DAV BOYS SENIOR SECONDARY SCHOOL, GPM

DEPARTMENT OF COMPUTER SCIENCE

FINAL-YEAR PROJECT

2025-26

BONAFIDE CERTIFICATE

INTERNAL ASSESMENT

CERTIFIED TO BE THE BONAFIDE WORK IN ______________________________

DONE BY _________________________ OF CLASS______ SECTION______ OF

____

DURING THE YEAR 20___- 20____

SIGNATURE OF PRINCIPAL
SCHOOL SEAL

SIGNATURE OF SUBJECT TEACHER
DESIGNATION: PGT/TGT

SUBMITTED FOR THE PRACTICAL EXAM HELD ON _________ AT

____

INTERNAL EXAMINER

EXTERNAL EXAMINER

CHIEF SUPERINTENDANT

DATE:

ACKNOWLEDEGMENT

I would first and foremost like to like to express my gratitude to the almighty for making this all possible, being with me every-step of the way and guiding me through this academic year.

I must also convey my thanks to my computer science teacher, Mrs. Karthika and the Head of the computer science department, Mrs. Hemalatha, for their guidance throughout the academic year and allowing this project to take form, without whom this project would not be possible

I also express my gratitude to the school management and the principal for giving me this opportunity

Finally, I extend my gratitude to my parents and all the people who were supportive in the effective completion of this project.

-Sraeshta Sabarish
XII-D

TABLE OF CONTENTS

BRIEF OVERVIEW

This project
'Chess Tournament Database Management System' is a comprehensive application with the aim of assistance in conducting chess tournaments with ease, and provides a consolidated system to keep track of the various rounds in a chess tournament and easily manage tournament pairings and results with minimal user effort.

This application features a robust back-end architecture supplied by Flask that ensures a seamless connection between the client-side interface and the database, which is set up using MySQL, to securely and reliably store the tournament pairing and results. all of this is put together using a HTML based front end, keeping it easy to run on any system without much difficulty

REASON FOR CHOOSING THIS PROJECT

While there exist a number of software in the global domain with the aim of providing a system for chess tournament management, most platforms are made solely for the purpose of managing tournaments of official status(FIDE), as a result of which most of these applications are overwhelming to setup in situations where the scrutiny of an international body is not involved.

Meanwhile, this project aims to provide a simple interface to amateur tournaments with minimal requirements and a extremely simple setup, while also providing everything required to conduct a chess tournament, while also being really lightweight and can virtually be run on most PC's keeping requirements minimal.

As a budding developer, I chose this project to explore database management, and web development frameworks

SOFTWARE AND HARDWARE REQUIREMENTS

This program being decently lightweight can run on any 64 or 32 bit operating system with python/MySQL installed, the recommended setup is as follows:

Software Requirements:

- Python
  - + Flask Framework
  - + MySQL Connector
- MySQL
- A web browser(that supports modern css)

Hardware Requirements:

- A computer with a mainstream OS(win, mac, or linux)
  - OS must support MySQL
- A minimum of 4GB RAM is recommended
- 6-8 GB of disc space is recommended for seamless storage of tournament data

Required Python Libraries

for the execution of the python application certain python libraries are prerequisite(mentioned above), they can be installed by running the following 'PIP' commands.

```
pip install flask
pip install mysql-connector-python
```

Note: PIP & Python must be added to path (Environmental variables) before the above commands may be executed

PROJECT CODE

Modules used:

Flask: A micro web framework used to handle server side application logic
MySQL Connector: python library to connect and interact with the MySQL database

