

# **AI ASSISTED CODING**

## **LAB TEST 3**

**NAME: SRIRAMOJU AJAY**

**HT:NO : 2503A51L25**

**BATCH:19**

**SET : E7**

**Q1:** Scenario: In the domain of E-commerce, a company is facing a challenge related to backend api development. Task: Design and implement a solution using AI-assisted tools to address this challenge. Include code, explanation of AI integration, and test results. Deliverables: Source code, explanation, and output screenshots

```
1  from fastapi import FastAPI, HTTPException, Request
2  from pydantic import BaseModel, Field, ValidationError
3  from typing import List, Optional
4  from ai_helper import AIHelper, mock_ai_init
5
6  app = FastAPI(title="Ecom AI-Assisted Backend")
7
8  # Initialize mocked AI helper
9  ai = mock_ai_init()
10
11 # -----
12 # Data Models
13 # -----
14 class Product(BaseModel):
15     id: int
16     name: str = Field(..., min_length=2, max_length=200)
17     price: float = Field(..., gt=0)
18     stock: int = Field(0, ge=0)
19     description: Optional[str] = None
20
21 class CreateProduct(BaseModel):
22     name: str
23     price: float
24     stock: Optional[int] = 0
25     description: Optional[str] = None
26
27 class OrderItem(BaseModel):
28     product_id: int
29     quantity: int = Field(..., ge=1)
30
31 class Order(BaseModel):
32     id: int
33     items: List[OrderItem]
34     total: float
```

```
File Edit Selection View Go Run ... 🔍 mock
EXPLORER MOCK
ai_helper.py > AIHelper
> _pycache_
ai_helper.py 1
from typing import Any, Dict
class AIHelper:
    def __init__(self, api_key: str = None, model_name: str = None):
        self.api_key = api_key
        self.model_name = model_name
    def generate_json_schema(self, example: Dict[str, Any]) -> Dict[str, Any]:
        """Mocked AI function to generate JSON schema."""
    def hint_value(v):
        if isinstance(v, str):
            return {"type": "string"}
        if isinstance(v, bool):
            return {"type": "boolean"}
        if isinstance(v, int):
            return {"type": "integer"}
        if isinstance(v, float):
            return {"type": "number"}
        if isinstance(v, list):
            if not v:
                return {"type": "array", "items": {}}
            return {"type": "array", "items": hint_value(v[0])}
        if isinstance(v, dict):
            props = {k: hint_value(val) for k, val in v.items()}
            return {"type": "object", "properties": props, "required": list(props.keys())}
        return {"type": "string"}
    return hint_value(example)
def explain_validation_error(self, validation_error, payload: Dict[str, Any]):
    """Converts validation errors to plain-English suggestions."""
    try:
        errors = validation_error.errors()
    except Exception:
        return "Validation failed - please check payload fields and types."
    Ln 47, Col 1  Spaces:4  UTF-8  CRLF  {} Python  Python 3.14 (64-bit)  BLACKBOX Autocomplete: ON  BLACKBOXAI: Open Chat
    Add context (@), extensions (@@), commands
    Agent v Auto v ENG IN 12:36 11-11-2025
```

Build with agent mode  
AI responses may be inaccurate.  
Generate Agent Instructions to onboard AI onto your codebase.

