Principles of Artificial Intelligence

Course Code: MEAD-605

Lab File

Submitted to: **Dr. Kalpana Johari**

Submitted by

Shantanu Shukla

Enrollment No. - 01211805424

in partial fulfillment for the award of the degree

of

Master of Technology - AI/DS

Center for Development of Advanced Computing, Noida

Affiliated to Guru Gobind Singh Indraprastha University

Index

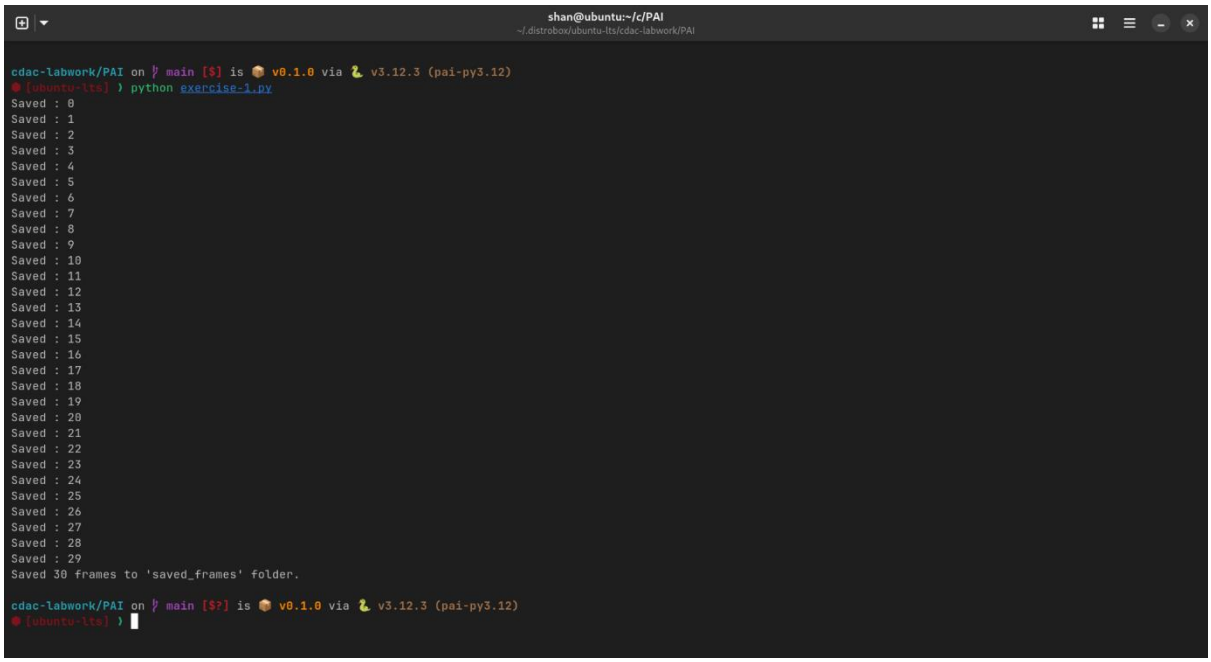
• ASSIGNMENT-1

Objective: Extraction of image frames from a video.

Code:

```
1. import cv2
2. import os
3.
4. # Set the path to the folder where you want to save the frames
5. output_folder = 'saved_frames_3'
6.
7. # Create the folder if it doesn't exist
8. if not os.path.exists(output_folder):
9.     os.makedirs(output_folder)
10.
11. # Open the video file
12. video_path = '3.mp4'  # Replace with your video path
13. cap = cv2.VideoCapture(video_path)
14.
15. frame_number = 0
16. saved_frame_count = 0
17.
18. # Loop through the video
19. while cap.isOpened():
20.     ret, frame = cap.read()
21.
22.     if not ret:
23.         break
24.
25.     # Check if the frame is the 10th one
26.     if frame_number % 10 == 0:
27.         frame_name = os.path.join(output_folder,
f'frame_{saved_frame_count}.jpg')  # Name the saved frame
28.         cv2.imwrite(frame_name, frame)  # Save the frame
29.         print(f"Saved : {saved_frame_count}")
30.         saved_frame_count += 1
31.
32.     frame_number += 1
33.
34. # Release the video capture object
35. cap.release()
36. cv2.destroyAllWindows()
37.
38. print(f"Saved {saved_frame_count} frames to '{output_folder}'
folder.")
39.
```

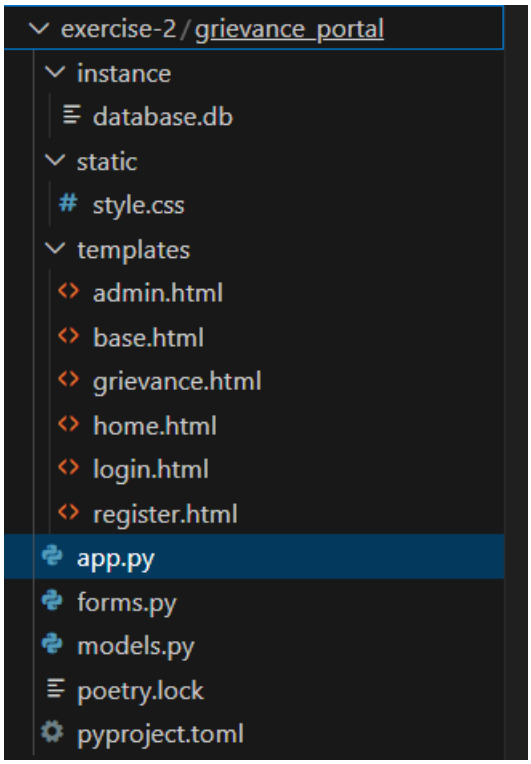
Output:


shan@ubuntu:~/c/PAI
~/.distrobox/ubuntu-lts/cdac-labwork/PAI
cdac-labwork/PAI on main [$] is v0.1.0 via v3.12.3 (pai-py3.12)
[ubuntu-lts]) python exercise-1.py
Saved : 0
Saved : 1
Saved : 2
Saved : 3
Saved : 4
Saved : 5
Saved : 6
Saved : 7
Saved : 8
Saved : 9
Saved : 10
Saved : 11
Saved : 12
Saved : 13
Saved : 14
Saved : 15
Saved : 16
Saved : 17
Saved : 18
Saved : 19
Saved : 20
Saved : 21
Saved : 22
Saved : 23
Saved : 24
Saved : 25
Saved : 26
Saved : 27
Saved : 28
Saved : 29
Saved 30 frames to 'saved_frames' folder.
cdac-labwork/PAI on main [$?] is v0.1.0 via v3.12.3 (pai-py3.12)
[ubuntu-lts])

• ASSIGNMENT-2

Objective: Build a Grievance Portal

Project Structure:



Sample Code:

Backend –

models.py – contains logic for our database models.

```
1. from flask_sqlalchemy import SQLAlchemy
2. from flask_login import UserMixin
3.
4. db = SQLAlchemy()
5.
6.
7. class User(db.Model, UserMixin):
8.     id = db.Column(db.Integer, primary_key=True)
9.     username = db.Column(db.String(150), nullable=False, unique=True)
10.    password = db.Column(db.String(150), nullable=False)
```

```
    # Set to True for admin users
    is_admin = db.Column(db.Boolean, default=False)


class Grievance(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    description = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'),
nullable=False)
    status = db.Column(db.String(50), default='Pending')

    user = db.relationship('User', backref='grievances')

    def __repr__(self):
        return f'<Grievance {self.title}>'

```

Frontend –

base.html – our base html template that other html files inherit from.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Grievance Portal</title>
    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.
css" rel="stylesheet">
    <!-- Custom CSS -->
    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
</head>

