

PLOTS LUMINEUX

# RAPPORT DE PROJET PERSONNEL

Jordan - Killian - Marceau - Charly - Enzo

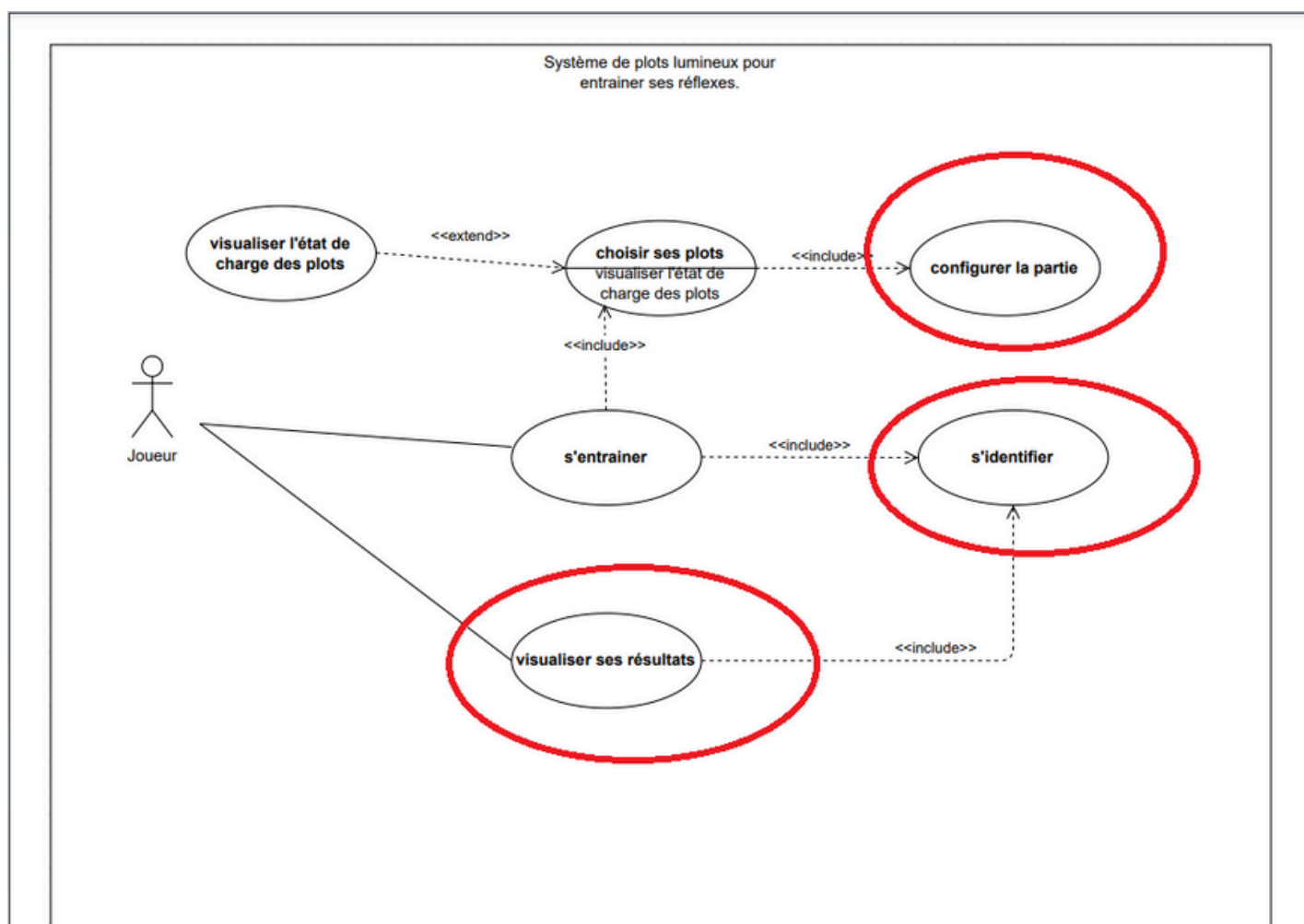
# SOMMAIRE

Introduction	1	Page 01/02
Différentes vues de l'application	2	Page 03/04/05/06/07
Base de données	3	Page 8
API	4	Page 9/10/11/12
Code c++ et QML	5	Page 13/14/15/16/17/18
Mise en commun / Etat d'avancement	6	Page 19/20
Conclusion	7	Page 21

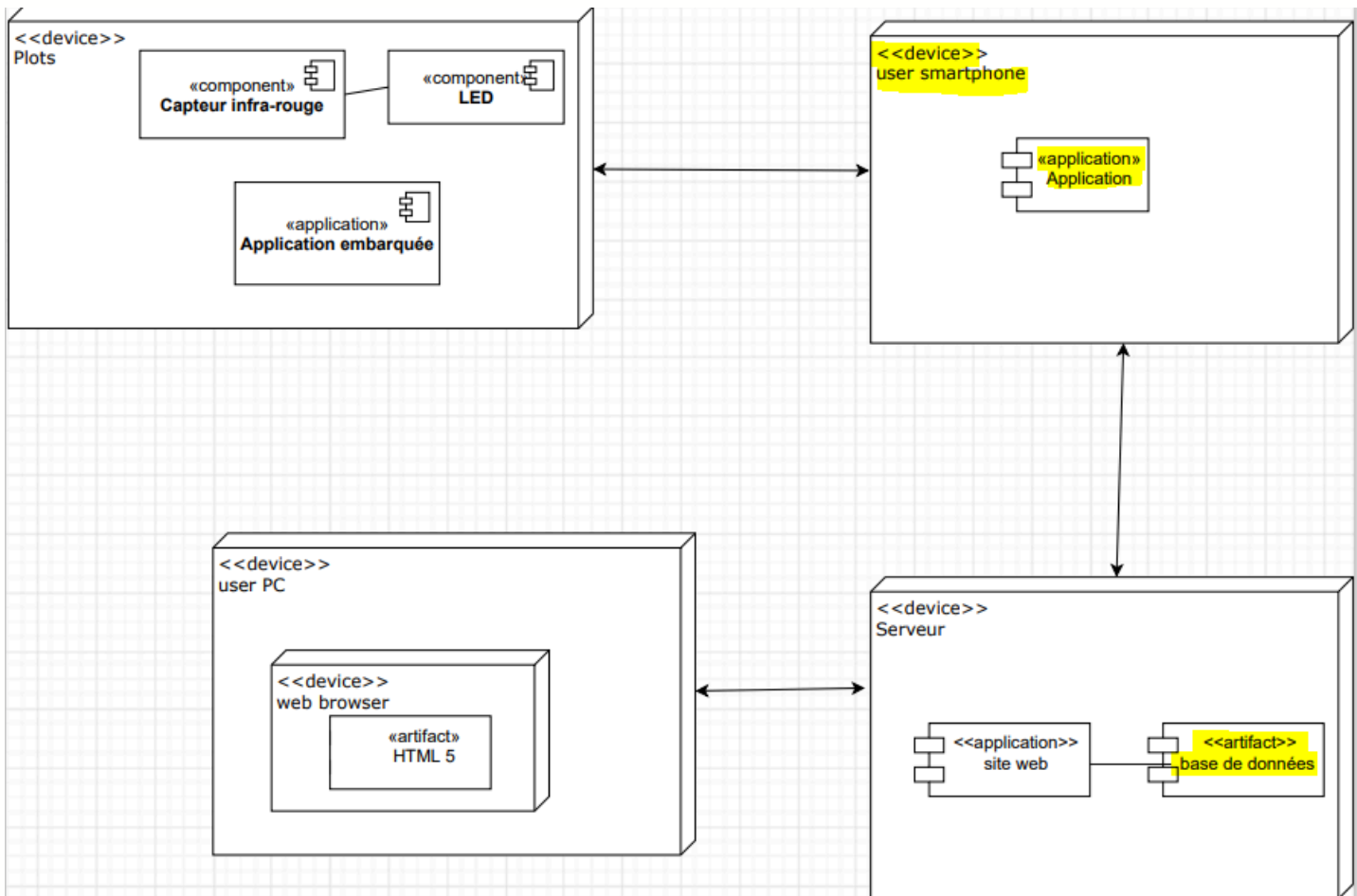
Dans ma contribution à la partie du projet qui était de ma responsabilité, j'ai pris en charge le développement de l'interface utilisateur. Cela comprenait la mise en place des fonctionnalités permettant aux utilisateurs de se connecter s'ils possédaient déjà un compte ou de s'inscrire s'ils étaient nouveaux. En outre, j'ai assumé la responsabilité de la logique nécessaire pour afficher les résultats des joueurs ainsi que pour configurer la partie.

