Software Engineering Assignment for Zerberus.ai

Objective:
To evaluate your knowledge of data structures, analytical thinking, and solution design.

Submission Email:
Send your completed assignment to careers@zerberus.ai with the subject line:
"Software Engineering Assignment - [Your Name]"

Deadline:
2 days from the date of receipt of an invitation

Assignment Overview:
Complete all mandatory tasks. Additional tasks will be considered as bonus efforts.

Mandatory Task:

Task 1: Implement a Queue Using Two Stacks (No Built-in Stack or Queue Functions)

Problem:
Create a MyQueue class to implement a **queue** using **two stacks**. However, you must **not** use any built-in stack or queue data structures provided by the language (like list's append/pop methods in Python). Instead, you must simulate the behavior of a stack by using a basic array or list and managing the operations manually.

The class should support the following operations:

1. enqueue(x): Add an element to the end of the queue.
2. dequeue(): Remove and return the front element of the queue.
3. peek(): Return the front element without removing it.
4. is_empty(): Check if the queue is empty.

Rules and Restrictions:

- You must simulate stack behaviour manually by using a basic array or list.
- Implement these stack operations yourself:
    - **Push**: Add an element to the top of the stack.

- **Pop**: Remove the top element from the stack.
- **Peek**: Return the top element of the stack without removing it.
- **Is Empty**: Check if the stack is empty.

- Use two such manually implemented stacks to create the queue.

Example Usage:

```
queue = MyQueue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print(queue.dequeue())  # Output: 1
print(queue.peek())     # Output: 2
print(queue.is_empty()) # Output: False
queue.dequeue()
queue.dequeue()
print(queue.is_empty()) # Output: True
```

Task 2: Employee Management System

Problem:
Design and implement a simple **Employee Management System** that interacts with a relational database. Your application should support the following operations:

1. **Add a New Employee**
   - Fields: id (auto-increment), name, email, department, salary.
   - Insert the employee's details into the database.
2. **Update Employee Details**
   - Allow updates to any field except id.
3. **Delete an Employee**
   - Remove an employee from the database using their id.
4. **List All Employees**
   - Fetch and display all employee records in a table-like format, sorted by their id.
5. **Calculate Average Salary by Department**
   - Fetch and display the average salary of employees for each department.

Requirements:

- Use a relational database (e.g., PostgreSQL or MySQL).
- Use any programming language/framework the candidate is comfortable with (e.g., Python, Java, Node.js).

- Ensure proper database connection handling and error handling.
- Use parameterized queries to prevent SQL injection.
- Provide a basic CLI or API interface to interact with the system.

Optional Task:(Choose 1 of 2)

Task 1: Design and Implement a Cache

- Implement an LRU (Least Recently Used) Cache with `get` and `put` operations.
- Key Concepts: Doubly linked list, hashmap, and time complexity optimization

Task 2: Design a Scheduler

Problem

Write a program to schedule tasks based on their dependencies. For example, task A depends on task B, so B should be executed before A. Use a **Topological Sort** algorithm.

Example:

Input: tasks = ['A', 'B', 'C'], dependencies = [('B', 'A'), ('C', 'B')] Output: ['C', 'B', 'A']