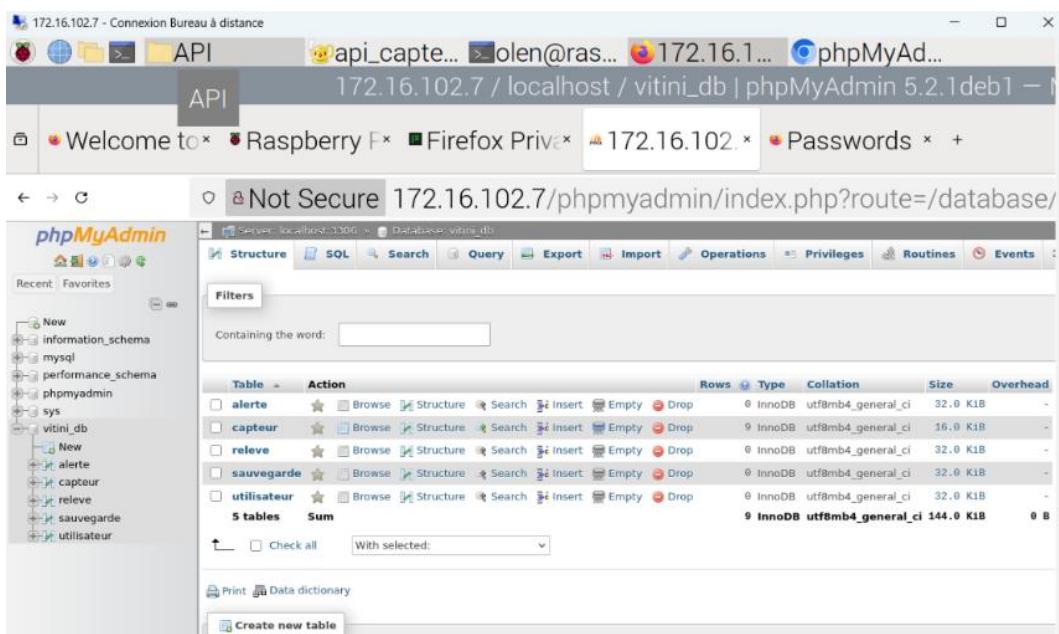
Ensuite, il faut installer PHP, car phpMyAdmin est écrit en PHP ainsi que ses modules. Il suffit de faire comme commande pour l'installer « sudo apt install php libapache2-mod-php php-mysql ». Quand c'est installer, on peut installer Maria DB qui est un système de gestion de base de données libre qui est compatible avec MySQL, il va donc permettre de stocker les données. Pour son installation, il faut mettre « sudo apt install mariadb-server », J'ai ensuite

```
olen@raspberrypi:~ $ php -v
PHP 8.2.28 (cli) (built: Mar 13 2025 18:21:38) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.2.28, Copyright (c) Zend Technologies
with Zend OPcache v8.2.28, Copyright (c), by Zend Technologies
```

suivi les instructions pour définir un mot de passe administrateur (root), supprimer les utilisateurs anonymes, désactiver l'accès distant à root et supprimer la base de test.

Enfin, dans les installations, il me reste à installer phpMyAdmin qui gère facilement la bases de données Maria DB via une interface web.

J'ai pour commencer utiliser comme commande « sudo apt install phpmyadmin », en attendant l'installation j'ai sélectionné Apache comme serveur web, j'ai choisi de configurer automatiquement base de données via dbconfig-common et un mot de passe m'a été demandé pour l'utilisateur phpMyAdmin. Je restarte le serveur Apache via « sudo systemctl restart apache2 » et enfin je vais sur internet, je mets mon adresse ip dans la barre de recherche avec /phpMyAdmin et je rentre mon identifiant ainsi que mon mot de passe, il ne reste plus qu'à importer le SQL de la base de données dans PhpMyAdmin qui se nomme vitini_db.



J'ai rencontré un problème durant le procédé qui est l'erreur "#1273 - Unknown collation: 'utf8mb4_0900_ai_ci'" qui signifie que le fichier SQL a été créé avec une version plus récente de MySQL/MariaDB, alors que ma Raspberry Pi 5 utilise une version plus ancienne qui ne supporte pas cette collation. Il fallait donc utiliser utf8mb4_general_ci au lieu de utf8mb4_0900_ai_ci dans le SQL de la base de données.