Les connaissances que j'ai acquises m'ont permis de les partager avec les autres membres de l'équipe, favorisant ainsi une meilleure compréhension globale du projet.

J'ai été disponible pour aider mes collègues en cas de difficultés ou de manque de compréhension, offrant ainsi mon assistance pour garantir une progression fluide du projet.



Côté déploiement, voici les parties sur lesquelles j'ai travaillé :



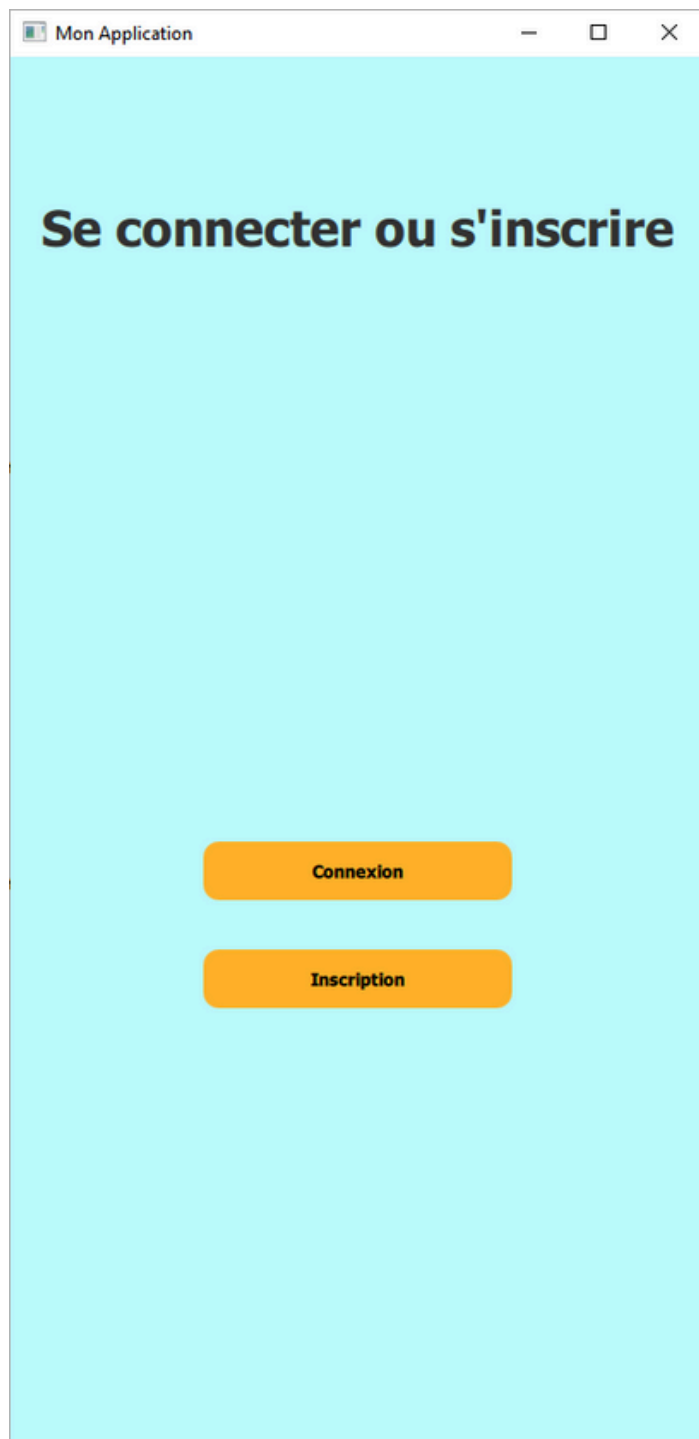
1 : Création de la base de données avec l'étudiant 5 pour savoir quelles données vont être importantes.

2 : Création de l'API qui va permettre de relier l'application à la base de données.

3 : Création du design et des fonctionnalités de l'application.