<body>
    <nav>
        <a href="{{ url_for('home') }}">Home</a>
        {% if current_user.is_authenticated %}
            <a href="{{ url_for('dashboard') }}">Dashboard</a>
            <a href="{{ url_for('logout') }}">Logout</a>
        {% else %}
            <a href="{{ url_for('login') }}">Login</a>
            <a href="{{ url_for('register') }}">Register</a>
        {% endif %}
    </nav>
    <div class="container">
        {% with messages =
get_flashed_messages(with_categories=true) %}
```

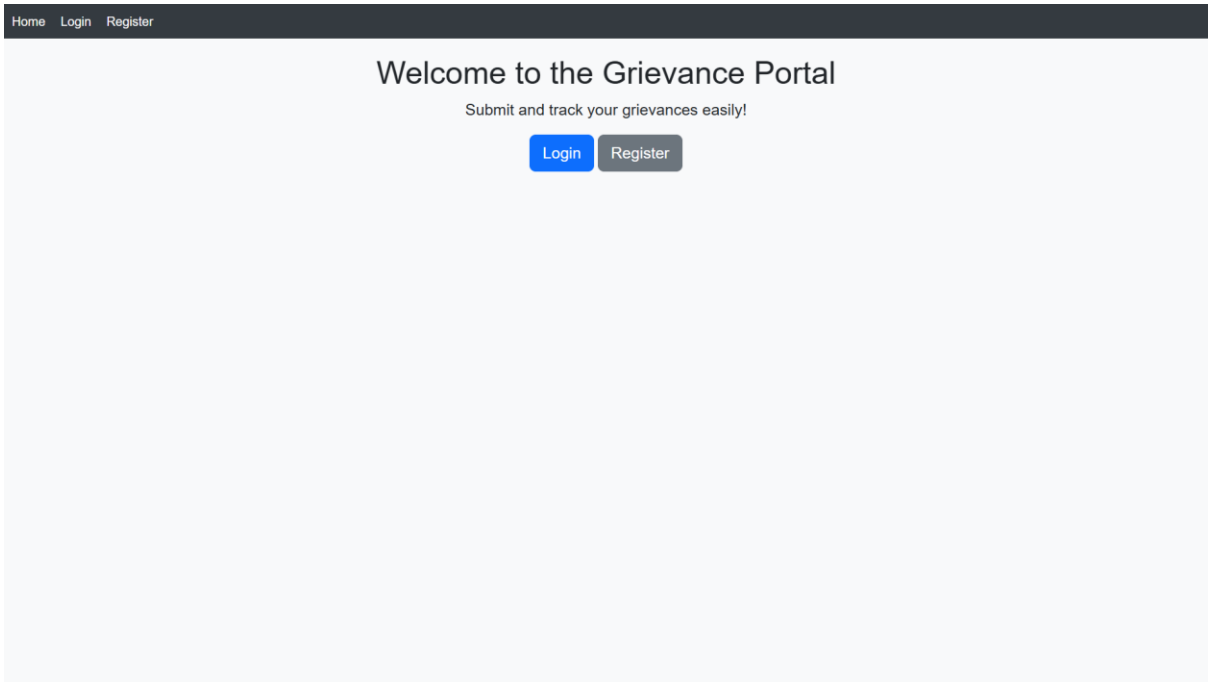
```
26.         {% if messages %}
27.             {% for category, message in messages %}
28.             <div class="alert {{ category }}">{{ message }}</div>
29.             {% endfor %}
30.         {% endif %}
31.         {% endwith %}
32.         {% block content %}{% endblock %}
33.     </div>
34.     <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle
.min.js"></script>
35. </body>
36. </html>
37.
```

style.css – our base css stylesheet.

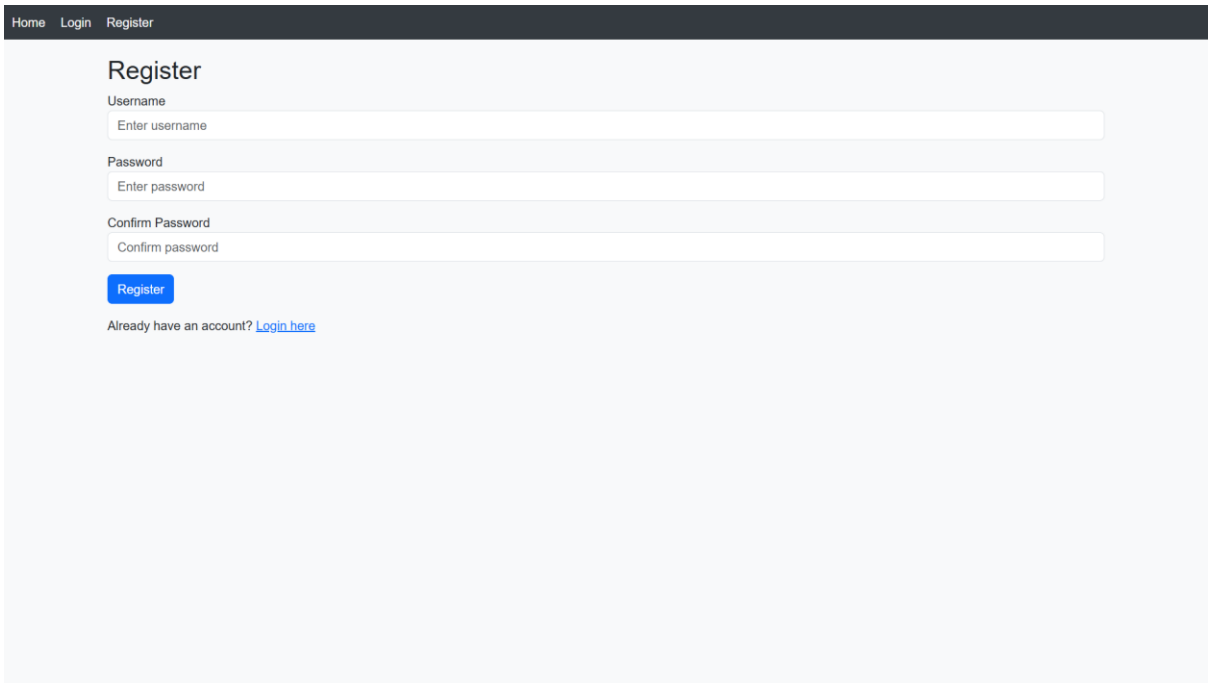
```
 1. body {
 2.     font-family: Arial, sans-serif;
 3.     background-color: #f8f9fa;
 4.     margin: 0;
 5.     padding: 0;
 6. }
 7.
 8. nav {
 9.     background-color: #343a40;