7.4.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Teste du fonctionnement d'Apache	Je mets l'IP de ma Raspberry dans la barre de recherche	Une page « It works! » d'Apache s'affiche	Conforme	OK
Teste du fonctionnement de PHP	Je mets l'IP de ma Raspberry dans la barre de recherche avec un /php	Une page PHP doit s'afficher	Conforme	OK
Teste du fonctionnement de Maria DB	Je mets mysql -u root -p	Ca m'affiche "Welcome to the MariaDB monitor. Commands end with ; or \g."	Conforme	OK
Teste du fonctionnement de phpMyAdmin	Je mets l'IP de ma Raspberry dans la barre de recherche avec un /phpMyAdmin	Une page phpMyAdmin doit s'afficher	Conforme	OK

Grâce à toutes ces installations, j'ai mis en place un environnement complet, fonctionnel et accessible permettant de gérer efficacement une base de données sur ma Raspberry Pi. L'utilisation de phpMyAdmin en interface graphique facilite les manipulations et les bases techniques (Apache, PHP, Maria DB) garantissent une structure solide pour le projet. Maintenant que la Raspberry est prête à utilisation, il faut mettre en place le code pour voir s'il est fonctionnel sur la Raspberry.

7.5 Etape 5– Migration du code vers la Raspberry Pi 5

Pour continuer, il faut appliquer le code du pc sur la Raspberry et voir si le code est toujours fonctionnel car on va stocker principalement tout notre système dessus pour utiliser le moins d'appareil possible.

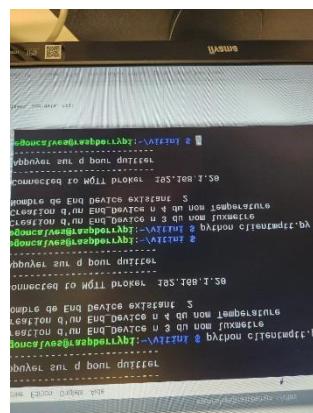
7.5.1 Objectifs

La difficulté avec ça, c'est que la Raspberry ne va pas avoir tout l'avantage qu'a un pc, il y aura donc certaine fonction qui ne marche pas et certain module vont devoir être importer.

7.5.2 Eléments de réalisation

Pour réaliser ce projet il faut un Raspberry déjà prêt à être utilisé, j'ai aussi installé un routeur Wi-Fi pour pouvoir connecter directement la Raspberry en Wi-Fi et non via l'Ethernet car la manipulation du LoRaWan sera plus rapide, simple et facile d'utilisation puisqu'on pourra récupérer les données n'importe où dans le bâtiment pour des tests et ne plus être bloqué par la distance qu'offre l'Ethernet. Il fallait maintenant télécharger les différents « import » pour que le code soit fonctionnel quand il est lancé.

Durant cette migration un problème est arrivé, il n'affichait pas les valeurs de mes différents capteurs, la raison était simple car la Raspberry de prenais pas l'« import » keyboard et ne pouvais pas donc boucler, il fallait donc ajouter une boucle avec un temps d'attente de 0.5sec. Quand cette boucle est ajoutée on peut voir que les différentes valeurs s'affichent bien sur le terminal.



```
olen@raspberrypi:~/var/www/projet_traitement/viticulture $ python clientmqtt.py
<Response [201]>
Creation d'un End_Device n 3 du nom luxmetre
Creation d'un End_Device n 4 du nom Temperature
Nombre de End Device existant 2

Connected to MQTT broker 192.168.1.20
-----
Appuyer sur q pour quitter
-----
Application: ID= 6 Nom= App_ORT_Lyon
End_Device : Nom= CapteurDidalab01
Capteur : Port= 3
Luminosite=> 1109123554
38.970588668408203
Application: ID= 6 Nom= App_ORT_Lyon
End_Device : Nom= CapteurDidalab01
Capteur : Port= 4
Temperature => 27.13
Application: ID= 6 Nom= App_ORT_Lyon
End_Device : Nom= CapteurDidalab01
Capteur : Port= 9
```

Figure 21 : Ajout de la boucle

Figure 20 Erreur dans l'affichage

7.5.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Teste si on affiche les données récupérer	Je lance le code, il doit réussir à se connecter au LoRaWan et donner les valeurs reçus	Le code réussis à se connecter au LoRaWan et il affiche bien les valeurs des différents capteurs toute les 1 secondes	Les données s'affichent	OK

On reçoit bien les différentes valeurs des capteurs via la Raspberry Pi 5, on va pourvoir maintenant mettre en place le pluviomètre qui n'est pas directement installer dans l'End Device et ensuite récupérer ses valeurs.