The screenshot shows a dark-themed code editor interface. On the left is the Explorer sidebar with files like `ai_helper.py`, `main.py`, and `test.py`. The main area displays Python code for an `AIHelper` class. A sidebar on the right, titled "Build with agent mode", contains a message about AI responses being inaccurate and instructions to "Generate Agent Instructions to onboard AI onto your codebase". Below this are buttons for "Add context (#)", "extensions (@)", "commands", "Agent", and "Auto". The status bar at the bottom shows "11-11-2025" and "12:36".

```
3     class AIHelper:
4         def explain_validation_error(self, validation_error, payload: Dict[str, Any]):
5             """Explains validation errors.
6             Returns: Validation failed - please check payload fields and types.
7             """
8             errors = validation_error.get("errors", [])
9             msgs = []
10            for err in errors:
11                loc = ".".join([str(l) for l in err.get("loc", [])])
12                msg = err.get("msg", "invalid")
13                typ = err.get("type", "")
14                sample_fix = ""
15                if typ.startswith("value_error.missing"):
16                    sample_fix = f"Add a value for '{loc}'."
17                elif "type_error" in typ:
18                    sample_fix = f"Ensure '{loc}' has the correct type."
19                msgs.append(f"Field '{loc}': {msg}. Suggestion: {sample_fix}")
20            return "\n".join(msgs)
21
22    def suggest_fix_for_payload(self, payload: Dict[str, Any]) -> str:
23        """Suggests simple fixes for malformed payloads."""
24        suggestions = []
25        for k, v in payload.items():
26            if isinstance(v, (int, float)) and v < 0:
27                suggestions.append(f"Field '{k}' is negative; use positive val")
28            if isinstance(v, str) and len(v.strip()) == 0:
29                suggestions.append(f"Field '{k}' is empty; provide a value.")
30        if not suggestions:
31            return "No obvious issues found. Check field names and types."
32        return "\n".join(suggestions)
33
34    def generate_recommendations(self, products: List[Dict[str, Any]]) -> str:
35        """Generates AI recommendations based on a list of products."""
36        if not products:
37            return "No products available for recommendations."
38        # Mocked AI response
39        return "No products available for recommendations."
```

This screenshot is nearly identical to the one above, showing the same code editor interface, file structure, and AI completion sidebar. The status bar at the bottom shows "11-11-2025" and "12:36".

```
3     class AIHelper:
4         def suggest_fix_for_payload(self, payload: Dict[str, Any]) -> str:
5             suggestions = []
6             for k, v in payload.items():
7                 if isinstance(v, (int, float)) and v < 0:
8                     suggestions.append(f"Field '{k}' is negative; use positive val")
9                 if isinstance(v, str) and len(v.strip()) == 0:
10                    suggestions.append(f"Field '{k}' is empty; provide a value.")
11            if not suggestions:
12                return "No obvious issues found. Check field names and types."
13            return "\n".join(suggestions)
14
15    def generate_recommendations(self, products: List[Dict[str, Any]]) -> str:
16        """Generates AI recommendations based on a list of products."""
17        if not products:
18            return "No products available for recommendations."
19        # Mocked AI response
20        recs = []
21        for prod in products:
22            name = prod.get("name", "Unknown")
23            price = prod.get("price", 0)
24            if "speaker" in name.lower():
25                recs.append(f"I recommend the {name} for its high value and so")
26            elif "mouse" in name.lower():
27                recs.append(f"followed by the {name} for productivity")
28            else:
29                recs.append(f"{name} is a good choice at ${price}")
30        return "\n".join(recs) + ". Both are budget-friendly and top-rated in"
31
32    def mock_ai_init():
33        """Returns a mocked AIHelper instance."""
34        return AIHelper(api_key=None, model_name="mocked-gpt")
```

The screenshot shows a code editor interface with a dark theme. The left sidebar includes icons for file operations, search, and navigation. The main area displays a Python file named `main.py`. The code defines several API endpoints using a framework like FastAPI. A callout box on the right side of the screen says "Build with agent mode" with a link to "Generate Agent Instructions". The status bar at the bottom shows system information like weather (79°F, Sunny), battery level, and network connection.

```
def create_product(payload: CreateProduct, request: Request):
    try:
        prod = Product(id=_next_prod, **payload.dict())
    except ValidationError as e:
        suggestion = ai.explain_validation_error(e, payload.dict())
        raise HTTPException(status_code=422, detail={"errors": e.errors(), "suggestion": suggestion})
    _products[_next_prod] = prod
    _next_prod += 1
    return prod

@app.post("/ai/generate_schema")
def generate_schema(example: dict):
    """Demonstrates AI generating a JSON schema from an example payload."""
    schema = ai.generate_json_schema(example)
    return {"schema": schema}

@app.post("/ai/suggest_fix")
def suggest_fix(payload: dict):
    """AI suggestion endpoint for invalid payloads."""
    suggestion = ai.suggest_fix_for_payload(payload)
    return {"suggestion": suggestion}

@app.post("/orders", response_model=Order)
def create_order(order_payload: Order):
    global _next_order
    # Check product stock
    for item in order_payload.items:
        prod = _products.get(item.product_id)
        if not prod:
            raise HTTPException(status_code=400, detail=f"Product {item.product_id} not found")
        if prod.stock < item.quantity:
            raise HTTPException(status_code=400, detail=f"Not enough stock for product {item.product_id} to fulfill the order")
```