10.     padding: 10px;
11. }
12.
13. nav a {
14.     color: white;
15.     text-decoration: none;
16.     margin-right: 15px;
17. }
18.
19. nav a:hover {
20.     text-decoration: underline;
21. }
22.
23. .container {
24.     margin-top: 20px;
25. }
26.
27. .alert {
28.     margin: 10px 0;
29. }
30.
31. .dashboard-table {
32.     margin-top: 20px;
33. }
34.
```

Output:

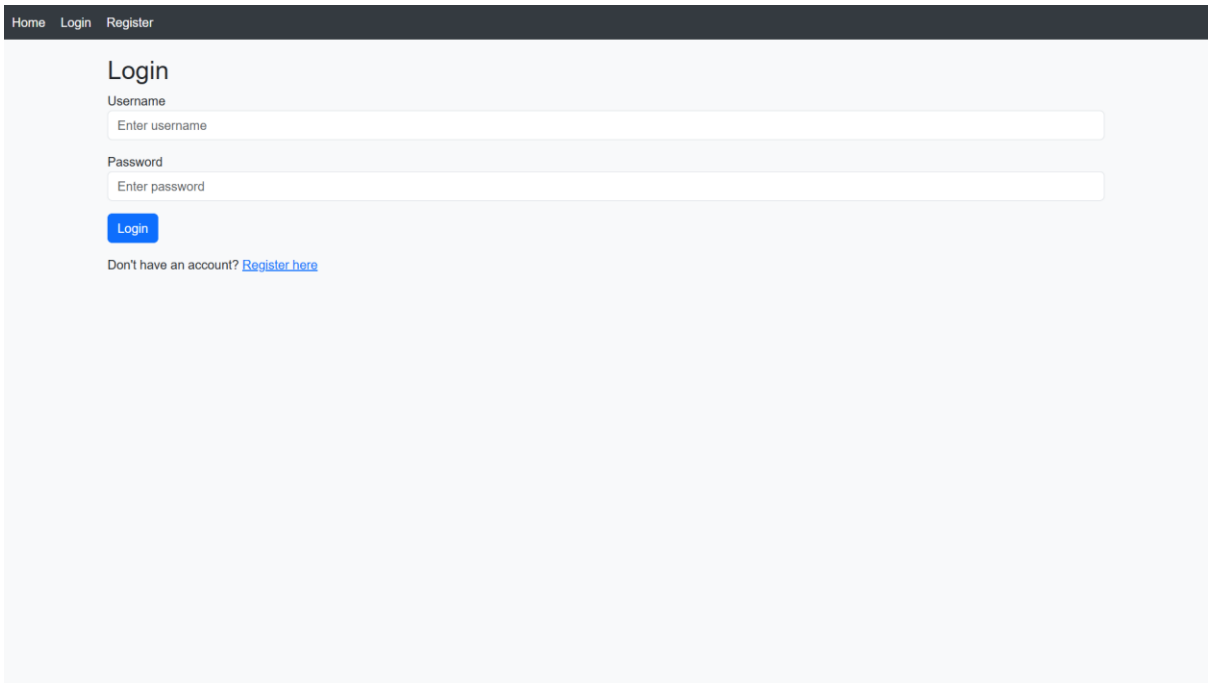
Homepage –



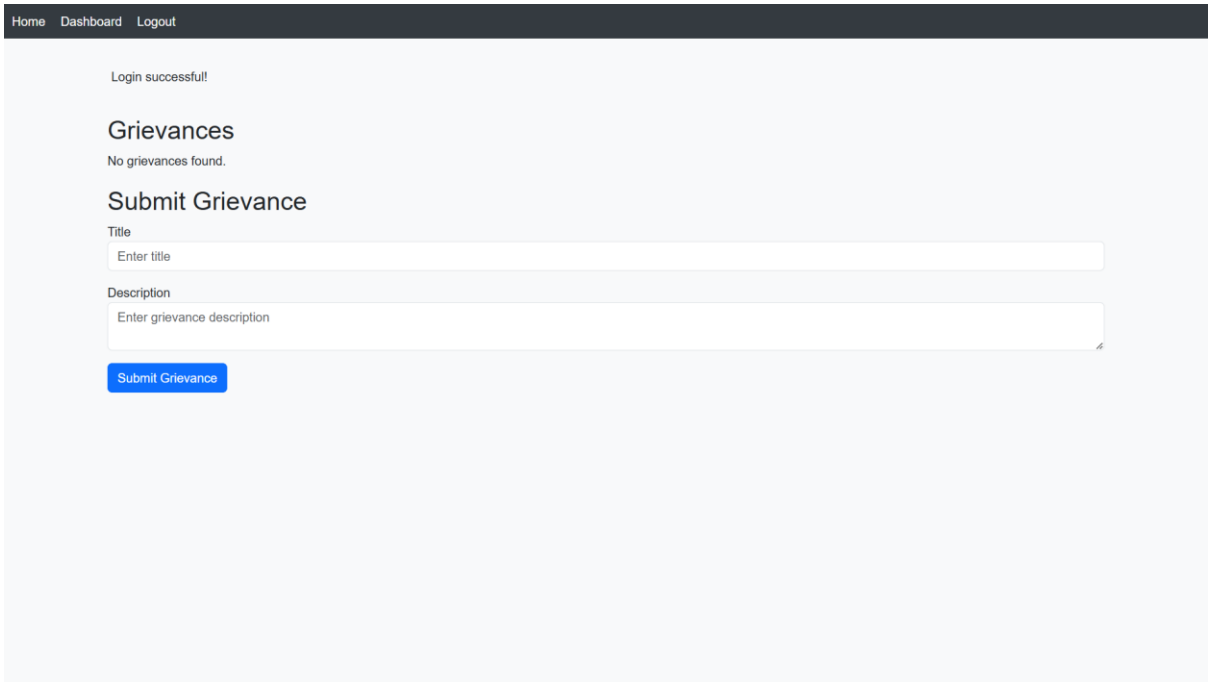
User Registration –

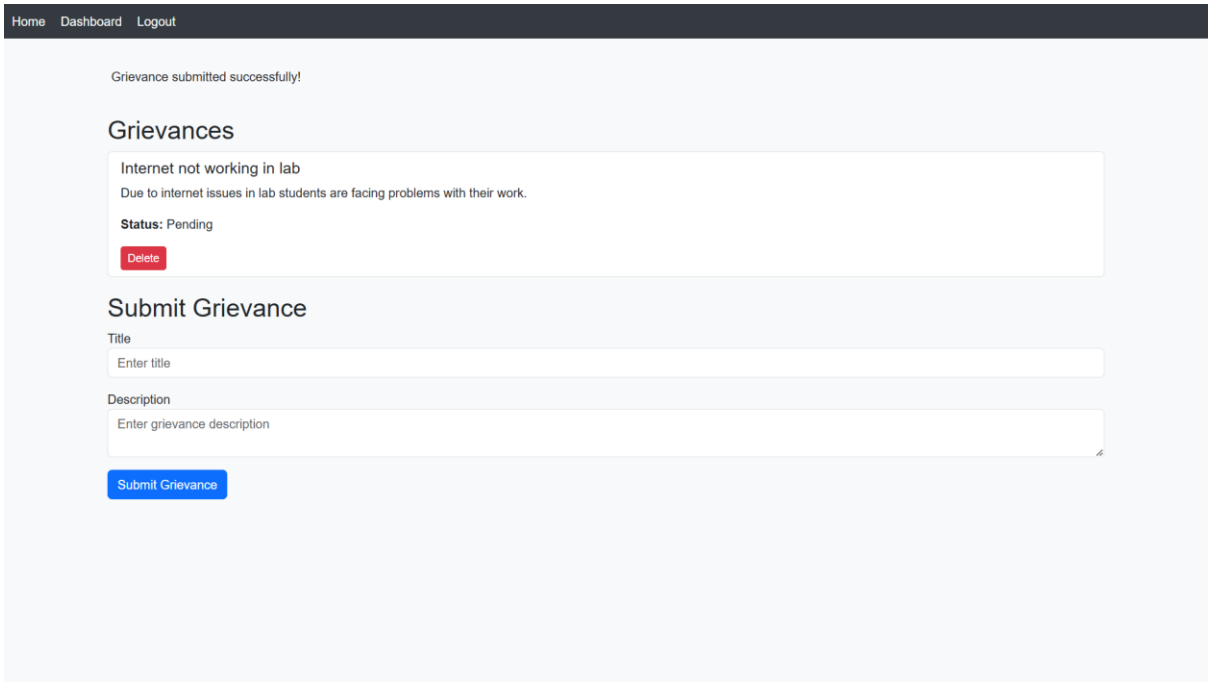


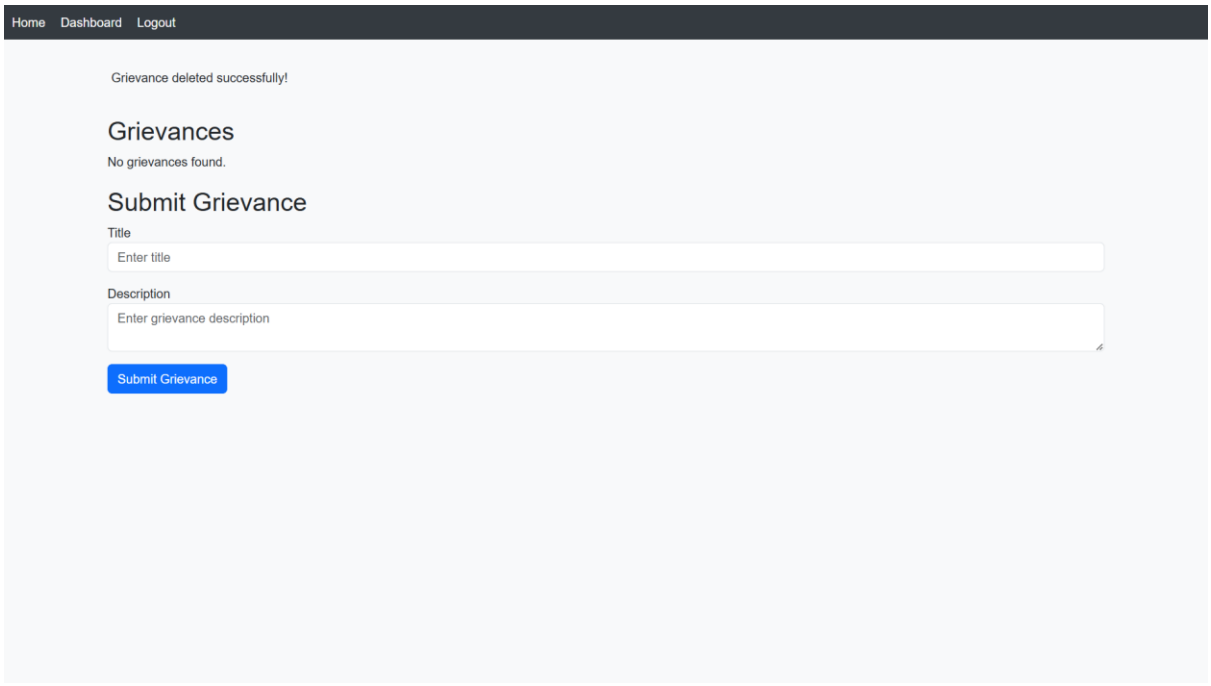
Login –



User Dashboard –




Home Dashboard Logout
Grievance submitted successfully!
Grievances
Internet not working in lab
Due to internet issues in lab students are facing problems with their work.
Status: Pending
Delete
Submit Grievance
Title
Enter title
Description
Enter grievance description
Submit Grievance


Home Dashboard Logout
Grievance deleted successfully!
Grievances
No grievances found.
Submit Grievance
Title
Enter title
Description
Enter grievance description
Submit Grievance

• ASSIGNMENT-3

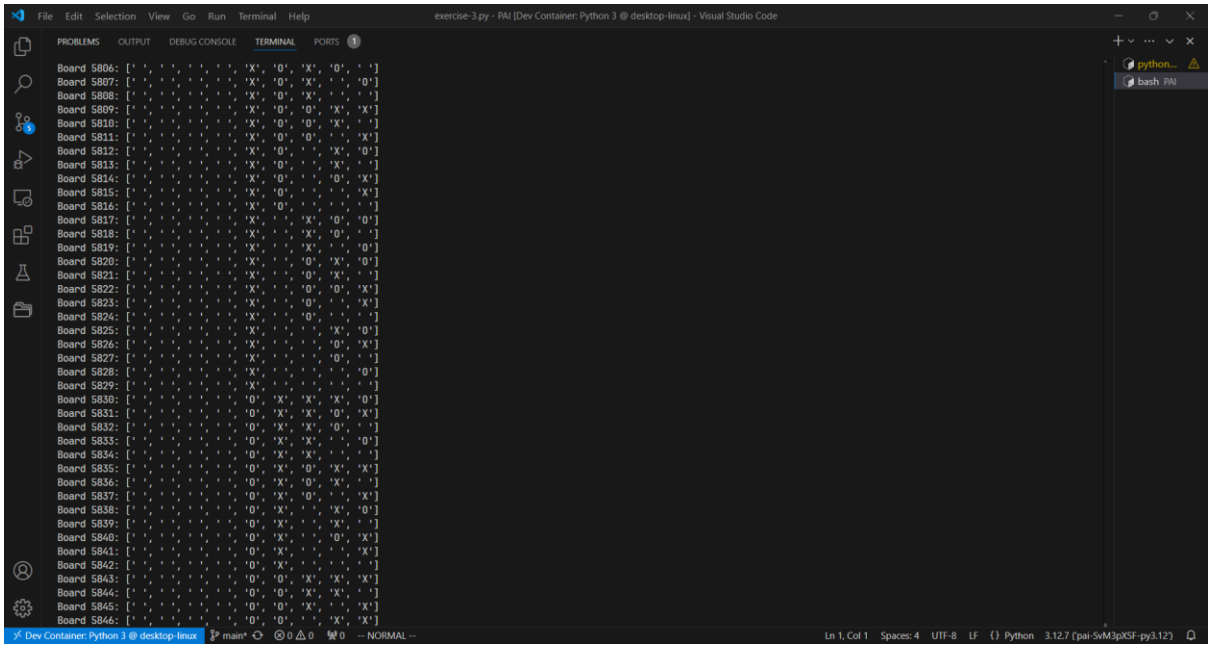
Objective: Generate all the possible states of Tic Tac Toe.

Code:

```
 1. from itertools import product
 2.
 3. # Function to check if a board configuration is valid based on Tic-
Tac-Toe rules
 4. def is_valid_board(board):
 5.     # Count the number of X's and O's
 6.     x_count = board.count('X')
 7.     o_count = board.count('O')
 8.
 9.     # Ensure the number of X's is equal to or one more than the number
of O's
10.     if not (x_count == o_count or x_count == o_count + 1):
11.         return False
12.
13.     # Check for winning conditions
14.     if check_winner(board, 'X') and check_winner(board, 'O'):
15.         return False  # Both players can't win simultaneously
16.     return True
17.
18. # Function to check if a player has won
19. def check_winner(board, player):
20.     # Win conditions for rows, columns, and diagonals
21.     win_conditions = [
22.         [0, 1, 2], [3, 4, 5], [6, 7, 8],  # Rows
23.         [0, 3, 6], [1, 4, 7], [2, 5, 8],  # Columns
24.         [0, 4, 8], [2, 4, 6]              # Diagonals
25.     ]
26.     for condition in win_conditions:
27.         if all(board[i] == player for i in condition):
28.             return True
29.     return False
30.
31. # Function to generate all valid Tic-Tac-Toe board configurations
32. def generate_all_boards():
33.     all_boards = []
34.     # Generate all combinations of 'X', 'O', and ' ' for the 9
positions
35.     for comb in product('XO ', repeat=9):
36.         board = list(comb)
37.         if is_valid_board(board):
38.             all_boards.append(board)
39.     return all_boards
40.
41. # Get all valid boards
```

