

AI ASSISTED CODING LAB EXAM

LAB EXAM NAME : AI Assisted Coding

ROLL NUMBER :2503a51l05

BATCH NO :24BTCAICSB19

NAME OF STUDENT: N. SATYA SRI CHARAN

PAPER SET NO :LAB 2

TASK 1: Q1: API Integration

- a. Use AI to generate Python code for calling a mock payment gateway API.
- b. Add exception handling and logging for failed transactions

1. PROMPT

Generate a Python program that integrates with a mock payment gateway API. The program should send payment details (amount and card number) to the API and receive the response. Also add exception handling for network errors or invalid responses, and include logging for failed transactions.

2. Code Generated

task1.py > ...

```
1  import requests
2  import logging
3
4  # Configure logging
5  logging.basicConfig(
6      filename="payment.log",
7      level=logging.INFO,
8      format="%(asctime)s - %(levelname)s - %(message)s"
9  )
10
11 def process_payment(amount, card_number):
12     url = "https://mock-payment-gateway.com/pay"
13     data = {
14         "amount": amount,
15         "card_number": card_number
16     }
17
18     try:
19         response = requests.post(url, json=data, timeout=5)
20         response.raise_for_status() # Raises exception for HTTP error
21         result = response.json()
22
23         if result.get("status") == "success":
24             logging.info(f"Payment successful for ₹{amount}")
25             return "Payment Successful"
26         else:
27             logging.error(f"Payment failed: {result}")
28             return "Payment Failed"
29
30     except requests.exceptions.RequestException as e:
31         logging.error(f"API Error: {e}")
32         return "Payment Failed due to API Error"
33     except Exception as e:
34         logging.error(f"Unexpected Error: {e}")
35         return "Payment Failed due to Unexpected Error"
```

```

11 def process_payment(amount, card_number):
34     logging.error(f"Unexpected Error: {e}")
35     return "Payment Failed due to Unexpected Error"
36
37
38 # Main Program
39 if __name__ == "__main__":
40     amount = 1000
41     card = "4111111111111111"
42     result = process_payment(amount, card)
43     print(result)
44

```

3. Output:

```

PROBLEMS OUTPUT Filter Code
[Running] python -u "c:\Users\N.SRICHARAN\Desktop\ai\ai lab test 4\task1.py"
Payment Failed due to API Error

[Done] exited with code=0 in 0.622 seconds

```

4. OBSERVATION:

- The program successfully **calls a payment gateway API** and sends payment details (amount and card number).
- When the API cannot be reached (as in the mock URL used), the program **handles the exception** gracefully and does not crash.
- The program **logs all important events**:
- Successful payments are logged as **INFO**.
- Failed transactions and API errors are logged as **ERROR**.
- The program returns appropriate messages on the console:
- "Payment Successful" for successful transactions.
- "Payment Failed" if the API returns a failure status.
- "Payment Failed due to API Error" if the request could not reach the server.

- "Payment Failed due to Unexpected Error" for any other unexpected issues.
- The program demonstrates the importance of **exception handling** and **logging** in real-world API integration to track transactions.
- Since a **mock URL** was used, the program currently shows "Payment Failed due to API Error" — this confirms that **error handling works correctly**.

Task 2:

Q2: Code Translation

- Translate an order-sorting function from Python to Java using AI.
- Validate correctness by generating sample input/output test cases

1. PROMPT:

Translate a Python order-sorting function to Java, including test cases and sample output.

2. Code generated:

```

task2.py > ...
1  import logging
2  import requests
3
4  # Configure logging
5  logging.basicConfig(filename='app.log', level=logging.INFO, format='%(asctime)s - %(le
6
7  # Part 1: Order Sorting Function
8  def sort_orders(orders):
9      return sorted(orders)
10
11 # Part 2: Mock Payment Gateway API Call
12 API_ENDPOINT = 'https://mock-payment-gateway.com/api/v1/transaction'
13
14 def call_payment_gateway(transaction_data):
15     try:
16         response = requests.post(API_ENDPOINT, json=transaction_data, timeout=10)
17         response.raise_for_status()
18         result = response.json()
19         if result.get('status') == 'success':
20             logging.info(f"Transaction successful: {result}")
21             return result
22         else:
23             logging.error(f"Transaction failed: {result}")
24             return None
25     except requests.exceptions.RequestException as e:
26         logging.error(f"Request failed: {e}")
27         return None
28
29 # Example usage
30 if __name__ == "__main__":
31     # Example for sorting orders
32     orders = [105, 101, 109, 110]
33     sorted_orders = sort_orders(orders)
34     print("Sorted Orders:", sorted_orders)

```

3. OUTPUT:

```

[Running] python -u "c:\Users\N.SRICHARAN\Desktop\ai\ai lab test 4\task2 .py"
Sorted Orders: [101, 105, 109, 110]

```

4. OBSERVATION:

- ❖ Combining related functionalities, like order sorting and payment processing, into a single program improves modularity and ease of maintenance while keeping code organized.
- ❖ Using clear function separation for each task (sorting, API calls) allows easy testing and reuse.
- ❖ Proper exception handling and logging are critical when working with external services like payment APIs to ensure errors are tracked and system reliability is maintained.
- ❖ Writing sample test cases or example usages in the combined program validates correctness and aids debugging.
- ❖ It is generally simpler to keep the combined program in a single language for coherence unless specific interoperability methods are used.
- ❖ AI-assisted code generation can accelerate translating or merging code across languages or functions but manual review is needed to ensure correctness and performance.