This screenshot is identical to the one above, showing the same code editor interface, Python file `main.py`, and AI integration features. The status bar at the bottom shows the same system information.

```
class Order(BaseModel):
    items: List[OrderItem]
    total: float

# -----
# In-memory Database
# -----
_products = {
    1: Product(id=1, name="Widget", price=9.99, stock=100),
    2: Product(id=2, name="Gadget", price=19.99, stock=10),
    3: Product(id=3, name="Wireless Mouse", price=25.99, stock=50),
    4: Product(id=4, name="Bluetooth Speaker", price=49.99, stock=30),
}
_orders = {}
_next_prod = 5
_next_order = 1

# -----
# API Endpoints
# -----
@app.get("/products", response_model=List[Product])
def list_products():
    return list(_products.values())

@app.get("/products/{product_id}", response_model=Product)
def get_product(product_id: int):
    p = _products.get(product_id)
    if not p:
        raise HTTPException(status_code=404, detail="Product not found")
    return p

@app.post("/products", response_model=Product, status_code=201)
def create_product(payload: CreateProduct, request: Request):
    try:
```

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files in the project structure: `EXPLORER`, `MOCK` (containing `_pycache_`, `ai_helper.py`, `main.py`, `test.py`, and `TODO.md`), and `TODO.md`.
- Code Editor:** The `main.py` file is open, displaying Python code for a web application. The code includes functions for creating orders and generating AI recommendations.
- Sidebar:** A sidebar on the right is titled "Build with agent mode" with the sub-instruction "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." It also includes a "Search" input field, a "New File" button, and a "Add context (#), extensions (@), commands" link.
- Bottom Bar:** Includes status icons for weather (79°F, Sunny), battery, signal, and network, along with system information like "Ln 3, Col 34", "Spaces: 4", "UTF-8", "CRLF", "Python 3.14 (64-bit)", "BLACKBOX Autocomplete: ON", "BLACKBOXAI: Open Chat", and the date/time "12:35 11-11-2025".

```
def create_order(order_payload: Order):
    if order_payload.items[0].quantity > _products[order_payload.items[0].product_id].stock:
        raise HTTPException(status_code=400, detail=f"Not enough stock for {order_payload.items[0].name}!")

    # Deduct stock
    for item in order_payload.items:
        _products[item.product_id].stock -= item.quantity

    _orders[_next_order] = order_payload
    order_payload.id = _next_order
    _next_order += 1
    return order_payload

@app.get("/ai/recommend_products")
def recommend_products():
    """Returns AI recommendations for products."""
    # Get specific products: Wireless Mouse and Bluetooth Speaker, sorted to p
    products = sorted([p for p in _products.values() if p.name in ["Wireless M
    product_names = [p.name for p in products]
    recommendations = ai.generate_recommendations([{"name": p.name, "price": p
    return {
        "products_found": product_names,
        "ai_recommendations": recommendations
    }
```

