FAKIRI YASSINE 2023

BTS SIO SLAM

ATELIER DE PROFESSIONNALISATION n°2

“PALACE GESTION”

1. Rappel du contexte :

Innovative IT est une entreprise technologique de pointe spécialisée dans le développement de solutions informatiques sur mesure. Fondée en 2020, la société est basée à Paris, en France, et est devenue un acteur incontournable dans le domaine des technologies de l'information.

Le Palace Hotel est un établissement hôtelier de luxe, reconnu pour sa qualité de service exceptionnelle et son cadre enchanteur. Situé au cœur de Paris, il offre une expérience unique de sophistication et de confort.

Afin de maintenir son statut de leader dans l'industrie hôtelière de luxe, le Palace Hotel cherche constamment à innover et à améliorer ses services. Ils ont identifié le besoin d'une application de gestion hôtelière sur mesure pour améliorer l'efficacité de leurs opérations et pour offrir une meilleure expérience à leurs clients. Cette application doit être capable de gérer les réservations, les clients et les chambres de l'hôtel, avec des options pour ajouter, mettre à jour et supprimer des données.

2. Rappel des missions

- Mission 1 : Conception et développement
    - Concevoir l'interface utilisateur avec C# Windows Forms (.NET Framework).
    - Développer des fonctionnalités pour gérer chambres, clients et réservations.
    - Créer la base de données.
    - Mettre en place l'authentification.

- Mission 2 : Assurance qualité et tests
    - Effectuer des tests pour garantir la fonctionnalité.
    - Utiliser Serilog pour la journalisation.
    - Utiliser SonarLint et SonarQube pour l'analyse du code.

- Mission 3 : Documentation et formation
    - Créer la documentation technique.
    - Préparer la documentation utilisateur.
    - Produire une vidéo explicative.

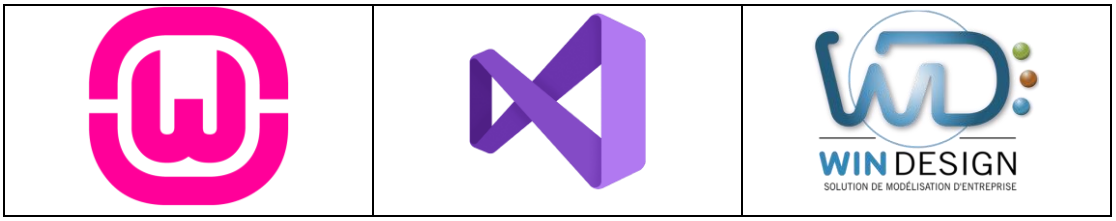
- Mission 4 : Déploiement et gestion de projet
    - Déployer la base de données en ligne.
    - Créer un fichier d'installation zip.
    - Utiliser Github pour la gestion des versions.
    - Utiliser Trello pour le suivi des tâches.

3. Outils utilisés :

- Microsoft Visual Studio 2019
- WampServer
- PhpMyAdmin
- Apache
- MySql
- Win'Design
- SandCastle
- Hostinger
- GitHub - Trello
- SonarLint - SonarQube
- Serilog

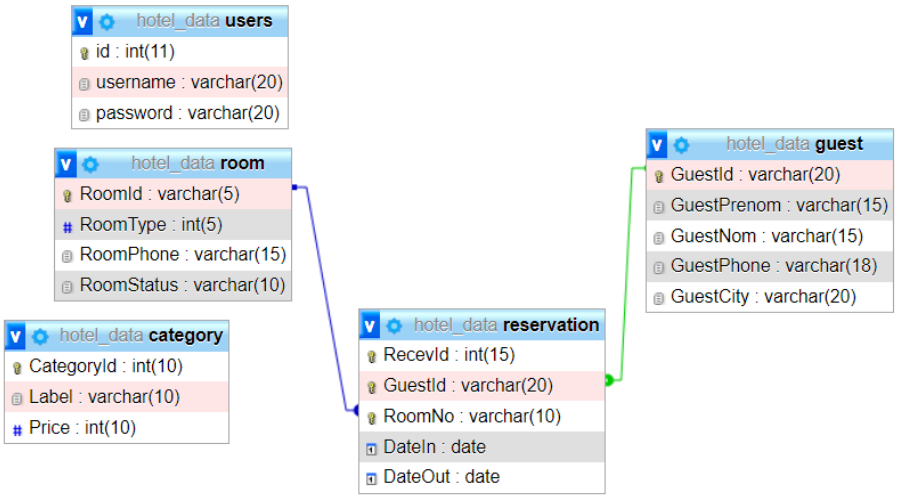
4. Mission 1 :

- Préparation de l'**environnement de travail** : Microsoft Visual Studio, lancement WampServer...



- Elaboration du **modèle conceptuel** (**MCD**) et du Modèle logique de données sur *Win'Design* qui nous permettent de générer automatiquement le script SQL.
- **Création de la BDD** : *hotel_data* avec encodage utf_8 sur phpMyAdmin en localhost
  Importation et exécution du script ce qui nous permet de créer la BDD qui correspond au modèle conceptuel.
  extrait du script :

```sql
CREATE TABLE IF NOT EXISTS `users` (
  `id` int(11) NOT NULL,
  `username` varchar(20) COLLATE utf8mb4_unicode_ci NOT NULL,
  `password` varchar(20) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```



- **Alimentation de la base de données :**

- Ajout utilisateur et mot de passe dans la table "*users*"
- Ajout des typologie et prix des chambres dans la table "*category*"
- Ajout de clients fictifs dans la table "*guests*"

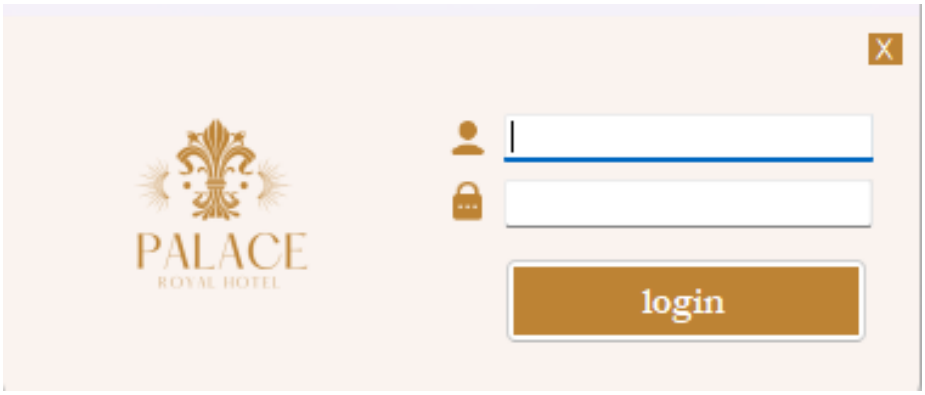
  ex.
```
INSERT INTO `guest`(`GuestId`, `GuestPrenom`, `GuestNom`, `GuestPhone`, `GuestCity`) VALUES ('897594','Jean','Dupont','0123456789','Annecy')
```
- Ajout de réservations fictifs dans la table "*reservation*"

- **Liaison Visual Studio à la base de données**
  - Pour cela j'ai créé la classe "**DBConnect**" qui permet la connexion et l'interaction avec la base de données, ainsi que l'ouverture et la fermeture de la connexion. J'ai utilisé "MySqlConnection" qui nécessite l'utilisation de la librairie *"MySql.Data.MySqlClient"* :

