RAPPORT DE PROJET

Création d'une application bureautique – Administrateur – Gestion des Matchs

Bachelor Développeur 22/24

Réalisé par :
MOHAMED ABDOU

Table des matières

Introduction :

Le présent rapport documente le développement et la mise en œuvre d'une application bureautique de gestion des matchs, conçue dans le cadre du projet STANIA. STANIA est une plateforme web dynamique dédiée à la gestion de matchs, aux paris sportifs et à la gestion de données relatives aux équipes et aux joueurs. Cette suite d'applications vise à faciliter et à améliorer l'expérience des utilisateurs engagés dans l'univers du sport, que ce soit en tant qu'administrateurs, parieurs ou fans.

Le projet STANIA est une initiative globale qui repose sur le développement de multiples solutions logicielles pour répondre aux besoins variés de sa communauté d'utilisateurs. L'application bureautique de gestion des matchs que nous abordons dans ce rapport est l'un des composants clés de cette suite. Elle s'adresse principalement aux administrateurs de la plateforme STANIA, leur offrant des outils puissants pour la gestion efficace des matchs sportifs.

Le développement de cette application bureautique est le fruit d'un travail méticuleux qui a mobilisé des compétences techniques en Python et l'utilisation de la bibliothèque Tkinter pour la création de l'interface utilisateur. Dans les sections à suivre, nous explorerons en détail les fonctionnalités de cette application, ses composants techniques, son intégration au sein de l'écosystème STANIA, ainsi que les perspectives d'amélioration pour répondre aux besoins futurs de l'utilisateur.

Ce rapport vise à offrir une vision d'ensemble complète de l'application de gestion des matchs, soulignant son importance dans le contexte de STANIA, et à fournir un guide pour ceux qui souhaitent comprendre son fonctionnement et ses possibilités d'extension.

Tkinter dans les applications bureautiques :

Tkinter, abréviation de "Tk interface," est une bibliothèque Python standard utilisée pour créer des interfaces graphiques utilisateur (GUI) dans les applications bureautiques. Tkinter repose sur le kit de développement d'interface utilisateur Tk, qui a été développé à l'origine pour le langage de programmation Tcl (Tool Command Language). Il fournit un ensemble de modules et de composants permettant aux développeurs de créer des fenêtres, des boutons, des zones de texte, des menus, des boîtes de dialogue et bien d'autres éléments d'interface utilisateur interactifs.

Les applications bureautiques sont des logiciels conçus pour faciliter et améliorer la productivité dans un environnement de travail. Tkinter est largement utilisé dans le développement de telles applications en raison de sa simplicité, de sa polyvalence et de son intégration native avec Python. Voici quelques points clés à retenir concernant l'utilisation de Tkinter dans les applications bureautiques :

1. **Facilité d'Utilisation** : Tkinter est apprécié pour sa simplicité, ce qui en fait un choix idéal pour les développeurs qui souhaitent créer des interfaces utilisateur conviviales sans avoir à gérer des détails complexes.

2. **Personnalisation** : Tkinter offre des possibilités de personnalisation importantes, permettant aux développeurs de concevoir des interfaces qui correspondent à l'esthétique de leur application ou de leur entreprise.

3. **Interaction Utilisateur** : Les applications bureautiques nécessitent souvent une interaction utilisateur efficace, et Tkinter propose une variété de widgets et d'éléments d'interface pour répondre à ces besoins.

4. **Intégration** : Tkinter s'intègre harmonieusement avec d'autres bibliothèques et frameworks Python, ce qui facilite l'ajout d'interfaces graphiques à des applications existantes.

5. **Portabilité** : Les applications développées avec Tkinter sont portables, ce qui signifie qu'elles peuvent être exécutées sur différentes plates-formes sans modifications significatives.

Dans le contexte de notre application de gestion des matchs pour STANIA, l'utilisation de Tkinter nous a permis de créer une interface utilisateur conviviale, tout en facilitant la gestion des matchs par les administrateurs de la plateforme.