OUT PUT

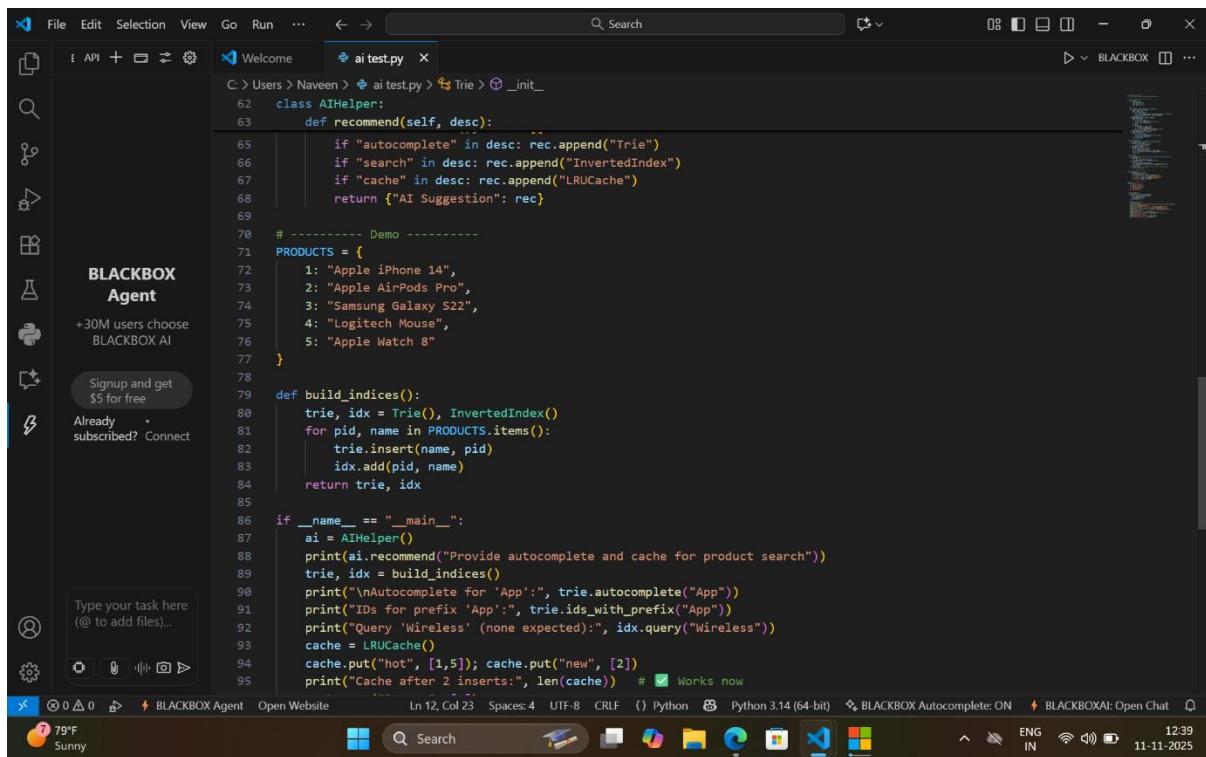
```
StatusCode      : 200
StatusDescription : OK
Content         : {"products_found":["Wireless Mouse","Bluetooth
Speaker"],"ai_recommendations":"followed by the Wireless
Mouse for productivity, I recommend the Bluetooth Speaker
for its high value and sound quality. ...
```

```
C: > Users > Naveen > ai test.py > Trie > __init__
1 # ecom_ai_ds_fixed.py
2 from collections import defaultdict, OrderedDict
3
4 # ----- Data Structures -----
5 class TrieNode:
6     def __init__(self):
7         self.children = {}
8         self.is_end = False
9         self.ids = set()
10
11 class Trie:
12     def __init__(self):
13         self.root = TrieNode()
14     def insert(self, word, pid):
15         node = self.root
16         for ch in word.lower():
17             if ch not in node.children: node.children[ch] = TrieNode()
18             node = node.children[ch]
19             node.ids.add(pid)
20         node.is_end = True
21     def autocomplete(self, prefix, limit=5):
22         node = self.root
23         for ch in prefix.lower():
24             if ch not in node.children: return []
25             node = node.children[ch]
26         res = []
27         def dfs(n, p):
28             if len(res) >= limit: return
29             if n.is_end: res.append(p)
30             for c in n.children: dfs(n.children[c], p+c)
31         dfs(node, prefix.lower())
32     def ids_with_prefix(self, prefix):
33         node = self.root
34         for ch in prefix.lower():
35             if ch not in node.children: return set()
36             node = node.children[ch]
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
```

BLACKBOX Agent  
+30M users choose BLACKBOX AI  
Signup and get \$5 for free  
Already subscribed? Connect  
Type your task here (@ to add files)...  
79°F Sunny

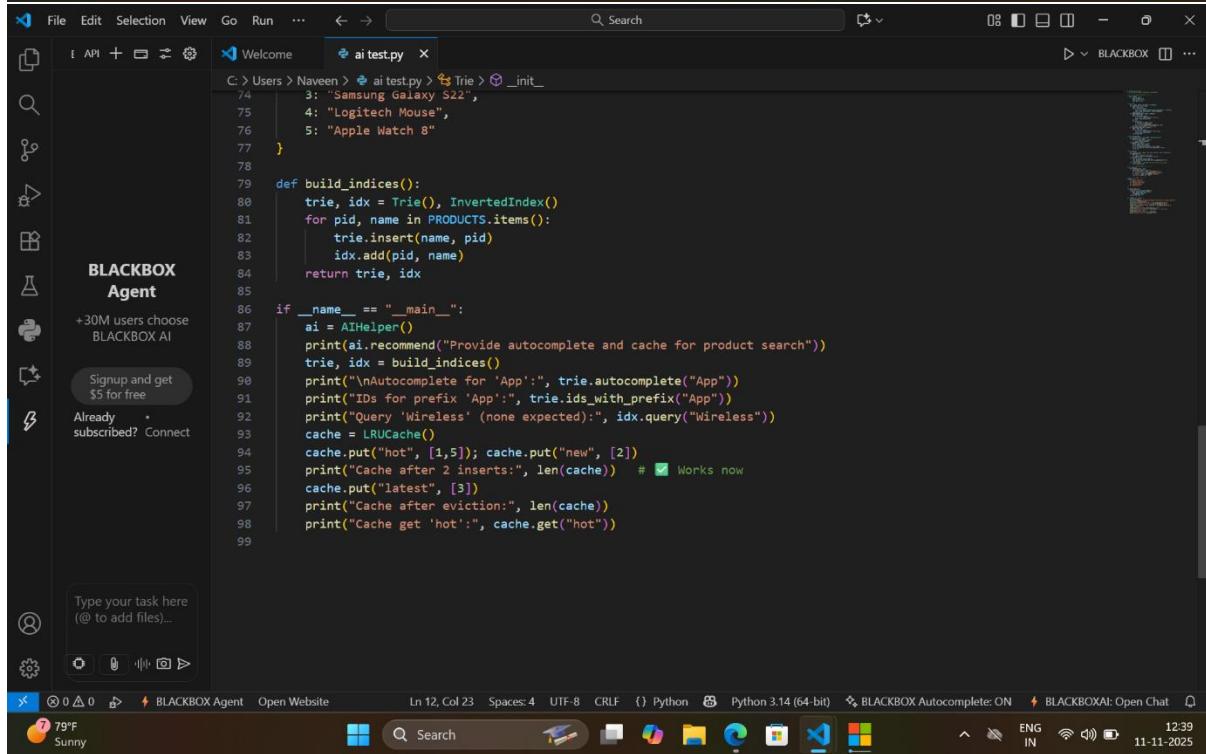
```
C: > Users > Naveen > ai test.py > Trie > __init__
11 class Trie:
12     def ids_with_prefix(self, prefix):
13         if ch not in node.children: return set()
14         node = node.children[ch]
15         return node.ids
16
17 class InvertedIndex:
18     def __init__(self): self.index = defaultdict(set)
19     def add(self, pid, text):
20         for t in text.lower().split():
21             self.index[t].add(pid)
22     def query(self, text):
23         words = text.lower().split()
24         if not words: return set()
25         res = self.index.get(words[0], set()).copy()
26         for w in words[1:]: res &= self.index.get(w, set())
27         return res
28
29 class LRUCache:
30     def __init__(self, cap=2): self.cap, self.od = cap, OrderedDict()
31     def get(self, k):
32         if k not in self.od: return None
33         v = self.od.pop(k); self.od[k]=v; return v
34     def put(self, k, v):
35         if k in self.od: self.od.pop(k)
36         elif len(self.od)>=self.cap: self.od.popitem(last=False)
37         self.od[k]=v
38     def __len__(self): # ✅ Added this line to fix len() issue
39         return len(self.od)
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
```