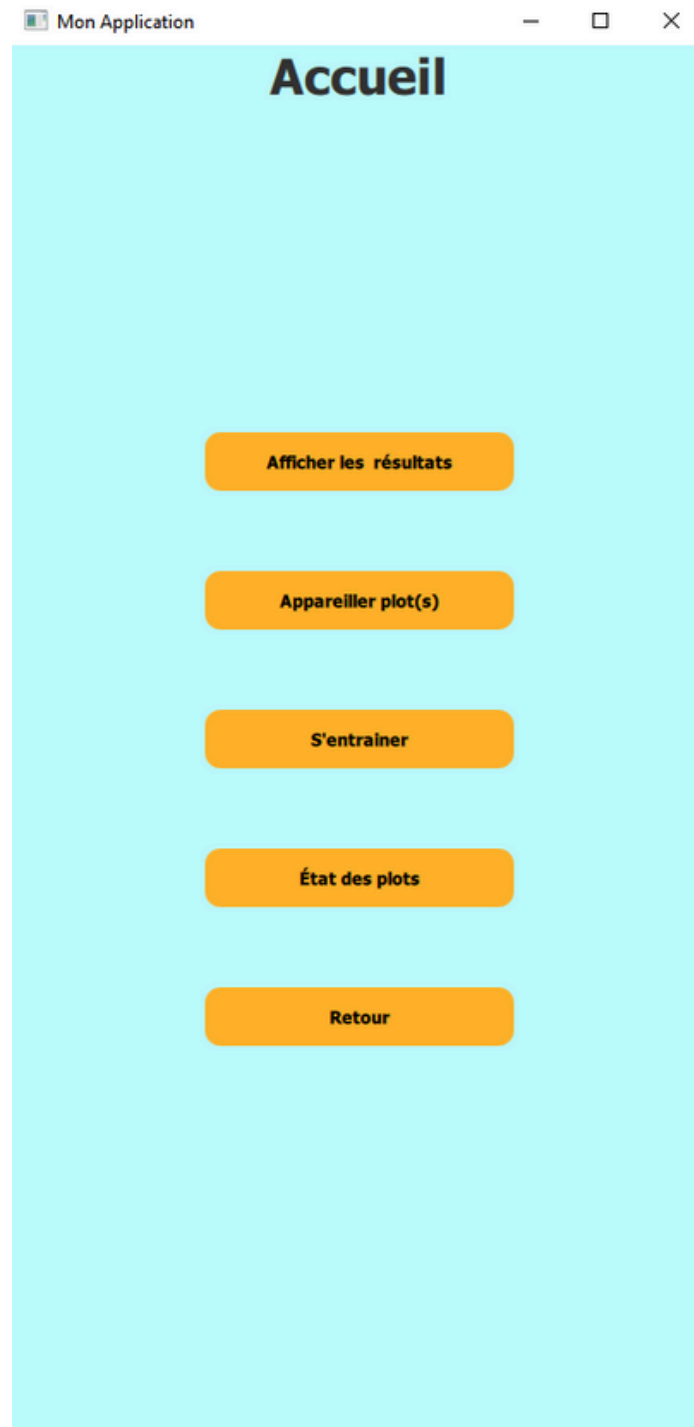
### 2.1 Connexion/Inscription

Je précise avant tout que l'intégralité des vues présentes dans ce rapport sont issus du QML, le langage permettant la création d'interface graphique en C++.



Ici on peut retrouver l'interface qui permet à l'utilisateur de s'inscrire ou se connecter au choix.

### 2.2 Accueil de l'application



Une fois inscrit ou connecter, l'utilisateur est redirigé vers la page d'accueil avec plusieurs possibilités qui sont :