Dans les sections suivantes de ce rapport, nous explorerons plus en détail comment Tkinter a été utilisé pour créer l'interface de l'application bureautique et comment cela a contribué à la réussite de notre projet.

Introduction à l'Utilisation de PyCharm pour le Développement de l'Application Bureautique

Le choix d'un environnement de développement intégré (IDE) joue un rôle crucial dans le processus de création d'une application bureautique. PyCharm, développé par JetBrains, est devenu un choix populaire parmi les développeurs Python en raison de ses fonctionnalités avancées, de son support complet de Python, et de sa facilité d'utilisation. Ce rapport examine l'utilisation de PyCharm comme éditeur principal pour le développement de notre application bureautique de gestion des matchs dans le contexte de la suite STANIA.

PyCharm offre un ensemble d'outils puissants qui facilitent la conception, le débogage et la gestion de projets Python, ce qui en fait un choix judicieux pour les développeurs soucieux de la qualité et de la productivité. Dans cette section, nous explorerons comment PyCharm a été utilisé pour optimiser le processus de développement de notre application bureautique.

Nous discuterons des caractéristiques et des avantages spécifiques de PyCharm, notamment :

1. **Intégration de Python** : PyCharm offre une intégration transparente avec Python, facilitant la gestion des dépendances, la création d'environnements virtuels, et l'exécution de scripts Python directement depuis l'IDE.

2. **Assistance à la Programmation** : L'IDE propose un ensemble d'outils pour l'autocomplétion, la vérification statique, et la suggestion de code, ce qui accélère le processus de développement et améliore la qualité du code.

3. **Débogage Avancé** : PyCharm permet un débogage avancé, offrant des fonctionnalités telles que le débogage à distance, le suivi des variables, et la possibilité de fixer des points d'arrêt pour détecter les erreurs plus rapidement.

4. **Gestion de Projet** : L'IDE simplifie la gestion de projets, y compris le suivi des versions, l'intégration avec des systèmes de contrôle de code source, et des outils de gestion de paquets.

5. **Interface Conviviale** : PyCharm offre une interface utilisateur intuitive, personnalisable et adaptable aux préférences de chaque développeur, améliorant ainsi le confort et la productivité de l'utilisateur.

Dans les sections suivantes de ce rapport, nous examinerons comment PyCharm a été utilisé pour le développement de notre application bureautique, en mettant en évidence les fonctionnalités spécifiques qui ont contribué à son succès et à son efficacité.

Exemple d’une fenêtre Tkinter code :

Utilisation de Pycharm interface

```
import tkinter as tk

fenetre = tk.Tk()

# Titre de la fenêtre
fenetre.title("Ma Fenêtre Tkinter")

# Dimensions de la fenêtre (largeur x hauteur)
fenetre.geometry("400x300")

# Boucle principale pour afficher la fenêtre
fenetre.mainloop()
```

But de l'Application Bureautique de Gestion des Matchs :

L'application bureautique que nous avons développée est un outil destiné à simplifier la gestion des données des matchs sportifs dans la base de données de la suite STANIA. Son rôle principal est de permettre aux administrateurs d'effectuer des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur les enregistrements de matchs. Plus spécifiquement, ses fonctions clés incluent :

1. **Création de Matchs** : Les administrateurs peuvent ajouter de nouveaux matchs à la base de données en spécifiant des détails tels que la date, l'heure, les équipes participantes et le lieu.
2. **Consultation des Matchs** : L'application offre aux administrateurs un moyen convivial de consulter les informations existantes sur les matchs enregistrés, fournissant un aperçu rapide et complet des données de match.
3. **Mise à Jour des Matchs** : Les administrateurs ont la possibilité de mettre à jour les enregistrements de matchs existants, que ce soit pour corriger des erreurs, mettre à jour les scores, ou modifier d'autres détails pertinents.
4. **Suppression de Matchs** : En cas de besoin, les administrateurs peuvent supprimer des matchs de la base de données, garantissant ainsi la précision et la pertinence des données.