BLACKBOX Agent  
+30M users choose BLACKBOX AI  
Signup and get \$5 for free  
Already subscribed? Connect  
Type your task here (@ to add files)...  
79°F Sunny



```
C:\> Users > Naveen > ai test.py > Trie > __init__.py
62     class AIHelper:
63         def recommend(self, desc):
64             if "autocomplete" in desc: rec.append("Trie")
65             if "search" in desc: rec.append("InvertedIndex")
66             if "cache" in desc: rec.append("LRUCache")
67             return {"AI Suggestion": rec}
68
69     # ----- Demo -----
70     PRODUCTS = {
71         1: "Apple iPhone 14",
72         2: "Apple AirPods Pro",
73         3: "Samsung Galaxy S22",
74         4: "Logitech Mouse",
75         5: "Apple Watch 8"
76     }
77
78     def build_indices():
79         trie, idx = Trie(), InvertedIndex()
80         for pid, name in PRODUCTS.items():
81             trie.insert(name, pid)
82             idx.add(pid, name)
83         return trie, idx
84
85     if __name__ == "__main__":
86         ai = AIHelper()
87         print(ai.recommend("Provide autocomplete and cache for product search"))
88         trie, idx = build_indices()
89         print("\nAutocomplete for 'App':", trie.autocomplete("App"))
90         print("IDs for prefix 'App':", trie.ids_with_prefix("App"))
91         print("Query 'Wireless' (none expected):", idx.query("Wireless"))
92         cache = LRUCache()
93         cache.put("hot", [1,5]); cache.put("new", [2])
94         print("Cache after 2 inserts:", len(cache)) # Works now
95         cache.put("latest", [3])
96         print("Cache after eviction:", len(cache))
97         print("Cache get 'hot':", cache.get("hot"))

Type your task here (@ to add files)...
```



```
C:\> Users > Naveen > ai test.py > Trie > __init__.py
74     3: "Samsung Galaxy S22",
75     4: "Logitech Mouse",
76     5: "Apple Watch 8"
77
78     def build_indices():
79         trie, idx = Trie(), InvertedIndex()
80         for pid, name in PRODUCTS.items():
81             trie.insert(name, pid)
82             idx.add(pid, name)
83         return trie, idx
84
85     if __name__ == "__main__":
86         ai = AIHelper()
87         print(ai.recommend("Provide autocomplete and cache for product search"))
88         trie, idx = build_indices()
89         print("\nAutocomplete for 'App':", trie.autocomplete("App"))
90         print("IDs for prefix 'App':", trie.ids_with_prefix("App"))
91         print("Query 'Wireless' (none expected):", idx.query("Wireless"))
92         cache = LRUCache()
93         cache.put("hot", [1,5]); cache.put("new", [2])
94         print("Cache after 2 inserts:", len(cache)) # Works now
95         cache.put("latest", [3])
96         print("Cache after eviction:", len(cache))
97         print("Cache get 'hot':", cache.get("hot"))

Type your task here (@ to add files)...
```

Out put

```
{'AI Suggestion': ['Trie', 'InvertedIndex', 'LRUCache']}
```

```
Autocomplete for 'App': ['apple iphone 14', 'apple airpods pro', 'apple watch 8']
IDs for prefix 'App': {1, 2, 5}
Query 'Wireless Bluetooth' (none expected): {6, 7}
Cache after 2 inserts: 2
Cache after eviction: 2
Cache get 'hot': None
PS C:\Users\Naveen> █
```