APP.PY

```
import mysql.connector
from mysql.connector import Error
import random
from flask import Flask, render_template, request, redirect, url_for, flash, make_response
import csv
import io

app = Flask(__name__)
app.secret_key = 'very_imp'


DB_CONFIG = {
        'host': 'localhost',
        'database': 'chess_tournament',
        'user': 'root',
        'password': 'root'
}

def get_db_connection():
        try:
                conn = mysql.connector.connect(**DB_CONFIG)
                return conn
        except Error as e:
                print(f"Error connecting to MySQL: {e}")
                return None

def calculate_points(tournament_id):
        conn = get_db_connection()
        if conn:
                cursor = conn.cursor(dictionary=True)
                query = """
                SELECT p.id, p.name, p.rating,
                COALESCE(SUM(CASE
                        WHEN m.player1_id = p.id THEN m.result
                        WHEN m.player2_id = p.id THEN 1 - m.result
                        END), 0) AS points
                FROM players p
                LEFT JOIN matches m ON (m.player1_id = p.id OR m.player2_id = p.id) AND m.result
IS NOT NULL
                WHERE p.tournament_id = %s
                GROUP BY p.id
                ORDER BY points DESC, p.rating DESC
                """
                cursor.execute(query, (tournament_id,))
                points = cursor.fetchall()
                cursor.close()
                conn.close()
                return points
        return []

def all_results_entered(tournament_id, round_num):
        conn = get_db_connection()
        if conn:
                cursor = conn.cursor(dictionary=True)
                cursor.execute("""
                        SELECT COUNT(*) AS pending
                        FROM matches
                WHERE tournament_id = %s AND round_num = %s AND result IS NULL AND player2_id
IS NOT NULL
            """, (tournament_id, round_num))
            pending = cursor.fetchone()['pending']
            cursor.close()
            conn.close()
            return pending == 0
        return False

def generate_pairings(tournament_id, round_num):
        conn = get_db_connection()
        if not conn:
            return False

        cursor = conn.cursor(dictionary=True)

        cursor.execute("SELECT COUNT(*) AS count FROM matches WHERE tournament_id = %s AND
round_num = %s", (tournament_id, round_num))
        if cursor.fetchone()['count'] > 0:
            conn.close()
            return False

        players = calculate_points(tournament_id)
        if round_num == 1:
            players.sort(key=lambda x: x['rating'], reverse=True)

        sorted_players = players

        n = len(sorted_players)
        if n < 2:
            conn.close()
            return False

        bye_player = None
        if n % 2 == 1:
            bye_player = sorted_players.pop()
            n -= 1
            cursor.execute("""
                INSERT INTO matches (tournament_id, round_num, player1_id, player2_id, result)
                VALUES (%s, %s, %s, NULL, 1.0)
            """, (tournament_id, round_num, bye_player['id']))

           if round_num == 1:
               num_groups = 2
        else:
            num_groups = 2 ** (round_num - 1)

           if num_groups > n:
               num_groups = n

        group_size = n // num_groups
        remainder = n % num_groups

        groups = []
        start = 0
        for i in range(num_groups):
            size = group_size + 1 if i < remainder else group_size
            groups.append(sorted_players[start:start + size])
            start += size

        for    i    in    range(0,      num_groups, 2):
            ifi +1 >=num_groups:
                break
            group_a = groups[i]
            group_b    =   groups[i + 1]


            random.shuffle(group_b)


            for a, b in zip(group_a, group_b):
                p1_id    =    min(a['id'],    b['id'])    p2_id    =
                max(a['id'], b['id']) cursor.execute("""

                    INSERT INTO matches (tournament_id, round_num, player1_id, player2_id,
result)
                    VALUES (%s, %s, %s, %s, NULL)
                """, (tournament_id, round_num, p1_id, p2_id))

        conn.commit()
        cursor.close()
        conn.close()
        return True

@app.route('/')
def index():
        conn = get_db_connection()
        if conn:
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT * FROM tournaments")
            tournaments = cursor.fetchall()
            cursor.close()
            conn.close()
            return render_template('index.html', tournaments=tournaments)
        flash('Database connection error')
        return render_template('index.html', tournaments=[])

@app.route('/create_tournament', methods=['GET', 'POST'])
def create_tournament():
        if request.method == 'POST':
            name = request.form.get('name', '').strip()
            num_rounds_str = request.form.get('num_rounds', '')

            if not name:
                flash('Tournament name is required!')
                return render_template('create_tournament.html')

            try:
                num_rounds = int(num_rounds_str)
                if num_rounds <= 0:
                    raise ValueError
            except ValueError:
                flash('Number of rounds must be a positive integer!')
                return render_template('create_tournament.html')

            conn = get_db_connection()
```

```
            if conn:
                try:
                    cursor = conn.cursor()
                    cursor.execute("INSERT INTO tournaments (name, num_rounds) VALUES (%s,
%s)", (name, num_rounds))
                    conn.commit()
                    tournament_id = cursor.lastrowid
                    cursor.close()
                    conn.close()
                    flash('Tournament created successfully!')
                    return redirect(url_for('tournament', id=tournament_id))
                except Error as e:
                    flash(f'Error creating tournament: {e}')
        else:
            flash('Database connection error')
    return render_template('create_tournament.html')

@app.route('/tournament/<int:id>', methods=['GET', 'POST'])
def tournament(id):
    conn = get_db_connection()
    if not conn:
        flash('Database connection error')
        return redirect(url_for('index'))

    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM tournaments WHERE id = %s", (id,))
    tour = cursor.fetchone()
    if not tour:
        flash('Tournament not found')
        conn.close()
        return redirect(url_for('index'))

    if request.method == 'POST':
        if 'add_player' in request.form:
            name  =  request.form.get('name',  '').strip()  rating_str  =
            request.form.get('rating', '') if not name:

                flash('Player name is required!')
            else:
                try:
                    rating = int(rating_str)
                    if rating < 0:
                        raise ValueError
                    cursor.execute("INSERT INTO players (tournament_id, name, rating)
VALUES (%s, %s, %s)", (id, name, rating))
                    conn.commit()
                    flash('Player added!')
                except ValueError:
                    flash('Rating must be a non-negative integer!')
                except Error as e:
                    flash(f'Error adding player: {e}')
        elif 'csv_file' in request.files:
            file = request.files['csv_file']
            if file and file.filename.endswith('.csv'):
                try:
                    csv_data = io.StringIO(file.stream.read().decode('utf-8'))
                    reader = csv.reader(csv_data)
                    header = next(reader, None) # Skip header if present
                    added = 0
                    for row in reader:
                        if len(row) >= 2:
                            name = row[0].strip()
                            rating_str = row[1].strip()
                            if not name:
                            continue
                            try:
                                rating = int(rating_str)
                                if rating < 0:
                                    raise ValueError
                                cursor.execute("INSERT INTO players (tournament_id, name,
rating) VALUES (%s, %s, %s)", (id, name, rating))
                                 added += 1
                               except ValueError:
                                pass
                    if added > 0:
                        conn.commit()
                        flash(f'{added} players imported from CSV!')
                    else:
                        flash('No valid players found in CSV!')
                except Exception as e:
                    flash(f'Error importing CSV: {e}')
            else:
                flash('Invalid file or no file selected!')
    cursor.execute("SELECT * FROM players WHERE tournament_id = %s", (id,))
    players = cursor.fetchall()

    standings = calculate_points(id)

    matches = {}
    for r in range(1, tour['current_round'] + 1):
        cursor.execute("""
            SELECT m.*, p1.name AS p1_name, p2.name AS p2_name FROM
            matches m LEFT JOIN players p1 ON m.player1_id = p1.id LEFT
            JOIN players p2 ON m.player2_id = p2.id WHERE
            m.tournament_id = %s AND m.round_num = %s

        """, (id, r))
        matches[r] = cursor.fetchall()

    cursor.close()
    conn.close()
    return render_template('tournament.html', tour=tour, players=players,
standings=standings, matches=matches)

@app.route('/delete_tournament/<int:id>', methods=['POST'])
def delete_tournament(id):
    conn = get_db_connection()
    if conn:
        cursor = conn.cursor()
        cursor.execute("DELETE FROM tournaments WHERE id = %s", (id,))
        conn.commit()
        cursor.close()
        conn.close()
        flash('Tournament deleted successfully!')
    else:
        flash('Database connection error')
        return redirect(url_for('index'))

@app.route('/generate_pairings/<int:id>')
def generate_pairings_route(id):
        conn = get_db_connection()
        if conn:
                cursor = conn.cursor(dictionary=True)
                cursor.execute("SELECT current_round, num_rounds FROM tournaments WHERE id = %s",
(id,))
                tour = cursor.fetchone()
                next_round = tour['current_round'] + 1
                if next_round > tour['num_rounds']:
                        flash('All rounds completed!')
                else:
                        if tour['current_round'] > 0 and not all_results_entered(id,
tour['current_round']):
                                flash('Cannot generate next round pairings until all results for the
current round are entered!')
                        elif generate_pairings(id, next_round):
                                cursor.execute("UPDATE tournaments SET current_round = %s WHERE id = %s",
(next_round, id))
                                conn.commit()
                                flash(f'Pairings generated for round {next_round}!')
                        else:
                                flash('Pairings already generated, not enough players, or error.')
                cursor.close()
                conn.close()
        else:
                flash('Database connection error')
        return redirect(url_for('tournament', id=id))

@app.route('/input_results/<int:id>/<int:round_num>', methods=['GET', 'POST'])
def input_results(id, round_num):
        conn = get_db_connection()
        if not conn:
                flash('Database connection error')
                return redirect(url_for('tournament', id=id))

        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
                SELECT m.*, p1.name AS p1_name, p2.name AS p2_name FROM
                matches m LEFT JOIN players p1 ON m.player1_id = p1.id LEFT
                JOIN    players    p2    ON    m.player2_id    =    p2.id    WHERE
                m.tournament_id = %s AND m.round_num = %s

        """, (id, round_num))
        matches = cursor.fetchall()

        if request.method == 'POST':
                updated = False
                for match in matches:
                        if match['player2_id'] is None:
                                continue     # Bye, already set
                        result_key = f"result_{match['id']}"
                        if result_key in request.form:
                                result_str = request.form[result_key]
                                if result_str == '':
                                        continue
```

```
                    try:
                        result = float(result_str)
                        if result not in [0.0, 0.5, 1.0]:
                            raise ValueError
                        cursor.execute("UPDATE matches SET result = %s WHERE id = %s",
(result, match['id']))
                          updated = True
                        except ValueError:
                        flash(f'Invalid result for match {match["id"]}: must be 0, 0.5, or
1!')
        if updated:
            conn.commit()
            flash('Results updated!')
        cursor.close()
        conn.close()
        return redirect(url_for('tournament', id=id))

    cursor.close()
    conn.close()
    return render_template('input_results.html', matches=matches, tour_id=id,
round_num=round_num)

@app.route('/export_standings/<int:id>')
def export_standings(id):
    standings = calculate_points(id)
    output = io.StringIO()
    writer = csv.writer(output)
    writer.writerow(['Name', 'Rating', 'Points'])
    for player in standings:
        writer.writerow([player['name'], player['rating'], player['points']])
    response = make_response(output.getvalue())
    response.headers["Content-Disposition"] = f"attachment; filename=standings_{id}.csv"
    response.headers["Content-type"] = "text/csv"
    return response

if __name__ == '__main__':
    app.run(debug=True)
```

INDEX.HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Tournaments</title>
<style>
body { font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 20px; }
h1 { color: #333; }
a { color: #007bff; text-decoration: none; }
a:hover { text-decoration: underline; }
ul { list-style-type: none; padding: 0; }
li { margin: 10px 0; background: #fff; padding: 10px; border-radius: 5px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); }
</style>
</head>
<body>
<h1>Tournaments</h1>
<a href="{{ url_for('create_tournament') }}">Create New Tournament</a>
<ul>
{% for t in tournaments %}
<li><a href="{{ url_for('tournament', id=t.id) }}">{{ t.name }}</a></li>
{% endfor %}
</ul>
{% with messages = get_flashed_messages() %}
{% if messages %}
<ul>
{% for message in messages %}
<li style="color: red;">{{ message }}</li>
{% endfor %}
</ul>
{% endif %}
{% endwith %}
</body>
</html>
```

CREATE_TOURNAMENT.HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Create Tournament</title>
<style>
body { font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 20px; }
h1 { color: #333; }
form { background: #fff; padding: 20px; border-radius: 5px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); width: 300px; }
input[type="text"], input[type="number"] { width: 100%; padding: 8px; margin: 10px 0; box-sizing: border-box; }
input[type="submit"] { background: #007bff; color: white; border: none; padding: 10px; cursor: pointer; }
input[type="submit"]:hover { background: #0056b3; }
</style>
</head>
<body>
<h1>Create Tournament</h1>
<form method="POST">
Name: <input type="text" name="name"><br>
Number of Rounds: <input type="number" name="num_rounds" min="1"><br>
<input type="submit" value="Create">
</form>
{% with messages = get_flashed_messages() %}
{% if messages %}
<ul>
{% for message in messages %}
<li style="color: red;">{{ message }}</li>
{% endfor %}
</ul>
{% endif %}
{% endwith %}
</body>
</html>
```

TOURNAMENT.HTML

```
<!DOCTYPE html>
<html>
<head>
<title>{{ tour.name }}</title>
<style>
body { font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 20px; }
h1, h2, h3 {color: #333;}
a { color: #007bff; text-decoration: none; }
a:hover { text-decoration: underline; }
form { background: #fff; border-radius: 5px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); margin-bottom: 20px; }
input[type="text"], input[type="number"], input[type="file"] { width: auto; padding: 8px; margin: 10px 0;}
input[type="submit"] { background: #007bff; color: white; border: none; padding: 10px; cursor: pointer; }
input[type="submit"]:hover { background: #0056b3;}
table {width: 100%; border-collapse: collapse; margin-bottom: 20px;}
th, td { border: 1px solid #ddd; padding: 12px; text-align: center; }
ul { list-style-type: none; padding: 0; }
li { margin: 5px 0; }
.tab { overflow: hidden; border: 1px solid #ccc; background-color: #f1f1f1;}
.tab button { background-color: inherit; float: left; border: none; outline: none; cursor: pointer; padding: 14px 16px; transition: 0.3s; font-size:
17px; }
.tab button.active { background-color: #ccc; }
.tabcontent { display: none; padding: 6px 12px; border: 1px solid #ccc; border-top: none; background: #fff; }
button.generate-btn { background: #28a745; color: white; border: none; padding: 10px 20px; cursor: pointer; margin-top: 20px; }
button.generate-btn:hover { background: #218838; }
button.input-btn { background: #ffc107; color: black; border: none; padding: 10px 20px; cursor: pointer; }
button.input-btn:hover { background: #e0a800; }
button.delete-btn { background: #dc3545; color: white; border: none; padding: 10px 20px; cursor: pointer; margin-top: 20px; }
button.delete-btn:hover {background: #c82333; }
button.back-btn { background: #6c757d; color: white; border: none; padding: 10px 20px; cursor: pointer; margin-top: 20px; }
button.back-btn:hover { background: #5a6268;}
</style>
</head>
<body>
<h1>{{ tour.name }} (Rounds: {{ tour.num_rounds }}, Current: {{ tour.current_round }})</h1>
{% with messages = get_flashed_messages() %}
{% if messages %}
<ul>
{% for message in messages %}
<li style="color: red;">{{ message }}</li>
{% endfor %}
</ul>
{% endif %}
{% endwith %}
```

```
<div class="tab">
<button class="tablinks" onclick="openTab(event, 'Players')">Players</button>
<button class="tablinks" onclick="openTab(event, 'Standings')">Standings</button>
{% for r in range(1, tour.current_round + 1) %}
<button class="tablinks" onclick="openTab(event, 'Round{{ r }}')">Round {{ r }}</button>
{% endfor %}
</div>
<div id="Players" class="tabcontent">
<h2>Players</h2>
<ul>
{% for p in players %}
<li>{{ p.name }} (Rating: {{ p.rating }})</li>
{% endfor %}
</ul>
<h3>Add Player</h3>
<form method="POST" style="padding: 20px;">
Name: <input type="text" name="name">
Rating: <input type="number" name="rating" min="0">
<input type="submit" name="add_player" value="Add">
</form>
<h3>Bulk Add via CSV</h3>
<form method="POST" enctype="multipart/form-data" style="padding: 20px;">
<input type="file" name="csv_file" accept=".csv">
<input type="submit" value="Import">
</form>
</div>
<div id="Standings" class="tabcontent">
<h2>Standings</h2>
<table>
<tr><th>Name</th><th>Rating</th><th>Points</th></tr>
{% for s in standings %}
<tr><td>{{ s.name }}</td><td>{{ s.rating }}</td><td>{{ s.points }}</td></tr>
{% endfor %}
</table>
<a href="{{ url_for('export_standings', id=tour.id) }}">Export Standings CSV</a>
</div>
```

```
{% for r, ms in matches.items() %}
<div id="Round{{ r }}" class="tabcontent ">
<h3>Round {{ r }} Pairings and Results</h3>
<table>
<tr><th>Player 1</th><th>Player 2</th><th>Result</th></tr>
{% for m in ms %}
<tr>
<td>{{ m.p1_name }}</td>
<td>{% if m.p2_name %}{{ m.p2_name }}{% else %}BYE{% endif %}</td>
<td>
{% if m.result is not none %}
{% if m.player2_id is none %}
Bye (1 point)
{% elif m.result == 1.0 %}
{{ m.p1_name }} wins
{% elif m.result == 0.5 %}
Draw
{% elif m.result == 0.0 %}
{{ m.p2_name }} wins
{% endif %}
{% else %}
Pending
{% endif %}
</td>
</tr>
{% endfor %}
</table>
<button class="input-btn" onclick="window.location.href='{{ url_for('input_results', id=tour.id, round_num=r) }}'">Input Results for Round
{{ r }}</button>
</div>
{% endfor %}
<button class="back-btn" onclick="window.location.href='{{ url_for('index') }}'">Back to Tournaments</button>
<form method="POST" action="{{ url_for('delete_tournament', id=tour.id) }}" style="display: inline;">
<button type="submit" class="delete-btn" onclick="return confirm('Are you sure you want to delete this tournament?');">Delete
Tournament</button>
</form>
{% if tour.current_round < tour.num_rounds %}
<button class="generate-btn" onclick="window.location.href='{{ url_for('generate_pairings_route', id=tour.id) }}'">Generate Pairings for
Next Round</button>
{% endif %}
```

```
<script>
function openTab(evt, tabName) {
var i, tabcontent, tablinks;
tabcontent = document.getElementsByClassName("tabcontent");
for (i = 0; i < tabcontent.length; i++) {
tabcontent[i].style.display = "none";
}
tablinks = document.getElementsByClassName("tablinks");
for (i = 0; i < tablinks.length; i++) {
tablinks[i].className = tablinks[i].className.replace(" active", "");
}
document.getElementById(tabName).style.display = "block";
evt.currentTarget.className += " active";
}

document.getElementsByClassName("tablinks")[0].click();
</script>

</body>
</html>
```

INPUT_RESULTS.HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Input Results</title>
<style>
body { font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 20px; }
h1 { color: #333; }
form { background: #fff; padding: 20px; border-radius: 5px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); }
table { width: 100%; border-collapse: collapse; margin-bottom: 20px; }
th, td { border: 1px solid #ddd; padding: 12px; text-align: center; } /* Centered text */
th { background-color: #f1f1f1; }
select { width: 150px; padding: 8px; }
input[type="submit"] { background: #007bff; color: white; border: none; padding: 10px; cursor: pointer; }
input[type="submit"]:hover { background: #0056b3; }
</style>
</head>
<body>
<h1>Input Results for Round {{ round_num }}</h1>
{% with messages = get_flashed_messages() %}
{% if messages %}
<ul>
{% for message in messages %}
<li style="color: red;">{{ message }}</li>
{% endfor %}
</ul>
{% endif %}
{% endwith %}
<form method="POST">
<table>
<tr><th>Player 1</th><th>Player 2</th><th>Result</th></tr>
{% for m in matches %}
{% if m.player2_id is not none %}
<tr>
<td>{{ m.p1_name }}</td>
<td>{{ m.p2_name }}</td>
<td>
<select name="result_{{ m.id }}">
<option value="" {% if m.result is none %}selected{% endif %}>Select</option>
<option value="1.0" {% if m.result == 1.0 %}selected{% endif %}>{{ m.p1_name }} wins</option>
<option value="0.5" {% if m.result == 0.5 %}selected{% endif %}>Draw</option>
<option value="0.0" {% if m.result == 0.0 %}selected{% endif %}>{{ m.p2_name }} wins</option>
</select>
</td>
</tr>
{% endif %}
{% endfor %}
</table>
<input type="submit" value="Save Results">
</form>
</body>
</html>
```

EXCECUTION

Set up database:

- Install MySQL
- Run the SQL schema file to set-up the required tables

Set up MySQL Connector:

- replace the placeholder in the app.py with the appropriate username and password for the mysql server you are running.

Run the application:

- run the app.py file using the appropriate python installation.

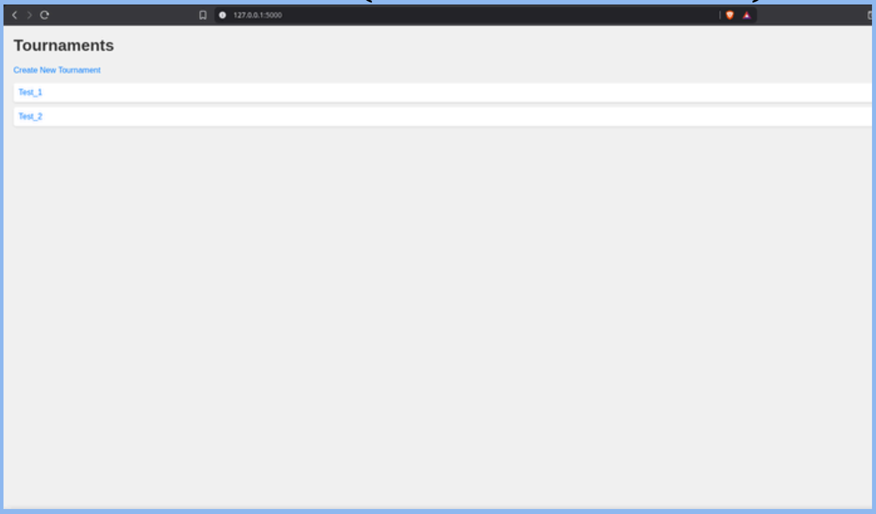
  The app will start running on localhost with the port 5000(http://127.0.0.1:5000).

Run the web interface:

- Open the URL in any web browser.

OUTPUT

Main Page(Index.Html)


127.0.0.1:5000

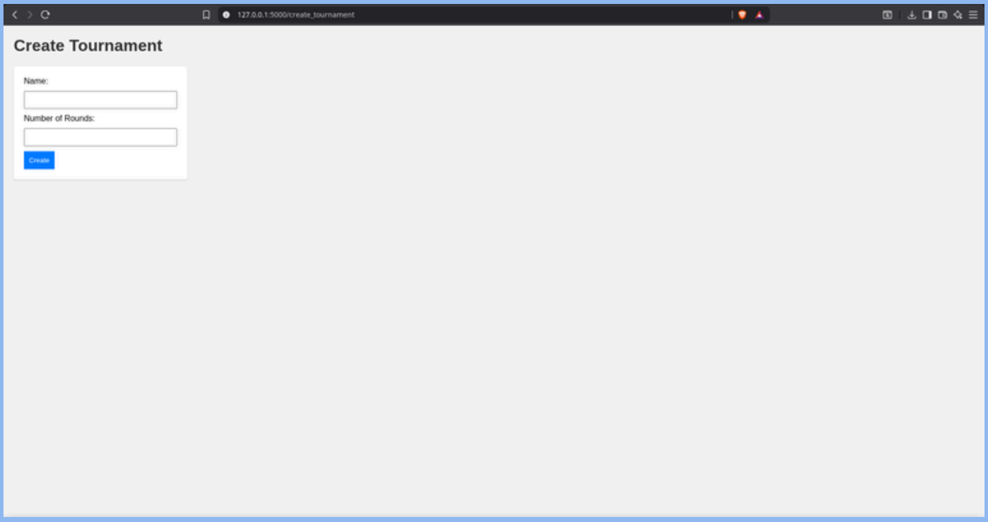
Tournaments

Create New Tournament

Test_1

Test_2

Create Tournament Page(create_tournament.html)


127.0.0.1:5000/create_tournament

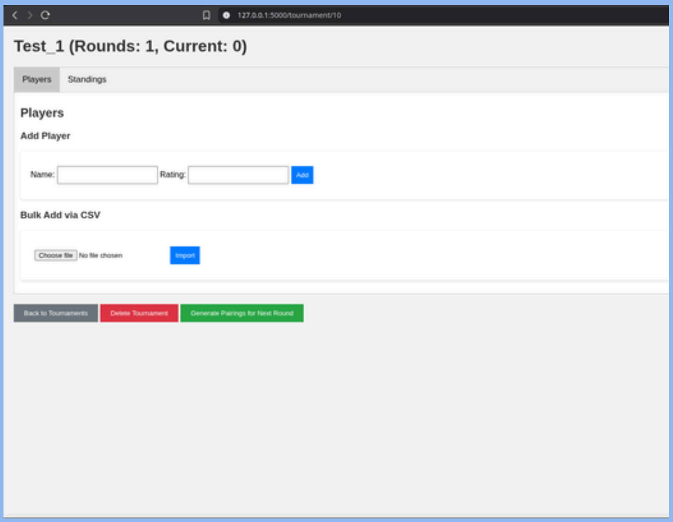
Create Tournament

Name:

Number of Rounds:

Create

Tournament Page(tournament.html)


127.0.0.1:5000/tournament/10

Test_1 (Rounds: 1, Current: 0)

Players | Standings

Players

Add Player

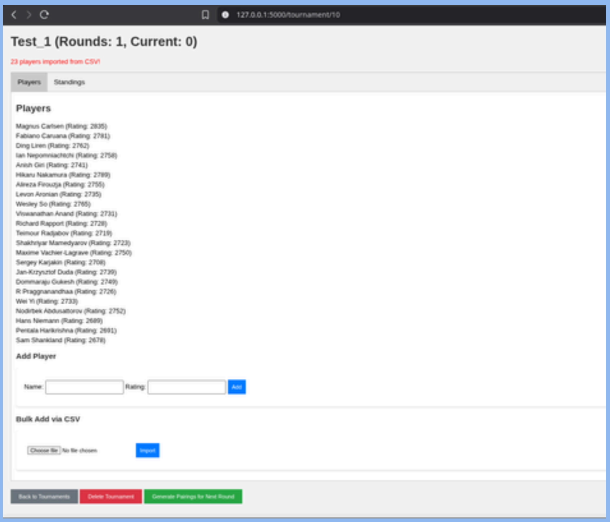
Name: Rating: Add

Bulk Add via CSV

Choose file No file chosen Import

Back to Tournaments | Delete Tournament | Generate Pairings for Next Round

Initial Page


127.0.0.1:5000/tournament/10

Test_1 (Rounds: 1, Current: 0)

23 players imported from CSV!

Players | Standings

Players

Magnus Carlsen (Rating: 2835)
Fabiano Caruana (Rating: 2781)
Ding Liren (Rating: 2762)
Ian Nepomniachtchi (Rating: 2758)
Anish Giri (Rating: 2741)
Hikaru Nakamura (Rating: 2789)
Alireza Firouzja (Rating: 2755)
Levon Aronian (Rating: 2735)
Wesley So (Rating: 2765)
Viswanathan Anand (Rating: 2731)
Richard Rapport (Rating: 2728)
Teimour Radjabov (Rating: 2719)
Shakhriyar Mamedyarov (Rating: 2723)
Maxime Vachier-Lagrave (Rating: 2750)
Sergey Karjakin (Rating: 2708)
Jan-Krzysztof Duda (Rating: 2739)
Dommaraju Gukesh (Rating: 2749)
R Praggnanandhaa (Rating: 2726)
Wei Yi (Rating: 2733)
Nodirbek Abdusattorov (Rating: 2752)
Hans Niemann (Rating: 2689)
Pentala Harikrishna (Rating: 2691)
Sam Shankland (Rating: 2678)

Add Player

Name: Rating: Add

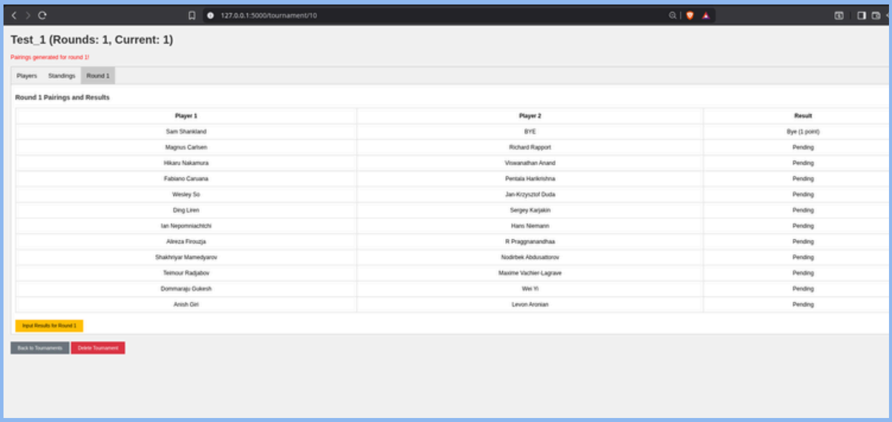
Bulk Add via CSV

Choose file No file chosen Import

Back to Tournaments | Delete Tournament | Generate Pairings for Next Round

Players Added

Tournament Page(tournament.html)


127.0.0.1:5000/tournament/10

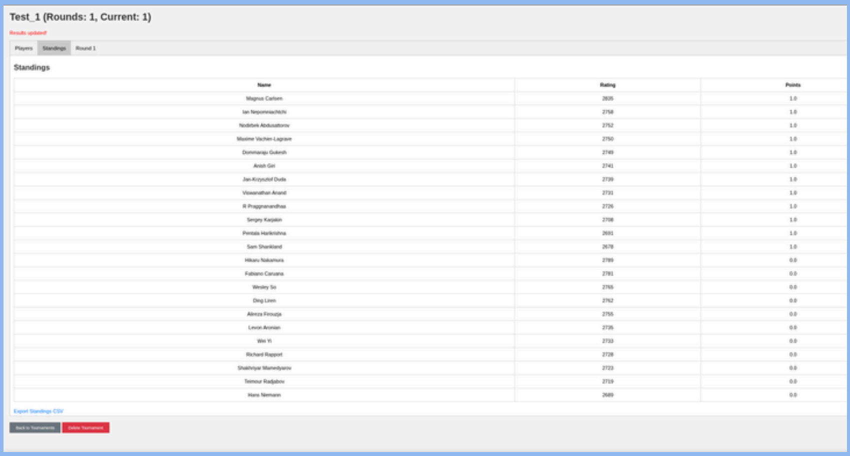
Test_1 (Rounds: 1, Current: 1)

Pairings generated for round 1!

Round 1 Pairings and Results

Player 1	Player 2	Result
Sam Shankland	BYE	Bye (1 point)
Magnus Carlsen	Richard Rapport	Pending
Hikaru Nakamura	Viswanathan Anand	Pending
Fabiano Caruana	Pentala Harikrishna	Pending
Wesley So	Jan-Krzysztof Duda	Pending
Ding Liren	Sergey Karjakin	Pending
Ian Nepomniachtchi	Hans Niemann	Pending
Alireza Firouzja	R Praggnanandhaa	Pending
Shakhriyar Mamedyarov	Nodirbek Abdusattorov	Pending