L'objectif principal de cette application est de simplifier et d'automatiser le processus de gestion des matchs pour les administrateurs, en éliminant la nécessité d'intervenir directement

dans la base de données ou d'utiliser des requêtes SQL complexes. Elle vise à améliorer l'efficacité de la gestion tout en minimisant les risques d'erreurs humaines.

Dans les sections suivantes de ce rapport, nous examinerons en détail comment l'application a été conçue pour atteindre ces objectifs, y compris les fonctionnalités spécifiques qui facilitent les opérations CRUD sur les matchs dans la base de données.

Réalisation et mise en Œuvre :

La partie code sur Tkinter :

La base de donnees et la creation de la table Match avec Mysqlite3 :

MySQLite3 est un système de gestion de base de données relationnelle (SGBDR) intégré à Python. Il offre une solution légère et simple pour la gestion des données, stockées dans un fichier unique, ce qui en fait un choix idéal pour les applications locales ou embarquées. Il prend en charge la création, la modification, la lecture et la suppression de données de manière efficace, tout en offrant une compatibilité SQL complète.

Code sur Pycharm :

Creation de la table des matchs :

```
import sqlite3

# Step 2: Establish a connection to the SQLite database (or create one if it doesn't exist)
conn = sqlite3.connect('db2')

# Step 3: Create a cursor object to interact with the database
cursor = conn.cursor()

# Step 4: Execute the SQL query to create the "match" table
create_table_query = '''
CREATE TABLE IF NOT EXISTS match (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    team1 TEXT,
    team2 TEXT,
    date_debut DATE,
    date_fin DATE,
    status TEXT,
    cote1 FLOAT,
    cote2 FLOAT,
    commentaires TEXT
);
'''
cursor.execute(create_table_query)

conn.commit()

conn.close()
```

Etablissement de la connection avec la base de donnees MySqlite3

```
import sqlite3

class Connection:

    def __init__(self):
        pass

    def conn(self):

        try:
            self.con = sqlite3.connect('db2')
            return self.con
        except sqlite3.Error as err:
            print(f"connection failed! {err}")
```

Le fichier Class.py qui contient les differents opérations CRUD sur la table des matchs :

- L'insertion :

```
def add(self,*args):
    try:
        con = self.conn()
        cur = con.cursor()
        cur.execute("INSERT INTO match(team1, team2, date_debut, date_fin, status, cote1, cote2, commentaires)
VALUES(?,?,?,?,?,?,?,?)",args)
        con.commit()
        return "The match has been sucesfully added"
    except sqlite3.Error as err:
        print(f"{err}")
    finally:
        self.con.close()
```

- La modification

```
def update(self,*args):
    try:
        con = self.conn()
        cur = con.cursor()
        cur.execute("UPDATE match SET team1 = ?, team2 = ?, date_debut = ?, date_fin = ?, status = ?, cote1 = ?, cote2 =
?, commentaires = ? WHERE id = ?",args)
        con.commit()
        return "Le match a été bien modifié"
    except sqlite3.Error as err:
        print(f"{err}")
    finally:
        self.con.close()
```

- La selection:

```
def select(self):
    try:
        cur = self.conn().cursor()
```

```
        cur.execute("SELECT * FROM match")
        return cur.fetchall()
    except sqlite3.Error as err:
        print(f"{err}")
    finally:
        self.con.close()
```

- La suppression:

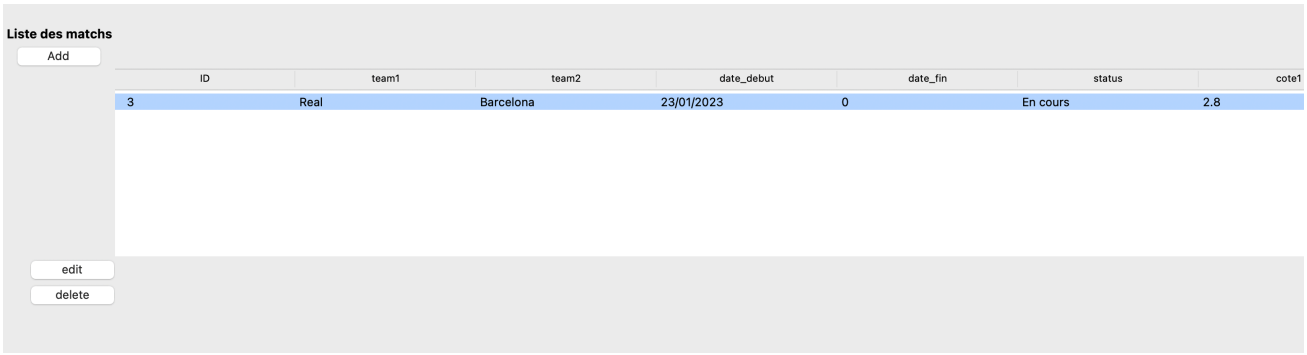
```
def delete(self,*args):
    try:
        con = self.conn()
        cur = con.cursor()
        cur.execute("DELETE FROM match WHERE id = ?",args)
        con.commit()
        return "Le match a été supprimé avec success"
    except sqlite3.Error as err:
        print(f"{err}")
    finally:
        self.con.close()
```

- Le formulaire de l'ajout d'un Match :

```
self.window_add.title("Ajouter un Match")
self.window_add.configure(padx=20, pady=20)

self.itm_add_label = ttk.Label(self.window_add, text="Ajouter match",
                               font=("Bahnschrift", 14, 'bold'))

self.itm_add_label.grid(row=0, column=1, columnspan=5)

self.team1_name_label = ttk.Label(self.window_add, text="Team 1", width=20)
self.team1_name_label.grid(row=1, column=1, columnspan=2)
self.team1_name = ttk.Entry(self.window_add, width=40)
self.team1_name.grid(row=1, column=3, columnspan=3)

self.team2_name_label = ttk.Label(self.window_add, text="Team 2", width=20)
self.team2_name_label.grid(row=2, column=1, columnspan=2)
self.team2_name = ttk.Entry(self.window_add, width=40)
self.team2_name.grid(row=2, column=3, columnspan=3)

self.date_debut_label = ttk.Label(self.window_add, text="Date de début", width=20)
self.date_debut_label.grid(row=3, column=1, columnspan=2)
self.date_debut = ttk.Entry(self.window_add, width=40)
self.date_debut.grid(row=3, column=3, columnspan=3)

self.date_fin_label = ttk.Label(self.window_add, text="Date de fin", width=20)
self.date_fin_label.grid(row=4, column=1, columnspan=2)
self.date_fin = ttk.Entry(self.window_add, width=40)
self.date_fin.grid(row=4, column=3, columnspan=3)

self.status_label = ttk.Label(self.window_add, text="status", width=20)
self.status_label.grid(row=5, column=1, columnspan=2)
self.status = ttk.Entry(self.window_add, width=40)
self.status.grid(row=5, column=3, columnspan=3)

self.cote1_label = ttk.Label(self.window_add, text="La cote de l'équipe 1", width=20)

self.cote1_label.grid(row=5, column=1, columnspan=2)
```

```
self.cote1 = ttk.Entry(self.window_add, width=40)
self.cote1.grid(row=5, column=3, columnspan=3)

self.cote2_label = ttk.Label(self.window_add, text="La cote de l'équipe 2", width=20)
self.cote2_label.grid(row=6, column=1, columnspan=2)
self.cote2 = ttk.Entry(self.window_add, width=40)
self.cote2.grid(row=6, column=3, columnspan=3)

self.commentaire_label = ttk.Label(self.window_add, text="Commentaires", width=20)
self.commentaire_label.grid(row=7, column=1, columnspan=2)
self.commentaire = ttk.Entry(self.window_add, width=40)
self.commentaire.grid(row=7, column=3, columnspan=3)



self.btn_add_product = ttk.Button(self.window_add, text="Ajouter le match",
                                  command=self.add_item)
self.btn_add_product.grid(row=8, column=1, columnspan=5)
```