7.6 Etape 6– Teste du pluviomètre sur la Raspberry Pi 5

Maintenant que le code principal est fonctionnel, il est temps d'ajouter les éléments nécessaires au bon fonctionnement du pluviomètre DFRobot SEN0575.

Le capteur sera dans un premier temps testé sur la Raspberry Pi 5, car le module final LoRaWAN (le EndDevice) agit comme une "boîte noire" difficilement accessible. En attendant plus d'informations du constructeur concernant son intégration, il a été décidé de procéder aux essais sur un environnement connu, stable et compatible : la Raspberry Pi.

7.6.1 Objectifs

L'objectif de cette étape est double :

1. Lire et enregistrer les données issues du pluviomètre :
 - Pluie cumulée (en mm),
 - Pluie tombée sur une durée d'une heure (en mm),
 - Nombre de basclements du seau.
2. Transmettre les données au code principal via MQTT, afin qu'elles puissent être :
 - Affichées en temps réel,
 - Intégrées dans le flux de données du système global.

Cela nécessitera le développement de deux scripts distincts :

- Le premier interagit directement avec le capteur via l'interface I2C,
- Le second lit les données extraites pour les formater et les transmettre via MQTT.

7.6.2 Eléments de réalisation

Pour commencer le projet, on va présenter le capteur de pluies qui est le DFRobot SEN0575. Le DFRobot SEN0575 est un pluviomètre numérique conçu pour mesurer la quantité de précipitations. C'est un capteur industriel de haute précision qui fonctionne selon le principe du godet basculant, et il est compatible avec des interfaces I2C ou UART (nous utiliserons pour ce projet le I2C). Le bus I2C, ou Inter-Integrated Circuit, est un protocole de communication série utilisé pour connecter plusieurs périphériques électroniques entre eux sur une même ligne de communication.

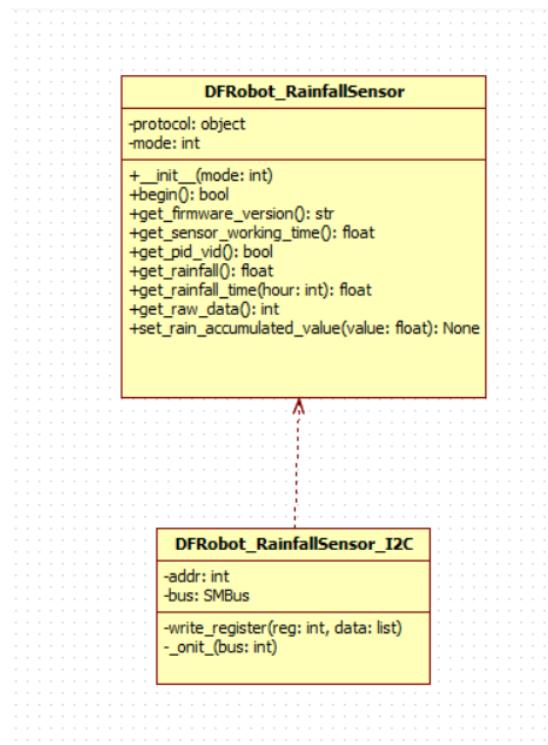


Cette adresse permet au maître de parler à un seul périphérique à la fois, même s'ils partagent les mêmes deux fils SDA/SCL.

Pour faire fonctionner ce capteur de pluie il est nécessaire de le faire fonctionner via deux codes. Le premier code va permettant de récupérer les données de précipitation (cumulée,

sur une durée, nombre de basculements) et de configurer le capteur via l'interface I2C en manipulant directement ses registres internes. Pour se faire on doit tout d'abord définir une classe mère `DFRobot_RainfallSensor` et une classe fille `DFRobot_RainfallSensor_I2C`. Puisqu'on utilise l'I2C, on utilise comme adresse par défaut 0x1D car c'est une adresse fixée en usine et on va utiliser la bibliothèque `smbus` qui permet de communiquer avec des périphériques I²C sous Linux.

Le code fournit par DfRobot fonctionne ainsi :



Lecture des données :

- Pluie cumulée (via registres de type *Input*),
- Pluie sur la dernière heure,
- Nombre de basculements du seau.

Réinitialisation des valeurs :

Permet de remettre à zéro le compteur après chaque envoi ou à intervalle régulier.

Configuration du capteur :

Définition du volume d'eau par basculement,

Paramétrage des heures de mesure.

Une fois les données récupérées via le capteur, un second script Python lit ces données puis les envoie à l'aide du client MQTT, qui est légèrement modifié pour ajouter la prise en charge du pluviomètre.