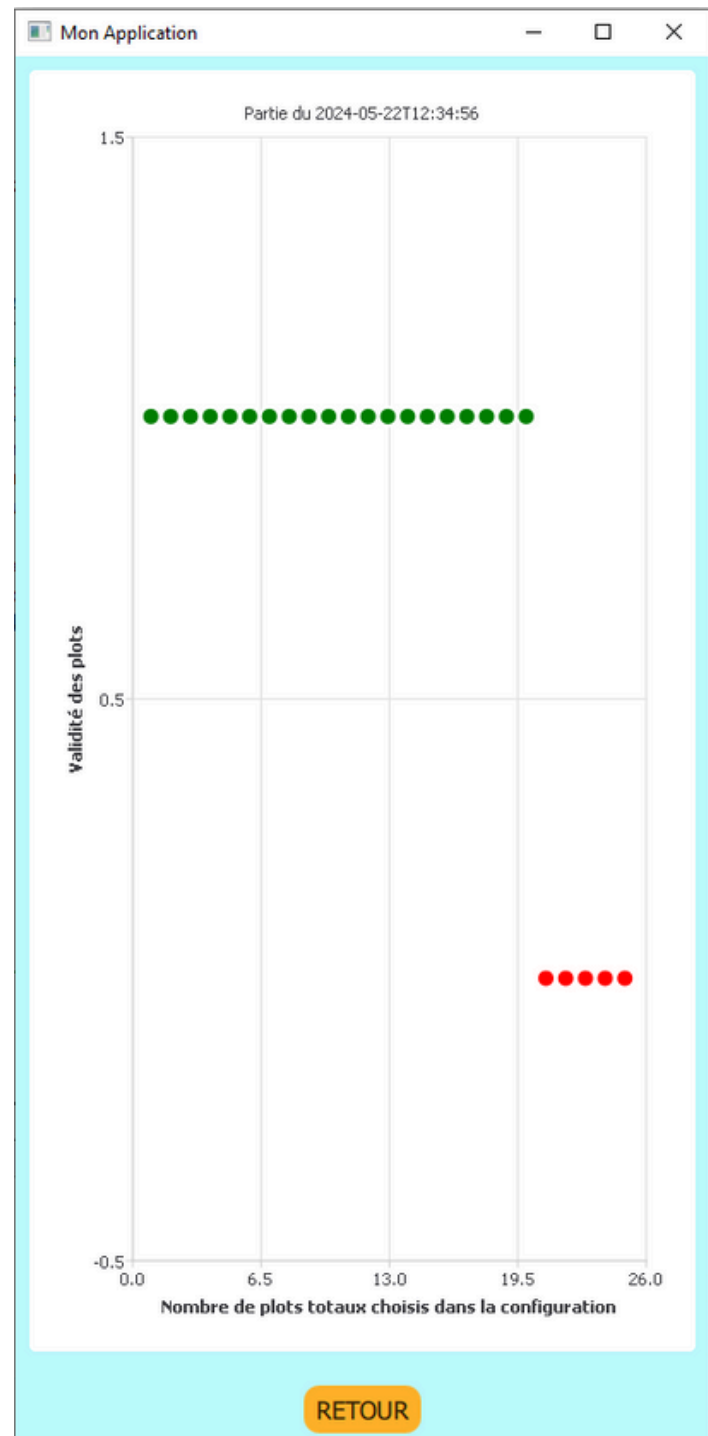
Afficher les résultats

Appareiller les plots

S'entraîner

Etat des plots

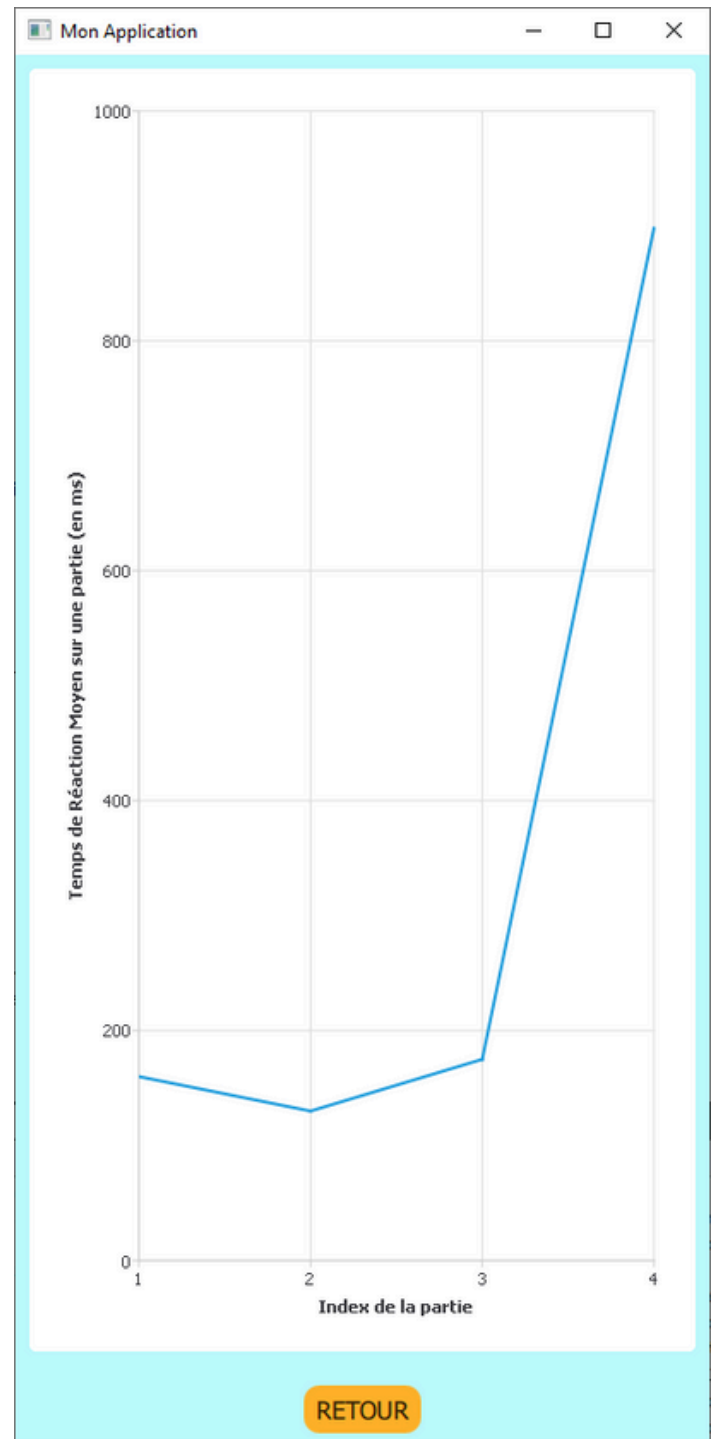
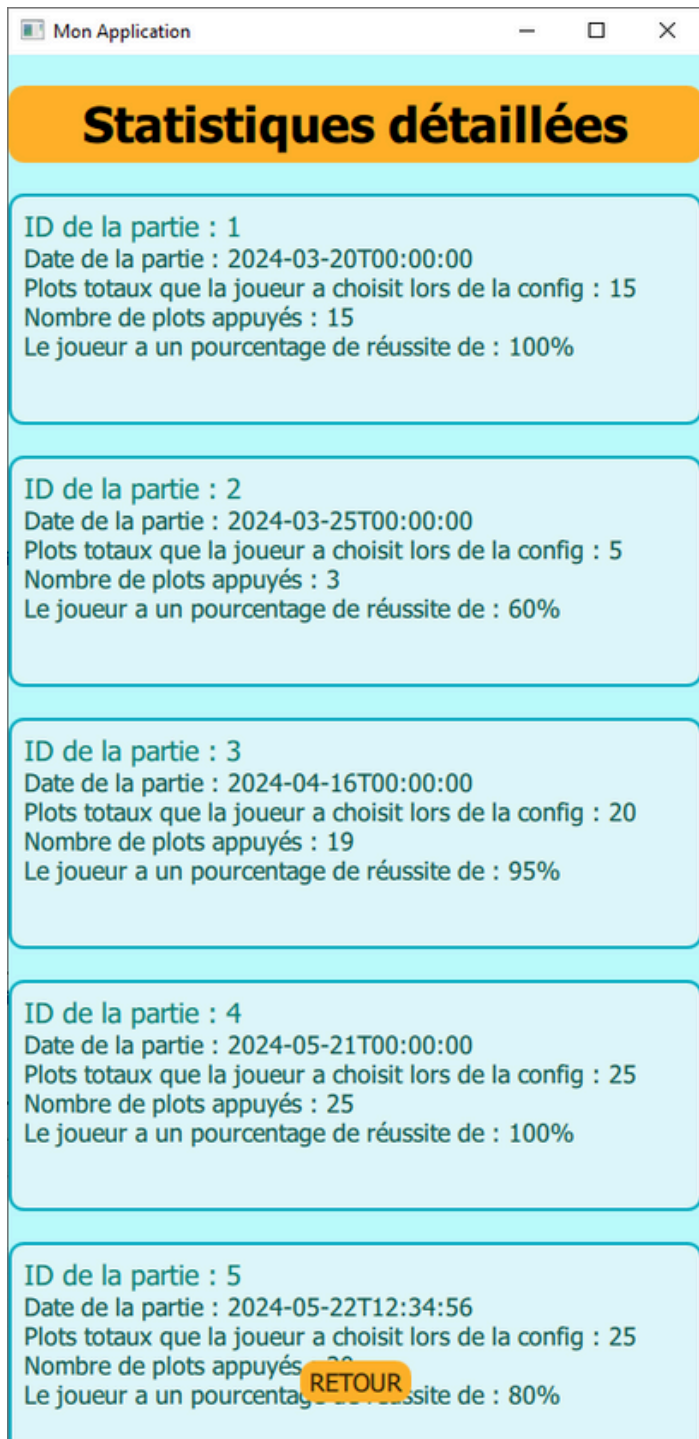
### 2.3 Afficher les résultats



Ici, il y a gauche l'interface de afficher les résultats avec de nouveaux plusieurs possibilités.

A droite il y a l'interface quand on clique sur une date avec un graphique qui représente en abscisse le nombre de plots que le joueur avait choisit lors de la configuration et en ordonné si le plot a bien été validé ou non.

