

AI ASSISTED CODING

LAB EXAM – 4

SET4

Name: Likhitha Pothunuri

Roll no: 2403A510D1

Batch 05

CSE 2nd year

Q1:

a) Prompt:

Translate the following PHP function into clean, idiomatic Node.js.

Maintain the same logic, conditions, and return values.

Also explain if any changes are needed in JavaScript related to typing or inputs.

PHP Code:

```
function calculateTotalValue($price, $quantity) {  
    if ($price < 0 || $quantity < 0) {  
        return -1; // invalid  
    }  
    return $price * $quantity;  
}
```

Code:

```
JS calculateTotalValue.js > ...
    Complexity is 5 Everything is cool!
1   function calculateTotalValue(price, quantity) {
2       if (price < 0 || quantity < 0) {
3           return -1; // invalid
4       }
5       return price * quantity;
6   }
7
8   module.exports = calculateTotalValue;
9
```

```
Welcome          calculateTotalValue.js      test.js
JS test.js > ...
1   const fn = require("./calculateTotalValue");
2
3   console.log(fn(10, 5));      // ?
4   console.log(fn(100, 0));     // ?
5   console.log(fn(-5, 10));     // ?
6   console.log(fn(20, -3));     // ?
7   console.log(fn(7, 2));       // ?
8   console.log(fn(0, 0));       // ?
```

Output:

```
● PS C:\Users\likhi\OneDrive\Desktop\AI_labexam4> node test.js
50
0
-1
-1
14
0
↖ PS C:\Users\likhi\OneDrive\Desktop\AI_labexam4>
```

Observation:

1. The Node.js version follows the same conditions and return values as the PHP function, ensuring that the core behavior remains unchanged.
2. PHP automatically converts strings to numbers, but JavaScript may return NaN, so the Node.js version should include additional checks to avoid unexpected results.
3. Running the same test cases in both languages helps verify that the outputs match, making it easier to identify errors and ensure a reliable migration.

b) Prompt:

Explain how to cross-validate whether the translated Node.js function behaves exactly the same as the original PHP function. Include the steps, reasoning, test cases, and how to compare outputs from both languages.

Explanation:

Step 1: Understand the original PHP function

First, note what the PHP function does:

- If any input is negative → return -1
- Otherwise → return **price × quantity**

This becomes your expected behavior.

Step 2: Create a list of test cases

Choose different types of inputs:

- calculateTotalValue(10, 5) → 50
- calculateTotalValue(100, 0) → 0
- calculateTotalValue(-5, 3) → -1

- calculateTotalValue(7, 2) → 14

These represent normal, boundary, and invalid cases.

Step 3: Run the same test cases in both PHP and Node.js

Execute the PHP function with the test inputs and write down the outputs.

Then run the Node.js version with the same inputs.

Step 4: Compare the results

If **every** output in Node.js matches the output in PHP, then the translation is correct.

For example:

Input	PHP Output	Node.js Output	Match?
(10,5)	50	50	✓
(-5,3)	-1	-1	✓

Matching results mean the logic is preserved.

Step 5: Consider type differences

PHP auto-converts strings to numbers.

JavaScript may return NaN.

So, if needed, add validation in Node.js.

Q2:

a) Prompt:

Convert the following SQL query into Django ORM syntax.

Maintain the same conditions, selected columns, joins, and ordering.

SQL Query:

```
SELECT id, name, price
FROM products
WHERE price > 100
ORDER BY price DESC;
```