Cela permet :

- Une centralisation des données,
- Une visualisation en temps réel sur l'interface du projet,
- Une intégration fluide dans l'architecture IoT mise en place.

7.6.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Teste si on affiche les données récupérer	Je lance le code, il doit réussir à se connecter au LoRaWan et donner les valeurs reçus du pluviomètre	Le code réussis à se connecter au LoRaWan et il affiche bien les valeurs du capteur	Les données s'affichent	OK

On reçoit bien les différentes valeurs du capteur via la Raspberry Pi 5, on va pourvoir donc maintenant envoyer ces différentes données sur la base de données pour pouvoir ensuite les utiliser sur la Raspberry.

7.7 Etape 8– Envoi vers la base de données

Maintenant que les différents capteurs sont fonctionnels, il faut maintenant ajouter le code qui permet d'envoyer les différentes valeurs des capteurs sur la base de données que l'on a apporté à l'étape 3, grâce à l'API qui a été fait par Aounou.

7.7.1 Objectifs

Pour réaliser cela il va falloir faire des ajouts dans le code qui vont passer par l'API et qui vont ensuite l'envoyer dans une base de données, il va permettre aussi de créer un nouveau capteur de donnée si le capteur n'existe pas dans la base de données et ces données seront après envoyer sur le site pour que le vigneron puisse voir les données de son champ.

7.7.2 Eléments de réalisation

Pour commencer, il faut intégrer l'API qui permet l'ajout des données à la base de données. Ensuite, dans le code, on définit un dictionnaire de configuration pour les capteurs. Ce dictionnaire contient des informations essentielles telles que :

- Le type de capteur,
- L'unité de mesure,
- Les valeurs minimale et maximale,
- Et le numéro de port auquel il est connecté.

À partir de ces données, on associe chaque port à une configuration précise.

On procède ensuite à la création d'instances des classes Capteur et Relevé, qui seront utilisées pour interagir avec l'API.

Une vérification est ensuite effectuée pour éviter la création en double d'un même capteur dans la base de données. Si le capteur n'existe pas encore, il est créé via l'API et son identifiant est stocké dans une structure appelée capteurs_crees, ce qui permet de ne pas le recréer inutilement lors des prochaines itérations.

Par la suite, le programme vérifie que la valeur mesurée est bien un nombre. Si c'est le cas, on prépare un objet JSON représentant le relevé, puis on l'envoie à l'API via la méthode api_releve.create().

Enfin, la définition de la boucle principale est modifiée pour vérifier si chaque port a bien un capteur enregistré dans la base de données (grâce à la fonction creer_capteur). Une fois cette vérification faite, le relevé correspondant est enregistré via la fonction enregister_releve.

7.7.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Teste si on affiche les données récupérer dans la base de données	Je lance le code, il doit réussir à se connecter au LoRaWan et doit mettre les valeurs reçus des capteurs dans la base de données	Le code réussis à se connecter au LoRaWan et on voit bien les données des capteurs dans la base de données	Les données sont bien stockées	OK

Notre chemin pour envoyer les données vers la base de données est achevé avec cette ajout de code, il reste maintenant à ajouter le panneau solaire a l'appareil.

7.8 Etape 9 – Lancement du code à l'allumage de la Raspberry pi5

Pour faciliter le fonctionnement du code, on va mettre en place un système qui va lancer directement le code clientmqtt.py dès l'allumage de la Raspberry pi5.

7.8.1 Objectifs

L'objectif de cette étape va permettre de faciliter l'accès au client en enlevant le fait que si la Raspberry pi5 s'éteint, s'il la rallume il n'aura pas besoin de démarrer le code par lui-même mais automatiquement.

7.8.2 Eléments de réalisation

Afin de garantir l'exécution automatique de mon script Python au démarrage de la Raspberry Pi 5, j'ai utilisé systemd, le gestionnaire de services présent sur la majorité des distributions Linux modernes.

Systemd est un système d'initialisation qui permet de lancer, arrêter et superviser des services dès le démarrage du système. L'objectif ici est de créer un fichier de service permettant à la Raspberry Pi d'exécuter mon script Python de manière autonome et fiable.

Pour cela, je vais créer un fichier de configuration systemd, appelé un service, qui décrit :

- Où se trouve mon script Python
- Ce qu'il doit exécuter
- Avec quel utilisateur,

Et en utilisant l'interpréteur Python de mon environnement virtuel (venv).

Ce fichier va permettre à la Raspberry Pi d'exécuter automatiquement mon script dès que le système a fini de démarrer.