```
class DBConnect
{
    // Créez une connexion à la base de données MySQL avec les informations d'identification spécifiées
    private readonly MySqlConnection connection = new MySqlConnection("datasource=localhost;port=3306;username=root;password=;database=hotel_data");
    /// <summary>
    /// Cette méthode retourne la connexion à la base de données
    /// </summary>
    /// <returns>MySqlConnection</returns>
    17 références | FakiriYassine, il y a 22 heures | 1 auteur, 2 modifications
    public MySqlConnection GetConnection()
    {
        return connection;
    }
    /// <summary>
    /// Cette méthode ouvre la connexion à la base de données si elle est fermée
    /// </summary>
    10 références | FakiriYassine, il y a 1 jour | 1 auteur, 1 modification
    public void OpenCon()
    {
        if (connection.State == System.Data.ConnectionState.Closed)
        {
            connection.Open();
        }
    }
    /// <summary>
    /// Cette méthode ferme la connexion à la base de données si elle est ouverte
    /// </summary>
    20 références | FakiriYassine, il y a 1 jour | 1 auteur, 1 modification
    public void CloseCon()
    {
        if (connection.State == System.Data.ConnectionState.Open)
        {
            connection.Close();
        }
    }
}
```

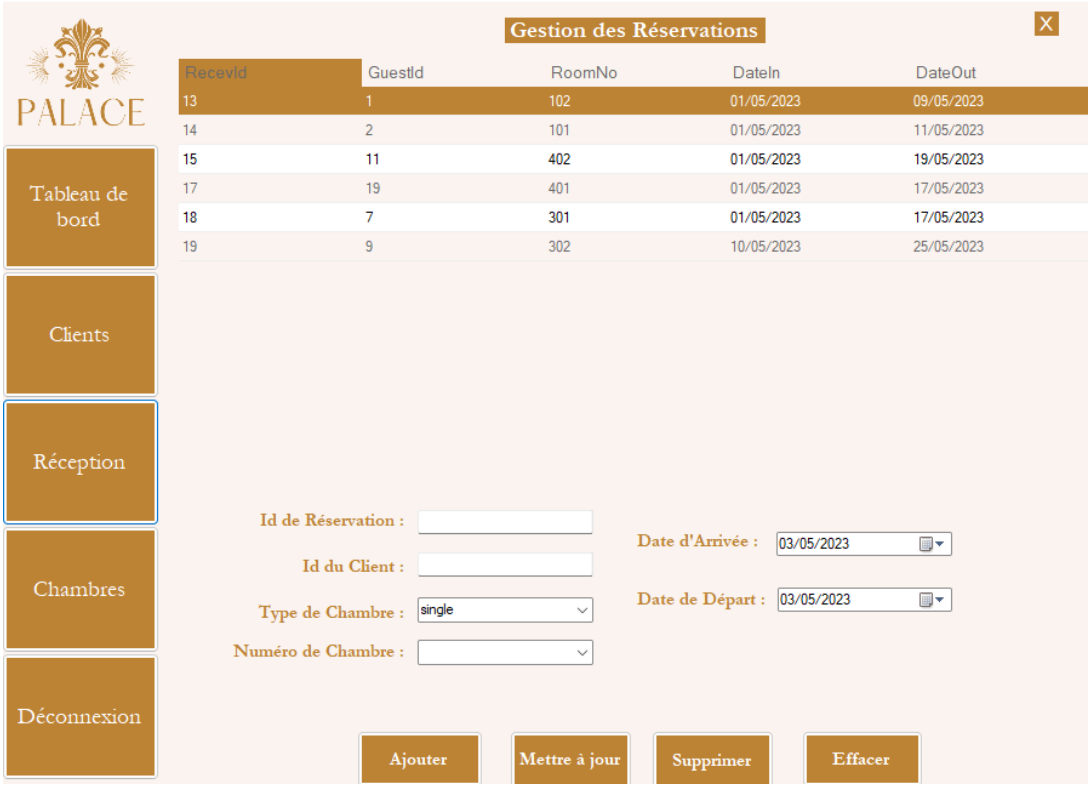
- **Création des différentes interfaces** en respectant la charte graphique :
  - Pour faire cela j'ai repris les couleurs du logo de l'hôtel qu'ils vont devenir les couleurs principaux de l'application :
    HEX #FAF3EF – RGB 250, 243, 239
    HEX #BD8334 – RGB 189, 131, 52
    Ensuite j'ai créé les différentes interfaces en utilisant la boite à outil de Windows Forms. C'est une bibliothèque graphique qui nous permet de construire des interfaces grâce à différents contrôles : Button, TextBox...

- *Fenêtre d'authentification :*

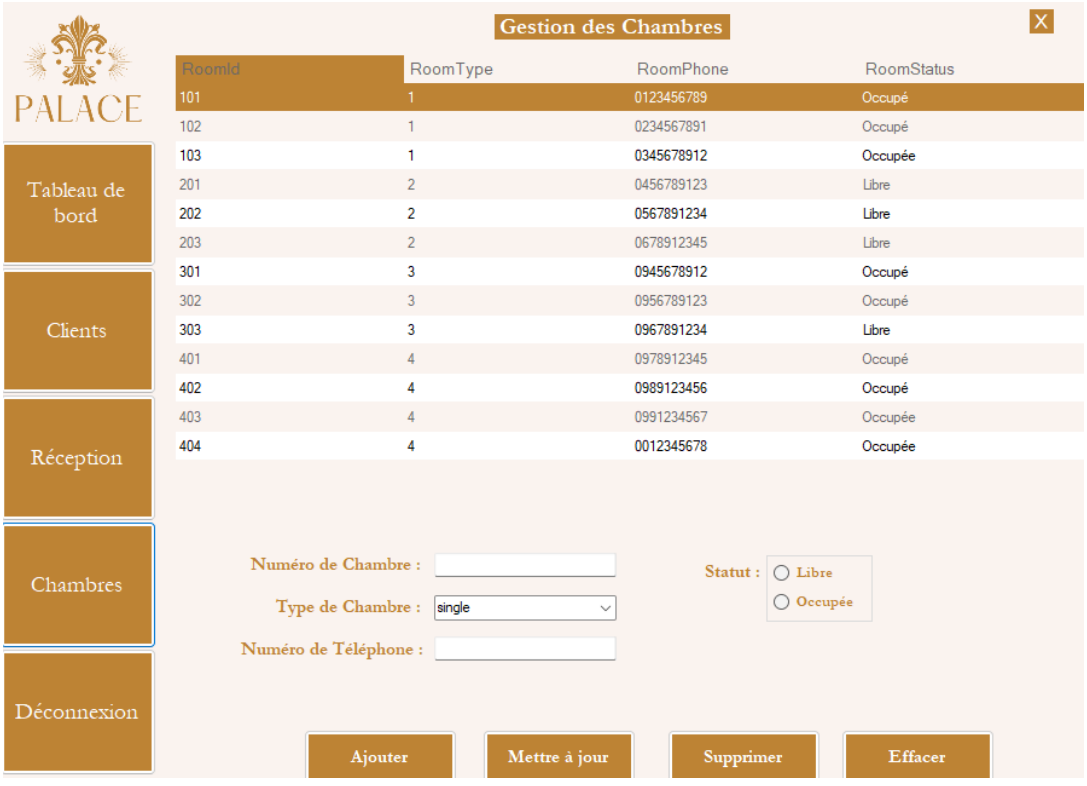


- *Fenêtre de gestion des clients :*

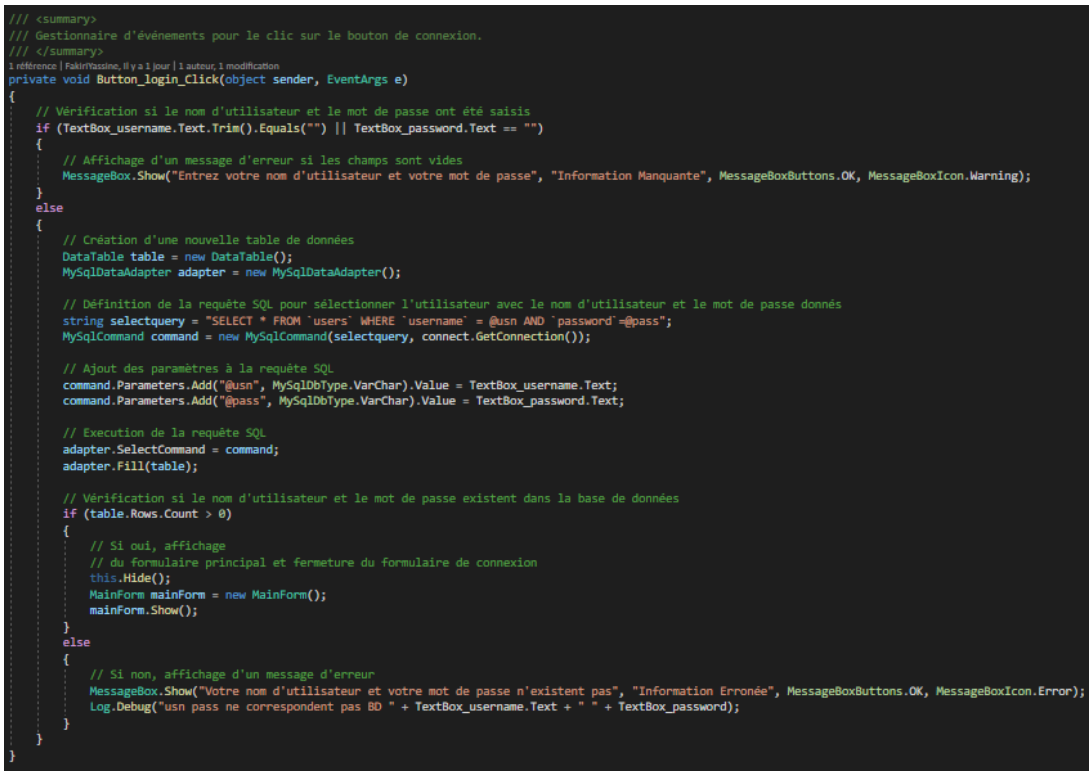
- *Fenêtre de gestion des réservations :*



- *Fenêtre de gestion des chambres :*

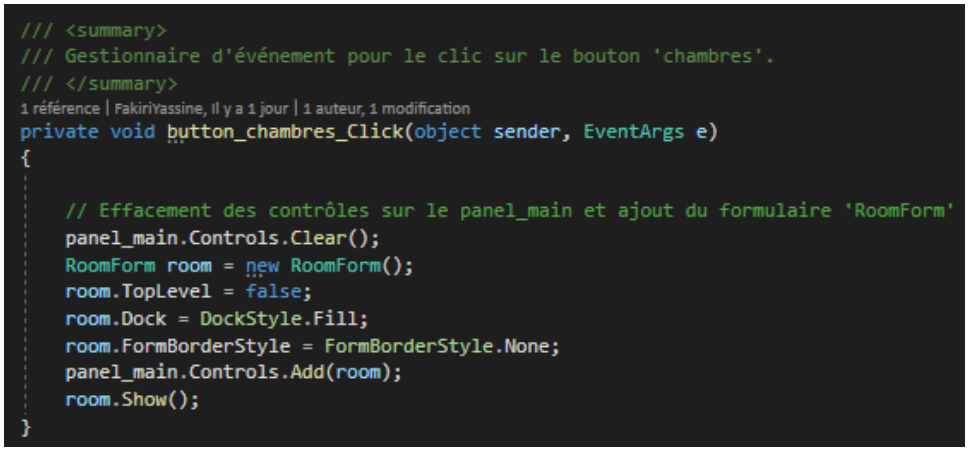


- **Gestion des événements du "*LoginForm*"** qui représente le formulaire d'authentification et nous permet de gérer les événements d'interface utilisateurs et interagir avec la base de données :
  Extrait de la classe "*LoginForm*" - événement click sur bouton Login :