Code:

```
⌚ sqltodjango.py > ...
1   from django.db.models import F
2   from ..models import Product
3
4   # Django ORM equivalent of:
5   # SELECT id, name, price
6   # FROM products
7   # WHERE price > 100
8   # ORDER BY price DESC;
9
10  # Option 1: Using values() to return dictionaries
11  products = Product.objects.filter(
12      price__gt=100
13  ).values('id', 'name', 'price').order_by('-price')
14
15  # Option 2: Using model instances instead of dictionaries
16  # (remove .values() call if you prefer this approach)
17  # products = Product.objects.filter(
18  #     price__gt=100
19  # ).order_by('-price')
20
21  # Iterating through results
22  for product in products:
23      print(product['id'], product['name'], product['price'])
24
```

```
sqlite3> .dump
1 """
2 Django ORM Conversion Demo
3 SQL to Django ORM Translation
4 """
5
6 # This demonstrates the Django ORM syntax equivalent to the SQL query
7 # Original SQL:
8 # SELECT id, name, price
9 # FROM products
10 # WHERE price > 100
11 # ORDER BY price DESC;
12
13 # Django ORM Equivalent:
14 # =====
15
16 print("=" * 60)
17 print("SQL to Django ORM Conversion")
18 print("=" * 60)
19
20 print("\n❑ ORIGINAL SQL QUERY:")
21 print("-" * 60)
22 sql_query = """
23 SELECT id, name, price
24 FROM products
25 WHERE price > 100
26 ORDER BY price DESC;
27 """
28 print(sql_query)
29
30 print("\n❑ DJANGO ORM EQUIVALENT:")
31 print("-" * 60)
32 django_orm = """
33 from django.db.models import F
```

```
sqltodjango_demo.py>...
31     print("-" * 60)
32     django_orm = """
33     from django.db.models import F
34     from .models import Product
35
36     # Option 1: Using values() to return dictionaries
37     products = Product.objects.filter(
38         price__gt=100
39     ).values('id', 'name', 'price').order_by('-price')
40
41     # Option 2: Using model instances
42     products = Product.objects.filter(
43         price__gt=100
44     ).order_by('-price')
45     """
46     print(django_orm)
47
48     print("\n📌 BREAKDOWN:")
49     print("-" * 60)
50     print("""
51     1. Product.objects.filter(price__gt=100)
52         └ Equivalent to: WHERE price > 100
53
54     2. .values('id', 'name', 'price')
55         └ Equivalent to: SELECT id, name, price
56
57     3. .order_by('-price')
58         └ Equivalent to: ORDER BY price DESC
59         └ (The '-' prefix indicates DESC order)
60     """)
61
62     print("\n✅ Conversion Complete!")
63     print("-" * 60)
64
```

Output:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE SQL HISTORY TASK MONITOR
PS C:\Users\likhi\OneDrive\Desktop\AI_labexam4> C:/Users/likhi/OneDrive/Desktop/AI_labexam4/.venv/Scripts/python.exe "C:/U
=====
SQL to Django ORM Conversion
=====

ORIGINAL SQL QUERY:
-----
SELECT id, name, price
FROM products
WHERE price > 100
ORDER BY price DESC;

DJANGO ORM EQUIVALENT:
-----
from django.db.models import F
from .models import Product

# Option 1: Using values() to return dictionaries
products = Product.objects.filter(
    price__gt=100
).values('id', 'name', 'price').order_by('-price')

# Option 2: Using model instances
products = Product.objects.filter(
    price__gt=100
).order_by('-price')

BREAKDOWN:
-----
1. Product.objects.filter(price__gt=100)
   └ Equivalent to: WHERE price > 100

2. .values('id', 'name', 'price')

BREAKDOWN:
-----
1. Product.objects.filter(price__gt=100)
   └ Equivalent to: WHERE price > 100

2. .values('id', 'name', 'price')
   └ Equivalent to: SELECT id, name, price

3. .order_by('-price')
   └ Equivalent to: ORDER BY price DESC
   └ (The '-' prefix indicates DESC order)

Conversion Complete!
  └ (The '-' prefix indicates DESC order)
  └ (The '-' prefix indicates DESC order)

Conversion Complete!
=====
PS C:\Users\likhi\OneDrive\Desktop\AI_labexam4>
```

Observation:

1. The Django ORM version preserves the exact logic of the SQL query.
2. SQL WHERE price > 100 becomes Django filter(price__gt=100).
3. SQL ordering ORDER BY price DESC becomes Django order_by('-price').
4. values() is used to return selected fields similar to SQL SELECT id, name, price.
5. ORM improves readability and integrates directly with Django models.

b) Prompt:

“State 3 limitations of automated code translation. Explain in simple points how AI-based translation can fail or produce incomplete results.”

3 Limitations:

a. AI may not fully understand business logic

Automated tools often convert syntax correctly but may miss hidden rules or special conditions in the original query. This can lead to **incorrect ORM logic** even if the translation looks correct.

b. Differences in language features can cause mismatches

SQL and ORM frameworks (like Django ORM) do not always map 1-to-1.

For example:

- SQL functions (COUNT, JOINs, subqueries) may not translate cleanly
- ORM may require additional filters or annotations
This can result in **partial or inaccurate translations.**
c. Translated code may run but not be optimal or secure
AI may produce ORM code that:
 - is inefficient
 - performs unnecessary queries
 - does not follow best practices
 - may introduce security issues (e.g., missing validations)Thus, manual review is still required to ensure **performance, security, and accuracy**