Project Report: HappyFox Backend Rule Processing System

1. Introduction

The Rule Processing System simulates Gmail-like email processing logic. It enables users to define custom rules (e.g., match by sender, subject, or date), and applies actions such as marking emails as read/unread or moving them to specific labels. It integrates with the Gmail API and uses an SQLite database to store and manage messages locally.

This project was implemented as part of a **pre-placement assignment** and includes a full backend pipeline along with unit and integration tests.

---

2. Project Architecture

```
HappyFox_Backend_task/
├── src/
│   ├── process_rules.py         # Core rule evaluation and Gmail API
integration
│   ├── fetch_and_store.py       # Gmail email fetching and local DB storage
│   └── inspect_db_mail.py       # Inspects processed/labelled emails in local
DB
│
├── tests/
│   ├── test_predicates.py       # Unit tests for predicate functions
│   ├── test_evaluate_rule.py    # Unit tests for evaluating rule structures
│   └── test_integration.py      # In-memory DB-based integration testing
│
├── config/
│   └── credentials.json         # OAuth2 credentials for Gmail API
│
├── emails.db                    # SQLite DB with fetched Gmail emails
├── rules.json                   # User-defined ruleset for email processing
├── pytest.ini                   # Pytest config file
└── .venv/                       # Virtual environment
```

---

3. Script Descriptions

3.1 `fetch_and_store.py`

- **Purpose**: Authenticates with Gmail and fetches recent emails.
- **Functionality**:
- Uses Gmail API to retrieve emails
- Extracts metadata such as sender, subject, date, snippet
- Stores all email data into a local SQLite database (`emails.db`)
- **Usage**:

```
python src/fetch_and_store.py
```

3.2 `process_rules.py`

- **Purpose**: Core logic for evaluating emails against defined rules and applying actions.
- **Key Responsibilities**:
- Loads `rules.json` which contains matching criteria and actions
- Applies rules using ALL/ANY logic
- Supports predicates like `contains`, `equals`, `less than`, etc.
- Calls Gmail API to mark emails read/unread or move to label
- Marks emails as `processed` in the database
- **Usage**:

```
python src/process_rules.py --rules rules.json
```

3.3 `inspect_db_mail.py`

- **Purpose**: Post-processing utility to inspect the local SQLite database
- **Functionality**:
- Displays emails that were processed
- Shows applied labels and timestamps
- **Usage**:

```
python src/inspect_db_mail.py
```

---

4. Rules & Configuration

`rules.json`

Defines how emails should be matched and what actions to perform. Example:

```
{
  "predicate": "any",
  "rules": [
    {"field": "From", "predicate": "contains", "value": "happyfox.com"},
  ],
  "actions": {
    "mark_as_read": true,
    "move_to": "Important"
  }
}
```

---

5. Testing

5.1 `test_predicates.py`

- Tests all predicate functions like `contains`, `equals`, `less than`, `greater than`
- Uses deterministic datetime values for reliability

5.2 `test_evaluate_rule.py`

- Unit tests for verifying rule evaluation on email objects
- Ensures correct logic for each rule structure

5.3 `test_integration.py`

- In-memory SQLite database integration
- Simulates full processing without touching actual Gmail API or DB file
- Verifies emails are correctly marked/moved based on rules

Run all tests using:

```
pytest
```

---

6. Challenges Faced

6.1 Date-Based Logic Ambiguity

- **Problem**: Initial confusion in interpreting rules like "less than 2 days"
- **Resolution**: Clarified that this means "received within the last 2 days"

6.2 Non-Deterministic Time in Tests

- **Problem**: Use of `datetime.now()` caused tests to fail randomly
- **Resolution**: Refactored predicates to accept a `now` argument for deterministic testing

6.3 Gmail API Access in Tests

- **Problem**: Integration tests should not make real API calls
- **Resolution**: Tests use in-memory DB and skip Gmail API calls completely

6.4 Rule Flexibility

- **Problem**: Handling various fields and predicates dynamically
- **Resolution**: Used a predicate function map and robust error handling

---

7. Conclusion

This project successfully implements the backend for Gmail-style rule processing with robust testing and Gmail API integration. The modular structure allows for easy extension, and the deterministic testing ensures reliability in production.