## 2.3 Afficher les résultats

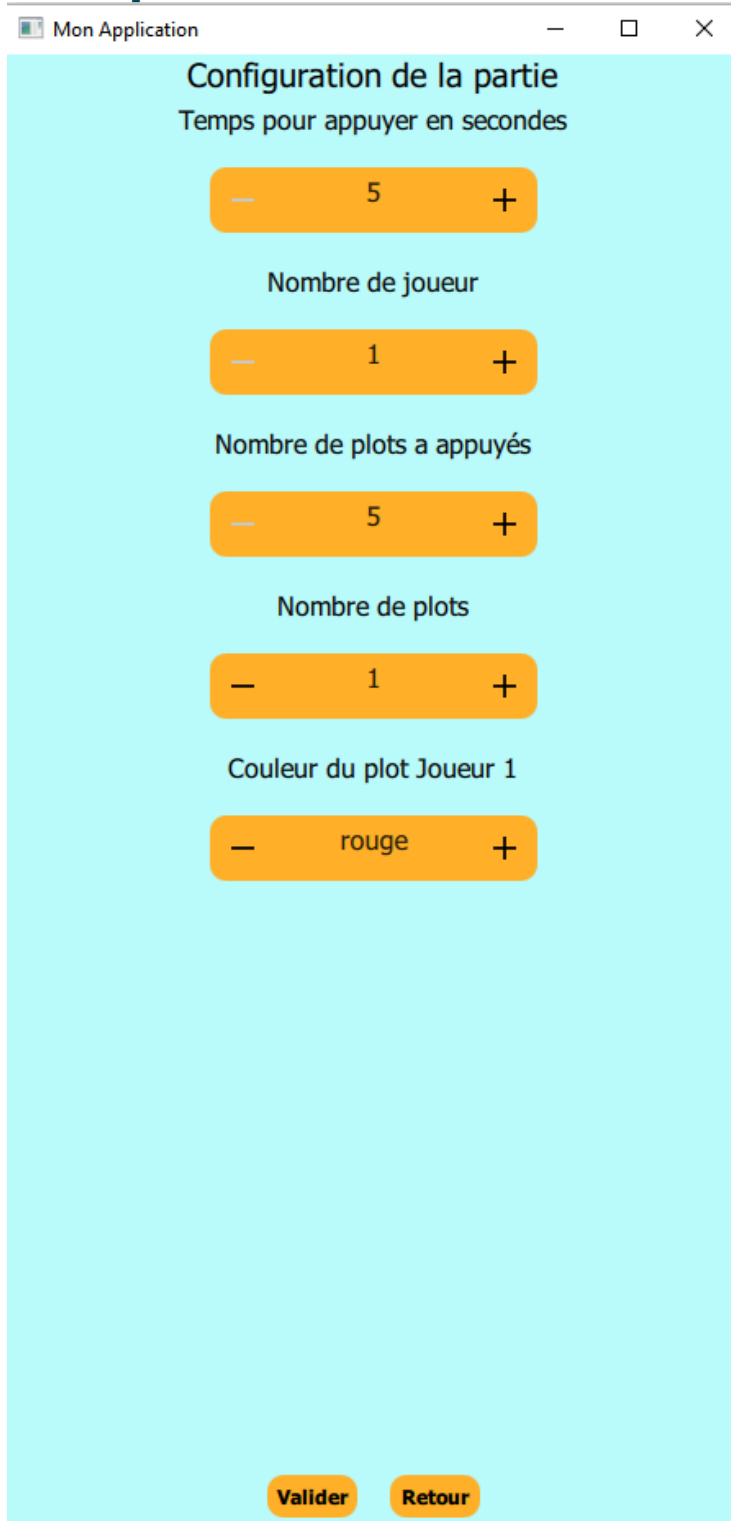


A gauche, l'interface des statistiques détaillées avec le pourcentage de réussite pour appuyer sur les plots.

A droite, il y a l'interface qui affiche le temps moyen de réaction via un graphique.



## 2.4 Configurer la partie



The screenshot shows a web application window titled "Mon Application". The main content area has a light blue background and is titled "Configuration de la partie". Below the title, there are five configuration options, each with a label and a value displayed in an orange button with minus and plus icons for adjustment:

- Temps pour appuyer en secondes**: Value 5
- Nombre de joueur**: Value 1
- Nombre de plots a appuyés**: Value 5
- Nombre de plots**: Value 1
- Couleur du plot Joueur 1**: Value rouge

At the bottom of the configuration area, there are two orange buttons: "Valider" and "Retour".

Ici, il y a l'interface de configuration permettant de lancer une partie avec plusieurs paramètres :

Le temps pour appuyer en secondes


































Le nombre de joueur

Le nombre de plots à appuyés

Le nombre de plots avec lesquels on va jouer la partie

La couleur du(des) plots pour jouer la partie

3.1 Base de données

←T→						idPartie	date	tempsReactionJ1	plotsAppuyes	identifiant	plotsTotaux	
<input type="checkbox"/>		Éditer		Copier		Supprimer	1	2024-03-20 05:12:50	160	15	d	20
<input type="checkbox"/>		Éditer		Copier		Supprimer	2	2024-03-21 00:00:00	150	33	aa	0
<input type="checkbox"/>		Éditer		Copier		Supprimer	3	2024-03-25 00:00:00	190	8	test	0
<input type="checkbox"/>		Éditer		Copier		Supprimer	4	2024-03-25 13:24:30	130	3	d	5
<input type="checkbox"/>		Éditer		Copier		Supprimer	5	2024-03-25 00:00:00	152	18	test	0
<input type="checkbox"/>		Éditer		Copier		Supprimer	6	2024-03-25 00:00:00	121	7	poiu5	0
<input type="checkbox"/>		Éditer		Copier		Supprimer	7	2024-03-25 00:00:00	163	5	test25	0
<input type="checkbox"/>		Éditer		Copier		Supprimer	9	2024-03-25 00:00:00	220	22	test25	0
<input type="checkbox"/>		Éditer		Copier		Supprimer	11	2024-03-25 00:00:00	230	22	poiu5	0
<input type="checkbox"/>		Éditer		Copier		Supprimer	12	2024-04-16 17:25:52	175	19	d	30
<input type="checkbox"/>		Éditer		Copier		Supprimer	15	2024-05-14 14:58:25	808	32	d	35

←T→				id	password	email	identifiant			
<input type="checkbox"/>		Éditer		Copier		Supprimer	22	1	1	1
<input type="checkbox"/>		Éditer		Copier		Supprimer	23	i	i	i
<input type="checkbox"/>		Éditer		Copier		Supprimer	24	f	f	f
<input type="checkbox"/>		Éditer		Copier		Supprimer	25	a	a	a
<input type="checkbox"/>		Éditer		Copier		Supprimer	26	d	d	d
<input type="checkbox"/>		Éditer		Copier		Supprimer	27	f	f	f
<input type="checkbox"/>		Éditer		Copier		Supprimer	28	t	t	t
<input type="checkbox"/>		Éditer		Copier		Supprimer	29	mot_de_passe	nouvel_utilisateur@example.com	nouvel_utilisateur
<input type="checkbox"/>		Éditer		Copier		Supprimer	30	mot_de_passe	nouvel_utilisateur@example.com	nouvel_utilisateur
<input type="checkbox"/>		Éditer		Copier		Supprimer	31	dd	qdsqdq@	dsqd
<input type="checkbox"/>		Éditer		Copier		Supprimer	32	aa	test@	test
<input type="checkbox"/>		Éditer		Copier		Supprimer	33	14	zz@	aa
<input type="checkbox"/>		Éditer		Copier		Supprimer	34	14	cerce@	adacxcxw
<input type="checkbox"/>		Éditer		Copier		Supprimer	35	25	dada@	test2
<input type="checkbox"/>		Éditer		Copier		Supprimer	36	14yyy	25iop@	testui
<input type="checkbox"/>		Éditer		Copier		Supprimer	37	azerty+	test@com	test25
<input type="checkbox"/>		Éditer		Copier		Supprimer	38	AZERTY*	test088@	test088
<input type="checkbox"/>		Éditer		Copier		Supprimer	39	12345.	poim@	poiu5
<input type="checkbox"/>		Éditer		Copier		Supprimer	40	12345.	ertete@	dadada@
<input type="checkbox"/>		Éditer		Copier		Supprimer	41	12345.	jep@	test0256
<input type="checkbox"/>		Éditer		Copier		Supprimer	42	test0705.	test0705@	test0705

Captures d’écran de la base de données de la table joueur et partie.