```
/// <summary>
/// Gestionnaire d'événements pour le clic sur le bouton de connexion.
/// </summary>
1 référence | FakiriYassine, Il y a 1 jour | 1 auteur, 1 modification
private void Button_login_Click(object sender, EventArgs e)
{
    // Vérification si le nom d'utilisateur et le mot de passe ont été saisis
    if (TextBox_username.Text.Trim().Equals("") || TextBox_password.Text == "")
    {
        // Affichage d'un message d'erreur si les champs sont vides
        MessageBox.Show("Entrez votre nom d'utilisateur et votre mot de passe", "Information Manquante", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
    else
    {
        // Création d'une nouvelle table de données
        DataTable table = new DataTable();
        MySqlDataAdapter adapter = new MySqlDataAdapter();

        // Définition de la requête SQL pour sélectionner l'utilisateur avec le nom d'utilisateur et le mot de passe donnés
        string selectquery = "SELECT * FROM `users` WHERE `username` = @usn AND `password`=@pass";
        MySqlCommand command = new MySqlCommand(selectquery, connect.GetConnection());

        // Ajout des paramètres à la requête SQL
        command.Parameters.Add("@usn", MySqlDbType.VarChar).Value = TextBox_username.Text;
        command.Parameters.Add("@pass", MySqlDbType.VarChar).Value = TextBox_password.Text;

        // Execution de la requête SQL
        adapter.SelectCommand = command;
        adapter.Fill(table);