```
42. valid_boards = generate_all_boards()
43. 
44. # Display all valid boards
45. for idx, board in enumerate(valid_boards):
46.     print(f"Board {idx + 1}: {board}")
47. 
```

Output:



• ASSIGNMENT-4

Objective: Implement A* search Algorithm.

Code:

```
import heapq

class Node:
    def __init__(self, state, parent=None, g=0, h=0):
        self.state = state  # The state represents the node (e.g., a position in a maze)
        self.parent = parent  # The parent node
        self.g = g  # Cost from start to the current node
        self.h = h  # Heuristic from current node to goal
        self.f = g + h  # Total cost (g + h)

    def __lt__(self, other):
        # Comparison operator for the priority queue (heapq)
        return self.f < other.f

def a_star_search(start, goal, get_neighbors, heuristic):
    """
    A* Search Algorithm to find the shortest path from start to goal.
    :param start: The start node (state)
    :param goal: The goal node (state)
    :param get_neighbors: Function to get neighbors of a node
    :param heuristic: Heuristic function to estimate the cost to the goal
    :return: List of states representing the path from start to goal, or None if no path found
    """
    # Initialize open and closed lists
    open_list = []
    closed_list = set()

    # Push the start node into the open list
    start_node = Node(start, None, 0, heuristic(start, goal))
    heapq.heappush(open_list, start_node)

    while open_list:
        # Get the node with the lowest f value
        current_node = heapq.heappop(open_list)

        # If goal is reached, reconstruct the path
        if current_node.state == goal:
            path = []
            while current_node:
                path.append(current_node.state)
                current_node = current_node.parent
```