Teimour Radjabov	Maxime Vachier-Lagrave	Pending
Dommaraju Gukesh	Wei Yi	Pending
Anish Giri	Levon Aronian	Pending

Round Pairings



Test_1 (Rounds: 1, Current: 1)

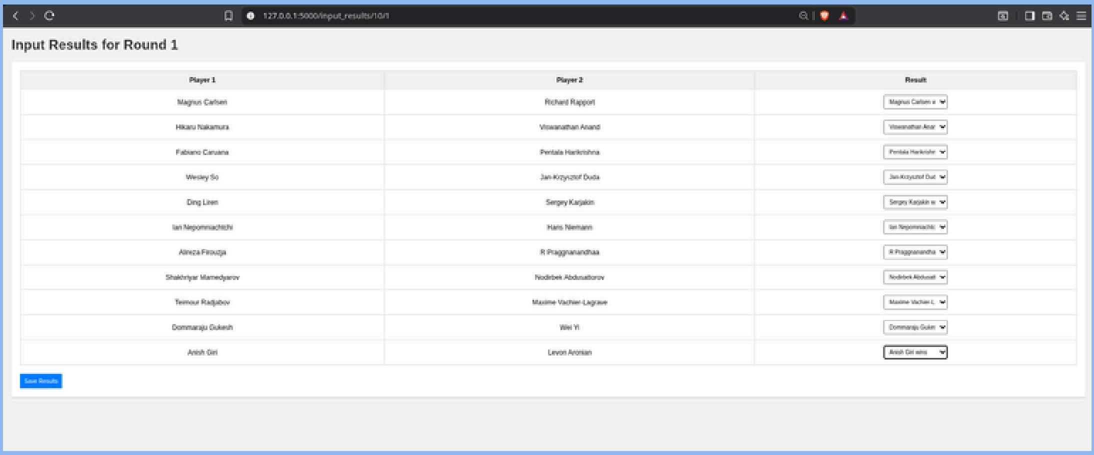
Results updated!

Standings

Name	Rating	Points
Magnus Carlsen	2835	1.0
Ian Nepomniachtchi	2758	1.0
Nodirbek Abdusattorov	2752	1.0
Maxime Vachier-Lagrave	2750	1.0
Dommaraju Gukesh	2749	1.0
Anish Giri	2741	1.0
Jan-Krzysztof Duda	2739	1.0
Viswanathan Anand	2731	1.0
R Praggnanandhaa	2726	1.0
Sergey Karjakin	2708	1.0
Pentala Harikrishna	2691	1.0
Sam Shankland	2678	1.0
Hikaru Nakamura	2789	0.0
Fabiano Caruana	2781	0.0
Wesley So	2765	0.0
Ding Liren	2762	0.0
Alireza Firouzja	2755	0.0
Levon Aronian	2735	0.0
Wei Yi	2733	0.0
Richard Rapport	2728	0.0
Shakhriyar Mamedyarov	2723	0.0
Teimour Radjabov	2719	0.0
Hans Niemann	2689	0.0

Tournament Standings

Results Input Page(input_results.html)


127.0.0.1:5000/input_results/10/1

Input Results for Round 1

Player 1	Player 2	Result
Magnus Carlsen	Richard Rapport	Magnus Carlsen w
Hikaru Nakamura	Viswanathan Anand	Viswanathan Anar
Fabiano Caruana	Pentala Harikrishna	Pentala Harikrishr
Wesley So	Jan-Krzysztof Duda	Jan-Krzysztof Dud
Ding Liren	Sergey Karjakin	Sergey Karjakin w
Ian Nepomniachtchi	Hans Niemann	Ian Nepomniachtc
Alireza Firouzja	R Praggnanandhaa	R Praggnanandha
Shakhriyar Mamedyarov	Nodirbek Abdusattorov	Nodirbek Abdusatt
Teimour Radjabov	Maxime Vachier-Lagrave	Maxime Vachier-L
Dommaraju Gukesh	Wei Yi	Dommaraju Guke
Anish Giri	Levon Aronian	Anish Giri wins

BIBLIOGRAPHY

1. Flask Documentation: https://flask.palletsprojects.com/en/stable/
2. Jet Brains: introduction to flask web applications https://www.jetbrains.com/help/pycharm/creating-web-application-with-flask.html
3. MySQL Connector Documentation: https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html
4. Planetscale.com: Integrating flask with MySQL: https://planetscale.com/learn/courses/mysql-for-python-developers/building-a-flask-app-with-mysql/
5. W3Schools.com: MySQL Relational database tutorial: https://www.w3schools.com/mysql/mysql_rdbms.asp
6. Chess Pairing Mechanisms : https://real.mtak.hu/80729/7/jXaio4T11ygd57-77-86.pdf
7. Guide to Swiss system pairings: https://www.chessmanager.com/uk-ua/blog/swiss-system