        // Vérification si le nom d'utilisateur et le mot de passe existent dans la base de données
        if (table.Rows.Count > 0)
        {
            // Si oui, affichage
            // du formulaire principal et fermeture du formulaire de connexion
            this.Hide();
            MainForm mainForm = new MainForm();
            mainForm.Show();
        }
        else
        {
            // Si non, affichage d'un message d'erreur
            MessageBox.Show("Votre nom d'utilisateur et votre mot de passe n'existent pas", "Information Erronée", MessageBoxButtons.OK, MessageBoxIcon.Error);
            Log.Debug("usn pass ne correspondent pas BD " + TextBox_username.Text + " " + TextBox_password);
        }
    }
}
```

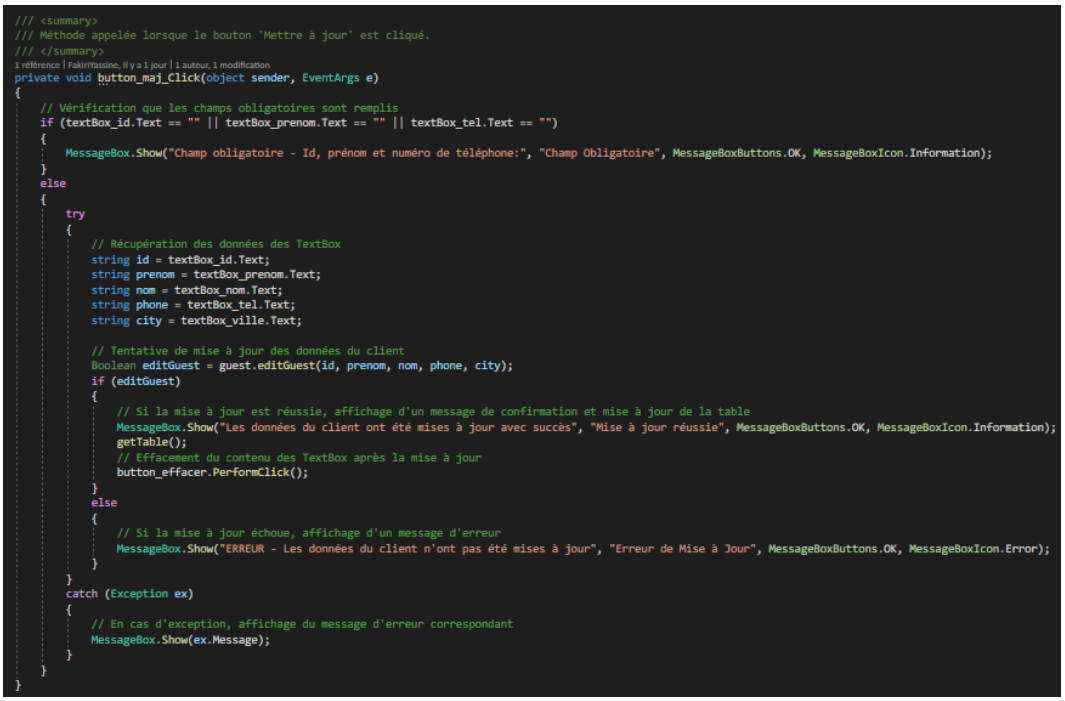
- **Gestion des événements du "*tableau de bord*"**, donc les événement lors de l'appuie sur les boutons : Chambres, Clients, Réception et Déconnexion
  extrait de la classe "*MainForm*" - événement click sur le bouton chambres



```
/// <summary>
/// Gestionnaire d'événement pour le clic sur le bouton 'chambres'.
/// </summary>
1 référence | FakiriYassine, Il y a 1 jour | 1 auteur, 1 modification
private void button_chambres_Click(object sender, EventArgs e)
{

    // Effacement des contrôles sur le panel_main et ajout du formulaire 'RoomForm'
    panel_main.Controls.Clear();
    RoomForm room = new RoomForm();
    room.TopLevel = false;
    room.Dock = DockStyle.Fill;
    room.FormBorderStyle = FormBorderStyle.None;
    panel_main.Controls.Add(room);
    room.Show();
}
```

Cette méthode efface tous les contrôles du panel_main, crée un nouveau RoomForm qui occupera toute la place dans le panel_main, puis ajoute et affiche le RoomForm.

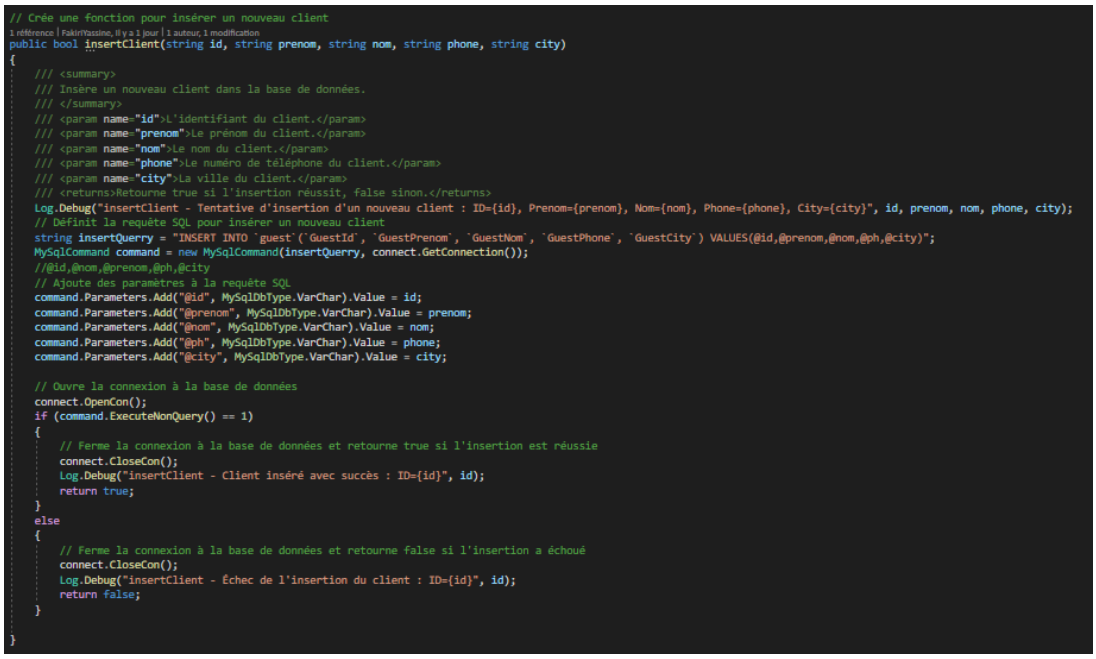
- **Gestion des événements du "*GuestForm*"**, donc les événement concernant l'appuie sur les différents boutons permettent l'ajout, la modification, la suppression des Clients
  extrait de la classe "*GuestForm*" - click bouton mise à jour client :



```
/// <summary>
/// Méthode appelée lorsque le bouton 'Mettre à jour' est cliqué.
/// </summary>
1 référence | FakirYassine, Il y a 1 jour | 1 auteur, 1 modification
private void button_maj_Click(object sender, EventArgs e)
{
    // Vérification que les champs obligatoires sont remplis
    if (textBox_id.Text == "" || textBox_prenom.Text == "" || textBox_tel.Text == "")
    {
        MessageBox.Show("Champ obligatoire - Id, prénom et numéro de téléphone:", "Champ Obligatoire", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        try
        {
            // Récupération des données des TextBox
            string id = textBox_id.Text;
            string prenom = textBox_prenom.Text;
            string nom = textBox_nom.Text;
            string phone = textBox_tel.Text;
            string city = textBox_ville.Text;

            // Tentative de mise à jour des données du client
            Boolean editGuest = guest.editGuest(id, prenom, nom, phone, city);
            if (editGuest)
            {
                // Si la mise à jour est réussie, affichage d'un message de confirmation et mise à jour de la table
                MessageBox.Show("Les données du client ont été mises à jour avec succès", "Mise à jour réussie", MessageBoxButtons.OK, MessageBoxIcon.Information);
                getTable();
                // Effacement du contenu des TextBox après la mise à jour
                button_effacer.PerformClick();
            }
            else
            {
                // Si la mise à jour échoue, affichage d'un message d'erreur
                MessageBox.Show("ERREUR - Les données du client n'ont pas été mises à jour", "Erreur de Mise à Jour", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        catch (Exception ex)
        {
            // En cas d'exception, affichage du message d'erreur correspondant
            MessageBox.Show(ex.Message);
        }
    }
}
```

Cette méthode récupère les informations entrées par l'utilisateur, tente de mettre à jour les informations du client dans la base de données et fournit des messages de confirmation ou d'erreur en fonction du succès de l'opération.

- **Création de la classe "*GuestClass*"** concernant l'interface Clients. Cette classe permet l'interaction avec la base de données, elle contient les quatre méthodes permettent d'ajouter, modifier, supprimer et récupérer un client de la base de données. Les requêtes sont paramétrées pour éviter les injections SQL
  extrait de la classe "*GuestClass*" – ajout d'un client :



