ДОДАТОК А

Код програмної реалізації

data_utility_classifier.py

```
from sklearn.ensemble import GradientBoostingClassifier
import joblib
import pandas as pd

class DataUtilityClassifier:
    def __init__(self):
        self.model = GradientBoostingClassifier()

    def train(self, data_path):
        df = pd.read_csv(data_path)
        X = df.drop(columns=["label"])
        y = df["label"]
        self.model.fit(X, y)
        joblib.dump(self.model, "model_classifier.joblib")

    def predict(self, features):
        if not hasattr(self, 'model'):
            self.model =
joblib.load("model_classifier.joblib")
        return self.model.predict([features])[0]
```

access_monitor.py

```
from datetime import datetime, timedelta, timezone
from typing import List

def simulate_access_log(file_id: str) -> dict:
    return {
        "file_id": file_id,
        "access_frequency": 27,
        "last_access_time": (datetime.now(timezone.utc) -
timedelta(days=2)).isoformat(),
        "source_type": 1,
        "size_MB": 340
    }

def process_logs(logs: List[dict]) -> List[dict]:
    result = []
    now = datetime.now(timezone.utc)
    for log in logs:
        dt = datetime.fromisoformat(log["last_access_time"])
```

```
            days_ago = (now - dt).days
            result.append({
                "file_id": log["file_id"],
                "access_frequency": log["access_frequency"],
                "last_access_days_ago": days_ago,
                "size_MB": log["size_MB"],
                "source_type": log["source_type"]
            })
        return result
```

policy_generator.py

```
class PolicyGenerator:
    def __init__(self):
        self.tiers = {
            "hot": "Hot",
            "cool": "Cool",
            "archive": "Archive"
        }

    def generate_policy(self, utility_score):
        if utility_score == "high":
            return self.tiers["hot"]
        elif utility_score == "medium":
            return self.tiers["cool"]
        else:
            return self.tiers["archive"]
```

orchestrator.py

```
from data_utility_classifier import DataUtilityClassifier
from access_monitor import AccessMonitor
from policy_generator import PolicyGenerator

class Orchestrator:
    def __init__(self):
        self.classifier = DataUtilityClassifier()
        self.monitor = AccessMonitor()
        self.policy_gen = PolicyGenerator()

    def run(self, object_id, features):
        self.monitor.log_access(object_id)
        score = self.classifier.predict(features)
        policy = self.policy_gen.generate_policy(score)
        return {
            "object_id": object_id,
            "storage_policy": policy,
```

```
            "accessed_at":
self.monitor.get_last_access(object_id)
        }
```

app.py

```
from orchestrator import Orchestrator

if __name__ == "__main__":
    orch = Orchestrator()
    sample_features = [0.75, 3, 120]  # utility_score,
access_count, days_since_last_access
    result = orch.run("file_123", sample_features)
    print(result)
```

auto_classifier.py

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient, BlobClient
from datetime import datetime, timedelta
import pandas as pd

# Параметри
storage_account_url = "https://
<your_storage_account>.blob.core.windows.net/"
container_name = "<your_container_name>"
days_threshold = 90  # Порогова кількість днів
target_tier = "Archive"  # Можливо також: "Hot", "Cool",
"Archive"

# Ініціалізація клієнта
credential = DefaultAzureCredential()
blob_service_client =
BlobServiceClient(account_url=storage_account_url,
credential=credential)
container_client =
blob_service_client.get_container_client(container_name)

# Дані класифікації (можна зчитати з БД або CSV)
classification_df = pd.read_csv("classification_results.csv")
# Очікується колонка 'blob_name', 'utility_class',
'last_access_date'

# Обробка кожного блобу
for index, row in classification_df.iterrows():
    blob_name = row["blob_name"]
    utility_class = row["utility_class"]
```

```
    last_access_str = row["last_access_date"]

    # Пропуск, якщо utility_class не "Low"
    if utility_class != "Low":
        continue

    # Перевірка дати останнього доступу
    last_access_date = datetime.strptime(last_access_str, "%Y-
%m-%d")
    if datetime.utcnow() - last_access_date >
timedelta(days=days_threshold):
        # Підготовка клієнта та зміна класу зберігання
        try:
            blob_client =
container_client.get_blob_client(blob_name)
            blob_client.set_standard_blob_tier(target_tier)
            print(f"[OK] Blob {blob_name} переведено до
{target_tier}")
        except Exception as e:
            print(f"[ERR] Не вдалося перевести {blob_name}:
{e}")
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	60 с.
	Інші документи	
2.	Презентаційні матеріали	15 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Мітрошкіна К.В.			Розробка системи динамічної оптимізації обсягів збереження даних у Data Lake на основі класифікації корисності даних
Перевірив		Єрохін М.А.			
Н.контр.		Малєєва І.А.			
Затв.		Золотухін О.В.			

Шифр групи	Код напр./ спец.
СШІм-23-2	122
ХНУРЕ кафедра ШІ	