## 4.1 API définition

Une API, ou **Interface de Programmation d'Applications** (Application Programming Interface en anglais), est un ensemble de définitions et de protocoles qui permet à une application logicielle de communiquer avec une autre.

1. **Méthodes HTTP** : Les actions que l'API peut effectuer, basées sur les verbes HTTP tels que :
  - **GET** : Récupérer des données.
  - **POST** : Créer de nouvelles ressources.
  - **PUT** : Mettre à jour des ressources existantes.
  - **DELETE** : Supprimer des ressources.
2. **Requêtes et Réponses** : Les messages envoyés et reçus par l'API. Une requête contient généralement un en-tête, une méthode, une URL, et parfois un corps de message. La réponse contient le statut de la requête (succès, erreur, etc.), ainsi que les données renvoyées par l'API.
3. **Formats de Données** : Les formats utilisés pour échanger les données, généralement JSON (JavaScript Object Notation) ou XML (eXtensible Markup Language).

## 4.2 API création d'une requête exemple

```

public function createJoueur($identifiant, $password, $email) {
    // Initialisation de la réponse
    $reponse = array('statut' => 1, 'message' => 'ok'); // 1 pour validée

    // Connexion à la BDD
    if (!$this->connexionBDD()) {
        $reponse['statut'] = 0; // Erreur
        $reponse['message'] = 'connexion bdd impossible';
        return $reponse;
    }

    // Vérifier si l'identifiant existe déjà
    $sql_check = 'SELECT COUNT(*) AS count FROM joueurs WHERE identifiant = :identifiant';
    $stmt_check = $this->bdd->prepare($sql_check);
    $stmt_check->execute(array(':identifiant' => $identifiant));
    $row = $stmt_check->fetch();

    if ($row['count'] > 0) {
        // L'identifiant existe déjà, renvoyer une erreur
        $reponse['statut'] = 0; // Erreur
        $reponse['message'] = 'Identifiant déjà utilisé';
        return $reponse;
    }

    // Hacher le mot de passe
    $hashed_password = password_hash($password, PASSWORD_DEFAULT);

    // Création de la requête
    $sql = 'INSERT INTO joueurs (identifiant, password, email) VALUES (:identifiant, :password, :email)';
    $stmt = $this->bdd->prepare($sql);

    // Exécution de la requête
    $res = $stmt->execute(array(':identifiant' => $identifiant, ':password' => $hashed_password, ':email' => $email));

    // Vérification de l'exécution de la requête
    if ($res === false) {
        $reponse['statut'] = 0; // Erreur
        $reponse['message'] = 'Requête erronée';
        return $reponse;
    } else if ($stmt->rowCount() === 0) { // Si aucun joueur ajouté
        $reponse['statut'] = 0; // Erreur
        $reponse['message'] = 'Non ajouté';
        return $reponse;
    }

    // Récupérer le joueur créé
    $reponse['id'] = intval($this->bdd->lastInsertId());

    // Attribution des données à la réponse
    return $reponse;
}

```

Dans cette capture d'écran, on crée une requête qui permet de créer un joueur avec un identifiant, un mot de passe et un email. Avant cela on doit se connecter à la base de données puis vérifier si l'identifiant existe déjà. Puis pour finir on exécute la requête en vérifiant son exécution et on récupère la valeur.

## 4.3 API HTTP POST exemple

```
<?php
// inclure les fonctions d'accès à la base de données
require("dataJoueurs.php");

// Récupérer le verbe (méthode http) de la requête
$request_method = $_SERVER["REQUEST_METHOD"];

$dataJoueurs = new DataJoueurs();

// test de la méthode de la requête http
switch ($request_method) {
    case 'POST': // Si c'est une méthode POST
        // Récupérer les données POST
        $identifiant = isset($_POST['identifiant']) ? $_POST['identifiant'] : '';
        $password = isset($_POST['password']) ? $_POST['password'] : '';
        $email = isset($_POST['email']) ? $_POST['email'] : '';

        if ($identifiant != '' && $password != '' && $email != '') {
            // Appel de la méthode createJoueur avec les données POST
            $reponse = $dataJoueurs->createJoueur($identifiant, $password, $email);
            echo json_encode($reponse);
        } else {
            // Si certaines données ne sont pas définies, renvoyer une erreur
            $reponse = array('statut' => 0, 'message' => 'Certains champs ne sont pas définis');
            header('Content-Type: application/json');
            http_response_code(400); // Mauvaise requete
            echo json_encode($reponse);
        }
        break;

    default:
        // Requête invalide
        header("HTTP/1.0 405 Method Not Allowed");
        break;
}
?>
```

## 4.3 API HTTP POST explication

Mon code commence par inclure le fichier **dataJoueurs.php**, qui contient les fonctions de base de données et la classe **DataJoueurs**.

Il récupère la méthode de requête HTTP (GET, POST, etc.) en utilisant `$_SERVER`.

Il crée une instance de la classe `DataJoueurs`.

Le script utilise une instruction `switch` pour gérer les différentes méthodes de requête HTTP. Dans ce cas, il ne gère que la méthode POST.

Il récupère les valeurs identifiant, password, et email de la requête POST. Si l'une de ces valeurs n'est pas définie, elle est par défaut une chaîne vide.

Il vérifie si les trois valeurs (identifiant, password, email) ne sont pas vides.

Si toutes les valeurs sont fournies, il appelle la méthode `createJoueur` de la classe `DataJoueurs` avec ces valeurs. La réponse de cette méthode est ensuite encodée en format JSON et renvoyée au client.  
php.

Si l'une des valeurs requises est manquante, il renvoie une réponse d'erreur avec un code de statut HTTP 400 et un message indiquant que certains champs ne sont pas définis.

Si la méthode de requête n'est pas POST, il renvoie un code de statut HTTP 405 Method Not Allowed.

## 5. 1 Explication code C++

La partie majeure du code définit la classe Controller qui gère la logique de communication entre les données des joueurs (DataJoueurs), les résultats (DataResultats), et l'interface utilisateur (via des signaux et des slots).

La classe Controller est dérivée de QObject, ce qui permet l'utilisation de faire des signaux et des slots de Qt pour faciliter la communication asynchrone.

