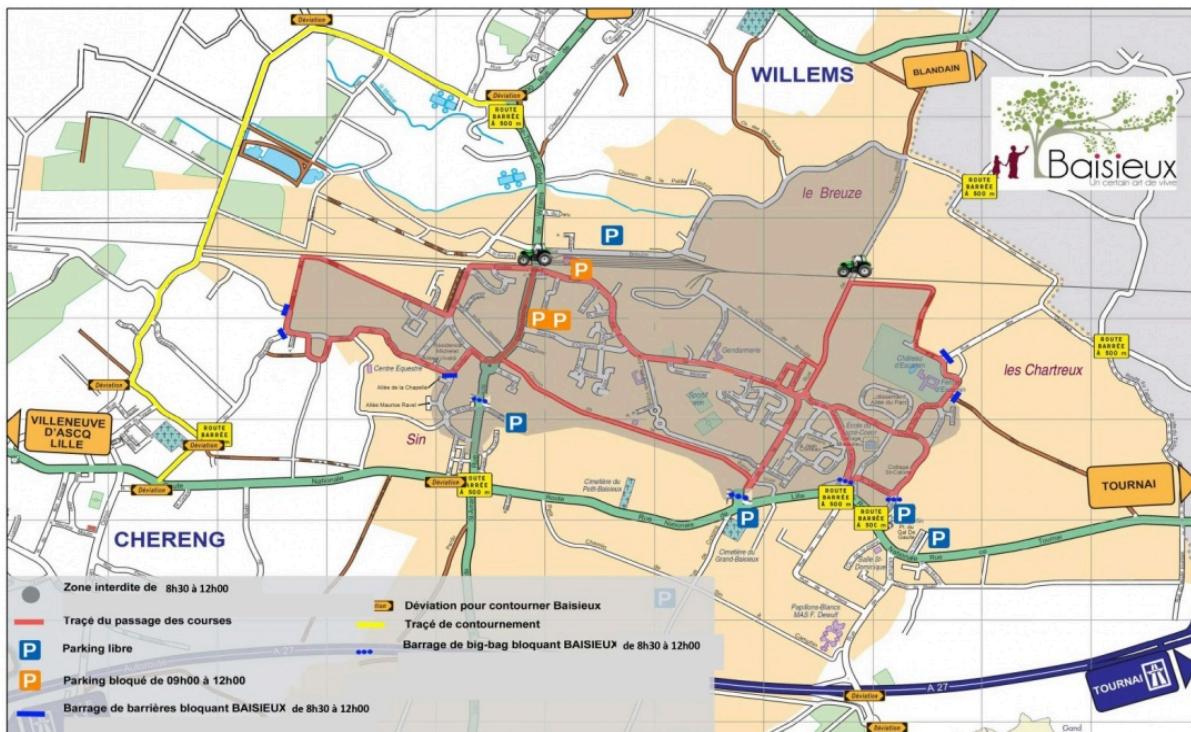


# COURSE DU CHICON

de Baisieux



# Rapport de projet : Gestion et chronométrage courses (Course du Chicon)

BTS CIEL Option A : Informatique et réseaux

Lycée Jean Rostand Roubaix

Professeurs ou Tuteurs responsables :

Serge Cuypers ; Christian Delvarre ; Laurent Neel ; François Pawlaczyk

Équipe chargée du projet :

- étudiant 1 : BOUGHEZAL Abdelrahim (application WEB et service REST)
- étudiant 2 : BOUMEDRAR Mohane (calculateur 'arrivée' et pistolet (ESP))
- étudiant 3 : BELGOUT Anis (application PC organisateur)

## Contexte du projet :

À Baisieux, l'association "LA COURSE DU CHICON" organise depuis plus de 20 ans l'événement éponyme, avec le soutien logistique et financier de la Mairie. Cette manifestation sportive a réuni en 2024 plus de 1000 participants, répartis entre deux courses adultes de 5 km et 10 km, ainsi que trois courses pour enfants (1500 m, 1000 m, 500 m). Pour les courses adultes, le chronométrage est effectué de manière professionnelle par un prestataire spécialisé, tandis que pour les enfants, ce dernier est encore réalisé manuellement.

## Analyse de l'existant :

Pour les adultes, un site web permet les inscriptions en ligne, et le chronométrage est pris en charge par un prestataire qui utilise une solution basée sur la technologie RFID.

Pour les enfants, les inscriptions sont réalisées via les écoles. Chaque participant fournit ses informations personnelles (nom, prénom, date de naissance, numéro de licence, club, numéro de course). Le jour de l'événement, un dossard est attribué à chaque enfant et inscrit sur une fiche, qui est ensuite répertoriée dans un tableau Excel avec les autres informations. À l'arrivée, le numéro de dossard et le temps du participant sont notés dans ce tableau. Les résultats sont ensuite publiés sur le site de l'association.

### Objectif du projet :

L'objectif principal du projet est de **moderniser** la gestion des courses pour enfants en proposant une solution complète, **à faible coût**, pour trois besoins essentiels.

Le premier besoin est d'**informatiser les inscriptions** des enfants. Actuellement, les inscriptions se font par les écoles et sont gérées manuellement, ce qui crée une charge de travail importante. Le projet propose donc une solution d'inscription en ligne pour simplifier cette tâche.

Le deuxième besoin est d'**automatiser le chronométrage des courses**, le temps de chaque participant sera enregistré de manière précise et rapide.

Enfin, le troisième objectif est d'**afficher les résultats** des courses en temps réel sur un écran, afin d'offrir une expérience plus interactive et professionnelle pour les spectateurs et les participants.

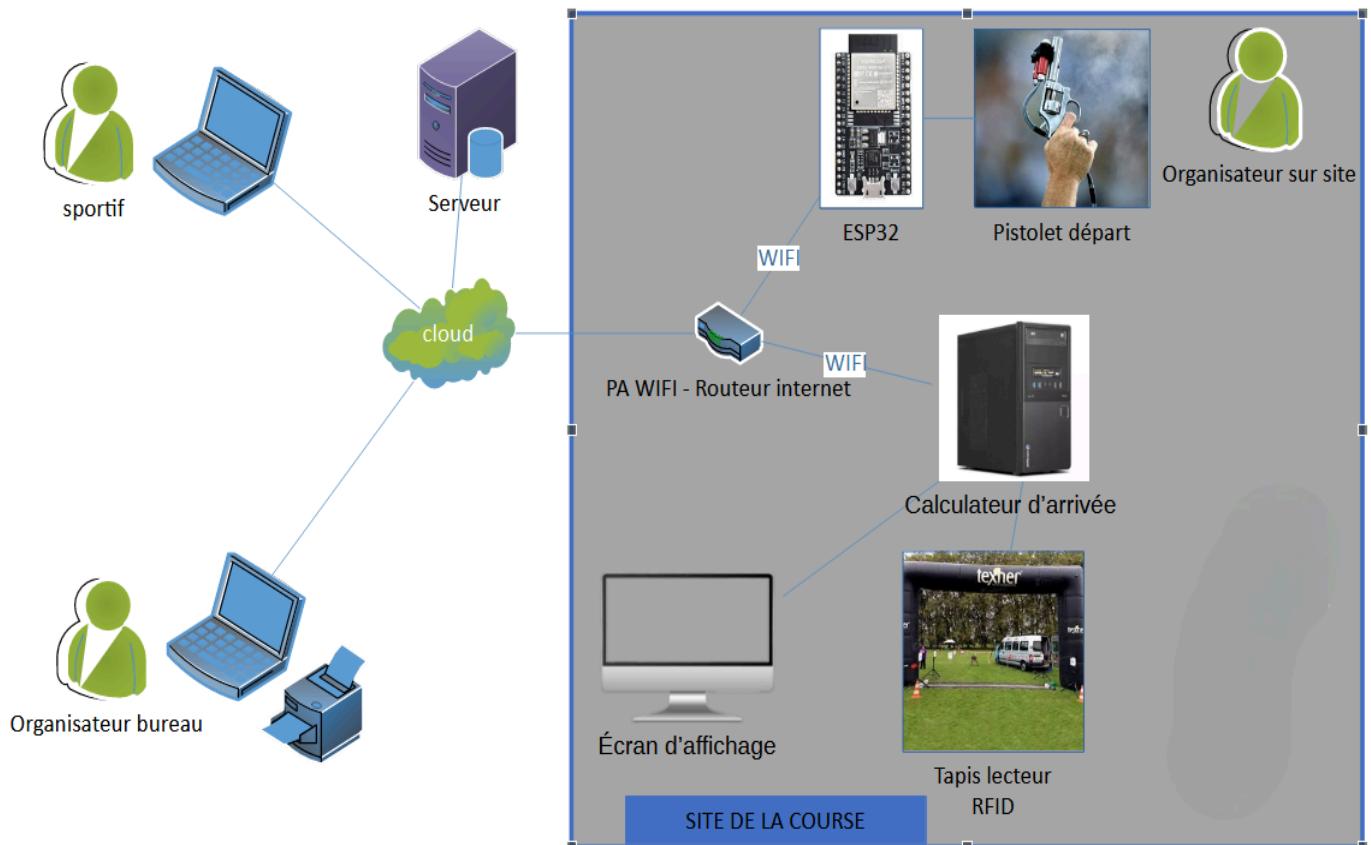


## Sommaire

Partie commune .....	5
Architecture physique.....	5
Espace de travail collaboratif.....	6
Répartition des tâches .....	

## Partie commune

### Architecture physique du système ou de la solution

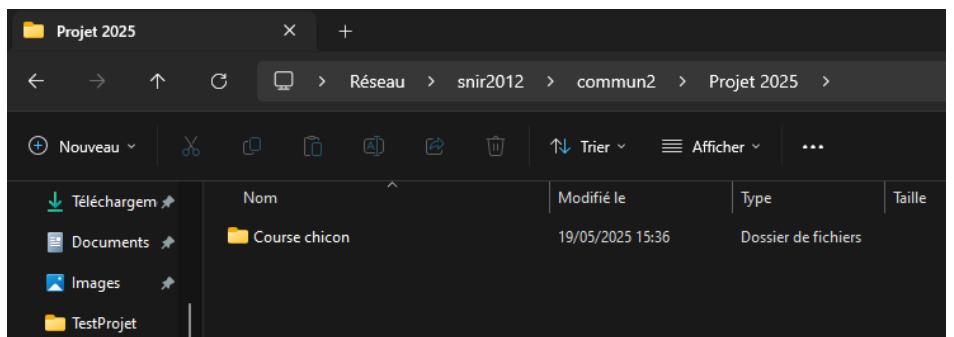


#### Spécification technique :

On impose l'utilisation du lecteur RFID et de son antenne-tapis de sol. On utilisera les Bandeaux Lumineux RS232 disponibles au Lycée ou un écran. Les échanges passent par une couche DAL via des **services REST**.

## Espace de travail collaboratif

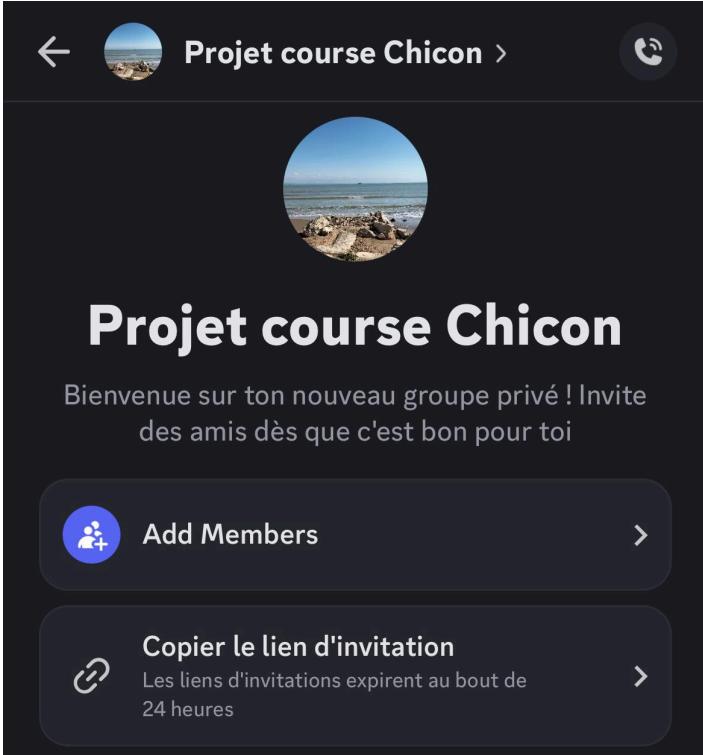
### 1. Google Drive & répertoire réseau



The screenshot shows a file explorer interface. On the left, there is a sidebar with several items: "Protocole MQTT", "chronometrage courses chicon(3étud) 2025", "Abdelrahim", "Mohane", "Anis", and "Rapport de projet". The main area displays a network path: "Réseau > snir2012 > commun2 > Projet 2025 >". Below this, a local folder structure is shown for "Projet 2025":

	Nom	Modifié le	Type	Taille
	Course chicon	19/05/2025 15:36	Dossier de fichiers	
	Documents			
	Images			
	TestProjet			

### 2. Discord



The screenshot shows a Discord server invite page for "Projet course Chicon". The page includes:

- A profile picture of a beach scene.
- The server name: "Projet course Chicon".
- A message: "Bienvenue sur ton nouveau groupe privé ! Invite des amis dès que c'est bon pour toi".
- A button: "Add Members".
- A button: "Copier le lien d'invitation" (Copy invitation link). A note below it says: "Les liens d'invitations expirent au bout de 24 heures".

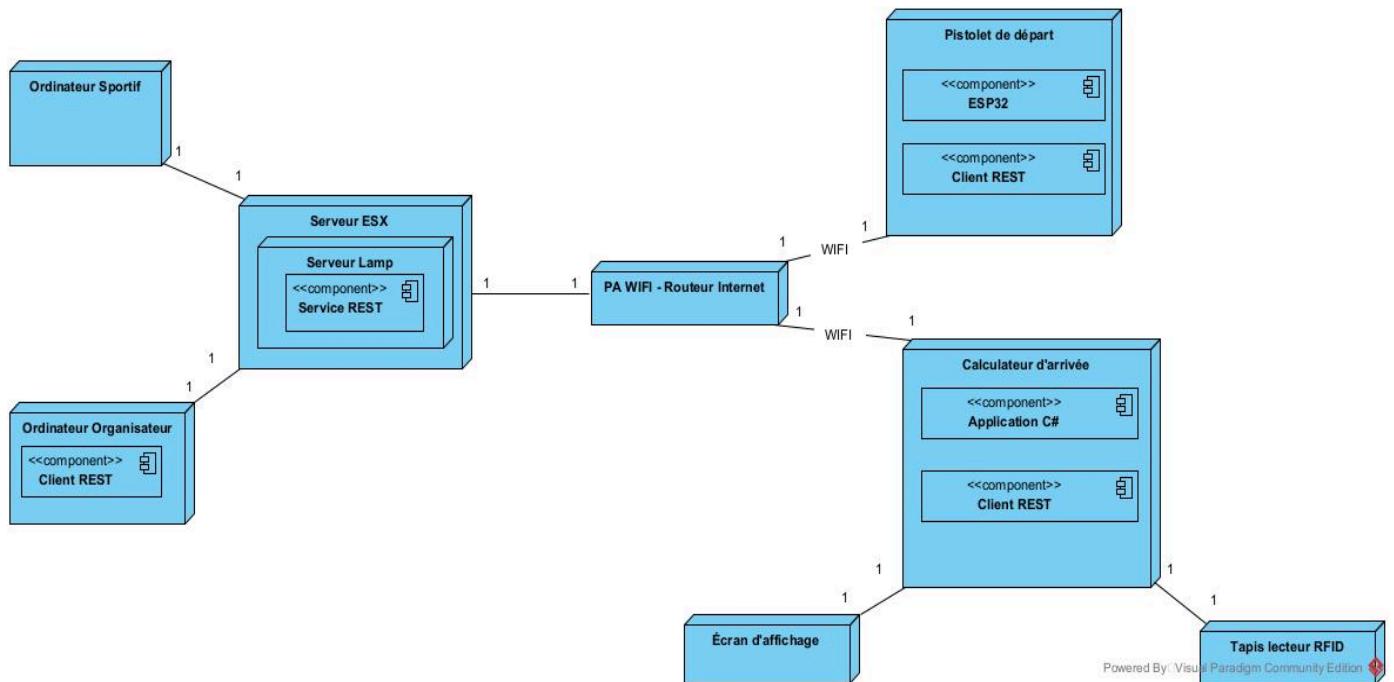
## Répartition des tâches et tâches communes

<b>Tâches communes</b>	durée
Lire, comprendre le dossier, compléter analyse, établir une planification type Gant.	10h
Rédiger le cahier de recette	4h
Préparation revues	10h
Effectuer les tests d'intégration et le <del>recettage</del>	20h
Préparation soutenance et rédaction rapport	16h
<b>Concevoir BDD</b>	<b>8H</b>
<b>Etudiant 1 : travail sur l'application WEB et service REST</b>	
Installation serveur WEB	10h
Choix technologie SW REST et installation serveur (si nécessaire)	5h
Prototyper les écrans utilisateur et proposer une première version de notice utilisateur	5h
Page accueil et présentation de la course.	10h
Page inscription en ligne	10h
Page identification	10h
Page état de l'inscription	10h
Page consultation résultat	10h
Service REST côté serveur	20h
<b>Etudiant 2 : travail sur calculateur 'arrivée' et pistolet (ESP)</b>	
Etude et mise en œuvre lecteur RFID (utilitaire fourni, réglage antenne etc...)	10h
Mise en œuvre API lecteur RFID et création d'un composant logiciel	20h
Mise en œuvre client REST, mise à jour temps chrono participant	20h
Côté Pistolet sur ESP, communication avec calculateur 'arrivée'	20h
Côté calculateur 'arrivée', serveur, communication avec ESP	20h
<b>Etudiant 3 : application PC organisateur</b>	
Prototyper les écrans utilisateur et proposer une première version de notice utilisateur	10h
Mise en œuvre client REST	10h
Création d'une course, visualiser et changer l'état de la course	10h
Visualiser les participants, valider une participation	15h
Bloquer inscription en ligne, affectation des dossards et RFID	15h
Gérer inscription dernière minute	20h
Validation démarrage course	10h

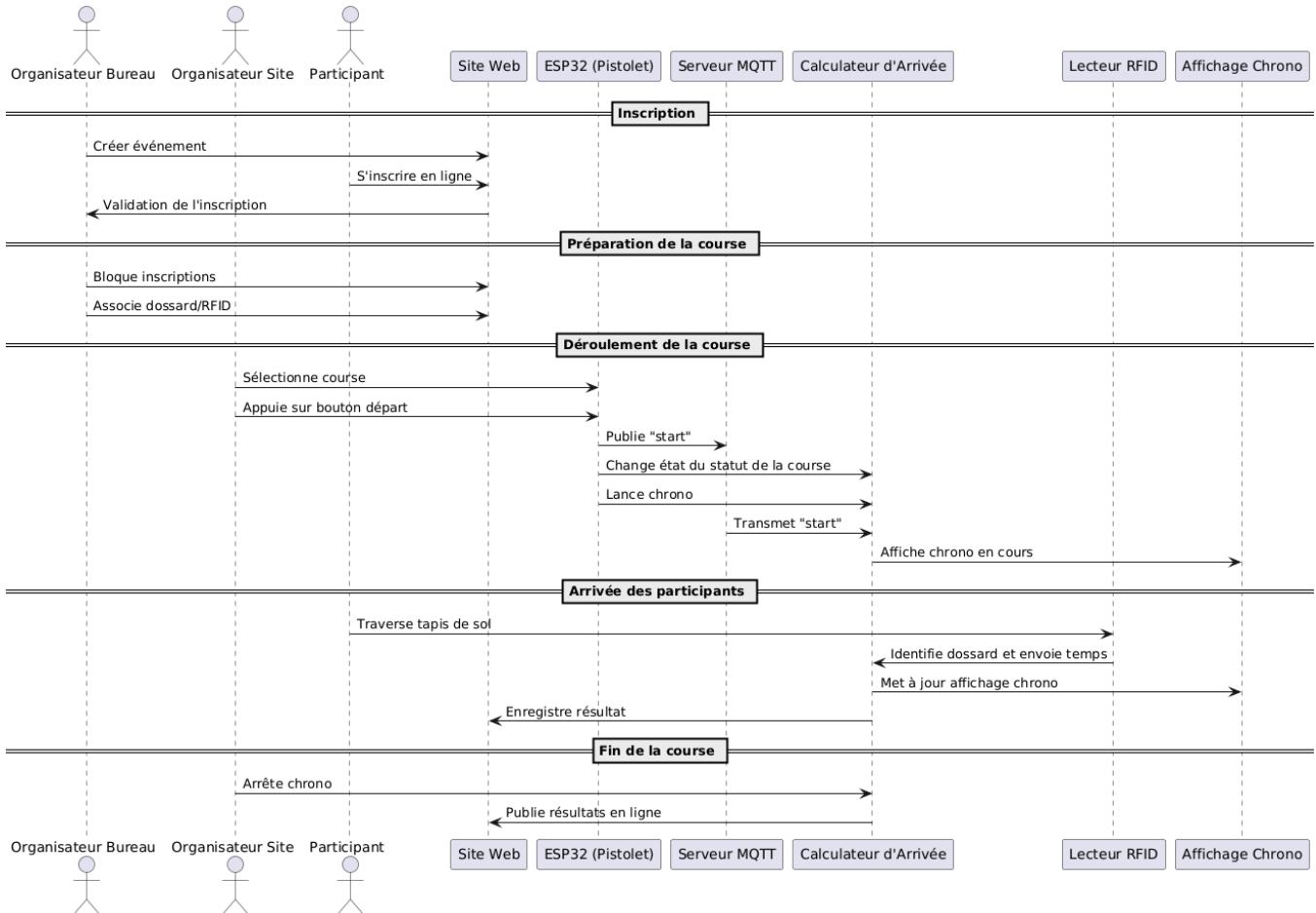
## Diagramme de cas d'utilisation



## Diagramme de déploiement

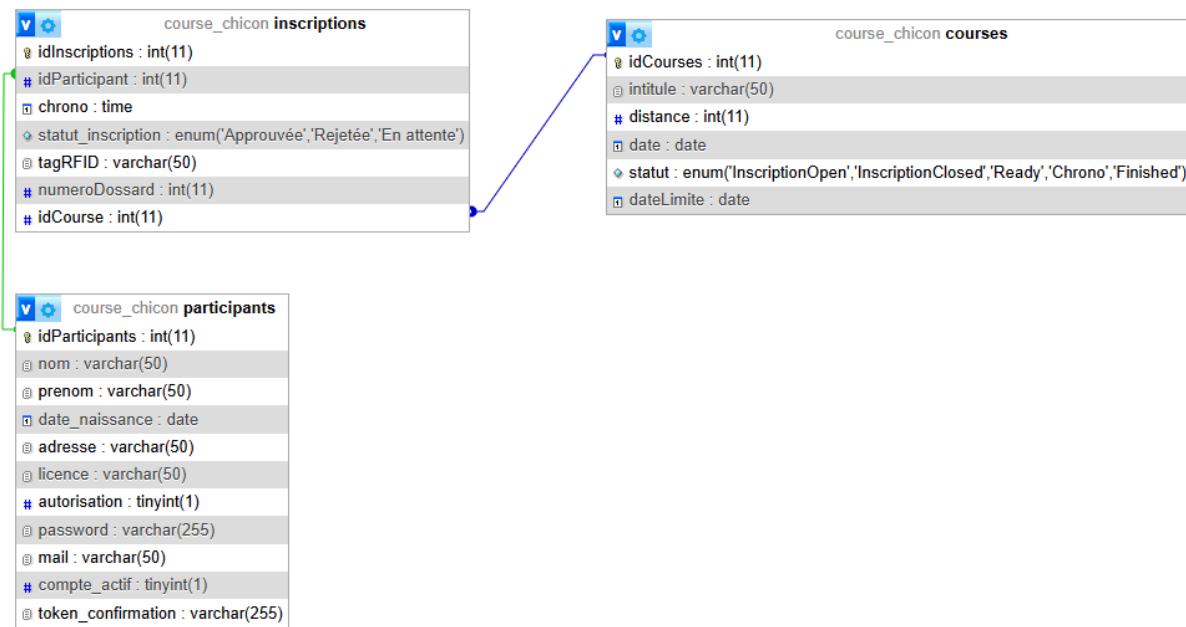


## Diagramme de séquence



## Base de données

Afin de garantir une gestion optimale de la sécurité et d'assurer une structuration logique des données, nous avons créé trois tables distinctes dans la base de données.



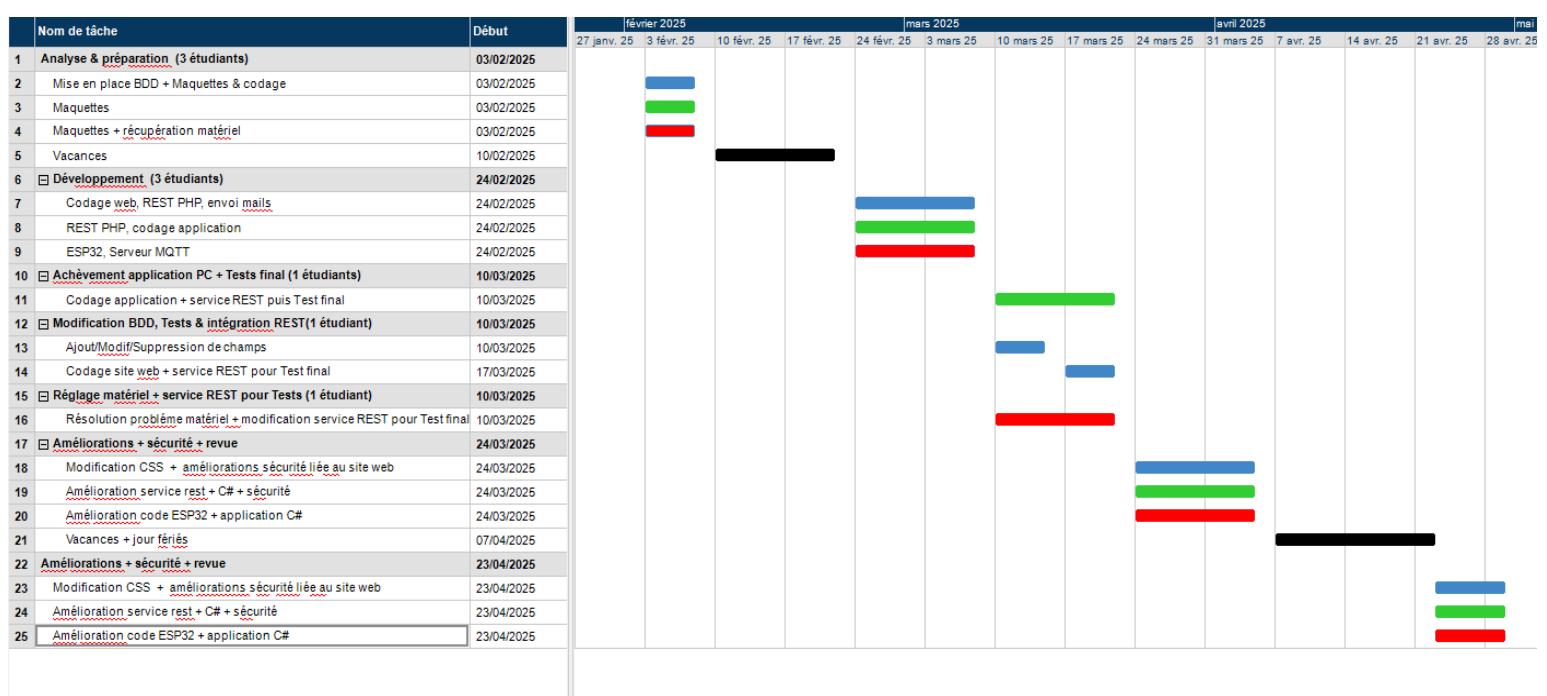
## Étude de coût

Comparaison prix serveur pour héberger base de données

	<b>OVHcloud</b>	<b>Ionos</b>	<b>AWS</b>
Coeurs	2	2	2
RAM (Gb)	4	4	4
Stockage SSD (Gb)	80 à 240	160	20 à 16 Tb
Prix/mois (TTC) (variable)	126,29€ (variable)	5€ pendant 6 mois puis 8€	61,733€ (variable)

## Planning

*Diagramme de gantt*



Fin de la partie commune

## BELGOUT Anis : étudiant 3, application PC organisateur

La partie de mon projet consiste en la création d'une application en **C#** (via Visual Studio) permettant la gestion de courses et d'inscriptions via une interface graphique.

L'application interagit avec une base de données située sur un serveur distant à l'adresse IP 172.16.245.15, en utilisant des services REST pour échanger des données avec le backend. Ces services REST sont responsables de l'accès et de la gestion des informations dans la base de données, qui est hébergée sur PHPMyAdmin.

L'application permet d'accomplir plusieurs tâches, telles que la gestion des dossards/tag RFID, la création de courses et l'inscription des participants.

Les technologies principales utilisées pour ce projet sont le langage **C#** pour la partie application (via **Visual Studio**), **PHPMyAdmin** pour la base de données, et des **services REST en PHP** pour la communication entre l'application et la base de données . Enfin, pour le transfert de fichiers entre le serveur local et le serveur de production, j'ai utilisé **FileZilla**, un client FTP populaire, pour envoyer les fichiers nécessaires au bon fonctionnement de l'application.

### Tâches professionnelles à réaliser pour ma partie :

Prototyper les écrans utilisateur et proposer une première version de notice utilisateur

Mise en œuvre client REST

Création d'une course, visualiser et changer l'état de la course

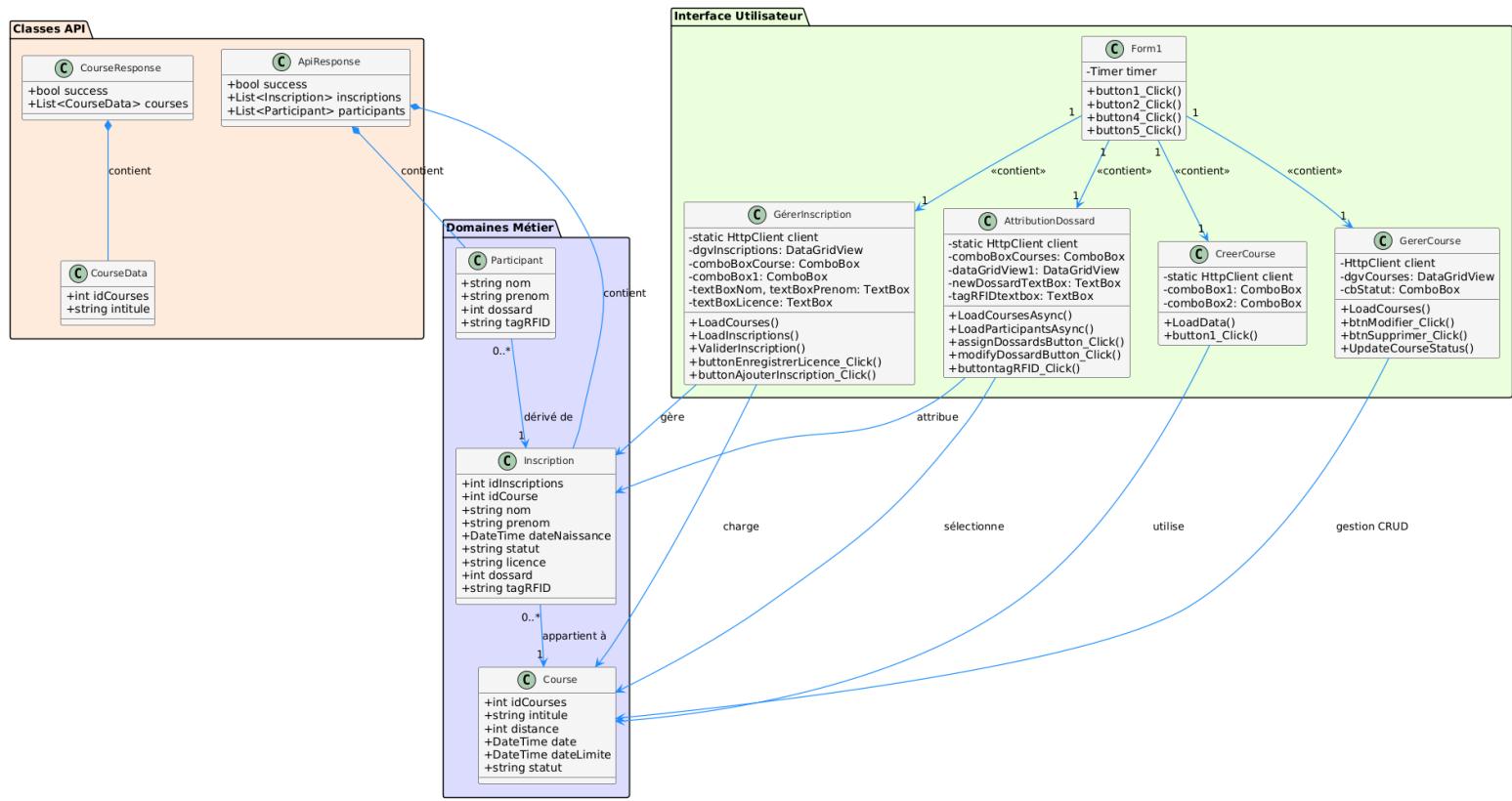
Visualiser les participants, valider une participation

Bloquer inscription en ligne, affectation des dossards et RFID

Gérer inscription dernière minute

Validation démarrage course

Diagramme de classe de l'application



## Écrans : Interface Utilisateur

## Écran principal - Form1.cs

Ce premier écran est le point d'entrée dans l'application. Il propose un menu avec plusieurs options qui permettent à l'utilisateur de créer/gérer les courses et les inscriptions mais également pour les dossards/tag RFID.



## Écran de création d'une course - CréerCourse.cs

Cet écran permet à l'utilisateur de créer une nouvelle course en saisissant les informations nécessaires (Intitulé, Distance, Date de départ de la course et Date limite d'inscription).

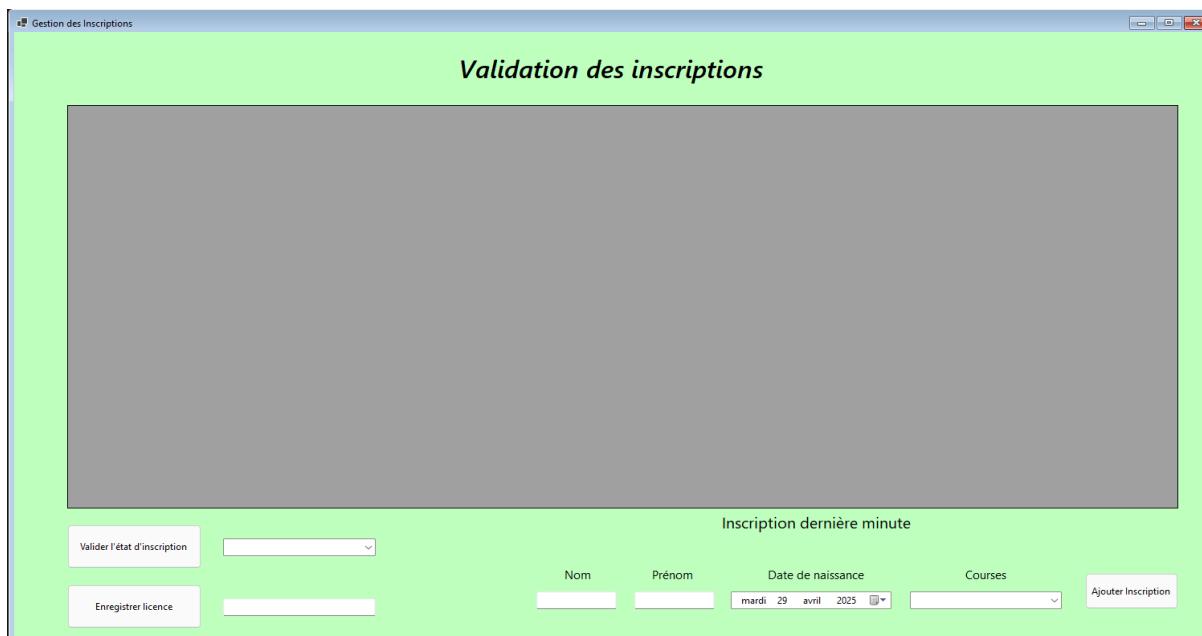
## Écran de gestion des courses - GérerCourse.cs

L'utilisateur peut visualiser les courses existantes, changer leur état (InscriptionOpen, InscriptionClosed, Ready, Chrono, Finished), ou supprimer une course.



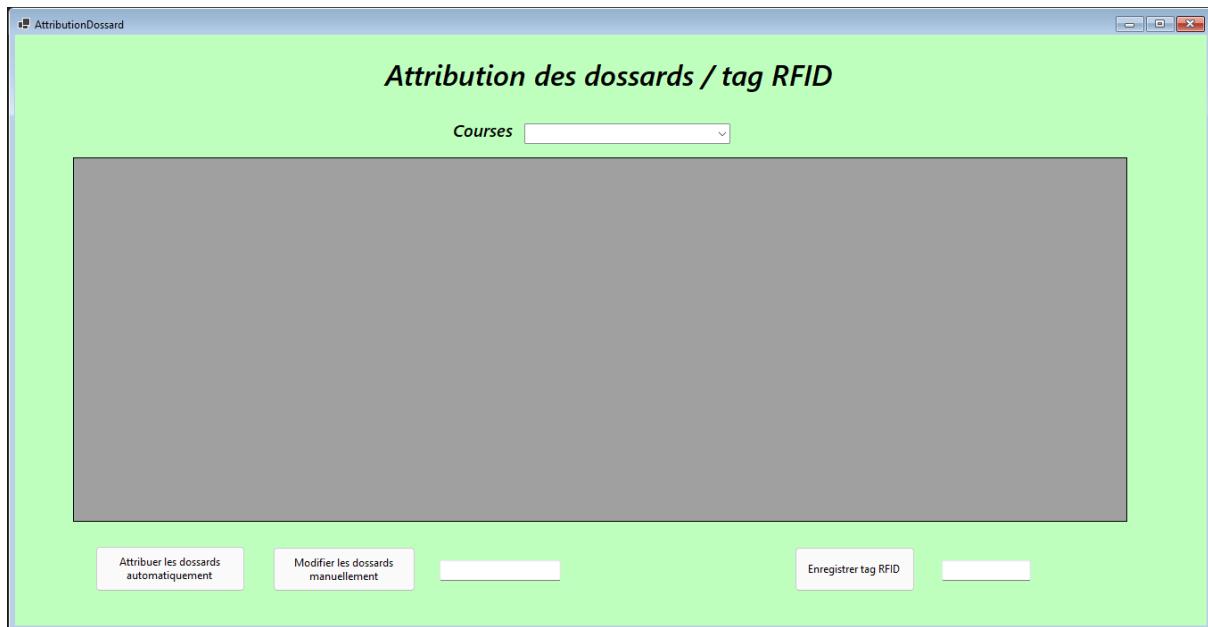
### Écran de gestion des inscriptions - GérerInscription.cs

Cet écran permet de consulter la liste des participants inscrits à une course et de valider ou rejeter leurs inscriptions. Il offre également la possibilité d'enregistrer des licences pour les coureurs. De plus, lorsque la course est en état "InscriptionClosed", cet écran permet d'effectuer des inscriptions de dernière minute.



### Écran de gestion des dossards - AttributionDossard.cs

Cet écran permet d'attribuer ou de modifier les dossards des participants inscrits à une course, ainsi que d'affecter un tag RFID.



### Partie codage

*Dans le cadre de mon projet, j'ai utilisé ChatGPT principalement comme un outil d'assistance pour résoudre certaines problématiques techniques. Toutefois, la grande majorité du travail a été réalisée par mes soins, et ChatGPT m'a uniquement aidé à surmonter quelques obstacles spécifiques en fournissant des suggestions ou en clarifiant des concepts.*

#### 1. CréerCourse.cs :

```
namespace TestProjet
{
    4 références
    public partial class CreerCourse : Form
    {
        private static readonly HttpClient client = new HttpClient(); // On crée un HttpClient static pour faire des appels HTTP vers une API.

        1 référence
        public CreerCourse()
        {
            InitializeComponent();
            LoadData(); // Charger les valeurs dans les ComboBox
        }

        //-----
        1 référence
        private void LoadData() // Remplit les deux ComboBox
        {
            // Remplit le Combobox Intitulé
            comboBox1.Items.Add("Mini-Chicons (6-8 ans)");
            comboBox1.Items.Add("Petits-Chicons (9-10 ans)");
            comboBox1.Items.Add("Grands-Chicons (11-12 ans)");

            // Remplit le ComboBox des distances
            comboBox2.Items.Add("500");
            comboBox2.Items.Add("1000");
            comboBox2.Items.Add("1500");
        }
    }
}
```

CreerCourse, qui est un formulaire permettant à l'utilisateur de définir les paramètres d'une

course. Cette classe inclut un objet HttpClient statique, que j'ai créé pour effectuer des appels HTTP vers une API distante. L'utilisation d'un HttpClient statique permet de réutiliser la même instance tout au long de l'application, ce qui optimise les performances en évitant de recréer cet objet à chaque appel HTTP.

Le constructeur de la classe CreerCourse appelle la méthode InitializeComponent(), qui est générée automatiquement pour initialiser les composants visuels du formulaire. Puis, j'ai utilisé la méthode LoadData() pour remplir les deux ComboBox du formulaire avec des valeurs spécifiques. Cette méthode a pour but de préparer l'interface en offrant à l'utilisateur des options de choix.

Dans la méthode LoadData(), j'ai renseigné les éléments du premier ComboBox (comboBox1), qui contient des groupes d'âge pour les participants à la course. Les choix sont les suivants : "Mini-Chicons (6-8 ans)", "Petits-Chicons (9-10 ans)", et "Grands-Chicons (11-12 ans)". J'ai ajouté ces options à l'aide de la méthode Items.Add(). Ensuite, j'ai rempli le deuxième ComboBox (comboBox2) avec des options de distances pour la course : 500 mètres, 1000 mètres et 1500 mètres, toujours via Items.Add().

Ainsi, cette méthode permet de charger les choix disponibles dans l'interface utilisateur avant que l'utilisateur ne fasse ses sélections. Une fois ces valeurs chargées dans les ComboBox, l'utilisateur peut choisir son groupe d'âge et la distance de la course avant de valider la création de celle-ci dans une étape suivante du processus.

(2/3)

```
private async void button1_Click(object sender, EventArgs e) // Bouton valider la course
{
    // Récupération des valeurs sélectionnées ou saisies par l'utilisateur

    string intitule = comboBox1.SelectedItem?.ToString();
    string distance = comboBox2.SelectedItem?.ToString();
    string date = datePicker1.Value.ToString("yyyy-MM-dd");
    string dateLimite = datePicker2.Value.ToString("yyyy-MM-dd");

    // Vérifie si tous les champs nécessaires sont remplis
    if (string.IsNullOrEmpty(intitule) || string.IsNullOrEmpty(distance))
    {
        MessageBox.Show("Veuillez remplir tous les champs !");
        return;
    }

    // Création de l'objet représentant la course afin de l'envoyer à l'API

    var newCourse = new
    {
        intitule = intitule,
        distance = int.Parse(distance),
        date = date,
        dateLimite = dateLimite,
    };
}
```

Ensuite, j'ai implémenté un événement qui se déclenche lorsqu'un utilisateur clique sur un bouton dans l'interface de l'application Windows Forms. Ce code permet de récupérer les informations saisies par l'utilisateur dans les différents éléments de l'interface et de les

préparer pour un traitement ultérieur, comme un envoi à une API ou un stockage dans une base de données.

Tout d'abord, je récupère les données saisies par l'utilisateur. L'intitulé de la course est sélectionné dans un menu déroulant, où l'utilisateur choisit le nom ou le titre de la course. La distance de la course est également sélectionnée dans un autre menu déroulant. Ces valeurs sont récupérées sous forme de chaînes de caractères, mais je les transforme ensuite en un format numérique, notamment la distance, que je convertis en entier.

Ensuite, je récupère la date de la course à partir d'un calendrier, grâce à un composant DateTimePicker. Cette date est ensuite formatée en une chaîne de caractères au format yyyy-MM-dd. Je fais de même pour la date limite d'inscription, qui est également sélectionnée à partir d'un autre calendrier et formatée de la même manière.

Avant de poursuivre avec le traitement des données, je vérifie que l'utilisateur a bien rempli les champs obligatoires. Si l'un des champs essentiels, comme l'intitulé ou la distance, est vide, un message d'alerte est affiché pour inviter l'utilisateur à remplir tous les champs nécessaires. Si les champs sont remplis correctement, je crée un objet représentant la course avec les informations suivantes : l'intitulé de la course, la distance convertie en entier, la date de la course, et la date limite d'inscription.

Cet objet est alors prêt à être utilisé pour des actions futures, comme l'envoi des données à une API ou leur enregistrement dans une base de données.

(3/3)

```
// Transforme en JSON
string json = JsonConvert.SerializeObject(newCourse);
MessageBox.Show("Données envoyées : " + json);

// Crée un objet StringContent
// la chaîne JSON représentant la course.
// Encoding.UTF8 , l'encodage des caractères, ici en UTF-8
// application/json , le type de contenu
var content = new StringContent(json, Encoding.UTF8, "application/json");

try
{
    // Envoie de la requête POST à l'API avec les données en JSON
    HttpResponseMessage response = await client.PostAsync("http://172.16.245.15/api/CreerCourse/create_course.php", content);

    // Récupération de la réponse brute (texte JSON renvoyé par l'API)
    string responseString = await response.Content.ReadAsStringAsync();

    MessageBox.Show("Réponse brute : " + responseString);

    // Déserialisation de la réponse en objet dynamique pour lire les champs
    dynamic responseJson = JsonConvert.DeserializeObject(responseString);

    if (responseJson.success == true)
    {
        MessageBox.Show("Course ajoutée avec succès !");
    }
    else
    {
        MessageBox.Show("Erreur : " + responseJson.message);
    }
}
catch (Exception ex)
{
    MessageBox.Show("Erreur de connexion : " + ex.Message);
}
```

Puis, après avoir créé l'objet représentant la course, je le sérialise en une chaîne JSON à l'aide de la méthode “JsonConvert.SerializeObject”. Cette chaîne est ensuite affichée à

l'utilisateur pour confirmation. Je crée un objet “StringContent” pour préparer les données à être envoyées à l'API. Je spécifie l'encodage en UTF-8 et le type de contenu comme étant du JSON.

Je tente ensuite d'envoyer les données en utilisant une requête HTTP POST via “HttpClient” à l'URL de l'API. La réponse de l'API est récupérée sous forme de chaîne JSON. Je l'affiche dans un message box pour informer l'utilisateur de la réponse brute de l'API.

Ensuite, je désérialise la réponse JSON en un objet dynamique pour vérifier si la requête a réussi. Si la propriété “success” dans la réponse est “true”, un message indique que la course a été ajoutée avec succès. Si la requête échoue, un message d'erreur est affiché avec le détail de l'échec.

Si une erreur se produit lors de l'envoi de la requête (par exemple, une erreur de connexion), un message d'erreur est affiché pour informer l'utilisateur du problème.

### Exemple avec test :

#### 1. Sélection de la course souhaitée

Création de la course

Intitulé: Mini-Chicons (6-8 ans)

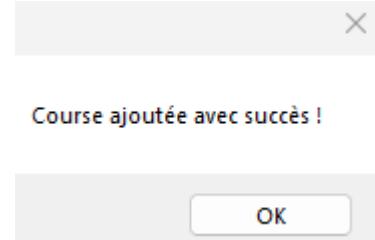
Distance (en mètres): 500

Date de départ de la course: samedi 12 juillet 2025

Date limite d'inscriptions: vendredi 11 juillet 2025

Valider la course

#### 2. Validation de la course



#### 3. La course est bien ajoutée dans la bdd en passant par un service REST

	Éditer	Copier	Supprimer	idCourses	intitule	distance	date	statut	dateLimite
<input type="checkbox"/>				76	Mini-Chicons (6-8 ans)	500	2025-03-25	Ready	2025-03-25
<input type="checkbox"/>				77	Petits-Chicons (9-10 ans)	1000	2025-03-25	Finished	2025-03-25
<input type="checkbox"/>				79	Grands-Chicons (11-12 ans)	1500	2025-03-25	Finished	2025-03-25
<input type="checkbox"/>				92	Mini-Chicons (6-8 ans)	500	2025-04-30	Ready	2025-04-29
<input type="checkbox"/>				97	Mini-Chicons (6-8 ans)	500	2025-07-12	InscriptionOpen	2025-07-11

### 2. GérerCourse.cs

(1/3)

```

<using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Newtonsoft.Json;

namespace TestProjet
{
    4 références
    public partial class GererCourse : Form
    {
        private HttpClient client = new HttpClient();

        1 référence
        public GererCourse()
        {
            InitializeComponent();
            LoadCourses();
        }

        //-----

        // CHARGEMENT DES COURSES
        3 références
        private async void LoadCourses()
        {
            try
            {
                string url = "http://172.16.245.15/api/GererCourse/getCourses.php"; // URL récupère liste des courses
                HttpResponseMessage response = await client.GetAsync(url); // envoie d'une requête GET à l'API pour récupérer les courses

                if (response.IsSuccessStatusCode)
                {
                    string jsonResponse = await response.Content.ReadAsStringAsync(); // Lire le contenu JSON de la réponse de l'API

                    List<Course> courses = JsonConvert.DeserializeObject<List<Course>>(jsonResponse); // Déserialisation du JSON en une liste d'objets de type "Course"

                    dgvCourses.DataSource = courses; // Affecter la liste des courses au DataGridView pour affichage
                    dgvCourses.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
                }
                else
                {
                    MessageBox.Show("Erreur lors du chargement des courses.");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Erreur : " + ex.Message);
            }
        }
    }
}

```

Tout d'abord, je charge les informations des courses en envoyant une requête HTTP GET à une API pour récupérer une liste de courses. La méthode LoadCourses est appelée lors de l'initialisation du formulaire. Elle envoie une requête à l'URL "http://172.16.245.15/api/GererCourse/getCourses.php" et, si la réponse est valide, elle récupère la réponse JSON sous forme de chaîne.

Ensuite, je désérialise cette chaîne JSON en une liste d'objets Course à l'aide de JsonConvert.DeserializeObject<List<Course>>. Cette liste est ensuite affectée au DataGridView (dgvCourses) pour être affichée à l'utilisateur. Je configure le mode de redimensionnement automatique des colonnes du DataGridView pour s'ajuster à la taille des données.

Si la requête échoue, un message d'erreur est affiché pour informer l'utilisateur. En cas d'exception (comme une erreur de connexion), un message d'erreur détaillant l'exception est également montré.

(2/3)

```

// -----
// BOUTON MODIFIER
// référence
private async void btnModifier_Click(object sender, EventArgs e)
{
    if (dgvCourses.SelectedRows.Count == 0) return; // Si aucune ligne n'est sélectionnée dans le DataGridView, on sort de la fonction

    int idCourse = Convert.ToInt32(dgvCourses.SelectedRows[0].Cells["idCourses"].Value); // Récupère l'ID de la course sélectionnée dans le DataGridView
    string newStatut = cbStatut.SelectedItem?.ToString() ?? ""; // Récupère le nouveau statut sélectionné dans la ComboBox

    if (string.IsNullOrEmpty(newStatut)) // Si aucun statut n'est sélectionné, on affiche un message d'erreur et on arrête l'exécution
    {
        MessageBox.Show("Sélectionnez un statut !");
        return;
    }

    await UpdateCourseStatus(idCourse, newStatut);
}

// -----
// BOUTON SUPPRIMER
// référence
private async void btnSupprimer_Click(object sender, EventArgs e)
{
    if (dgvCourses.SelectedRows.Count == 0) return; // Si aucune ligne n'est sélectionnée dans le DataGridView, on quitte la méthode

    int idCourse = Convert.ToInt32(dgvCourses.SelectedRows[0].Cells["idCourses"].Value); // Récupère l'ID de la course sélectionnée dans le DataGridView

    // Affiche une boîte de dialogue de confirmation pour s'assurer que l'utilisateur veut vraiment supprimer la course
    DialogResult result = MessageBox.Show("Voulez-vous vraiment supprimer cette course ?", "Confirmation", MessageBoxButtons.YesNo);

    if (result == DialogResult.No) return;

    try
    {
        string url = $"http://172.16.245.15/api/GererCourse/deleteCourse.php?idCourse={idCourse}"; // Prépare l'URL de l'API pour envoyer la requête DELETE avec l'ID de la course dans l'URL
        HttpResponseMessage response = await client.DeleteAsync(url); // Effectue la requête HTTP DELETE de manière asynchrone

        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Course supprimée !");
            LoadCourses();
        }
        else
        {
            MessageBox.Show("Erreur lors de la suppression.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur : " + ex.Message);
    }
}

```

### Bouton "Modifier"

Le bouton **Modifier** permet de changer le statut d'une course sélectionnée dans le DataGridView. Lorsque l'utilisateur clique sur le bouton, le code vérifie si une ligne a été sélectionnée. Si aucune ligne n'est sélectionnée, la méthode s'arrête immédiatement. Ensuite, l'ID de la course sélectionnée est récupéré et le nouveau statut est extrait de la ComboBox. Si aucun statut n'est sélectionné, un message d'erreur apparaît pour inviter l'utilisateur à choisir un statut. Si un statut valide est sélectionné, la méthode `UpdateCourseStatus` est appelée pour mettre à jour le statut de la course dans la base de données.

### Bouton "Supprimer"

Le bouton **Supprimer** permet de supprimer une course sélectionnée dans le DataGridView. Après vérification qu'une ligne est bien sélectionnée, l'ID de la course est récupéré. Une boîte de dialogue de confirmation s'affiche pour confirmer si l'utilisateur souhaite réellement supprimer la course. Si l'utilisateur confirme la suppression, une requête HTTP DELETE est envoyée à l'API pour supprimer la course de la base de données. Si la suppression est réussie, un message indique que la course a été supprimée, et la liste des courses est rechargée avec la méthode `LoadCourses`. En cas d'erreur, un message d'erreur est affiché, et une gestion d'exception est prévue pour capturer et afficher toute erreur éventuelle.

```
// MISE À JOUR DU STATUT
// Référence
private async Task UpdateCourseStatus(int courseId, string newStatus)
{
    try
    {
        string url = "http://172.16.245.15/api/GererCourse/UpdateCourseStatus.php"; // L'URL de l'API qui va recevoir la requête de mise à jour
        var data = new { idCourse = courseId, statut = newStatus }; // Création d'un objet anonyme contenant l'ID de la course et le nouveau statut
        var json = JsonConvert.SerializeObject(data); // Sérialisation de l'objet en JSON pour l'envoyer dans la requête HTTP
        var content = new StringContent(json, Encoding.UTF8, "application/json"); // Création d'un objet StringContent qui contiendra les données en JSON et les envoie avec le bon encodage (UTF-8)

        HttpResponseMessage response = await client.PostAsync(url, content); // Envoi de la requête HTTP POST à l'API avec les données en JSON

        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Statut mis à jour !");
            LoadCourses();
        }
        else
        {
            MessageBox.Show("Erreur de mise à jour.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur : " + ex.Message);
    }
}
```

### Méthode "UpdateCourseStatus"

La méthode **UpdateCourseStatus** va permettre de mettre à jour le statut d'une course dans la base de données via une API. Elle prend en paramètres l'ID de la course et le nouveau statut. Un objet anonyme est créé avec ces deux informations, puis cet objet est sérialisé en JSON pour pouvoir être envoyé dans la requête HTTP. La méthode utilise `StringContent` pour préparer les données en format JSON avec l'encodage UTF-8, ce qui est nécessaire pour une communication correcte avec l'API.

Ensuite, une requête HTTP POST est envoyée à l'API avec ces données. Si la mise à jour est réussie (si le code de statut HTTP de la réponse est positif), un message indique à l'utilisateur que le statut a été mis à jour, et la liste des courses est rechargée en appelant la méthode `LoadCourses`. Si une erreur survient, un message d'erreur est affiché. En cas d'exception, un message d'erreur avec le détail de l'exception est également montré.

### Exemple d'utilisation :

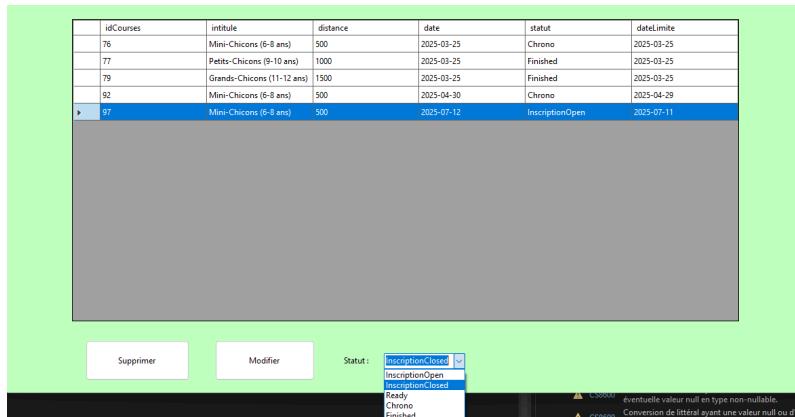
#### 1. Sélectionnez la ligne de la course choisie



#### 2.Modifier le statut souhaité

(exemple : IncriptionOpen → IncriptionClosed)

## 3. Validation du statut



Statut mis à jour !



## 4. Changement dans la base de donnée

Éditer  Supprimer 97 Mini-Chicons (6-8 ans) 500 2025-07-12 IncriptionClosed 2025-07-11

3. GérerInscription.cs

(1/5)

```
4 références
public partial class GérerInscription : Form
{
    private static readonly HttpClient client = new HttpClient();
    1 référence
    public GérerInscription()
    {
        InitializeComponent();
        LoadInscriptions();
        LoadCourses();
    }

    // CHARGE LES COURSES
    1 référence
    private async void LoadCourses()
    {
        try
        {
            string url = "http://172.16.245.15/api/GérerInscription/getCourses.php";
            HttpResponseMessage response = await client.GetAsync(url);

            if (response.IsSuccessStatusCode)
            {
                string json = await response.Content.ReadAsStringAsync();
                var apiResponse = System.Text.Json.JsonSerializer.Deserialize<CourseResponse>(json, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

                if (apiResponse != null && apiResponse.success && apiResponse.courses != null)
                {
                    comboBoxCourse.DataSource = apiResponse.courses;
                    comboBoxCourse.DisplayMember = "intitule"; // Affiche le nom de la course
                    comboBoxCourse.ValueMember = "idCourses"; // Stocke l'ID de la course
                }
            }
            else
            {
                MessageBox.Show("Erreur lors du chargement des courses.");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Erreur : " + ex.Message);
        }
    }
}
```

Premièrement, j'ai implémenté une fonctionnalité pour charger des informations sur des courses depuis l' API et les afficher dans un ComboBox dans le formulaire Windows Forms. Lors de l'initialisation du formulaire, j'appelle la méthode LoadCourses.

Cette méthode envoie une requête HTTP à l'API pour récupérer la liste des courses. Si la requête aboutit, les données renvoyées sont déserialisées en objets C# via JsonSerializer. Ensuite, j'affiche les courses dans le ComboBox, avec le nom de la course comme texte et l'ID de la course comme valeur.

L'objectif de cette approche est de rendre l'application dynamique en récupérant les informations directement depuis l'API, ce qui permet de maintenir facilement les données sans avoir à modifier le code à chaque mise à jour des courses.

(2/5)

```
// CHARGE LES INSCRIPTIONS
4 références
private async void LoadInscriptions()
{
    try
    {
        string url = "http://172.16.245.15/api/GererInscription/getInscriptions.php"; // URL de l'API pour récupérer les inscriptions
        HttpResponseMessage response = await client.GetAsync(url); // Envoi d'une requête HTTP GET à cette URL

        if (response.IsSuccessStatusCode)
        {
            string json = await response.Content.ReadAsStringAsync(); // Lit la réponse en format texte (JSON)

            // Déserialiser en utilisant la classe ApiResponse
            var apiResponse = System.Text.Json.JsonSerializer.Deserialize<ApiResponse>(json, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

            // Vérifier si la réponse est valide et contient des inscriptions
            if (apiResponse != null && apiResponse.success && apiResponse.Inscriptions != null)
            {
                dgvInscriptions.DataSource = apiResponse.Inscriptions;
                dgvInscriptions.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
            }
            else
            {
                MessageBox.Show("Aucune inscription trouvée.");
            }
        }
        else
        {
            MessageBox.Show("Erreur lors de la récupération des inscriptions.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur : " + ex.Message);
    }
}
```

Ensuite, j'ai créé une fonctionnalité pour charger les inscriptions depuis l' API getInscriptions et les afficher dans un DataGridView dans le formulaire Windows Forms.

Lors de l'initialisation, la méthode LoadInscriptions est appelée. Cette méthode envoie une requête HTTP à l'API pour récupérer la liste des inscriptions. Si la requête réussit, la réponse est lue au format JSON et déserialisée en objets C# à l'aide de JsonSerializer. Ensuite, je vérifie si les inscriptions sont présentes et valides.

Si c'est le cas, elles sont affichées dans le DataGridView avec la méthode DataSource. Si aucune inscription n'est trouvée, un message d'erreur est affiché. Si une erreur survient pendant la récupération des données, elle est capturée et un message d'erreur est affiché également. L'objectif de cette fonctionnalité est de permettre à l'utilisateur de visualiser les inscriptions existantes de manière dynamique, directement récupérées depuis l'API.

(3/5)

```

// BOUTON VALIDER STATUT D'INSCRIPTION

1 référence
private void ValiderInscriptionButton_Click(object sender, EventArgs e)
{
    if (dgvInscriptions.SelectedRows.Count == 0) return;

    // Récupérer l'ID de l'inscription sélectionnée
    int idInscription = Convert.ToInt32(dgvInscriptions.SelectedRows[0].Cells["idInscriptions"].Value);

    // Vérifier si un statut a été sélectionné
    if (comboBox1.SelectedItem == null)
    {
        MessageBox.Show("Veuillez choisir un statut pour l'inscription.");
        return;
    }

    string statut = comboBox1.SelectedItem.ToString(); // Récupérer le statut sélectionné

    // Demander confirmation
    DialogResult result = MessageBox.Show("Valider cet état d'inscription ? ", "Confirmation", MessageBoxButtons.YesNo);
    if (result == DialogResult.No) return;

    // Appeler la méthode pour valider l'inscription avec le statut sélectionné
    ValiderInscription(idInscription, statut);
}

// ENVOYER LA MISE À JOUR DU STATUT D'INSCRIPTION À L'API

1 référence
private async Task ValiderInscription(int idInscription, string statut)
{
    try
    {
        string url = $"http://172.16.245.15/api/GererInscription/validerInscription.php?idInscription={idInscription}&statut={statut}";

        // Envoie d'une requête HTTP de type PUT (utilisée ici pour signaler une mise à jour)
        // Pas de contenu dans le corps donc on envoie "null"
        HttpResponseMessage response = await client.PutAsync(url, null);

        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("état d'inscription validée !");
            LoadInscriptions();
        }
        else
        {
            MessageBox.Show("Erreur lors de la validation.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur : " + ex.Message);
    }
}

```

Par la suite, j'ai implémenté une fonctionnalité pour valider le statut d'une inscription. Lorsqu'on clique sur le bouton "Valider Statut d'Inscription", plusieurs étapes sont effectuées. Tout d'abord, je vérifie si une ligne d'inscription est sélectionnée dans le DataGridView. Si aucune ligne n'est sélectionnée, la méthode s'arrête. Ensuite, je récupère l'ID de l'inscription sélectionnée. Si aucun statut n'est choisi dans le ComboBox, un message d'erreur apparaît et la méthode s'arrête.

Avant de valider, j'affiche une fenêtre de confirmation. Si l'utilisateur choisit "Non", rien n'est modifié. Si l'utilisateur confirme, la méthode ValiderInscription est appelée pour envoyer une requête HTTP PUT à l'API, mettant ainsi à jour le statut de l'inscription. Si la mise à jour réussit, un message de succès est affiché et la liste des inscriptions est rechargée.

Cette fonctionnalité permet de mettre à jour facilement les statuts des inscriptions via l'API, tout en garantissant que l'utilisateur confirme bien chaque modification avant de l'appliquer.

(4/5)

```
// ENREGISTRER LICENCE
1 référence
private async void buttonEnregistrerLicence_ClickAsync(object sender, EventArgs e)
{
    if (dgvInscriptions.SelectedRows.Count == 0)
    {
        MessageBox.Show("Veuillez sélectionner une inscription.");
        return;
    }

    int idInscription = Convert.ToInt32(dgvInscriptions.SelectedRows[0].Cells["idInscriptions"].Value);
    string licence = textBoxLicence.Text.Trim();

    if (string.IsNullOrEmpty(licence))
    {
        MessageBox.Show("Veuillez entrer un numéro de licence.");
        return;
    }

    try
    {
        string url = "http://172.16.245.15/api/GererInscription/enregistrerLicence.php";
        var data = new { idInscription = idInscription, licence = licence };
        var jsonContent = new StringContent(System.Text.Json.JsonSerializer.Serialize(data), Encoding.UTF8, "application/json");

        HttpResponseMessage response = await client.PutAsync(url, jsonContent);

        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Licence enregistrée !");
            LoadInscriptions(); // Recharge les inscriptions pour voir la mise à jour
        }
        else
        {
            MessageBox.Show("Erreur lors de l'enregistrement.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur : " + ex.Message);
    }
}
```

Pour continuer, j'ai créé une fonctionnalité pour enregistrer un numéro de licence pour une inscription. Lorsqu'on clique sur le bouton "Enregistrer Licence", plusieurs étapes sont réalisées. Tout d'abord, je vérifie si une inscription est sélectionnée dans le DataGridView. Si aucune inscription n'est choisie, un message d'erreur apparaît et la méthode s'arrête. Ensuite, je récupère l'ID de l'inscription sélectionnée et le numéro de licence saisi dans le TextBox. Si le champ de la licence est vide, un message d'erreur est affiché et l'exécution s'arrête.

Si tout est correct, je crée un objet contenant l'ID de l'inscription et la licence, que je sérialise en JSON. Cet objet est envoyé via une requête HTTP PUT à l'API pour enregistrer la licence. Si l'enregistrement réussit, un message de confirmation est affiché et la liste des inscriptions est rechargée pour refléter la mise à jour. En cas d'erreur lors de l'enregistrement, un message d'erreur est affiché.

Cette fonctionnalité permet d'ajouter facilement une licence à une inscription, tout en garantissant que les données saisies sont valides avant de les envoyer à l'API.

(5/5)

```

// AJOUTER INSCRIPTION DERNIERE MINUTE
1 référence
private async void buttonAjouterInscription_ClickAsync(object sender, EventArgs e)
{
    string nom = textBoxNom.Text.Trim();
    string prenom = textBoxPrenom.Text.Trim();
    DateTime dateNaissance = dateTimePickerNaissance.Value;
    int idCourse = (int)comboBoxCourse.SelectedValue;

    // Valider les champs nécessaires
    if (string.IsNullOrEmpty(nom) || string.IsNullOrEmpty(prenom))
    {
        MessageBox.Show("Veuillez renseigner le nom et le prénom.");
        return;
    }

    try
    {
        string url = "http://172.16.245.15/api/GererInscription/ajouterInscription.php";

        // Préparer les données au bon format
        var data = new
        {
            nom,
            prenom,
            date_naissance = dateNaissance.ToString("yyyy-MM-dd"), // Format correct
            idCourse = idCourse
        };

        // Sérialiser en JSON (Utilisation de Newtonsoft.Json pour éviter les problèmes)
        string jsonData = JsonConvert.SerializeObject(data);
        var jsonContent = new StringContent(jsonData, Encoding.UTF8, "application/json");

        // DEBUG : Vérifier le JSON envoyé
        MessageBox.Show("Données envoyées :\n" + jsonData, "DEBUG");

        // la requête POST
        HttpResponseMessage response = await client.PostAsync(url, jsonContent);

        // Lire la réponse du serveur
        string responseContent = await response.Content.ReadAsStringAsync();

        // DEBUG : Afficher la réponse du serveur
        MessageBox.Show("Réponse du serveur :\n" + responseContent, "DEBUG");

        // Vérifier si la requête a réussi
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Inscription ajoutée avec succès !");
            LoadInscriptions(); // Recharge la liste des inscriptions
        }
        else
        {
            MessageBox.Show("Erreur lors de l'ajout de l'inscription.\nDétails : " + responseContent);
        }
    }
    catch (Exception ex)
    {
        // Capturer et afficher toute erreur
        MessageBox.Show("Erreur : " + ex.Message);
    }
}

```

Puis j'ai implémenté une fonctionnalité pour ajouter une inscription de dernière minute à une course. Lorsqu'on clique sur le bouton "Ajouter Inscription", plusieurs étapes sont réalisées. Tout d'abord, je récupère les données saisies dans les champs du formulaire, comme le nom, le prénom, la date de naissance et la course sélectionnée. Si le nom ou le prénom sont vides, un message d'erreur apparaît et l'exécution s'arrête.

Ensuite, je prépare un objet avec ces données et je m'assure que la date de naissance est formatée correctement. Cet objet est ensuite sérialisé en JSON avec

`JsonConvert.SerializeObject`. Après cela, j'envoie une requête HTTP POST à l'API pour ajouter l'inscription.

Si la requête réussit, un message de confirmation est affiché et la liste des inscriptions est rechargée. Si une erreur survient pendant l'envoi, un message détaillant l'erreur est affiché.

Cette fonctionnalité permet d'ajouter une inscription de manière fluide, tout en garantissant que les données saisies sont valides avant d'être envoyées à l'API.

#### Exemple d'utilisation :

The image consists of three vertically stacked screenshots of a web application interface:

- Screenshot 1:** A registration form titled "Inscription dernière minute". It has four input fields: "Nom" (Guillaume), "Prénom" (Eric), "Date de naissance" (mercredi 15 mai 2019), and "Courses" (Mini-Chicons (6-8 ans)). To the right is a button labeled "Ajouter Inscription".
- Screenshot 2:** A validation step titled "Valider l'état d'inscription". It shows a dropdown menu with two options: "Approuvée" (selected) and "Rejetée".
- Screenshot 3:** A final step titled "Enregistrer licence". It shows a text input field containing "LIC10552".

#### 4. AttributionDossard.cs : (1/ )

```
// Charge la liste des courses et remplit le ComboBox.
1 référence
private async Task LoadCoursesAsync()
{
    try
    {
        // Utilisez l'URL appropriée pour obtenir les courses.
        HttpResponseMessage response = await client.GetAsync("http://172.16.245.15/api/GererCourse/getCourses.php");
        string responseString = await response.Content.ReadAsStringAsync();
        var courses = JsonConvert.DeserializeObject<List<CourseData>>(responseString);

        if (courses != null && courses.Count > 0)
        {
            comboBoxCourses.DataSource = courses;
            comboBoxCourses.DisplayMember = "intitule"; // Affiche le nom de la course.
            comboBoxCourses.ValueMember = "idCourses"; // L'ID de la course.
        }
        else
        {
            MessageBox.Show("Aucune course trouvée.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur lors du chargement des courses : " + ex.Message);
    }
}
```

Premièrement, je fais un appel HTTP à une API pour obtenir la liste des courses disponibles je décode ensuite la réponse au format JSON pour en extraire les données que j'affiche dans un ComboBox cela me permet de charger automatiquement les courses depuis le serveur et de les présenter à l'utilisateur sous forme de liste déroulante où chaque course est identifiée par son nom et son identifiant

```
private async Task LoadParticipantsAsync()
{
    try
    {
        int selectedCourseId = 0;
        // Vérifie le type de SelectedValue.
        if (comboBoxCourses.SelectedValue is int)
        {
            selectedCourseId = (int)comboBoxCourses.SelectedValue;
        }
        else if (comboBoxCourses.SelectedValue is CourseData course)
        {
            selectedCourseId = course.idCourses;
        }
        else
        {
            return; // Si aucune valeur valide, ne rien faire.
        }

        // Envoie une requête GET à l'API avec l'ID de la course pour récupérer les participants inscrits.
        HttpResponseMessage response = await client.GetAsync($"http://172.16.245.15/api/AttributionDossard/get_inscriptions.php?courseId={selectedCourseId}");

        // Lit la réponse JSON sous forme de chaîne.
        string responseString = await response.Content.ReadAsStringAsync();

        if (!response.IsSuccessStatusCode)
        {
            MessageBox.Show($"Erreur HTTP {response.StatusCode}");
            return;
        }

        // Déserialise la réponse JSON en objet ApiResponse contenant la liste des participants
        var apiResponse = JsonConvert.DeserializeObject<ApiResponse>(responseString);
        if (apiResponse != null && apiResponse.success && apiResponse.participants != null)
        {
            dataGridView1.DataSource = apiResponse.participants; // Affiche les participants dans la DataGridView
            dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
            dataGridView1.Refresh();
        }
        else
        {
            dataGridView1.DataSource = null;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur lors du chargement des participants : " + ex.Message);
    }
}
```

Ensuite, je récupère l'identifiant de la course sélectionnée dans le ComboBox puis j'envoie une requête GET à l'API pour obtenir la liste des participants inscrits à cette course je lis la réponse au format JSON et je la transforme en objets C# si les données sont valides je les affiche dans un tableau avec la DataGridView ce qui permet de visualiser facilement les participants liés à la course choisie cela rend l'interface interactive et connectée aux données en temps réel

(3/)

```
// BOUTON ATTRIBUTION AUTOMATIQUE DOSSARD
1 référence
private async void assignDossardsButton_Click(object sender, EventArgs e)
{
    try
    {
        if (comboBoxCourses.SelectedItem is CourseData selectedCourse) // Vérifie si un élément est sélectionné dans le ComboBox des courses
        {
            int courseId = selectedCourse.idCourses; // Récupère l'ID de la course sélectionnée
            var content = new StringContent($"courseId={courseId}", Encoding.UTF8, "application/x-www-form-urlencoded"); // Prépare le contenu
            HttpResponseMessage response = await client.PostAsync("http://172.16.245.15/api/AttributionDossard/assign_dossards.php", content);
            string responseString = await response.Content.ReadAsStringAsync(); // Lit la réponse de l'API (au cas où on voudrait l'utiliser)

            await LoadParticipantsAsync(); // Recharge immédiatement la liste des participants
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur lors de l'attribution des dossards : " + ex.Message);
    }
}
```

Pour continuer , je créer un bouton qui permet de déclencher l'attribution automatique des dossards lorsque l'utilisateur clique sur le bouton je vérifie d'abord qu'une course est sélectionnée puis j'envoie son identifiant à l'API via une requête POST pour lancer l'attribution des dossards après la réponse je recharge immédiatement la liste des participants pour afficher les nouvelles attributions dans le tableau cela permet d'automatiser le processus de manière simple et rapide depuis l'interface.

```
// BOUTON ATTRIBUER DOSSARD MANUELLEMENT
1 référence
private async void modifyDossardButton_Click(object sender, EventArgs e)
{
    try
    {
        if (dataGridView1.SelectedRows.Count > 0 && !string.IsNullOrEmpty(newDossardTextBox.Text))
        {
            // Récupère l'ID de l'inscription sélectionnée
            int inscriptionId = (int)dataGridView1.SelectedRows[0].Cells["idInscriptions"].Value;

            // Convertit le texte du nouveau dossier en entier
            int newDossardNumber = int.Parse(newDossardTextBox.Text);

            // Crée le contenu de la requête POST
            var content = new StringContent($"idInscriptions={inscriptionId}&dossard={newDossardNumber}", Encoding.UTF8, "application/x-www-form-urlencoded");

            // Envoie la requête POST à l'API pour modifier le numéro de dossier
            HttpResponseMessage response = await client.PostAsync("http://172.16.245.15/api/AttributionDossard/update_dossards.php", content);

            // Lit la réponse renournée par l'API
            string responseString = await response.Content.ReadAsStringAsync();

            // Déserialise la réponse JSON pour la lire facilement
            dynamic responseJson = JsonConvert.DeserializeObject(responseString);

            if (responseJson.success == true)
            {
                MessageBox.Show("Dossard modifié avec succès !");
                await LoadParticipantsAsync();
            }
            else
            {
                MessageBox.Show("Erreur : " + responseJson.message);
            }
        }
        else
        {
            MessageBox.Show("Sélectionnez une ligne et entrez un nouveau numéro de dossier.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur de connexion : " + ex.Message);
    }
}
```

Pour modifier manuellement un numéro de dossard, j'ai mis en place un bouton qui permet d'envoyer la nouvelle valeur à l'API lorsque l'utilisateur clique dessus. Je commence par vérifier qu'une ligne est bien sélectionnée dans le tableau et qu'un nouveau numéro de dossard a été saisi dans le champ prévu. Ensuite, je récupère l'identifiant de l'inscription sélectionnée et je construis une requête POST avec cet identifiant ainsi que le nouveau numéro de dossard. Cette requête est envoyée à l'API via une URL dédiée à la mise à jour des dossards. Une fois la réponse de l'API reçue, je la lis et la désérialise pour vérifier si l'opération a réussi. Si c'est le cas, j'affiche un message de confirmation et je recharge immédiatement la liste des participants pour refléter les changements dans le tableau. Sinon, un message d'erreur est affiché. Ce bouton permet donc de modifier rapidement un dossard individuel de manière sécurisée et interactive.

(5/5)

```
private async void buttonTagRFID_Click(object sender, EventArgs e)
{
    // Récupère le texte du champ de saisie du tag RFID, sans les espaces autour
    string tagRFID = tagRFIDtextbox.Text.Trim();

    // Vérifie que le champ n'est pas vide
    if (string.IsNullOrEmpty(tagRFID))
    {
        MessageBox.Show("Le champ Tag RFID est vide !");
        return;
    }

    try
    {
        // Récupère la ligne sélectionnée dans le DataGridView
        var selectedRow = dataGridView1.SelectedRows[0];

        // Récupère l'identifiant de l'inscription à partir de la ligne sélectionnée
        var idInscriptions = selectedRow.Cells["idInscriptions"].Value;

        // Prépare le corps de la requête HTTP
        var content = new StringContent($"idInscriptions={idInscriptions}&tagRFID={tagRFID}", Encoding.UTF8, "application/x-www-form-urlencoded");

        // Envoie la requête POST à l'API PHP pour mettre à jour le tag RFID
        HttpResponseMessage response = await client.PostAsync("http://172.16.245.15/api/AttributionDossard/update_tagRFID.php", content);

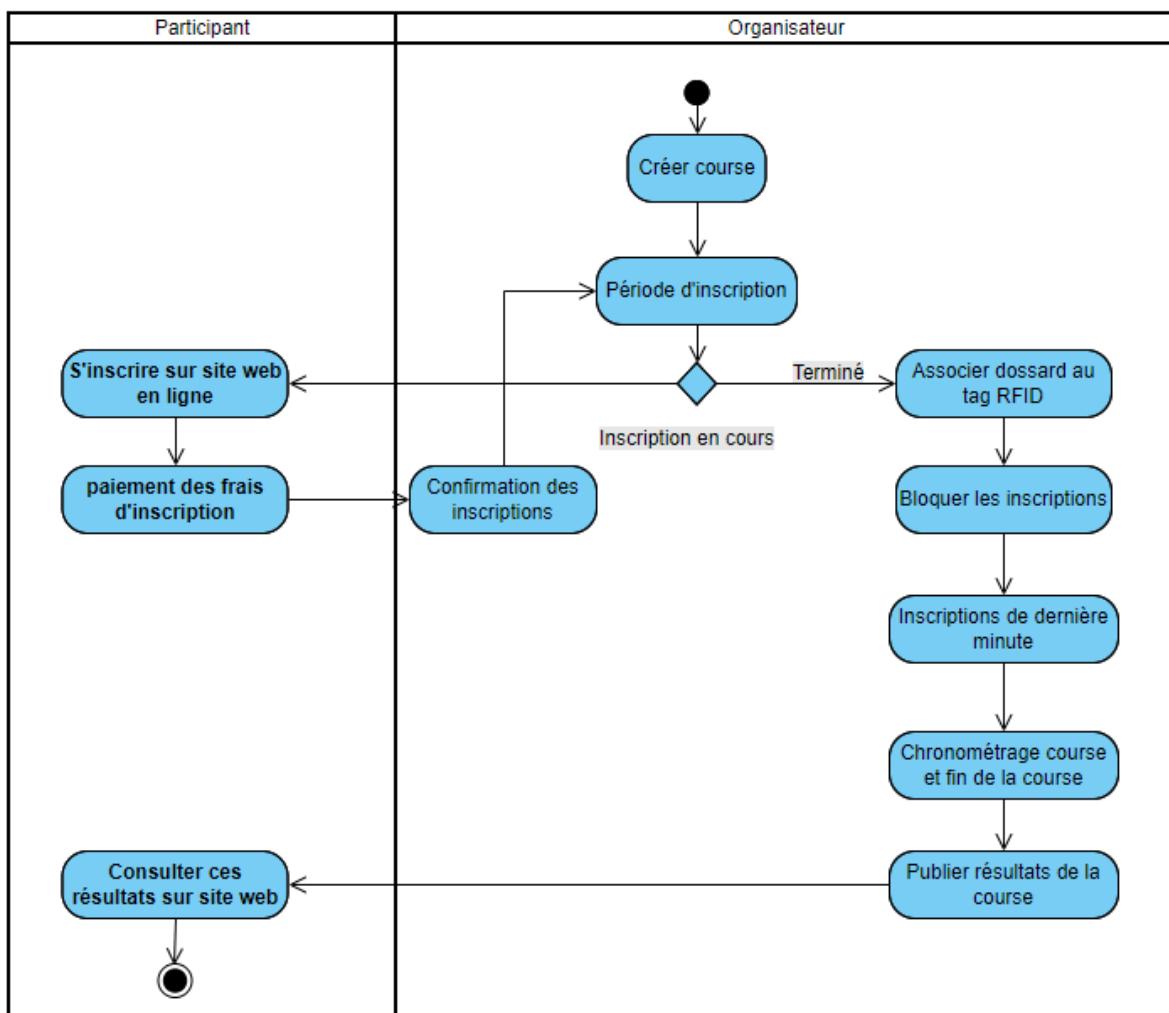
        // Lit la réponse de l'API en tant que chaîne de caractères
        string responseString = await response.Content.ReadAsStringAsync();

        // Convertit la chaîne JSON en un objet dynamique pour lire la réponse
        dynamic responseJson = JsonConvert.DeserializeObject(responseString);

        if (responseJson.success == true)
        {
            MessageBox.Show("Tag RFID mis à jour avec succès !");
            await LoadParticipantsAsync();
        }
        else
        {
            MessageBox.Show("Erreur : " + responseJson.message);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erreur de connexion : " + ex.Message);
    }
}
```

Pour continuer, je crée un bouton qui permet de mettre à jour le tag RFID d'un participant. Lorsque l'utilisateur clique sur ce bouton, je commence par récupérer le contenu saisi dans le champ dédié au tag RFID, en supprimant les éventuels espaces inutiles. Je vérifie ensuite que ce champ n'est pas vide. Si c'est le cas, je récupère la ligne sélectionnée dans le tableau des participants, puis j'en extrais l'identifiant de l'inscription. J'envoie ensuite ces informations (l'identifiant et le tag RFID) à l'API via une requête POST pour mettre à jour le tag dans la base de données. Une fois la réponse reçue, je vérifie si la mise à jour a été effectuée avec succès. Si c'est le cas, j'affiche un message de confirmation à l'utilisateur et je recharge immédiatement la liste des participants pour que les changements soient visibles. Cela permet de gérer les tags RFID de manière simple et directe depuis l'interface.

### Résumé diagramme d'activité

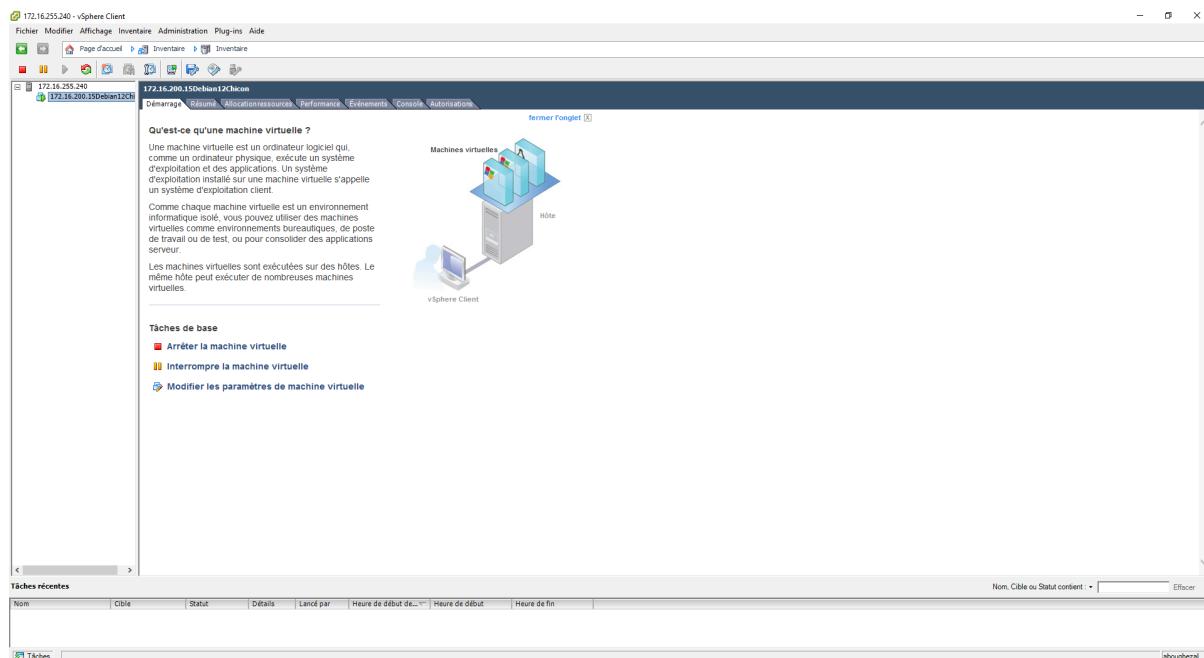


## Abderrahim Boughezal : Étudiant 1, application web et services REST

La partie de mon projet consiste à créer une application web et des services REST afin de gérer la communication avec la base de données. L'application web permet d'enregistrer l'inscription des participants dans la base de données via un formulaire. Les participants doivent ensuite confirmer leur inscription en envoyant leur licence par mail à l'adresse indiquée sur le formulaire d'inscription qui ensuite est vérifiée par l'étudiant 3 (Organisateur bureau). Les participants ont un espace personnel sur lequel ils peuvent vérifier le statut de leur inscription (en attente, approuvée, non approuvée) et sont notifiés par mail du changement du statut de l'inscription. Les **services REST** permettent aux étudiants 2 et 3 (organisateur bureau et organisateur sur site) d'interagir avec la base de données via différentes requêtes **HTTP** (PUT,POST,GET,DEL) sous format **JSON**. Les technologies utilisées pour la création de l'application web sont : le langage **PHP** pour le back-end ainsi que le framework **Bootstrap** pour le front-end et le responsive. J'ai également utilisé l'api **Brevo** et la librairie **PHP-mailer** pour la gestion de l'envoi des mails.

### Installation de la machine virtuelle et du serveur web

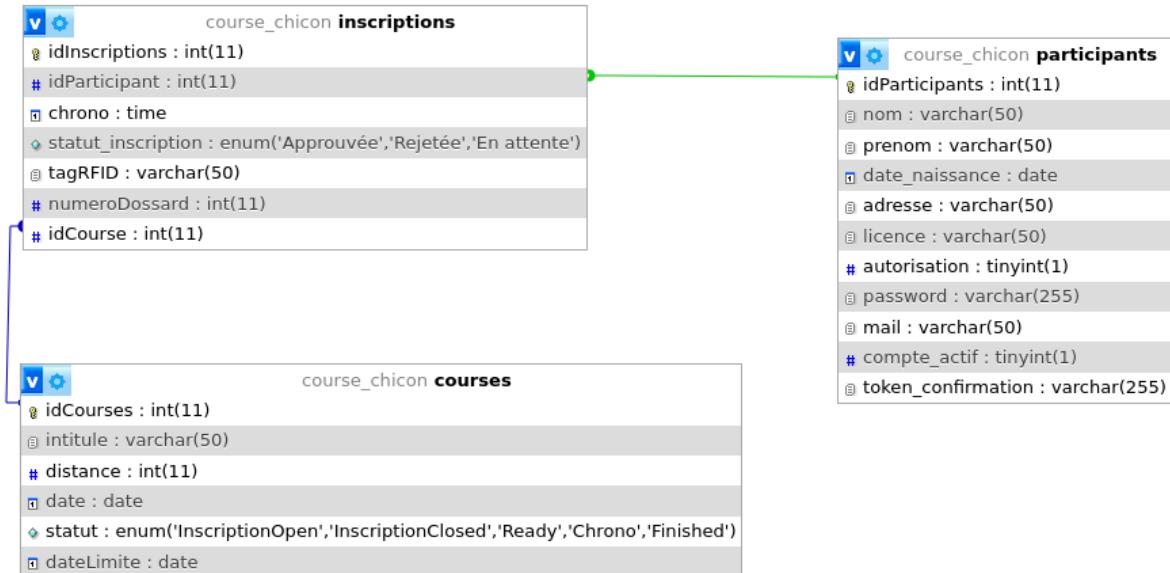
Dans un souci de respect du cahier des charges, qui exige une solution économique pour la gestion des courses, j'ai opté pour l'installation d'une machine virtuelle Debian 12 sur un serveur physique. L'adresse IP de cette machine virtuelle, modifiée en cours de projet, est désormais : 172.16.245.15/24.



Puis j'ai installé en ligne de commandes les paquets nécessaires au serveur web (LAMP), apache2, phpmyadmin et mariadb. afin de pouvoir créer la base de données.

## Création de la base de données

Afin de garantir une gestion optimale de la sécurité et d'assurer une structuration logique des données, j'ai créé trois tables distinctes dans la base de données.



## Maquettage écran

### 1. Page d'accueil

- Rôle : Point d'entrée du site, présentation de l'événement et des principales actions disponibles.
- Éléments clés : Introduction, visuel événementiel, boutons vers l'inscription et les résultats, navigation principale.

### 2. Page d'inscription

- Rôle : Permet à un utilisateur de s'inscrire à la course.
- Éléments clés : Formulaire (informations personnelles, choix de la course), rappel d'envoi de la licence, validation des données.

### 3. Page de connexion

- Rôle : Permettre la connexion à l'espace personnel sécurisé.
- Éléments clés : Formulaire de connexion, lien « Mot de passe oublié ? ».

### 4. Espace personnel (*Mise à jour avec détails*)

- Rôle : Interface dédiée à chaque participant pour suivre et gérer ses inscriptions.
- Éléments clés :
  - Affichage du statut de connexion (connecté/déconnecté, bienvenue, etc.).

- Liste des courses auxquelles l'utilisateur est inscrit avec détails (nom, date, statut d'inscription : en attente, approuvée, non approuvée).
- Possibilité d'annuler une inscription à une course (bouton « Annuler » à côté de chaque course).
- Notifications ou informations si le statut d'inscription change (exemple : email envoyé en cas de validation ou refus).

## 5. Page des résultats

- Rôle : Affichage des classements et performances des participants.
- Éléments clés : Liste filtrable des courses, accès au détail des résultats pour chaque course.

## 6. Page de contact

- Rôle : Permettre aux utilisateurs de contacter l'organisation.
- Éléments clés : Coordonnées, formulaire de contact, carte d'accès à l'événement.

## 7. Page du règlement

- Rôle : Informer sur le cadre légal et les règles de participation.
- Éléments clés : Texte du règlement organisé en articles, accessible à tous.

## 8. Lien RGPD

- Rôle : Accès aux informations sur la gestion des données personnelles, respect des obligations légales.
- Éléments clés : Lien vers la page RGPD.

## Éléments ergonomiques et conformité

- Menu de navigation en haut de page pour accéder rapidement à toutes les sections principales du site : Accueil, Inscription, Contact, Règlement, Parcours, Résultats et RGPD.
- Responsive design : Les maquettes ont été pensées pour une utilisation confortable sur ordinateur, tablette et smartphone.

The screenshot displays the following sections of the website:

- Homepage:** Shows the logo 'LA COURSE DU CHICON' and a brief description of the race.
- Registration Form:** A form titled 'Inscription à la course' with fields for name, address, phone number, email, and race selection.
- Profile Management:** A section titled 'Mon Profil' showing personal information (Name: SOUHEZAN, Address: 123 rue de la Paix, Email: rastimousouhezan@gmail.com, Date of birth: 1999-12-25) and race registrations (Petit-Chicon (9-10 ans) Distance: 1000m, Date: 2025-09-25).
- Connection Page:** A 'Connexion' form with fields for Email and Mot de passe, and a 'Se connecter' button.
- Race Map:** A map titled 'Course du Chicon' showing the route through various neighborhoods.
- Results Table:** A table titled 'Résultats de la Course Petits-Chicons (9-10 ans)' listing participants by rank, bib number, name, and time.
- Rules Section:** A detailed document titled 'Règlement de la Course du Chicon' containing articles from 1 to 6.
- Results Filter:** A sidebar titled 'Résultats des Courses' with filters for race and date.
- Contact Us:** A section with a map and contact details.

## Organisation technique du projet (Arborescence)

L’arborescence du site a été pensée pour séparer de façon claire la logique métier, la gestion des données, les éléments d’interface et les fonctionnalités principales. Voici comment elle est organisée :

## 1. Dossiers principaux

- **api/** : Contient les fichiers liés aux services REST (API), utilisés par les organisateurs pour interagir avec la base de données via des requêtes HTTP.
- **assets/** : Accueille les ressources statiques (images, fichiers CSS, JavaScript, etc.) nécessaires au bon affichage et au design du site.
- **config/** : Stocke les fichiers de configuration (connexion à la base de données, configuration d'email, etc.).
- **dal/** : (Data Access Layer) Contient les classes/fichiers qui gèrent l'accès aux données pour séparer la logique d'accès à la base du reste de l'application.
- **models/** : Décrit la structure des objets métier utilisés dans le site (participant, inscription, course, etc.).
- **PHPMailer/** : Bibliothèque utilisée pour gérer l'envoi d'e-mails (notifications, confirmation d'inscription, etc.).

## 2. Fichiers principaux à la racine

- **.htaccess** : Fichier de configuration Apache, essentiel pour la sécurité et la gestion des URLs.
- **index.php** : Page d'accueil du site, point d'entrée principal.

- navbar.php & footer.php : Composants réutilisables pour la barre de navigation (menu) et le pied de page (footer).
- login.php / logout.php : Gestion de la connexion et de la déconnexion des utilisateurs.
- forgot\_password.php / reset\_password.php : Fonctionnalités pour la récupération et la réinitialisation de mot de passe.
- profile.php : Page espace personnel du participant, avec gestion du profil et suivi des inscriptions.
- inscription\_course.php : Formulaire d'inscription à la course.
- confirmation.php : Page de confirmation d'inscription, affichée après la validation.
- activer.php : Activation du compte via un lien envoyé par e-mail.
- annuler\_inscription.php : Permet à l'utilisateur d'annuler son inscription à une course.
- supprimer\_compte.php : Gestion de la suppression du compte utilisateur.
- contact.php : Page de contact, avec formulaire ou informations d'organisation.
- reglement.php : Affichage du règlement officiel de la course.
- parcours.php : Présentation du parcours de la course.
- resultats.php / consulter\_resultats.php : Accès aux résultats (public ou après recherche/filtrage).

### 3. Organisation logique

- Séparation du code serveur (API, modèles, DAL) et du code affichage (pages PHP pour l'utilisateur).
- Centralisation des composants réutilisables (navbar, footer) pour un site cohérent et facile à maintenir.
- Respect du MVC simplifié : accès aux données dans `dal/`, logique métier dans `models/`, présentation via les fichiers PHP principaux.

```

/
├── api/          (Services API REST)
├── assets/       (Ressources statiques : images, CSS, JS)
├── config/       (Fichiers de configuration)
├── dal/          (Data Access Layer : accès BDD)
├── models/        (Modèles métier)
├── PHPMailer/    (Librairie d'envoi d'e-mails)
├── .htaccess     (Configuration Apache)
├── index.php     (Page d'accueil)
├── navbar.php    (Barre de navigation)
├── footer.php    (Pied de page)
├── login.php     (Connexion)
├── logout.php    (Déconnexion)
├── forgot_password.php (Mot de passe oublié)
├── reset_password.php (Réinitialisation du mot de passe)
├── inscription_course.php (Formulaire d'inscription)
├── confirmation.php (Confirmation d'inscription)
├── activer.php    (Activation du compte)
├── annuler_inscription.php (Annulation d'une inscription)
└── supprimer_compte.php (Suppression du compte)

```

profile.php	(Espace personnel utilisateur)
contact.php	(Page de contact)
reglement.php	(Règlement de la course)
parcours.php	(Présentation du parcours)
resultats.php	(Liste des résultats)
consulter_resultats.php	(Consultation des résultats détaillés)
config_mail.php	(Configuration email)

## Développement des fonctionnalités principales

### Développement de la fonctionnalité d'inscription des participants

L'inscription d'un participant à la course se fait entièrement en ligne via un formulaire sécurisé. J'ai développé ce module pour qu'il soit à la fois convivial pour l'utilisateur, robuste en cas d'erreur, et conforme aux exigences de sécurité du projet. Le processus d'inscription s'appuie sur deux fichiers principaux :

- inscription\_course.php (affichage et gestion du formulaire),
- confirmation.php (traitement des données et gestion de la confirmation).

#### 1. Affichage du formulaire et expérience utilisateur (inscription\_course.php)

L'utilisateur accède à un formulaire où il renseigne ses informations : nom, prénom, date de naissance, adresse, email, mot de passe, choix de la course (récupéré dynamiquement via l'API interne), et accepte éventuellement la publication de ses résultats.

Points clés du formulaire :

- Liste déroulante des courses à jour : Les courses disponibles sont récupérées en temps réel via un appel à l'API interne (`courseServiceOpen.php`). Cela garantit que seules les courses encore ouvertes apparaissent.
- Gestion des erreurs et pré-remplissage : Si l'utilisateur commet une erreur (ex : champ manquant ou mal rempli), le formulaire affiche un message d'erreur et pré-remplit les champs déjà saisis grâce à l'utilisation de variables de session. L'utilisateur ne perd donc pas ce qu'il avait déjà saisi.
- Message de rappel administratif : Un message d'information rappelle à l'utilisateur qu'il doit envoyer sa licence sportive par e-mail à l'adresse **courseduchicon.baisieux @gmail.com** pour que son inscription soit effectivement validée par l'organisation.
- 

#### 2. Traitement serveur, sécurité et validation (confirmation.php)

Lorsque le formulaire est soumis, la logique de vérification et d'enregistrement s'effectue côté serveur :

### a) Récupération des données et contrôles

Chaque champ du formulaire est contrôlé en détail :

- Nom, prénom, date de naissance, adresse, course, email : Le script vérifie que chaque champ est renseigné et correctement formaté.
- Email : Validation de la syntaxe et vérification que l'adresse n'est pas déjà utilisée (consultation en base).
- Mot de passe : Le mot de passe doit comporter au moins 8 caractères, contenir un chiffre et un caractère spécial. Ce sont des critères de sécurité imposés pour renforcer la protection des comptes.

Si une ou plusieurs erreurs sont détectées, elles sont mémorisées en session, l'utilisateur est redirigé vers le formulaire qui affiche précisément les messages à corriger.

### b) Enregistrement du participant et de son inscription

Si toutes les données sont valides :

- Hachage du mot de passe : Le mot de passe est sécurisé avant d'être stocké en base (fonction PHP `password_hash`).
- Création d'un token de confirmation : Génération d'un token aléatoire unique, qui servira à confirmer la création du compte par mail : sécurité et lutte contre les faux comptes.
- Insertion en base de données : Création de l'utilisateur, puis de son inscription à la course choisie (statut mis à "En attente" tant que la licence n'est pas vérifiée par l'organisateur).

### c) Envoi du mail de confirmation

Le participant reçoit automatiquement un email qui contient :

- Un message de bienvenue
- Un lien d'activation personnalisé (avec le token)

Ce lien permet à l'utilisateur d'activer son compte via une page dédiée (`activer.php`), garantissant que l'adresse mail utilisée existe bien.

### d) Gestion des retours à l'utilisateur

- Si tout s'est bien passé, un message de confirmation s'affiche pour prévenir que l'email a bien été envoyé.

## 3. Aspects techniques et sécurité

- Validation des données côté serveur, pour éviter toute triche ou faille due à JavaScript désactivé.
- Gestion propre des erreurs et retours utilisateur via la session, pour une expérience fluide.

- Hashage systématique des mots de passe et utilisation d'un token de confirmation pour éviter les inscriptions frauduleuses.
- Pas de création de compte en double : l'unicité de l'email est vérifiée avant toute insertion.
- Préparation pour la gestion RGPD : consentement explicite sur la diffusion des résultats, et données effaçables par l'organisation.

Code php du script de vérification (page confirmation.php)

```
// Vérification des champs obligatoires et validation du mot de passe
if (isset($_POST['email']) && trim($_POST['email']) != '') {
    $email = trim($_POST['email']);
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $errors[] = "L'email n'est pas valide.";
    } elseif ($participantDAL->checkEmailExists($email)) {
        $errors[] = "Cet email est déjà utilisé." }
if (isset($_POST['password']) && trim($_POST['password']) != '') {
    $password = trim($_POST['password']);
    if (strlen($password) < 8) {
        $errors[] = "Le mot de passe doit contenir au moins 8 caractères.";
    } elseif (!preg_match("/\d/", $password)) {
        $errors[] = "Le mot de passe doit contenir au moins un chiffre.";
    } elseif (!preg_match("/[@$!%*?&]/", $password)) {
        $errors[] = "Le mot de passe doit contenir au moins un caractère spécial." }
// Hashage du mot de passe et génération d'un token unique
$hashedPassword = password_hash($password, PASSWORD_DEFAULT);
$token_confirmation = bin2hex(random_bytes(16));
// Création du participant et insertion en base
try {
    $participant = new Participant();
    // ... (attribution des propriétés, voir code complet)
    $participantDAL->insert($participant);
    // Envoi du mail de confirmation avec lien d'activation
    $mail = new Mail();
    $sujet = "Confirmez votre inscription";
    $message = "Pour activer votre compte, cliquez ici : <a href='http://172.16.245.15/activer.php?token=$token_confirmation'>Activer mon compte</a>";
    $mail->envoyerMail($email, $sujet, $message);
    $_SESSION['confirmation_message'] = "Un email vous a été envoyé pour confirmer votre compte.";
} catch (PDOException $e) {
    $_SESSION['errors'] = ["Une erreur technique est survenue. Veuillez réessayer plus tard."];}

```

Code html d'un champ de saisie:

```
<label for="nom" class="form-label">Nom</label>
<input type="text" name="nom" class="form-control" id="nom"
value="php if (isset($old['nom'])) echo htmlspecialchars($old['nom']); ?&gt;" required&gt;</pre

```

Réponse JSON de du service REST CourseServiceOpen.php

```
1  [
2   {
3     "idCourses": 76,
4     "intitule": "Mini-Chicons (6-8 ans)",
5     "distance": 500,
6     "date": "2025-03-25",
7     "statut": "InscriptionOpen",
8     "dateLimite": "2025-03-25"
9   },
10  {
11    "idCourses": 77,
12    "intitule": "Petits-Chicons (9-10 ans)",
13    "distance": 1000,
14    "date": "2025-03-25",
15    "statut": "InscriptionOpen",
16    "dateLimite": "2025-03-25"
17  }
```

## Mise en place de l'envoi d'e-mails : PHPMailer et Brevo

Afin d'automatiser l'envoi des e-mails dans l'application (confirmation des inscriptions, notifications, etc.), l'application s'appuie sur la bibliothèque PHPMailer combinée au service SMTP de Brevo. La version gratuite de ce service permet l'envoi de 300 mails par jour.

PHPMailer permet de gérer l'ensemble des aspects techniques nécessaires à un envoi conforme aux standards actuels :

- Mise en forme HTML,
- Encodage des caractères (UTF-8),
- Support de l'authentification et des connexions sécurisées,
- Gestion avancée des erreurs.

Fonctionnement dans le projet:

1. Chargement des modules nécessaires
  - Je commence par importer PHPMailer et ses modules ("Exception" et "SMTP") dans ma classe `Mail`.
2. Configuration de l'outil
  - Dans le constructeur de la classe, je configure l'utilisation du serveur SMTP de Brevo.  
Cela consiste à :
    - Définir l'adresse du serveur Brevo
    - Saisir mes identifiants fournis par Brevo (nom d'utilisateur et clé API)
    - Définir le port sécurisé (587) et la protection STARTTLS
3. Rédaction et envoi du mail
  - Quand l'application doit envoyer un email (par exemple lors de l'inscription ou de la confirmation du compte), elle appelle la fonction `envoyerMail`.
  - Cette fonction ajoute le destinataire, le sujet, le contenu HTML du message puis tente d'envoyer le mail via Brevo.
  - Si tout fonctionne, le mail est expédié immédiatement.
  - Si une erreur survient (mauvaise configuration, erreur réseau...), l'utilisateur reçoit un message d'erreur générique et le détail de l'erreur est gardé côté serveur.

Avantages de cette approche

- Les utilisateurs reçoivent plus sûrement leurs e-mails (le passage par Brevo limite l'arrivée en spam).
- Tout est automatisé : l'envoi du mail se fait sans intervention humaine dès qu'une action le nécessite (inscription, changement de statut, etc.).
- Les erreurs sont gérées proprement : l'utilisateur n'a pas accès au détail technique, ce qui sécurise l'application.

Exemple dans le projet

Lorsqu'un nouvel utilisateur s'inscrit, la méthode suivante est appelée :

php

```
$mail = new Mail();
$mail->envoyerMail($email, $sujet, $message);
```

Cela déclenche tout le mécanisme d'envoi : création du message, connexion sécurisée au serveur Brevo, vérification, et remise du mail.

```
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

require '../PHPMailer/src/PHPMailer.php';
require '../PHPMailer/src/Exception.php';
require '../PHPMailer/src/SMTP.php';

class Mail {
    private $mail;

    public function __construct($subject = "Confirmation de votre inscription") {
        $this->mail = new PHPMailer(true);
        // Configuration SMTP Brevo
        $this->mail->isSMTP();
        $this->mail->Host = 'smtp-relay.brevo.com';
        $this->mail->SMTPAuth = true;
        $this->mail->Username = 'xxxxxx@smtp-brevo.com'; // identifiant
        $this->mail->Password = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'; // clé API
        $this->mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
        $this->mail->Port = 587;
        $this->mail->CharSet = 'UTF-8';
        $this->mail->setFrom('courseduchicon.baisieux@gmail.com');
    }

    public function envoyerMail($destinataire, $sujet, $message) {
        try {
            $this->mail->addAddress($destinataire);
            $this->mail->isHTML(true);
            $this->mail->Subject = $sujet;
            $this->mail->Body = $message;
            $this->mail->send();
            return true;
        } catch (Exception $e) {
            return false; // Message d'erreur logué côté serveur
        }
    }
}
```

## Activation du compte utilisateur

L'activation du compte repose sur un lien unique envoyé par e-mail lors de l'inscription. Ce lien contient un token généré aléatoirement et associé au participant.

Lorsque l'utilisateur clique sur ce lien :

- Le script récupère le token passé dans l'URL.
- Vérification en base : le token est recherché dans la table des utilisateurs.
  - Si le token existe, le compte est activé (mise à jour du statut en base de données).
  - L'utilisateur voit un message de succès et est invité à se connecter.

- Si le token n'existe pas (lien déjà utilisé, expiré, ou erroné), un message d'erreur approprié est affiché.
- Sécurité : ce mécanisme garantit que l'adresse e-mail appartient bien à l'utilisateur, évitant ainsi les inscriptions frauduleuses.

Cette étape est indispensable pour valider définitivement une inscription et garantir l'intégrité de la base de données utilisateurs.

```
if (isset($_GET['token'])) {
    $token = $_GET['token'];
    $participant = $participantDAL->getByToken($token);
    if ($participant) {
        // Activation du compte dans la base de données
        $participantDAL->activerCompte($participant->mail);
        $message = "Votre compte a été activé avec succès. Vous pouvez maintenant vous connecter.";
        $alert_class = "alert-success";
    } else {
        $message = "Token invalide ou expiré.";
        $alert_class = "alert-danger";
    } } else {
    $message = "Aucun token de confirmation trouvé.";
    $alert_class = "alert-warning"; }
```

## Fonctionnalité de connexion utilisateur

L'authentification des participants est gérée depuis une page de connexion sécurisée, accessible après activation du compte.

Le participant renseigne son e-mail et son mot de passe, qui sont ensuite vérifiés côté serveur.

## Extrait de code représentant la logique de connexion

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $email = trim($_POST['email']);
    $password = trim($_POST['password']);

    if (!$email || !$password) {
        $message = 'Tous les champs sont obligatoires.';
    } else {
        $participant = $participantDAL->getByEmail($email);
        if ($participant) {
            if ($participant->compte_actif != 1) {
                $message = "Votre compte n'est pas activé.";
            } elseif (password_verify($password, $participant->password)) {
                // Connexion réussie
                $_SESSION['user_id'] = $participant->idParticipants;
                header('Location: profile.php');
                exit;
            } else {
                $message = 'Mot de passe incorrect.'; } } else {
                $message = "Aucun compte trouvé avec cet e-mail."; }}
```

## Explication

- L'utilisateur saisit ses identifiants via un formulaire.
- Le script vérifie si le compte existe et s'il est activé.
- Le mot de passe renseigné est comparé au mot de passe hashé en base grâce à la fonction `password_verify`.
- En cas d'erreur (compte inactif, mauvais mot de passe, compte inexistant), un message adapté est affiché.
- Lors d'une authentification réussie, la session utilisateur est ouverte et l'accès à l'espace personnel est autorisé.

## Réinitialisation du mot de passe (processus "mot de passe oublié")

Pour garantir la sécurité et faciliter la gestion des comptes, la plateforme intègre une fonctionnalité de réinitialisation du mot de passe via un lien sécurisé envoyé par e-mail.

## Schéma du processus

1. Demande de réinitialisation :  
L'utilisateur indique son e-mail sur la page « mot de passe oublié ». Un lien de réinitialisation personnalisé lui est envoyé s'il existe en base.
2. Lien avec token sécurisé :  
Le lien contient un token unique, généré par un HMAC, qui permet de vérifier à la fois l'identité de l'utilisateur et la validité du lien (limitation de durée).
3. Vérification & changement de mot de passe :  
Lorsqu'il clique sur le lien, le script contrôle le token, l'intégrité des données et la validité temporelle. Après validation, l'utilisateur peut saisir un nouveau mot de passe, qui est immédiatement mis à jour en base (après hashage).

Extrait de code condensé (vérification du token et changement de mot de passe)

```
if (isset($_GET['token'])) {
    $decodedToken = base64_decode($_GET['token']);
    list($email, $timestamp, $hmac) = explode('|', $decodedToken);

    if ((time() - (int)$timestamp) < EXPIRATION &&
        hash_equals($hmac, hash_hmac('sha256', $email . '|' . $timestamp, $secretKey))) {
        // Afficher formulaire de changement de mot de passe
        // Après soumission :
        $participantDAL->updatePassword($email, $newPassword);
    } else {
        echo "Lien invalide ou expiré.";
    }
}
```

## Explication

- Le lien transmis par e-mail contient un token sécurisé, limitant la réinitialisation à un temps précis.

- À l'ouverture du lien, le script vérifie que le token est valide, que l'e-mail existe, et que le lien n'a pas expiré.
- Après validation, l'utilisateur peut choisir un nouveau mot de passe, qui sera à nouveau hashé avant d'être stocké.

## Sécurité

- Les mots de passe ne sont jamais stockés en clair : tous sont hashés.
- Les tokens de réinitialisation sont uniques, sécurisés et expirent rapidement pour éviter toute exploitation malveillante.
- Les messages d'erreur restent génériques pour ne pas divulguer d'informations techniques sensibles.

## Présentation de la page "Mon profil" ([profile.php](#))

La page "Mon profil" permet à chaque utilisateur inscrit et connecté de gérer et suivre ses informations et inscriptions de manière autonome. L'accès à cette page est réservé aux personnes authentifiées.

### Fonctionnalités principales

- Affichage des informations personnelles
  - Nom, prénom, email et date de naissance de l'utilisateur sont affichés dans la section supérieure de la page, permettant à chacun de vérifier les données liées à son compte.
- Liste des inscriptions aux courses
  - La page affiche pour chaque inscription :
    - Le nom de la course, la distance et la date.
    - Le statut d'inscription :
      - En attente (badge jaune),
      - Approuvée (badge vert),
      - Non inscrit (badge gris).
    - Un bouton permet à l'utilisateur d'annuler facilement son inscription à une course, après confirmation.
- Actions sur le compte
  - L'utilisateur a la possibilité de :
    - Se déconnecter via un bouton dédié,
    - Supprimer son compte (et toutes ses inscriptions), après une confirmation pour éviter toute suppression accidentelle.

### Sécurité et protection

- L'accès à la page est strictement réservé aux utilisateurs authentifiés (présence de la variable `$_SESSION['user_email']`).
- Les actions sensibles (suppression, annulation) sont protégées par des demandes de confirmation.

**Mon Profil**

**Informations personnelles**

Nom : BOUGHEZAL  
Prénom : Abderrahim  
Email : rahimboughezal@gmail.com  
Date de naissance : 1999-12-28

**Mes inscriptions aux courses**

Course : Petits-Chicons (9-10 ans) Distance : 1000 m Date : 2025-03-25  
Statut : Approuvée

[Annuler l'inscription](#)

[Se déconnecter](#) [Supprimer mon compte](#)

La page “Résultats des courses” permet aux utilisateurs de consulter les classements et temps réalisés sur chaque course, une fois celle-ci terminée.

#### Fonctionnalités principales

- Affichage uniquement des courses terminées
  - Seules les courses ayant le statut “Finished” dans la base de données sont affichées dans la liste. Cela garantit que seuls les résultats officiels et complets sont disponibles à la consultation, évitant toute confusion avec les courses à venir ou en cours.
- Filtre par course
  - Un menu déroulant permet à l’utilisateur de sélectionner une course précise ou de consulter l’ensemble des courses terminées.
  - Lorsqu’une course est sélectionnée, la page met à jour dynamiquement la liste des résultats correspondants. Sinon, toutes les courses terminées sont listées.
- Présentation claire des courses
  - Pour chaque course affichée : le nom, la distance et la date de la course sont indiqués.
  - Un bouton “Consulter” permet d’accéder au détail du classement pour la course choisie.
- Détail des résultats
  - En cliquant sur “Consulter”, l’utilisateur accède à une page dédiée affichant le classement détaillé : forment une table avec le rang, le numéro de dossard, le nom, prénom et le chrono de chaque participant.
  - Ces résultats sont présentés de façon lisible et hiérarchisée, comme le montre la capture d’écran :

*Par respect des règles de confidentialité et conformément au RGPD, la page "Résultats des courses" affiche le nom et le prénom des participants uniquement si ceux-ci ont donné leur consentement lors de l'inscription. Pour les autres, l'information nominative est masquée et remplacée par "XXXXX".*

*Cette approche permet de garantir un classement cohérent et transparent, tout en respectant la volonté et la vie privée de chaque participant.*

RGPD

## Réglementation sur la Protection des Données Personnelles

Dans de nombreux pays, la protection des données personnelles est régie par des lois strictes. Par exemple, en **Europe**, le **Règlement Général sur la Protection des Données (RGPD)** encadre le traitement des données personnelles.

- **Le consentement des participants** : En vertu du RGPD et d'autres lois sur la protection des données, vous devez obtenir le **consentement explicite des participants** pour publier leurs données personnelles (y compris leurs noms et prénoms). Cela peut être fait via une clause dans le formulaire d'inscription à la course, où vous informez les participants que leurs noms et résultats seront publiés publiquement.
- **Légalité et transparence** : Vous devez informer les participants de la manière dont leurs données seront utilisées, pourquoi vous les collectez, et combien de temps elles seront conservées. Cela doit être fait de manière claire et transparente, généralement via une politique de confidentialité.

## 2. But de la Publication des Informations

- Si vous souhaitez publier uniquement les **résultats de la course** (par exemple, les temps et positions des participants), cela peut être plus acceptable et couramment pratiqué, à condition que vous respectiez les règles de protection des données. Cependant, vous devez également veiller à ne pas inclure d'informations sensibles ou inutiles qui ne sont pas pertinentes pour le résultat (par exemple, l'adresse ou le numéro de téléphone).
- Si la publication des noms et prénoms est liée à un **but commercial**, comme la publicité ou la promotion de la course, vous devrez peut-être obtenir un consentement plus explicite ou un contrat avec chaque participant pour l'utilisation de leurs données à des fins commerciales.

## 3. Exceptions au Consentement

Certaines situations permettent la publication de données personnelles sans consentement explicite :

- **Intérêt légitime** : Si la publication des résultats est dans l'intérêt légitime de l'organisateur (par exemple, pour la transparence des résultats d'une compétition

sportive), cela peut être permis, mais il faut en tenir compte avec soin et s'assurer que cela n'entrave pas les droits des participants.

- **Publicité sportive** : Dans le cadre d'événements sportifs, il est souvent acceptable de publier des informations sur les performances des participants (noms, résultats) tant que cela ne va pas à l'encontre de leur vie privée.

#### 4. Conditions Pratiques pour la Publication des Données

- **Formulaire d'inscription** : Incluez une section claire dans le formulaire d'inscription à la course où les participants acceptent explicitement que leurs noms et résultats soient publiés publiquement. Il est préférable d'inclure une case à cocher pour marquer leur consentement.
- **Politique de confidentialité** : Fournissez un accès facile à une politique de confidentialité qui explique comment vous collectez, utilisez, et partagez les informations des participants. Assurez-vous d'inclure des informations sur le fait que les noms et les résultats des participants seront publiés sur le site web, les réseaux sociaux ou tout autre support de l'événement.
- **Retrait des informations** : Permettez aux participants de retirer leur consentement à tout moment avant l'événement si cela est possible, ou en fonction de vos politiques internes.

#### 5. Exemple de texte pour le consentement des participants

Voici un exemple de texte que vous pouvez inclure dans le formulaire d'inscription :

*"En vous inscrivant à cet événement, vous acceptez que vos données personnelles, y compris votre nom et vos résultats de course, soient publiées publiquement sur notre site web et d'autres supports liés à l'événement. Vos données seront utilisées uniquement pour les besoins de cet événement. Si vous ne souhaitez pas que vos informations soient publiées, veuillez nous en informer à l'avance."*

## Boumedrar Mohane : Étudiant 2, Calculateur ‘arrivée’ et pistolet (ESP)

Ma contribution au projet a porté sur deux éléments techniques essentiels du système de chronométrage :

- Un **pistolet de départ connecté**, basé sur un **ESP32**, utilisé par l'organisateur pour lancer la course. Lors de l'appui, l'ESP32 envoie une requête HTTP à l'API REST qui déclenche le départ, met à jour le statut de la course, et démarre le chronomètre qui apparaît sur un écran géant.
- Un **calculateur d'arrivée** utilisant un **lecteur RFID R640** relié à une **antenne-tapis de sol**, connecté au réseau local. Lorsqu'un coureur franchit la ligne d'arrivée, son badge RFID est détecté et son chrono est transmis automatiquement à la base de données via des **services web (API REST)**. Le serveur enregistre alors l'heure d'arrivée dans la base de données.

Les **technologies utilisées** sont :

- Client Lourd en C# pour gérer Le RFID et Le chronomètre
- Lecteur **RFID R640** (Ethernet),
- **Antenne-tapis RFID**,
- Microcontrôleur **ESP32** (programmation en C++ via Arduino IDE),
- **API REST** pour la communication avec le serveur,
- Requêtes **HTTP POST/GET** pour l'envoi des données.

Pendant le projet, j'ai réalisé plusieurs tests pour vérifier la détection stable des badges, le bon fonctionnement du pistolet, et la réception correcte des données sur le serveur.

J'ai également utilisé **ChatGPT** comme outil d'assistance pour m'aider à structurer certaines requêtes HTTP, comprendre des exemples en JSON et optimiser le code. Toutes les décisions et la mise en œuvre technique ont été faites par moi-même.

### Tâches professionnelles réalisées

Installer et configurer le lecteur RFID R640

Gérer la communication réseau et les requêtes vers l'API

- 
- Développer le code ESP32 pour le pistolet (départ)
  - Envoyer les données UID et les horodatages vers la base
  - Tester le système en conditions réelles
  - Collaborer avec les camarades (web et PC organisateur)

## Présentation

1) ESP32 et