Je commence par créer le fichier de service en ligne de commande :

`sudo nano /etc/systemd/system/viticulture.service`

Puis je vais mettre dans ce fichier les instructions nécessaires au lancement du script.

Voici ce que je mets :

```
GNU nano 7.2                               viticulture.service *
[Unit]
Description=Service VITICULTURE - Client MQTT
After=network.target

[Service]
Type=simple
User=olen
WorkingDirectory=/var/www/projet_traitement/viticulture
ExecStart=/home/myPyEnv/mon_env/bin/python /var/www/projet_traitement/viticulture>
Restart=always
Environment=PYTHONUNBUFFERED=1

[Install]
WantedBy=multi-user.target
```

Explication de chaque ligne du fichier :

[Unit] : donne le nom du service et indique qu'il faut attendre que le réseau soit prêt.

User=olen : indique que c'est l'utilisateur olen qui exécute le script.

WorkingDirectory : c'est là où se trouve le script.

ExecStart : c'est la commande exacte à lancer. Ici, on appelle le python du venv.

Restart=always : ton script redémarre s'il plante.

Environment=PYTHONUNBUFFERED=1 : ça permet d'avoir des logs en temps réel.

[Install] : dit que ce service doit se lancer au démarrage (mode multi-user).

Après qu'on est mis en place ce fichier, il ne manque plus que de le mettre en fonction en faisant

sudo systemctl daemon-reload = permet de recharger le service.

sudo systemctl enable viticulture.service = permet de le lancer au démarrage

7.8.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Teste du lancement du code	Je redémarre la Raspberry pi5 et je regarde la base de données des relevés	La base de données affiche de nouvelles données sans que je lance le code par moi-même	Je n'ai pas eu besoin de lancer le code manuellement et de nouvelle valeurs son arrivé dans la base de données	OK

Grâce à l'installation de ce fichier, le code se lance automatiquement sans intervention du vigneron, même en cas de redémarrage de la Raspberry Pi 5 dû à une coupure de courant ou à un autre incident.

8 Intégration

8.1 Partie Web (Aounou et Bastien)

Nous avons travaillé ensemble sur la base de données afin qu'elle puisse être constituée des tables suivantes : capteur, relevé, utilisateur, alerte et sauvegarde. Ensuite, lors de l'intégration sur la Raspberry, nous avons fait face à une difficulté, le format d'écriture n'était pas le même ce qui nous a poussé changer celui du fichier.

Pour utiliser les méthodes liées aux tables de la base de données, il fallait passer par l'API qui fonctionnait comme un contrôleur en MVC. Ainsi, pour que Bastien puisse l'utiliser, un guide a été créé et fournis pour chaque end-point de l'API.

De ce fait, il y avait les en-têtes nécessaires ainsi que toutes les informations commentées qu'il fallait pour utiliser les end-points. Bastien n'avait plus qu'à s'inspirer des fichiers guides pour coder le site en l'utilisant en fonction de ce qu'il voulait faire.

8.2 Envoi des données via (Aounou et Ethan)

Pour utiliser l'API en python Aounou a implémenté des classes ReleveAPI et CapteurAPI qui permettent d'utiliser les fonctions CRUD avec facilité à partir d'un script Python. Ainsi, il suffisait de les instancier et d'utiliser les méthodes get(), post(), put(), delete() pour exécuter les requêtes http correspondantes. Le script peut donc récupérer les capteurs, voir s'ils correspondent à ceux présents, sinon en créer puis envoyer les relevés.

9 Annexes

9.1 Source utiliser

Candidat : Goncalves Ethan

J'ai utilisé différente source pour m'aider dans mes codes :

<https://www.youtube.com/watch?v=eBePt0tAtSw&t=210s> : pour l'installation de mqtt sur ma Raspberry.

<https://www.ibm.com/docs/fr/ibm-mq/9.2?topic=problems-resolving-problem-mqtt-client-does-not-connect> pour essayer de résoudre le fait que je ne récupère pas les données sur ma Raspberry, j'ai voulu regarder si c'était un problème lié à la connexion mqtt.

https://tecadmin.net/resolved-unknown-collation-utf8mb4_0900_ai_ci/ pour régler le problème avec ma base de données.

https://wiki.dfrobot.com/SKU_SEN0575_Gravity_Rainfall_Sensor wiki pour toutes les informations sur le capteur.

<https://www.tderflinger.com/fr/utiliser-systemd-demarrer-application-python-virtualenv> : pour lancer clientmqtt.py au démarrage de la Raspberry Pi5