**Constructeur :** Initialise les pointeurs dataJoueurs et dataResultats, et connecte les signaux statistiquesChanger et creationJoueurReussie à leurs slots respectifs afficherStatistiques et lireCreateJoueurReussie.

```
Controller::Controller(DataJoueurs* dataJoueurs, DataResultats* dataResultats, QObject *parent) : QObject(parent)
{
    this->dataJoueurs = dataJoueurs;
    this->dataResultats = dataResultats;
    this->joueurConnecte = nullptr;
    connect(dataResultats, &DataResultats::statistiquesChanger, this, &Controller::afficherStatistiques);
    connect(dataJoueurs, &DataJoueurs::creationJoueurReussie, this, &Controller::lireCreateJoueurReussie);
}
```

**createJoueur :** Appelle la méthode createJoueur de dataJoueurs pour créer un nouveau joueur avec l'identifiant, le mot de passe et l'email fournis.

```
void Controller::createJoueur(const QString &identifiant, const QString &password, const QString &email)
{
    dataJoueurs->createJoueur(identifiant, password,email);

    //connect(dataJoueurs, &DataJoueurs::creationJoueurReussie, this, &Controller::lireCreateJoueurReussie);
}
```

## 5. 1 Explication code C++

connexionJoueur : Appelle la méthode connexionJoueur de dataJoueurs pour connecter un joueur et connecte le signal connexionReussie au slot lireConnexionJoueurReussie.

```
void Controller ::connexionJoueur(const QString &identifiant, const QString &password)
{
    dataJoueurs->connexionJoueur(identifiant, password);
    connect(dataJoueurs, &DataJoueurs::connexionReussie, this, &Controller::lireConnexionJoueurReussie);
}
```

getStatistiques : Efface la liste des résultats et demande les statistiques via dataResultats.

```
void Controller::getStatistiques()
{
    qDeleteAll(listeResultats);
    listeResultats.clear();

    //lire data BDD
    dataResultats->getStatistiques();
}
```



## 5. 1 Explication code C++

`afficherStatistiquesUnJoueur` : Traite et affiche les statistiques d'un joueur spécifique.

```
void Controller::afficherStatistiques(QJsonDocument jsonData){
    QJsonObject dataJson = jsonData.object(); // Accéder directement à l'objet principal sans "partie"
    QJsonArray listePartiesJson = dataJson["statistiques"].toArray();
    qDebug()<<listePartiesJson;

    for(int i = 0; i< listePartiesJson.size(); i++ ){
        listeResultats.append(new Resultat(listePartiesJson[i].toObject()));
    }

    for(int i=0;i<listeResultats.size();i++){
        qDebug()<< listeResultats[i]->toString();
    }

    emit listeResultatsChanged();
}
```

La plupart des autres méthodes que j'ai faite sont des méthodes qui servent à émettre un signal lorsqu'une action a été faite.

Emit est souvent utilisé pour gérer des événements comme les clics de bouton, les changements de données, ou d'autres actions spécifiques de l'utilisateur.

## 5. 2 Explication code QML

```
#include "controller.h"
#include <QObject>

Controller::Controller(DataJoueurs* dataJoueurs, DataResultats* dataResultats, QObject *parent) : QObject(parent)
{
    this->dataJoueurs = dataJoueurs;
    this->dataResultats = dataResultats;
    this->joueurConnecte = nullptr;
    connect(dataResultats, &DataResultats::statistiquesChanger, this, &Controller::afficherStatistiques);
    connect(dataJoueurs, &DataJoueurs::creationJoueurReussie, this, &Controller::lireCreateJoueurReussie);
}

void Controller::createJoueur(const QString &identifiant, const QString &password, const QString &email)
{
    dataJoueurs->createJoueur(identifiant, password,email);

    //connect(dataJoueurs, &DataJoueurs::creationJoueurReussie, this, &Controller::lireCreateJoueurReussie);
}

void Controller::lireCreateJoueurReussie()
{
    emit creationJoueurReussie();
}

void Controller::lireCreateJoueurEchoue()
{
    emit creationJoueurEchoue();
}

void Controller::connexionJoueur(const QString &identifiant, const QString &password)
{
    dataJoueurs->connexionJoueur(identifiant, password);

    connect(dataJoueurs, &DataJoueurs::connexionReussie, this, &Controller::lireConnexionJoueurReussie);
}
}
```

Pour cette application, j'ai codé pour faire en sorte que le lien entre les deux codes se fait via le code C++ et l'interface utilisateur (dans le code QML). Le code QML déclenche des actions en réponse aux interactions de l'utilisateur, tandis que le code C++ gère la logique et communique avec les services réseau pour effectuer les opérations nécessaires, comme l'inscription d'un nouveau joueur.

## 5. 2 Explication code QML

```

        errorMessage = "veuillez entrer un identifiant.";
        return;
    }

    if (email === "") {
        errorMessage = "Veuillez entrer une adresse email.";
        return;
    }

    if (!email.includes("@")) {
        errorMessage = "L'adresse email est incorrecte.";
        return;
    }

    if (password === "") {
        errorMessage = "Veuillez entrer un mot de passe.";
        return;
    }

    if (confirmPassword === "") {
        errorMessage = "Veuillez confirmer votre mot de passe.";
        return;
    }

    if (identifiant.length < 5 || identifiant.length > 20) {
        errorMessage = "L'identifiant doit contenir entre 5 et 20 caractères.";
        return;
    }

    if (password.length < 5 || password.length > 20) {
        errorMessage = "Le mot de passe doit contenir entre 5 et 20 caractères.";
        return;
    }

    var specialChars = /^[A-Za-z0-9]/;
    if (!specialChars.test(password)) {
        errorMessage = "Le mot de passe doit contenir au moins un caractère spécial.";
        return;
    }

    if (password !== confirmPassword) {
        errorMessage = "Les mots de passe ne correspondent pas.";
        return;
    }

    // Utilisation des valeurs des champs de texte pour l'inscription
    creationJoueurReussie();
    controller.createJoueur(identifiant, password, email);
}

```

Ici on peut voir toutes les conditions qu'une personne doit remplir pour créer un utilisateur. Mot de passe avec caractère spécial, email avec un @, la taille de l'identifiant...

Une fois ces conditions remplies le joueur peut alors être créé. Avec `controller.createJoueur(identifiant,password,email)`.

## 5. 2 Explication code QML

```

import QtQuick 1.0
title: "Accueil application"

Component.onCompleted: {
    controller.get;
}

Rectangle {
    anchors.fill: parent
    color: "#BBFAFC"

    RowLayout {
        id: barreAccueilIdentification
        anchors.horizontalCenter: parent.horizontalCenter
        Text {
            text: qsTr("Accueil")
            font.pointSize: 24
            font.bold: true
            color: "#333333"
        }
    }
}

Column {
    spacing: 40
    anchors.centerIn: parent

    Button {
        width: 200
        height: 50
        text: "Afficher les résultats"
        onClicked: stackView.push("afficherLesResultats.qml")
        background: Rectangle {
            color: "#FFB22A"
            radius: 10
        }
        contentItem: Text {
            text: parent.text
            font.bold: true
            color: "black"
            horizontalAlignment: Text.AlignHCenter
            verticalAlignment: Text.AlignVCenter
        }
    }

    Button {
        width: 200
        height: 50
        text: "Appareiller plot(s)"
        onClicked: stackView.push("appareillerLesPlots.qml")
        background: Rectangle {
            color: "#FFB22A"
            radius: 10
        }
        contentItem: Text {
            text: parent.text
        }
    }
}

