Detailed Report: EMIS Process Automation with Python, JSON, and GitHub Copilot

Prepared by: Francisco José Nardi Filho

Introduction

- **Objective:** Optimize EMIS processes with automation and onboarding support.
- **Focus Areas:**
    - Automating repetitive JSON file creation.
    - Enhancing new developer onboarding.
    - Leveraging GitHub Copilot.
- **Structure:** Analysis of problems and a detailed practical solution.

Part 1: Problem Identification, Research, and Onboarding Strategy

1.1 Problem Context

- **What is EMIS?** A configurable banking integration system using JSON-based low-code processes (MOVE, SEND).
- **Key Challenges:**
    - Manual JSON creation is repetitive and error-prone.
    - Onboarding is slow due to limited documentation.
- **Impact:** Delays and team overload at Bank of America.

1.2 Research: Key Questions and Insights

- **Q1: Challenges & Adaptation**
    - *Prompt:* "What challenges do new developers face in EMIS?"
    - *Insight:* Steep learning curve; suggests training and mentorship.
- **Q2: Support Materials**
    - *Prompt:* "How can we improve learning resources?"
    - *Insight:* Use visual playbooks and Copilot integration.

- **Q3: Onboarding Efficiency**
    - *Prompt:* "How to streamline onboarding?"
    - *Insight:* Shadowing, pair programming, and updated docs.
- **Q4: Automation Tools**
    - *Prompt:* "What tools can automate tasks?"
    - *Insight:* Python, Copilot, and Git for efficiency.

1.3 Resolution Strategy

- **Decision:** Hire a new developer with Copilot support.
- **Goals:**
    - Reduce learning time from months to weeks.
    - Minimize manual errors.
    - Boost productivity with JSON automation.
- **Justification:** Addresses documentation and task repetition issues.

Part 2: Practical Solution - JSON File Automation

2.1 Case Description

- **Problem:** Manual creation of JSONs for MOVE/SEND (currencies: DKK, NOK, etc.; types: MT300, MT202, BMC) is inefficient.
- **Objective:** Automate with Python, supported by Copilot.

2.2 Step-by-Step Resolution

Step 1: Data Collection

- **Action:** Gather currencies, operation types, and ID ranges.
- **Details:**
    - Currencies: DKK, NOK, SEK, etc. (12 total).
    - Types: MT300, MT202, BMC (3 total).
    - IDs: MT300 (MOVE 15302-15313, SEND 15314-15325), etc.
- **Copilot:** "List currencies and types with ID ranges."

Step 2: Logic Planning

- **Action:** Design the Python script structure.
- **Details:**
    - Use lists for currencies and dictionaries for types.
    - Implement loops for dynamic ID generation.
    - Add `date_now` for timestamps.
- **Copilot:** "Generate a script for currency/type-based JSONs."

Step 3: Script Development

In [ ]:

```
# Initialize data structures
import json
from datetime import datetime, timedelta

currencys = ["DKK", "NOK", "SEK", "AUD", "CAD", "SGD", "NZD", "CHF", "JPY", "MXN", "ZAR", "CNY"]
tipos = [
    {"name": "MT300", "move_start": 15302, "send_start": 15314},
    {"name": "MT202", "move_start": 15326, "send_start": 15338},
    {"name": "BMC", "move_start": 15350, "send_start": 15362}
]
parent_id = 5199
system_resource_id = 5719
```

In [ ]:

```
# Define timestamp function
def date_now(offset=0):
    dt = datetime.now() + timedelta(minutes=offset)
    return f"{(int(dt.timestamp()) * 1000) - 3900}"

# Loop through types and currencies
for tipo in tipos:
    for idx, currency in enumerate(currencys):
```

```
        move_id = tipo["move_start"] + idx
        send_id = tipo["send_start"] + idx
```

In [ ]:

```
# Generate and save JSON files
file_pattern = f"bmlnyc07{currency}{tipo['name'].lower()}*.txt"
move_json = {
    "ProcessId": move_id,
    "Description": f"Feed to ESI - CashPro - {currency} - {tipo['name']} Move",
    "DateLastUpdate": date_now(move_id)
}
with open(f"Process_{move_id}.json", "w", encoding="utf-8") as f:
    json.dump(move_json, f, indent=2)

send_json = {
    "ProcessId": send_id,
    "Description": f"Feed to ESI - CashPro - {currency} - {tipo['name']} Send",
    "DateLastUpdate": date_now(send_id)
}
with open(f"Process_{send_id}.json", "w", encoding="utf-8") as f:
    json.dump(send_json, f, indent=2)
```

- **Copilot Use:** Suggested loops and `%CURRENCY%` replacement; optimized JSON formatting.

Step 4: Validation & Testing

- **Action:** Test and validate generated files.
- **Details:**
    - Run in a controlled environment.
    - Verify 72 files (12 currencies x 3 types x 2 processes).
    - Check ID consistency and currency substitution.
- **Copilot:** "Validate JSON generation for each currency/type."

Step 5: Integration & Scalability

- **Action:** Integrate into workflow and plan expansions.
- **Details:**
    - Save in Git for version control.
    - Add support for new currencies/types by updating lists.
    - Document in an internal manual.
- **Copilot:** "Suggest scaling for new currencies/types."

2.3 Observed Benefits

- **Automation:** 72 JSONs in under 1 minute, eliminating errors.
- **Learning:** New developer gained hands-on EMIS skills with Copilot.
- **Scalability:** Script adaptable with minimal adjustments.

Conclusion

- **Part 1:** Research identified onboarding and task challenges, guiding Copilot use.
- **Part 2:** Automation reduced errors and sped up processes.
- **Impact:** Copilot enhances learning and productivity; continuous use recommended.