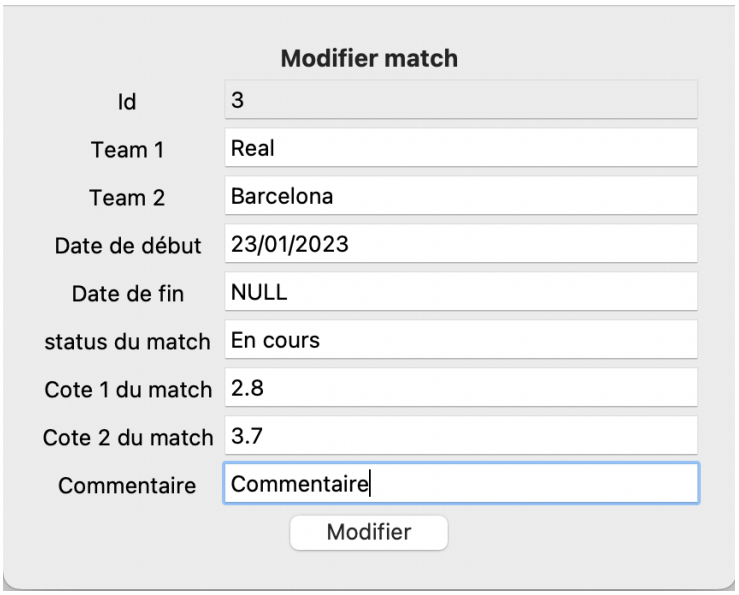
Screens de l'application



En cliquant sur Add :

En cliquant sur Édit :

Avant de clique sur le bouton édit, il faut sélectionner le match a modifier.



En cliquant sur delete :

On remarque donc la suppression du match.

Conclusion

Lorsqu'il s'agit de développer une application bureautique avec Tkinter, cette bibliothèque offre une solution robuste pour créer des interfaces utilisateur conviviales. Tkinter, étant la bibliothèque GUI standard de Python, présente plusieurs avantages significatifs pour le développement d'applications bureautiques.

Tout d'abord, Tkinter est facile à prendre en main, ce qui en fait un excellent choix pour les débutants en programmation. Sa simplicité permet de créer rapidement des interfaces utilisateur fonctionnelles sans une courbe d'apprentissage abrupte. En utilisant Tkinter, il est possible de concevoir des applications bureautiques pour la gestion de documents, de données, des tableurs, des éditeurs de texte, et bien plus encore.

De plus, Tkinter est livré avec une vaste bibliothèque de widgets prédéfinis, tels que des boutons, des listes, des champs de texte, des menus, etc., facilitant ainsi la création d'interfaces riches et interactives. Ces éléments peuvent être agencés et personnalisés pour répondre aux besoins spécifiques de l'application bureautique en cours de développement.

Tkinter facilite également l'intégration avec d'autres bibliothèques et modules Python, ce qui permet d'ajouter des fonctionnalités avancées à l'application bureautique. De plus, étant une bibliothèque standard, Tkinter est généralement déjà installé avec Python, éliminant ainsi le besoin de téléchargement ou d'installation supplémentaire.

En somme, Tkinter apporte une solution simple et efficace pour le développement d'applications bureautiques avec Python. Son ensemble d'outils conviviaux et sa polyvalence en font un choix judicieux pour créer des logiciels de bureau destinés à des tâches variées, offrant une base solide pour les développeurs cherchant à construire des applications bureautiques fonctionnelles et intuitives.