```

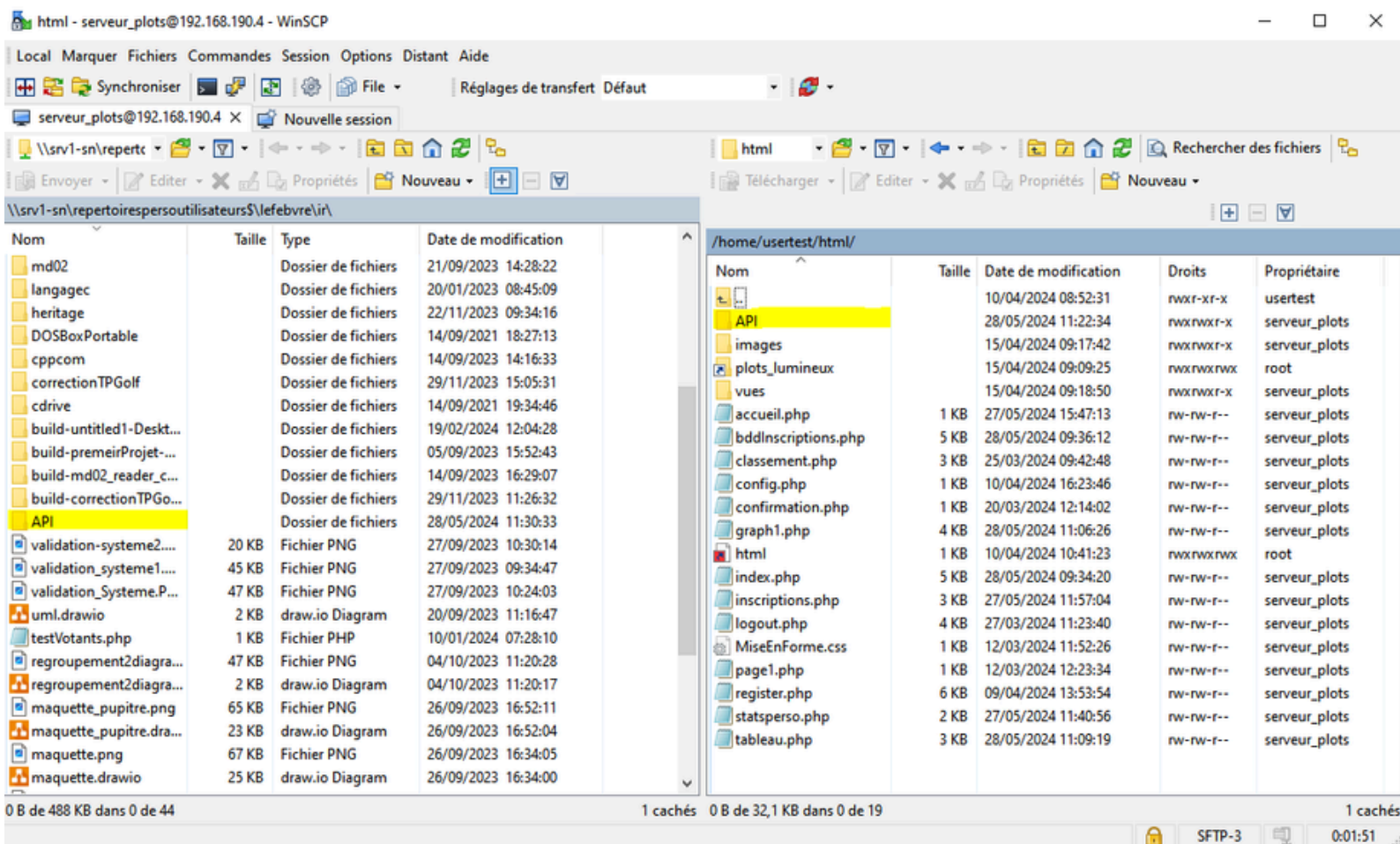
La page principale de l'application sert d'accueil, avec un titre et plusieurs options de navigation. Les boutons permettent aux utilisateurs de naviguer facilement entre différentes pages de l'application.

Utilisation d'une palette de couleurs cohérente (fond bleu clair, boutons orange) et d'une mise en page simple et centrée pour une interface propre.

Utilisation de StackView : Permet de gérer la navigation entre différentes pages de manière fluide et maintient un historique de navigation.

En résumé, ce design offre une interface utilisateur simple, intuitive pour naviguer dans l'application, en utilisant des éléments de base de QtQuick et QtQuick.Controls.

## 6.1 Mise en commun



Nous avons mis l'API qui a été faite dans le serveur via WinSCP. Puis modifier le PDO ( interface pour accéder à une base de données depuis PHP)

## 6.2 Etat d'avancement personnel

### Ce qui a été fait :

#### Identifier un utilisateur :

Une personne peut soit si elle le souhaite créer un utilisateur avec un identifiant, un email ainsi qu'un mot de passe. Il y a aussi la possibilité de se connecter avec l'identifiant et le mot de passe créer avant. Ces données sont bien stockées dans la base de données plots\_lumineux du serveur dans la table joueur.

#### Sélectionner un mode et configurer l'entraînement :

Le joueur a la possibilité de configurer sa partie avec plusieurs paramètres comme le temps pour appuyer sur un plot, le nombre de joueur, le nombre de plots à appuyer, le nombre de plots connectés en BLE et la couleur du plot.

#### Afficher les résultats/statistiques :

Les résultats des parties stockés dans la BDD, sont visibles dans un tableau avec la date, le nombre de plots appuyés et le temps de réaction moyen sur la partie. Un graphique a aussi été fait montre la courbe de progression du joueur. Les statistiques détaillées sont aussi disponibles avec notamment son pourcentage de réussite.

#### Transmettre les résultats au serveur :

Les données ajoutées via l'application sont bien ajoutées dans la BDD du serveur et on peut aussi utilisés les données de cette BDD.

### Ce qui n'a pas été fait ou terminé :

#### La mise en relation avec les autres parties de l'application:

Le fait de pouvoir appareiller/visualiser les plots et s'entraîner n'a pas été mis en commun avec ma partie.

#### La création de partie via l'application :

Le joueur est ne peut que créer des parties via la base de données.

# CONCLUSION

Pour conclure sur ce projet ma partie personnelle est fonctionnelle dans sa globalité :

La partie identifier un utilisateur est fonctionnelle.  
Le fait de sélectionner un mode et configurer l'entraînement est opérationnel.

Le fait d'afficher les résultats ainsi que les statistiques est aussi fonctionnelle.

La transmission avec le serveur fonctionne.

La mise en relation avec les autres parties de l'application n'a pas été effectué ainsi que la création de partie via l'application

J'ai vraiment apprécié ce projet qui m'aura permis d'appréhender un protocole inconnu à mes yeux ainsi que d'approfondir mes connaissances déjà acquises grâce à ces deux années de BTS.

# 07