#Team Formation Tool POC:

# CRUD: Player profiles.

# create\_team\_rosters(team\_id): Assist in creating balanced team rosters based on player skills and statistics.

# track\_team\_formation\_changes(change\_data): Track changes and updates to team formations.

#Code:

#To tackle this problem statement effectively, we'll structure our solution into the following core components:

# \*\*Player Class\*\*: Represents a player profile with methods for creation, reading, updating, and deleting profiles.

# \*\*Team Class\*\*: Manages team rosters and operates on a collection of Player objects.

# \*\*FormationTracker Class\*\*: Tracks and records changes in team formations.

# \*\*RosterBalancer Algorithm\*\*: Algorithm for balancing teams based on player skills and statistics.

#Here is the complete Python code implementing these functionalities:

from typing import List, Dict

class Player:

    def \_\_init\_\_(self, player\_id: int, name: str, skill\_level: int):

        self.player\_id = player\_id

        self.name = name

        self.skill\_level = skill\_level

    def \_\_str\_\_(self):

        return f"{self.name} (Skill Level: {self.skill\_level})"

    def update\_profile(self, \*\*kwargs):

        for key, value in kwargs.items():

            setattr(self, key, value)

class Team:

    def \_\_init\_\_(self, team\_id: int):

        self.team\_id = team\_id

        self.players = []

    def add\_player(self, player: Player):

        if player not in self.players:

            self.players.append(player)

    def remove\_player(self, player\_id: int):

        self.players = [player for player in self.players if player.player\_id != player\_id]

    def get\_average\_skill(self):

        total\_skill = sum(player.skill\_level for player in self.players)

        return total\_skill / len(self.players) if self.players else 0

    def \_\_str\_\_(self):

        return f"Team {self.team\_id} with players: {[str(player) for player in self.players]}"

class FormationTracker:

    def \_\_init\_\_(self):

        self.changes = []

    def record\_change(self, team\_id: int, change\_data: str):

        self.changes.append((team\_id, change\_data))

    def get\_changes(self):

        return self.changes

class RosterBalancer:

    def create\_balanced\_teams(self, players: List[Player], num\_teams: int) -> Dict[int, Team]:

        teams = {i: Team(i) for i in range(num\_teams)}

        sorted\_players = sorted(players, key=lambda x: x.skill\_level, reverse=True)

        for i, player in enumerate(sorted\_players):

            teams[i % num\_teams].add\_player(player)

        return teams

def display\_menu():

    print("\nMenu:")

    print("1. Create Player")

    print("2. Create Team")

    print("3. Add Player to Team")

    print("4. Remove Player from Team")

    print("5. Display Team Average Skill")

    print("6. Display All Changes")

    print("7. Create Balanced Teams")

    print("8. Exit")

def main():

    players = []

    teams = {}

    tracker = FormationTracker()

    balancer = RosterBalancer()

    while True:

        display\_menu()

        choice = input("Choose an option: ")

        if choice == '1':

            try:

                player\_id = int(input("Enter player ID: "))

                name = input("Enter player name: ")

                skill\_level = int(input("Enter player skill level: "))

                player = Player(player\_id, name, skill\_level)

                players.append(player)

                print(f"Player created: {player}")

            except ValueError:

                print("Invalid input. Please enter numeric values for ID and skill level.")

        elif choice == '2':

            try:

                team\_id = int(input("Enter team ID: "))

                if team\_id not in teams:

                    teams[team\_id] = Team(team\_id)

                    print(f"Team {team\_id} created.")

                else:

                    print("Team ID already exists.")

            except ValueError:

                print("Invalid input. Please enter a numeric value for team ID.")

        elif choice == '3':

            try:

                team\_id = int(input("Enter team ID to add player to: "))

                player\_id = int(input("Enter player ID to add: "))

                player = next((p for p in players if p.player\_id == player\_id), None)

                if player and team\_id in teams:

                    teams[team\_id].add\_player(player)

                    tracker.record\_change(team\_id, f"Added player {player\_id}")

                    print(f"Player {player.name} added to Team {team\_id}.")

                else:

                    print("Player or Team not found.")

            except ValueError:

                print("Invalid input. Please enter numeric values for IDs.")

        elif choice == '4':

            try:

                team\_id = int(input("Enter team ID to remove player from: "))

                player\_id = int(input("Enter player ID to remove: "))

                if team\_id in teams:

                    teams[team\_id].remove\_player(player\_id)

                    tracker.record\_change(team\_id, f"Removed player {player\_id}")

                    print(f"Player {player\_id} removed from Team {team\_id}.")

                else:

                    print("Team not found.")

            except ValueError:

                print("Invalid input. Please enter numeric values for IDs.")

        elif choice == '5':

            try:

                team\_id = int(input("Enter team ID to display average skill: "))

                if team\_id in teams:

                    average\_skill = teams[team\_id].get\_average\_skill()

                    print(f"Average skill for Team {team\_id}: {average\_skill:.2f}")

                else:

                    print("Team not found.")

            except ValueError:

                print("Invalid input. Please enter a numeric value for team ID.")

        elif choice == '6':

            changes = tracker.get\_changes()

            print("Formation Changes:")

            for change in changes:

                print(f"Team {change[0]}: {change[1]}")

        elif choice == '7':

            try:

                num\_teams = int(input("Enter number of teams to create: "))

                if num\_teams > 0:

                    balanced\_teams = balancer.create\_balanced\_teams(players, num\_teams)

                    for team in balanced\_teams.values():

                        print(team)

                else:

                    print("Number of teams must be positive.")

            except ValueError:

                print("Invalid input. Please enter a numeric value for the number of teams.")

        elif choice == '8':

            print("Exiting...")

            break

        else:

            print("Invalid option. Please try again.")

if \_\_name\_\_ == '\_\_main\_\_':

    main()