**Employee Data Management System**

**Problem Statement:** Employee Data Management

**Objective**: Design and implement an asynchronous C# application to manage employee data by generating comprehensive employee reports. The system must efficiently fetch data from multiple heterogeneous sources, process it into a unified report, and store the results in a file or database. The application should leverage asynchronous programming to optimize performance, ensure scalability, and provide robust error handling to maintain reliability in a real-world enterprise environment.

**Background**: In a corporate setting, HR departments often need to generate reports that combine employee information from various systems, such as HR databases, payroll systems, and training platforms. These systems are typically accessed via APIs or database queries, which are I/O-bound operations with varying response times. Sequential data retrieval would lead to significant delays, impacting user experience and system efficiency. Additionally, errors such as invalid employee IDs, network failures, or file access issues are common and must be handled gracefully to prevent system crashes.

**Requirements**:

**1. Data Sources:**

- Employee Profile: Retrieve employee details (e.g., ID, name, department) from an HR database. This simulates a database query or API call that returns structured data.

- Payroll Details: Fetch payroll information (e.g., salary, last pay date) from a payroll system. This simulates an external API or system call with potential latency.

- Training Records: Obtain training records (e.g., list of completed courses) from a training management system. This simulates another I/O-bound operation.

- Each data source operation must be asynchronous to reflect real-world I/O-bound behavior (e.g., network latency, database query time).

**2. Parallel Data Fetching:**

- Fetch data from the HR, payroll, and training systems concurrently to minimize total latency.

- Use `Task.WhenAll` to execute these independent operations in parallel, reducing the overall time compared to sequential execution.

- Simulate varying response times for each source (e.g., 1000ms for HR, 800ms for payroll, 1200ms for training) to mimic real-world variability.

**3. Data Processing:**

- Combine the fetched data into a single `EmployeeReport` object containing:

- Employee ID (int)

- Name (string)

- Department (string)

- Salary (decimal)

- Completed Courses (list of strings)

- Report Generation Date (DateTime)

- Process the data asynchronously to simulate computation or transformation (e.g., formatting, validation, or aggregation).

- Ensure the processing step depends on the completion of all data fetch tasks.

**4. Data Storage:**

- Save the generated report to a file or database, simulated as an asynchronous file write operation (e.g., writing to a text file named `employee\_report\_{id}.txt`).

- The saved report should include a formatted summary of the employee’s data (e.g., name, department, salary, courses, and generation date).

- Handle potential I/O errors (e.g., file access denied, disk full) during the save operation.

**5. Error Handling:**

- Task-Specific Errors: Each data fetch operation should handle specific errors:

- Invalid employee IDs (e.g., negative or zero) should throw an `ArgumentException`.

- File operations should handle `IOException` or `FileNotFoundException`.

- Log errors with meaningful messages (e.g., “Failed to fetch profile for ID {id}: {error}”).

- Aggregate Errors: When using `Task.WhenAll`, catch `AggregateException` to handle cases where one or more tasks fail, logging each inner exception.

- General Errors: Catch unexpected exceptions in the main workflow to prevent crashes and return a null or empty result to indicate failure.

- Ensure partial failures (e.g., one system is down) are logged but do not prevent processing of successful tasks when possible.

**6. Performance Optimization:**

- Minimize total execution time by running independent data fetches in parallel.

- Ensure the system is non-blocking, allowing the main thread to remain responsive (suitable for UI or server applications).

- Design the system to scale for processing multiple employees or additional data sources in the future.

**7. C# Async Programming Features:**

- Use `Task` for operations without return values (e.g., saving data).

- Use `Task<T>` for operations with return values (e.g., `Task<EmployeeProfile>`, `Task<PayrollDetails>`).

- Employ `async/await` to write readable, asynchronous code that handles I/O-bound operations efficiently.

- Leverage `Task.WhenAll` to execute multiple tasks concurrently and wait for their completion.

- Implement robust exception handling using `try/catch` blocks for both individual tasks and aggregate task failures.

**Constraints**:

- Assume data sources are I/O-bound with simulated delays (e.g., 800–1200ms per operation).

- Handle invalid inputs (e.g., negative employee IDs) by throwing exceptions and logging errors.

- Ensure file operations are robust against common issues (e.g., missing directories, permission errors).

- Simulate data sources using `Task.Delay` for simplicity, but design the system to work with real APIs or databases.

- The system must remain stable even if one or more data sources fail.

**Expected Output:**

- For a valid employee ID (e.g., 123):

- Generate and save a report file (e.g., `employee\_report\_123.txt`) with formatted employee data.

- Display a console summary, e.g.:

Report saved for employee ID 123

Generated Report: Employee\_123, Engineering, $75,000.00, Courses: C# Basics, Async Programming

- File content example:

Report for Employee\_123 (ID: 123)

Department: Engineering

Salary: $75,000.00

Courses: C# Basics, Async Programming

Generated: 2025-07-29 14:21:00

- For an invalid employee ID (e.g., -1):

- Log errors for each failed data fetch and indicate report generation failure:

Failed to fetch profile for ID -1: Invalid employee ID

Failed to fetch payroll for ID -1: Invalid employee ID

Failed to fetch training records for ID -1: Invalid employee ID

One or more data fetch operations failed:

Error: Invalid employee ID

Error: Invalid employee ID

Error: Invalid employee ID

Failed to generate report for invalid ID.