```
42.             return path[::-1]  # Return reversed path (from start to
goal)
43.
44.         closed_list.add(current_node.state)
45.
46.         # Get the neighbors of the current node
47.         for neighbor, cost in get_neighbors(current_node.state):
48.             if neighbor in closed_list:
49.                 continue
50.
51.             g = current_node.g + cost
52.             h = heuristic(neighbor, goal)
53.             neighbor_node = Node(neighbor, current_node, g, h)
54.
55.             # Add the neighbor to the open list if it is not already
there
56.             if all(neighbor_node.f < node.f for node in open_list):
57.                 heapq.heappush(open_list, neighbor_node)
58.
59.     return None  # Return None if no path found
60.
61. # Example heuristic function (Manhattan distance for grid-based
pathfinding)
62. def heuristic(state, goal):
63.     x1, y1 = state
64.     x2, y2 = goal
65.     return abs(x1 - x2) + abs(y1 - y2)
66.
67. # Example function to get neighbors (4-directional movement for a
grid)
68. def get_neighbors(state):
69.     neighbors = []
70.     x, y = state
71.     # Move up, down, left, right (grid-based example)
72.     for dx, dy, cost in [(-1, 0, 1), (1, 0, 1), (0, -1, 1), (0, 1,
1)]:
73.         neighbor = (x + dx, y + dy)
74.         neighbors.append((neighbor, cost))
75.     return neighbors
76.
77. # Example usage
78. start = (0, 0)  # Starting position (x, y)
79. goal = (4, 4)  # Goal position (x, y)
80.
81. path = a_star_search(start, goal, get_neighbors, heuristic)
82.
83. if path:
84.     print("Path found:", path)
85. else:
86.     print("No path found")
```

Output:

```
pai-py3.12vscode → /workspaces/cdac-labwork/PAI (main) $ python exercise-4.py
Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]
```

• ASSIGNMENT-5

Objective: Implement AO* search algorithm.

Code:

```
class AONode:
    def __init__(self, name, heuristic_cost):
        self.name = name
        self.heuristic_cost = heuristic_cost  # Heuristic cost of the node
        self.successors = []  # List of AND/OR successor groups
        self.solved = False  # Whether the node is solved
        self.best_successor = None  # Best successor group (optimal path)

    def add_successors(self, successors):
        """
        Add a successor group to the node.
        Each successor group is a list of nodes (AND branch).
        """
        for group in successors:
            if not all(isinstance(child, AONode) for child in group):
                raise ValueError("All children in successors must be instances of AONode.")
        self.successors.extend(successors)

def ao_star(node, trace_path=[]):
    """
    The AO* algorithm to find the optimal solution path.
    """
    if node.solved:
        return node.solved

    print(f"Visiting Node: {node.name}")
    trace_path.append(node.name)

    # If it's a leaf node, mark it as solved
    if not node.successors:
        node.solved = True
        trace_path.pop()
        return True

    # Evaluate all successor groups to find the best one
    min_cost = float('inf')
    best_successor = None

    for successors in node.successors:
        # Compute the total cost for this group
        total_cost = sum(child.heuristic_cost for child in successors)
        if total_cost < min_cost:
            min_cost = total_cost
            best_successor = successors

    # Set the best successor group
    node.best_successor = best_successor
```

```
48.
49.     # Recursively solve the best successor group
50.     all_solved = True
51.     for child in best_successor:
52.         if not ao_star(child, trace_path):
53.             all_solved = False
54.
55.     # If all successors in the best group are solved, mark the node as solved
56.     node.solved = all_solved
57.     if all_solved:
58.         node.heuristic_cost = min_cost
59.
60.     trace_path.pop()
61.     return node.solved
62.
63. def print_solution(node):
64.     """
65.     Print the solution path found by the AO* algorithm.
66.     """
67.     if not node or not node.best_successor:
68.         return
69.     print(f"Node {node.name} -> ", [child.name for child in
node.best_successor])
70.     for child in node.best_successor:
71.         print_solution(child)
72.
73. # Example usage
74. if __name__ == "__main__":
75.     # Creating an example And-Or graph
76.     A = AONode("A", 10)
77.     B = AONode("B", 8)
78.     C = AONode("C", 7)
79.     D = AONode("D", 6)
80.     E = AONode("E", 5)
81.
82.     # Define successors (AND/OR groups)
83.     A.add_successors([[B, C]])  # AND group: A -> {B AND C}
84.     A.add_successors([[D]])    # OR group: A -> D
85.     B.add_successors([[E]])    # OR group: B -> E
86.
87.     # Run AO* algorithm
88.     trace_path = []
89.     ao_star(A, trace_path)
90.
91.     print("\nSolution Path:")
92.     print_solution(A)
```

Output:

```
pai-py3.12vscode → /workspaces/cdac-labwork/PAI (main) $ python exercise-5.py
Visiting Node: A
Visiting Node: D

Solution Path:
Node A ->  ['D']
```

• ASSIGNMENT-6

Objective: Implement BFS algorithm.

Code:

```
from collections import deque

def bfs_maze(maze, start, goal):
    """
    Perform BFS to find the shortest path in a maze from start to goal.

    :param maze: 2D list representing the maze (0 = open, 1 = wall)
    :param start: Tuple (row, col) representing the start position
    :param goal: Tuple (row, col) representing the goal position
    :return: List representing the path from start to goal or None if no path exists
    """
    # Directions (up, down, left, right)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    # Queue for BFS (FIFO)
    queue = deque([(start, [start])])  # (current position, path taken to reach that position)
    visited = set()  # Set to track visited positions

    while queue:
        current_pos, path = queue.popleft()  # Dequeue the first node in the queue

        # Goal test
        if current_pos == goal:
            return path  # Return the path if the goal is found

        visited.add(current_pos)  # Mark the position as visited

        # Generate neighbors (adjacent cells)
        for direction in directions:
            new_row, new_col = current_pos[0] + direction[0], current_pos[1] + direction[1]