```
// Crée une fonction pour insérer un nouveau client
1 référence | FakirMassine, Il y a 1 jour | 1 auteur, 1 modification
public bool insertClient(string id, string prenom, string nom, string phone, string city)
{
    /// <summary>
    /// Insère un nouveau client dans la base de données.
    /// </summary>
    /// <param name="id">L'identifiant du client.</param>
    /// <param name="prenom">Le prénom du client.</param>
    /// <param name="nom">Le nom du client.</param>
    /// <param name="phone">Le numéro de téléphone du client.</param>
    /// <param name="city">La ville du client.</param>
    /// <returns>Retourne true si l'insertion réussit, false sinon.</returns>
    Log.Debug("insertClient - Tentative d'insertion d'un nouveau client : ID={id}, Prenom={prenom}, Nom={nom}, Phone={phone}, City={city}", id, prenom, nom, phone, city);
    // Définit la requête SQL pour insérer un nouveau client
    string insertQuerry = "INSERT INTO `guest`(`GuestId`, `GuestPrenom`, `GuestNom`, `GuestPhone`, `GuestCity`) VALUES(@id,@prenom,@nom,@ph,@city)";
    MySqlCommand command = new MySqlCommand(insertQuerry, connect.GetConnection());
    //@id,@nom,@prenom,@ph,@city
    // Ajoute des paramètres à la requête SQL
    command.Parameters.Add("@id", MySqlDbType.VarChar).Value = id;
    command.Parameters.Add("@prenom", MySqlDbType.VarChar).Value = prenom;
    command.Parameters.Add("@nom", MySqlDbType.VarChar).Value = nom;
    command.Parameters.Add("@ph", MySqlDbType.VarChar).Value = phone;
    command.Parameters.Add("@city", MySqlDbType.VarChar).Value = city;

    // Ouvre la connexion à la base de données
    connect.OpenCon();
    if (command.ExecuteNonQuery() == 1)
    {
        // Ferme la connexion à la base de données et retourne true si l'insertion est réussie
        connect.CloseCon();
        Log.Debug("insertClient - Client inséré avec succès : ID={id}", id);
        return true;
    }
    else
    {
        // Ferme la connexion à la base de données et retourne false si l'insertion a échoué
        connect.CloseCon();
        Log.Debug("insertClient - Échec de l'insertion du client : ID={id}", id);
        return false;
    }
}
```

- **Gestion des événements du “*RoomForm*”** , donc les événement concernant l’appuis des boutons permettent l’ajout, la modification, la suppression des Chambres
  extrait de la classe “*RoomForm*" - click bouton permettent l’ajout client :

```
/// <summary>
/// Handler pour l'événement Click du bouton d'enregistrement.
/// </summary>
1 référence | FakiriYassine, Il y a 1 jour | 1 auteur, 1 modification
private void button_enregistrer_Click(object sender, EventArgs e)
{
    string no = textBox_id.Text;
    int type = Convert.ToInt32(comboBox_chambreType.SelectedValue.ToString());
    string ph = textBox_tel.Text;
    string status = radioButton_libre.Checked ? "Libre" : "Occupée";

    try
    {
        if (room.addRoom(no, type, ph, status))
        {
            MessageBox.Show("Chambre ajoutée avec succès", "Ajouter une Chambre", MessageBoxButtons.OK, MessageBoxIcon.Information);
            getRoomList();
            button_effacer.PerformClick();
        }
        else
        {
            MessageBox.Show("La chambre n'a pas été ajoutée", "Ajouter une Chambre", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Cette méthode récupère les informations entrées par l'utilisateur, tente d'ajouter une nouvelle chambre à l'aide de ces informations et affiche un message à l'utilisateur pour indiquer si l'ajout de la chambre a réussi ou échoué.

- **Création de la classe "*RoomClass*"** concernant l'interface des Chambres. Cette classe permet l'interaction avec la base de données, elle contient les méthodes permettent d'ajouter, modifier, supprimer et récupérer une chambre de la base de données. Les requêtes sont paramétrées pour éviter les injections SQL
  extrait de la classe "*RoomClass*" – ajout d'une chambre :

```
/// <summary>
/// Ajoute une nouvelle chambre.
/// </summary>
/// <param name="no">Numéro de la chambre.</param>
/// <param name="type">Type de la chambre.</param>
/// <param name="phone">Numéro de téléphone de la chambre.</param>
/// <param name="status">Statut de la chambre.</param>
/// <returns>Booléen indiquant si l'ajout a réussi.</returns>
1 référence | FakiriYassine, Il y a 1 jour | 1 auteur, 1 modification
public bool addRoom(string no, int type, string phone, string status)
{
    string insertQuerry = "INSERT INTO `room`(`RoomId`, `RoomType`, `RoomPhone`, `RoomStatus`) VALUES (@no,@type,@ph,@sts)";
    MySqlCommand command = new MySqlCommand(insertQuerry, connect.GetConnection());
    //@no,@type,@ph,@sts
    command.Parameters.Add("@no", MySqlDbType.VarChar).Value = no;
    command.Parameters.Add("@type", MySqlDbType.Int32).Value = type;
    command.Parameters.Add("@ph", MySqlDbType.VarChar).Value = phone;
    command.Parameters.Add("@sts", MySqlDbType.VarChar).Value = status;

    connect.OpenCon();
    if (command.ExecuteNonQuery() == 1)
    {
        connect.CloseCon();
        // La chambre a été ajoutée avec succès
        return true;
    }
    else
    {
        connect.CloseCon();
        // L'ajout de la chambre a échoué
        return false;
    }
}
```

Cette méthode ajoute une nouvelle chambre à la base de données en insérant les informations fournies. Elle retourne un booléen indiquant si l'opération d'ajout a réussi ou non.

- **Gestion des événements du "*ReservationForm*"**, donc les événement concernant l'appuis des boutons permettent l'ajout, la modification, la suppression des Réservations.
  extrait de la classe "*ReservationForm*" - click bouton permettent l'ajout d'une réservation :

```
/// <summary>
/// Événement appelé lorsqu'on clique sur le bouton "Ajouter".
/// Ajoute une nouvelle réservation avec les informations saisies par l'utilisateur.
/// </summary>
1 référence | FakirYassine, Il y a 1 jour | 1 auteur, 1 modification
private void button_ajouter_Click(object sender, EventArgs e)
{
    try
    {
        string guestId = textBox_clientId.Text;
        string roomNo = comboBox_ChambreNo.SelectedValue.ToString();
        DateTime dIn = dateTimePicker_dateArrivee.Value;
        DateTime dOut = dateTimePicker_dateDepart.Value;

        // La date d'arrivée doit être égale ou supérieure à la date d'aujourd'hui et
        // la date de départ doit être égale ou supérieure à la date d'arrivée
        if (dIn < DateTime.Today)
        {
            MessageBox.Show("La date d'arrivée de la réservation doit être aujourd'hui ou après", "Date invalide",
                MessageBoxButtons.OK, MessageBoxIcon.Error);

        }
        else if (dOut < dIn)
        {
            MessageBox.Show("La date de départ de la réservation doit être la même que la date d'arrivée ou après",
                "Date invalide", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
            if (reservation.addReserv(guestId, roomNo, dIn, dOut) && reservation.setReservRoom(roomNo, "Occupé"))
            {
                getReservTable();
                MessageBox.Show("Nouvelle réservation ajoutée avec succès", "Ajout de réservation",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show("La réservation n'a pas été ajoutée avec succès", "Erreur de réservation",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Erreur lors de l'ajout de la réservation", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Cette fonction est déclenchée quand on clique sur le bouton "Ajouter". Elle crée une nouvelle réservation, vérifie la validité des dates, puis informe l'utilisateur si l'opération a réussi ou échoué.

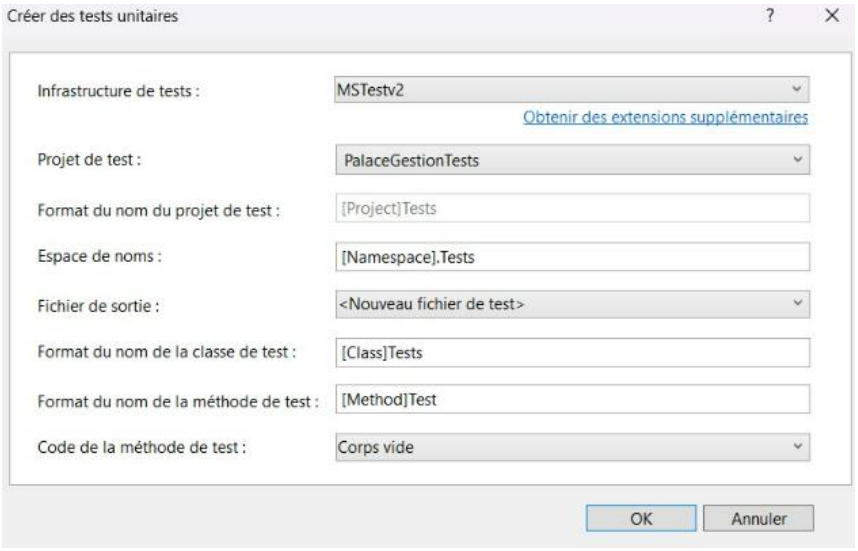
- **Création de la classe "*ReservationClass*"** concernant l'interface permettent de gérer les réservations. Cette classe permet l'interaction avec la base de données, elle contient les méthodes permettent d'ajouter, modifier, supprimer et récupérer une réservation de la base de données. Les requêtes sont paramétrées pour éviter les injections SQL
  extrait de la classe "*ReservationClass*" modification d'une réservation :

```
/// <summary>
/// Modifie une réservation existante.
/// </summary>
/// <param name="reserId">ID de la réservation.</param>
/// <param name="guestId">ID du client.</param>
/// <param name="roomNo">Numéro de la chambre.</param>
/// <param name="dateIn">Date d'arrivée.</param>
/// <param name="dateOut">Date de départ.</param>
/// <returns>Booléen indiquant si la modification a réussi.</returns>
1 référence | FakriYassine, Il y a 1 jour | 1 auteur, 1 modification
public bool editReserv(int reserId, string guestId, string roomNo, DateTime dateIn, DateTime dateOut)
{
    // La requête SQL
    string insertQuerry = "UPDATE `reservation` SET `GuestId`=@Gid,`RoomNo`=@Rno,`DateIn`=@Din,`DateOut`=@Dout WHERE `RecevId`=@rid";
    // Crée une commande SQL
    MySqlCommand command = new MySqlCommand(insertQuerry, connect.GetConnection());
    // Ajoute les paramètres à la commande SQL
    command.Parameters.Add("@rid", MySqlDbType.Int32).Value = reserId;
    command.Parameters.Add("@Gid", MySqlDbType.VarChar).Value = guestId;
    command.Parameters.Add("@Rno", MySqlDbType.VarChar).Value = roomNo;
    command.Parameters.Add("@Din", MySqlDbType.Date).Value = dateIn;
    command.Parameters.Add("@Dout", MySqlDbType.Date).Value = dateOut;

    // Ouvre la connexion à la base de données
    connect.OpenCon();
    // Exécute la commande SQL et vérifie si la requête a réussi
    if (command.ExecuteNonQuery() == 1)
    {
        connect.CloseCon();
        return true;
    }
    else
    {
        connect.CloseCon();
        return false;
    }
}
```

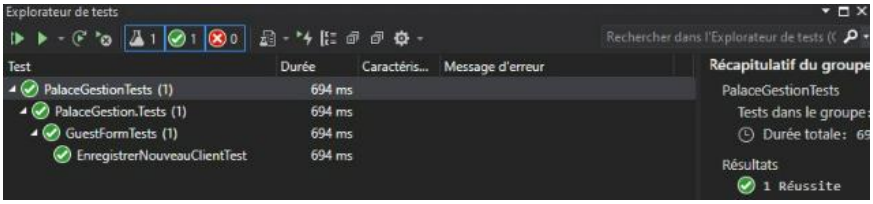
Cette fonction modifie une réservation existante en mettant à jour les informations fournies. Elle retourne un booléen indiquant si la modification a réussi ou non.

5. Mission 2

- J’ai effectué des **tests** pour garantir le bon fonctionnement de l’application, cela m’a permis de m’assurer que l’application répond aux exigences et se comporte comme prévu. Cela m’a permis de détecter des erreurs et garantir un bon fonctionnement du produit. Mise en place du test unitaire, création de la classe *“GuestFormTests”* qui permet de tester teste l'enregistrement d'un nouveau client dans la classe *“GuestFormTests”*. Il fournit des valeurs de test, appelle la méthode *“EnregistrerNouveauClient()”* et vérifie si l'opération a réussi avec *"Assert.IsTrue(result)*". Pour éviter toute sorte de problème j’ai réalise les tests sur une base de données dédié : *hotel-data_test*.

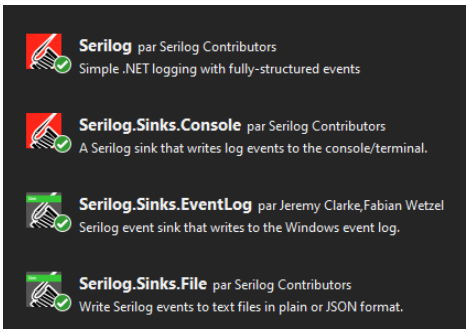



```
namespace PalaceGestion.Tests
{
    [TestClass()]
    0 références | FakiriYassine, il y a 2 jours | 1 auteur, 1 modification
    public class GuestFormTests
    {


        [TestMethod()]
        0 références | FakiriYassine, il y a 2 jours | 1 auteur, 1 modification
        public void EnregistrerNouveauClientTest()
        {

            var id = "test12345";
            var prenom = "testPrenom";
            var nom = "testNom";
            var phone = "testPhone";
            var city = "testCity";

            var guestForm = new GuestForm();


            var result = guestForm.EnregistrerNouveauClient(id, prenom, nom, phone, city);


            Assert.IsTrue(result);
        }
    }
}
```



- Utilisation de *Serilog* pour la **Journalisation** :
    - Pour faciliter la détection, ainsi que la résolution des problèmes dans l'application j'ai utilisé la bibliothèque *Serilog* qui m'as permis de capturer et d'enregistrer des informations détaillées sur le comportement de l'application. Cela permet de faciliter le débogage, identifier les erreurs et les résoudre. Cela améliore également la maintenance de l'application.

    - J'ai commencé par m'assurer que *Serilog* étais bien installe ainsi que les packages *Serilog.Sinks.Console* et *Serilog.Sinks.File*, qui permettent d'écrire les logs dans la console et dans les fichiers.



    - J'ai configuré *Serilog* dans le fichier "*Program.cs*" en définissant les options de journalisation, y compris l'écriture des logs dans la console, les fichiers "log.txt" et "errorlog.txt", ainsi que dans l'EventLog Windows.

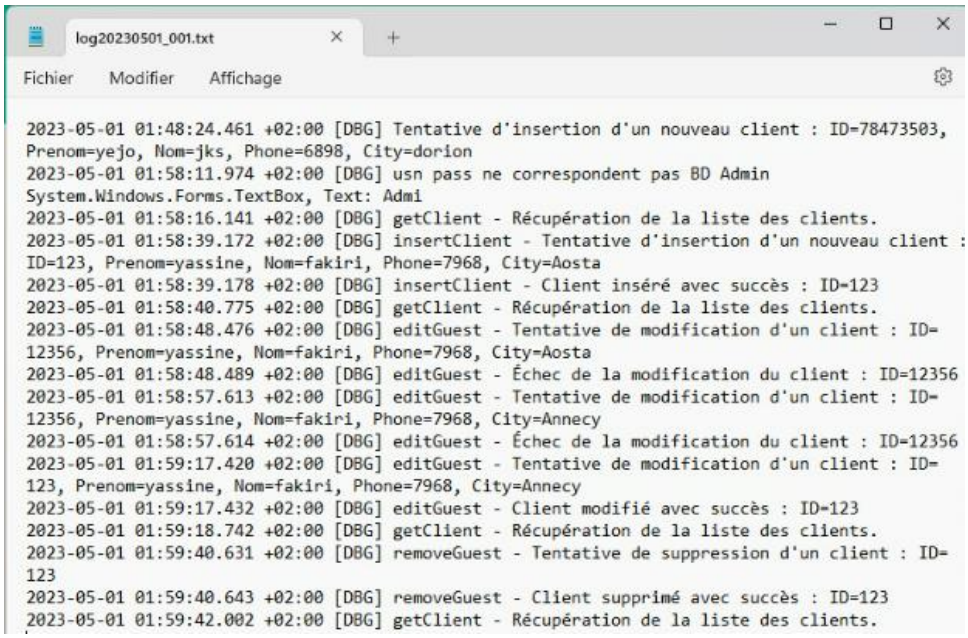
```
0 références | FakirYassine, Il y a 1 jour | 1 auteur, 2 modifications
static class Program
{
    /// <summary>
    /// Point d'entrée principal de l'application.
    /// </summary>
    [STAThread]
    0 références | FakirYassine, Il y a 1 jour | 1 auteur, 2 modifications
    static void Main()
    {
        // Configuration du logger Serilog pour écrire les logs dans la console,
        // les fichiers log.txt et errorlog.txt, ainsi que dans l'EventLog Windows
        Log.Logger = new LoggerConfiguration()
            .MinimumLevel.Verbose()
            .WriteTo.Console()
            .WriteTo.File("logs/log.txt",
            rollingInterval: RollingInterval.Day)
            .WriteTo.File("logs/errorlog.txt",
            restrictedToMinimumLevel: Serilog.Events.LogEventLevel.Information)
            .WriteTo.EventLog("NombreCache",
            manageEventSource: true,
            restrictedToMinimumLevel: Serilog.Events.LogEventLevel.Fatal)
            .CreateLogger();
```

    - Ensuite j'ai intégré des messages de **log** dans différentes parties du code de la classe "*GuestClass*". Par exemple, lors de l'insertion, la modification ou la suppression d'un client dans la base de données.
      Extrait classe "*GuestClass*" :

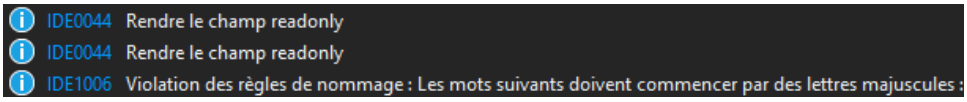
```
// Ouvre la connexion à la base de données
connect.OpenCon();
if (command.ExecuteNonQuery() == 1)
{
    // Ferme la connexion à la base de données et retourne true si l'insertion est réussie
    connect.CloseCon();
    Log.Debug("insertClient - Client inséré avec succès : ID={id}", id);
    return true;
}
else
{
    // Ferme la connexion à la base de données et retourne false si l'insertion a échoué
    connect.CloseCon();
    Log.Debug("insertClient - Échec de l'insertion du client : ID={id}", id);
    return false;
}
```

logs lors de l'ajout d'un client.

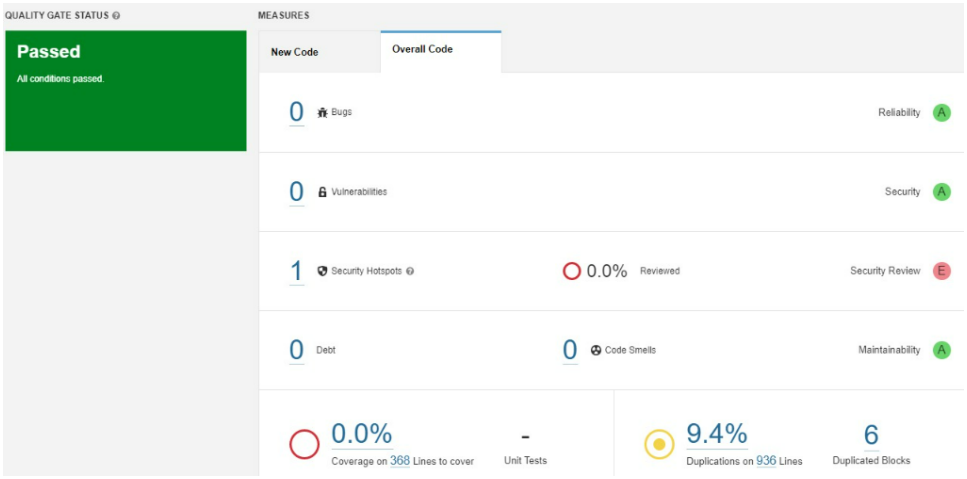
- Fichier des logs qui m'as permis de vérifier ce qui se passe pendent l'ajout, la modification et la suppression d'un client :



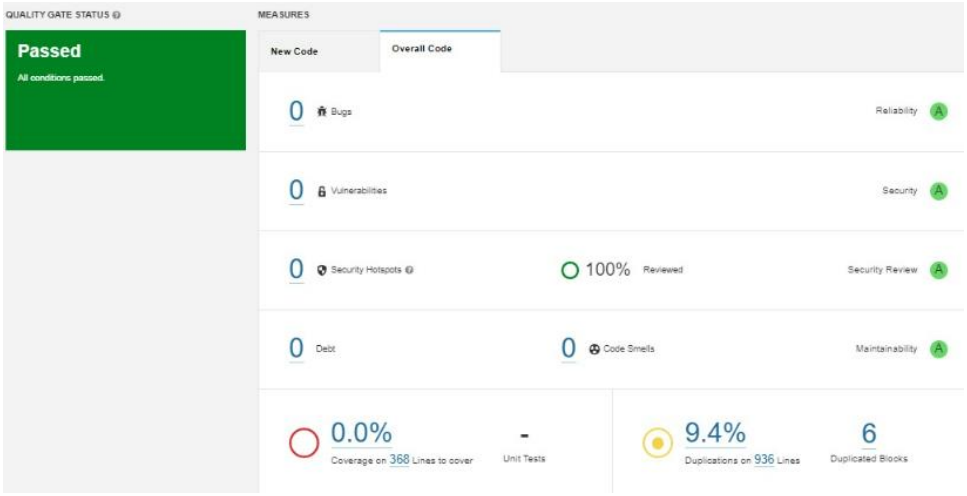
- Utilisation de *SonarLint* et *SonarQube* pour l'**analyse du code** :
  - J'ai nettoyé le code grâce à l'outil intégré à l'IDE ainsi qu'au plugin SonarLint extrait des messages :



- Ensuite j'ai analyse le code avec *SonarQube* qui m'as permis d'analyser le code en profondeur et de détecter des problèmes ainsi que de me fournir des conseils, il m'a permis également de corriges des problèmes de sécurité :
  Avant correctif :



Après correctif :

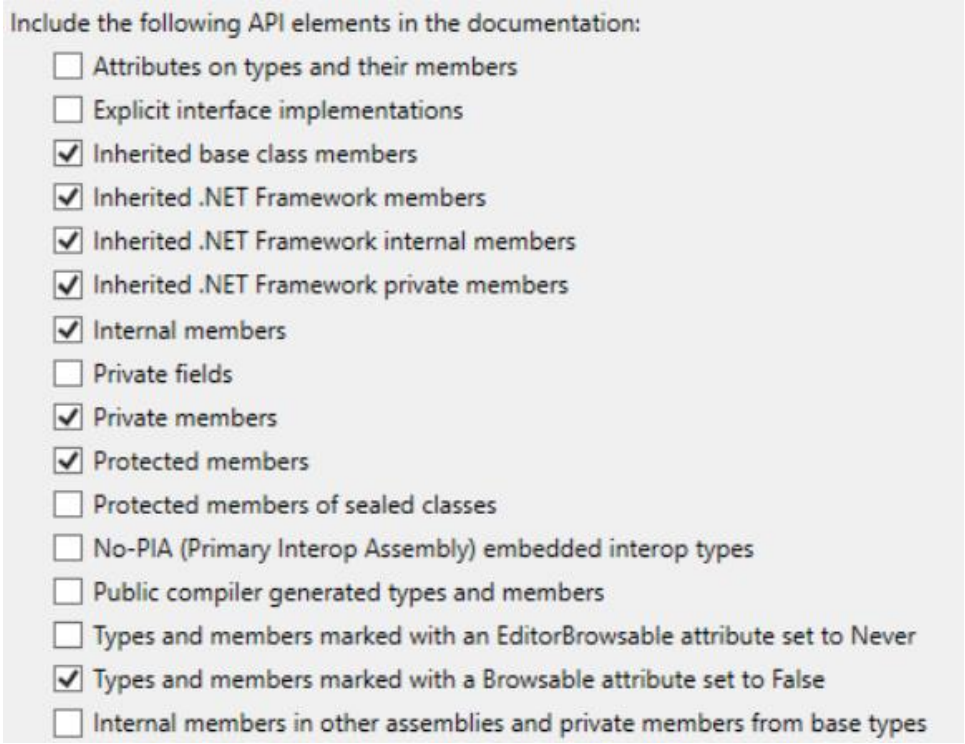


L'utilisation de ces outils m'as aidé à maintenir un code de bonne qualité et sécurisé, ce qui réduit les risques et rend l'application plus fiable.

6. Mission 3

- Créer la **documentation technique** : Dans cette étape, l'objectif est de créer une documentation technique détaillée pour les développeurs et les parties prenantes du projet. Elle doit également contenir des commentaires détaillés sur les différentes classes, méthodes et fonctions du code, afin de faciliter la compréhension et la

maintenance du projet. Pour cela j'ai utilisé *SandCastle* :
Je l'ai configuré de sorte à générer la documentation technique relative aux classes privés, publiques et internes :



- Préparer la **documentation utilisateur et produire une vidéo explicative** :
    - L'objectif ici est de fournir aux utilisateurs finaux des ressources complémentaires pour les aider à utiliser le logiciel. La documentation utilisateur doit expliquer les fonctionnalités du logiciel et comment les utiliser, en utilisant un langage simple et accessible. Les instructions étape par étape, les captures d'écran et les exemples concrets sont essentiels pour faciliter la compréhension.
    - En complément de la documentation écrite, une vidéo explicative a été réalisée pour présenter les principales fonctionnalités du logiciel et montrer comment les utiliser dans des situations réelles. Pour créer cette vidéo, la solution en ligne de capture vidéo screencapture.com a été utilisée. Cette vidéo permet aux utilisateurs de visualiser le logiciel en action et de se familiariser rapidement avec son fonctionnement.

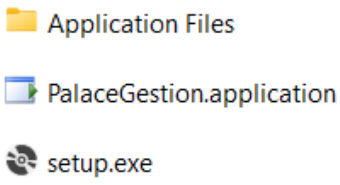
7. Mission 4

- J'ai entrepris le **déploiement de la base de données** en ligne sur un serveur web, en utilisant l'hébergeur *Hostinger* pour cette tâche. Pour cela, j'ai d'abord exporté le script de ma base de données locale pour ensuite l'importer sur phpMyAdmin de *Hostinger*. Une fois cette étape réalisée, j'ai créé le lien de connexion en utilisant le code suivant :

```
private readonly MySqlConnection connection =
    new MySqlConnection("datasource=onedomaine.tech;port=3306;username=u571651200_adminbddy2;password=**********;database=u571651200_hoteldata");
```

Cependant, malgré mes efforts, je n'arrivais pas à établir une connexion avec la base de données en ligne. J'ai passé beaucoup de temps à chercher la source du problème, et j'ai finalement découvert qu'un paramètre caché était responsable de cet échec. En effet, pour autoriser les connexions à distance, il était nécessaire d'ajuster ce paramètre caché dans les options de l'hébergeur *Hostinger*. Après avoir modifié ce paramètre, la connexion a finalement fonctionné comme prévu, permettant ainsi l'accès à la base de données en ligne.

- **Création du fichier d'installation zip** : Afin de faciliter l'installation du logiciel, un fichier zip contenant tous les fichiers nécessaires au fonctionnement du programme a été créé. Les utilisateurs peuvent simplement télécharger ce fichier, le décompresser et lancer l'application sans avoir à effectuer d'autres étapes d'installation complexes. Ce fichier a été ajouté dans le répertoire *publish* du projet :



- Utilisation de *Github* pour la gestion des versions et de *Trello* pour le suivi des tâches : J'ai employé Trello pour organiser et suivre les différentes tâches du projet.