            # Check if the new position is within bounds and not a wall
            if (0 <= new_row < len(maze)) and (0 <= new_col < len(maze[0])) and maze[new_row][new_col] == 0:
                new_pos = (new_row, new_col)
                if new_pos not in visited:
                    visited.add(new_pos)
                    queue.append((new_pos, path + [new_pos]))  # Enqueue the new position with updated path
```

```
38.
39.     return None  # Return None if no path is found
40.
41.
42. # Example usage
43. maze = [
44.     [0, 0, 0, 0, 0],
45.     [0, 1, 1, 1, 0],
46.     [0, 1, 0, 0, 0],
47.     [0, 1, 1, 1, 0],
48.     [0, 0, 0, 0, 0]
49. ]
50.
51. start = (0, 0)  # Starting position (row, col)
52. goal = (4, 4)  # Goal position (row, col)
53.
54. path = bfs_maze(maze, start, goal)
55.
56. if path:
57.     print("Path found:", path)
58. else:
59.     print("No path found")
60.
```

Output:

```
pai-py3.12vscode ➜ /workspaces/cdac-labwork/PAI (main) $ python exercise-6.py
Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]
pai-py3.12vscode ➜ /workspaces/cdac-labwork/PAI (main) $
```

• ASSIGNMENT-7

Objective: Implement DFS algorithm.

Code:

```
 1. def dfs_maze(maze, start, goal):
 2.     """
 3.     Perform DFS to find the shortest path in a maze from start to
goal.
 4.
 5.     :param maze: 2D list representing the maze (0 = open, 1 = wall)
 6.     :param start: Tuple (row, col) representing the start position
 7.     :param goal: Tuple (row, col) representing the goal position
 8.     :return: List representing the path from start to goal or None if
no path exists
 9.     """
10.     # Directions (up, down, left, right)
11.     directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
12.
13.     # Stack for DFS (LIFO)
14.     stack = [(start, [start])]  # (current position, path taken to
reach that position)
15.     visited = set()  # Set to track visited positions
16.
17.     while stack:
18.         current_pos, path = stack.pop()  # Pop the last node in the
stack
19.
20.         # Goal test
21.         if current_pos == goal:
22.             return path  # Return the path if the goal is found
23.
24.         visited.add(current_pos)  # Mark the position as visited
25.
26.         # Generate neighbors (adjacent cells)
27.         for direction in directions:
28.             new_row, new_col = current_pos[0] + direction[0],
current_pos[1] + direction[1]
29.
30.             # Check if the new position is within bounds and not a
wall
31.             if (0 <= new_row < len(maze)) and (0 <= new_col <
len(maze[0])) and maze[new_row][new_col] == 0:
32.                 new_pos = (new_row, new_col)
33.                 if new_pos not in visited:
34.                     visited.add(new_pos)
35.                     stack.append((new_pos, path + [new_pos]))  # Push
the new position to the stack with updated path
36.
```

```
37.     return None  # Return None if no path is found
38.
39.
40. # Example usage
41. maze = [
42.     [0, 0, 0, 0, 0],
43.     [0, 1, 1, 1, 0],
44.     [0, 1, 0, 0, 0],
45.     [0, 1, 1, 1, 0],
46.     [0, 0, 0, 0, 0]
47. ]
48.
49. start = (0, 0)  # Starting position (row, col)
50. goal = (4, 4)  # Goal position (row, col)
51.
52. path = dfs_maze(maze, start, goal)
53.
54. if path:
55.     print("Path found:", path)
56. else:
57.     print("No path found")
58.
```

Output:

```
pai-py3.12vscode ➜ /workspaces/cdac-labwork/PAI (main) $ python exercise-7.py
Path found: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4), (4, 4)]
pai-py3.12vscode ➜ /workspaces/cdac-labwork/PAI (main) $
```

• ASSIGNMENT-8

Objective: Implement N Queens problem.

Code:

```
1. def print_solution(board):
2.     """Print the chessboard."""
3.     for row in board:
4.         print(" ".join("Q" if col else "." for col in row))
5.     print()
6.
7. def is_safe(board, row, col, n):
8.     """
9.     Check if placing a queen at board[row][col] is safe.
10.    """
11.    # Check the current column
12.    for i in range(row):
13.        if board[i][col]:
14.            return False
15.
16.    # Check the upper-left diagonal
17.    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
18.        if board[i][j]:
19.            return False
20.
21.    # Check the upper-right diagonal
22.    for i, j in zip(range(row, -1, -1), range(col, n)):
23.        if board[i][j]:
24.            return False
25.
26.    return True
27.
28. def solve_n_queens(board, row, n):
29.    """
30.    Solve the N Queens problem using backtracking.
31.    """
32.    if row >= n:
33.        print_solution(board)
34.        return True
35.
36.    res = False
37.    for col in range(n):
38.        if is_safe(board, row, col, n):
39.            # Place the queen
40.            board[row][col] = True
41.
42.            # Recurse for the next row
43.            res = solve_n_queens(board, row + 1, n) or res
44.
45.            # Backtrack and remove the queen
```

```
46.                 board[row][col] = False
47.
48.     return res
49.
50. def n_queens(n):
51.     """
52.     Initialize the board and solve the N Queens problem.
53.     """
54.     board = [[False] * n for _ in range(n)]
55.     if not solve_n_queens(board, 0, n):
56.         print("No solution exists.")
57.
58. # Example Usage
59. if __name__ == "__main__":
60.     N = 8  # Change this to solve for a different board size
61.     n_queens(N)
62.
```

Output: