**RTR Payment Simulation Project Documentation**Rouvin Rebello

**GitHub:** <https://github.com/Rouvin-rebello/RTR_Payment_Simulation>

**Project Description:**  
This project simulates a complete end-to-end Real-Time Rail (RTR) payment ecosystem that adheres to ISO 20022 financial messaging standards. It is designed for testing, analysis, and demonstration of interbank credit transfer flows in a controlled environment.

**Objective:**  
To build a modular simulation framework that replicates the real-time payment lifecycle, spanning customer initiation, interbank processing, settlement, and analytics, enabling testing, validation, and educational use of ISO 20022 payment standards.

**Core Modules & Functionality:**

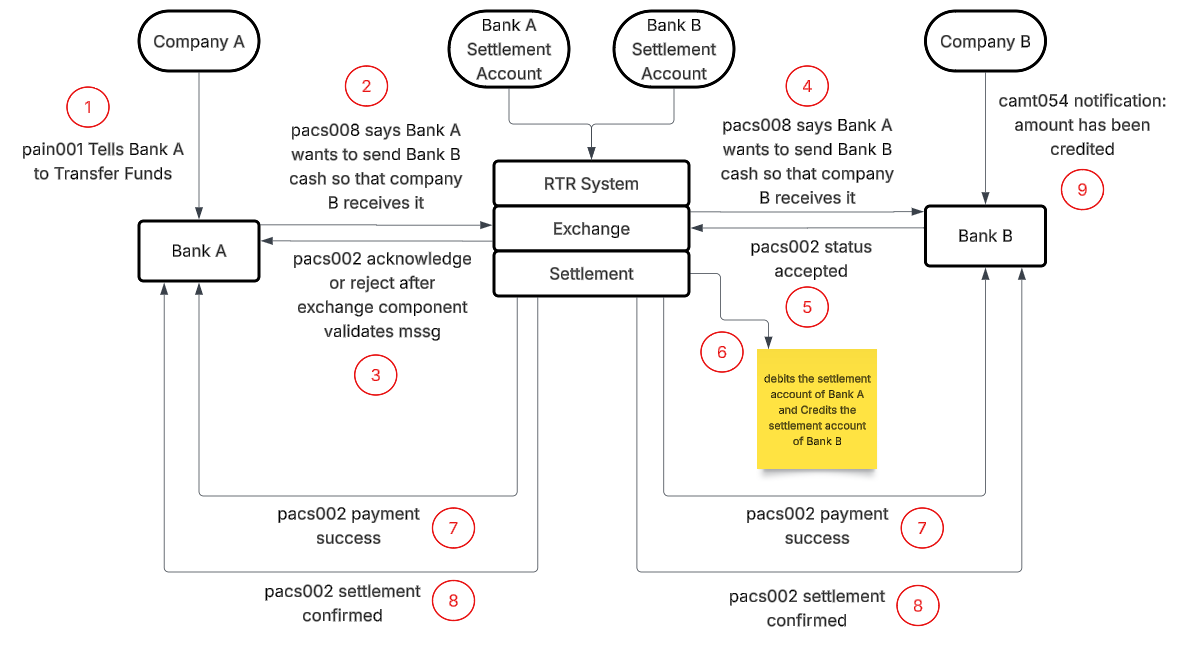
1. **Payment Simulation Components:**
   * **Agent\_Debtor\_Simulator.py:** Simulates the originating bank by parsing PAIN.001 messages and generating PACS.008 interbank credit transfer messages.
   * **Agent\_Creditor\_Simulator.py:** Acts as the receiver bank, processing PACS.008 messages and generating acknowledgement (PACS.002) and credit notification (CAMT.054) messages.
2. **Core Payment Engine:**
   * **RTR\_Exchange\_Processor.py:** Central orchestrator that routes, validates, and settles payments. Collaborates with settlement and message generation modules to simulate the full lifecycle of a payment.
   * **RTR\_Settlement\_Processor.py:** Ensures atomicity, consistency, and security in transactional processing by validating users, checking balances, and updating accounts using exclusive locks.
3. **ISO 20022 Message Generators:**
   * **ISO20022\_Pain001\_Generator.py** – Constructs customer-initiated payment instructions.
   * **ISO20022\_Pacs008\_Generator.py** – Builds and logs interbank transfer messages.
   * **ISO20022\_Pacs002\_Generator.py** – Creates acknowledgment messages (accepted/rejected).
   * **ISO20022\_Camt054\_Generator.py** – Generates credit notifications upon settlement.
4. **Database Management:**
   * **payment\_system.db** – SQLite database that stores user credentials, BIC codes, and transaction records.
   * **db\_setup.py**, **db\_manager.py**, **db\_check\_records.py** – Handle database initialization, reset, and inspection.
5. **Data Processing & Analysis:**
   * **Analytics\_ETL.py** – Extracts, transforms, and loads raw log data into structured formats (JSON/CSV).
   * **Analytics\_analyze\_transactions.py** – Performs statistical analysis, trend detection, and anomaly identification using Matplotlib and Seaborn.
   * **Transaction Analysis Workbook.twb** – Provides interactive dashboards and visual reporting tools for stakeholders.
6. **User Interface:**
   * **payment\_interface.py** – A GUI-based frontend allowing users to log in, initiate payments, and track the 16-stage payment lifecycle with integrated analytics triggers.
7. **Logging & Auditing:**
   * **settlement log.txt** – Captures chronological logs for every key action, message creation, and transaction step for auditability and debugging.

**Key Use Cases:**

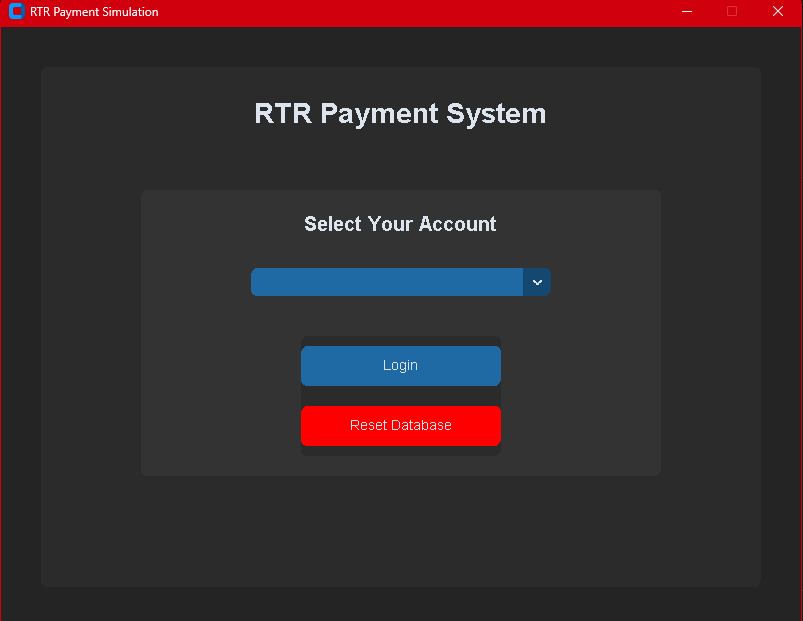
* Educational simulations for financial systems and ISO 20022 training.
* End-to-end functional and integration testing for payment infrastructure.
* Benchmarking transaction processing under varying conditions.
* Data analytics and anomaly detection for operational insights.

**Impact:**  
This simulation platform enables financial institutions, developers, and analysts to explore and evaluate ISO 20022-compliant real-time payment flows in a fully offline, testable, and extensible environment. It also provides a strong foundation for expanding into live system integrations, compliance testing, and research on payment behavior and anomalies.

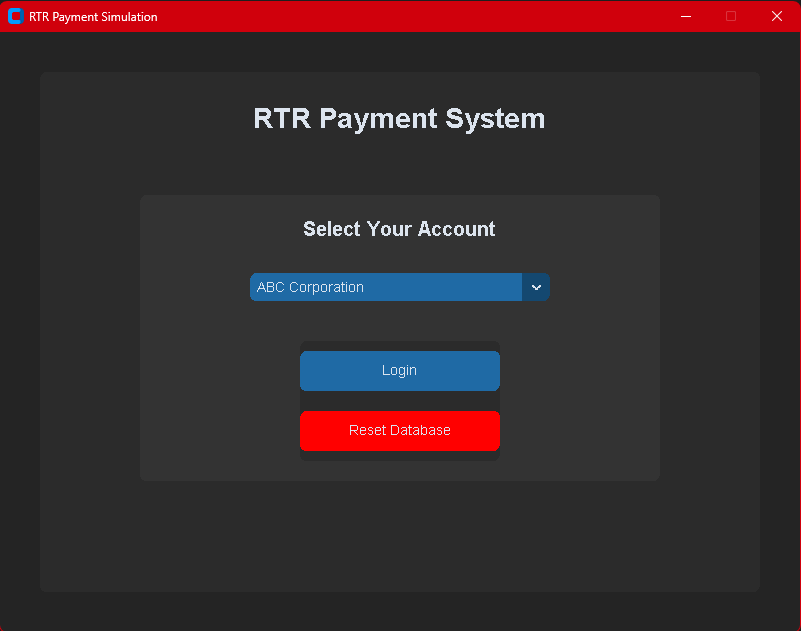
**Push Transaction with pacs008**

****

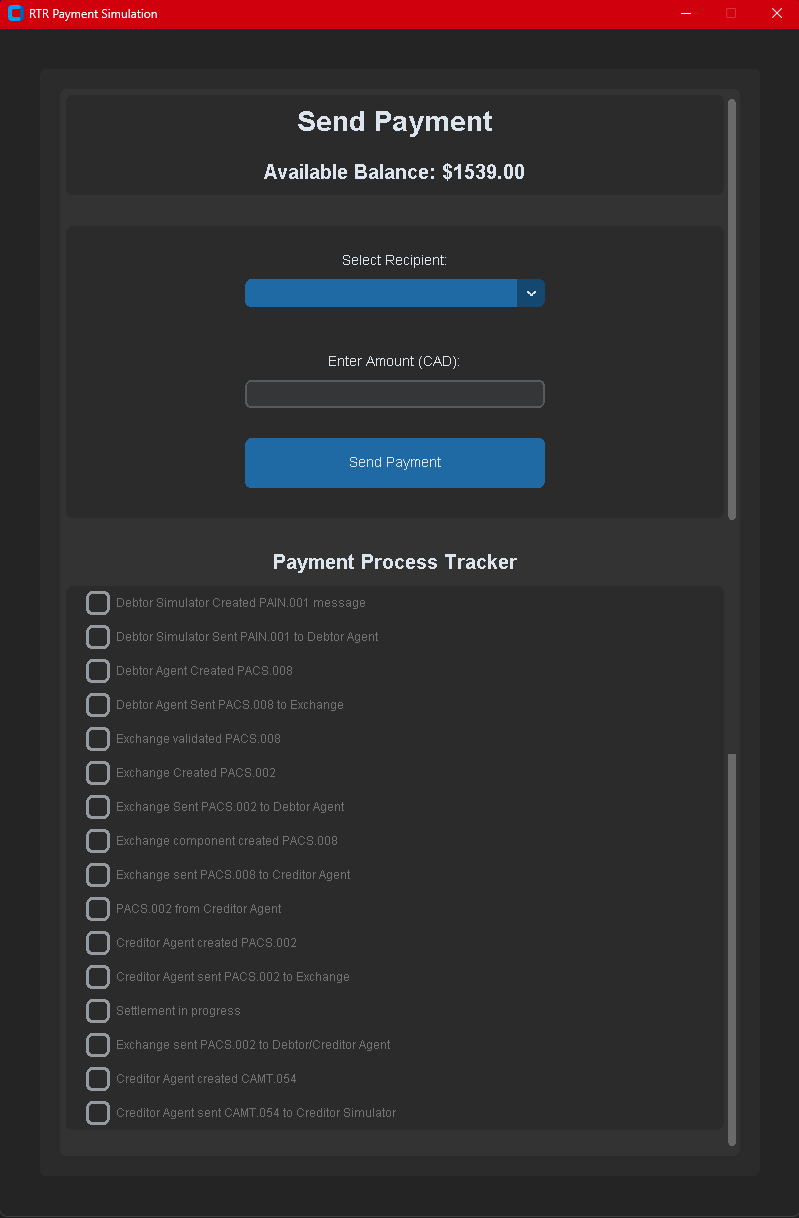
**Project Demonstration**

****

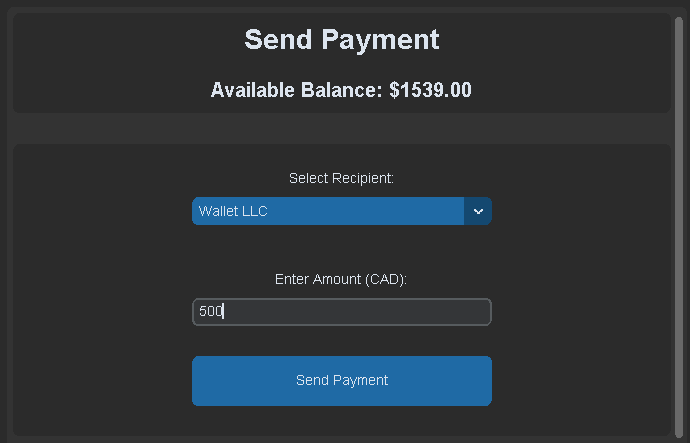
A user (Company) is prompted with a simple login page



After logging in, users can send an amount to another user:

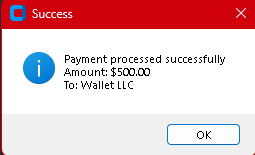


In this case, ABC Corporation will send Wallet LLC $500

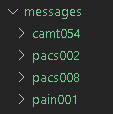
****

After clicking the “Send Payment” button, the following steps are executed, followed by a confirmation message:

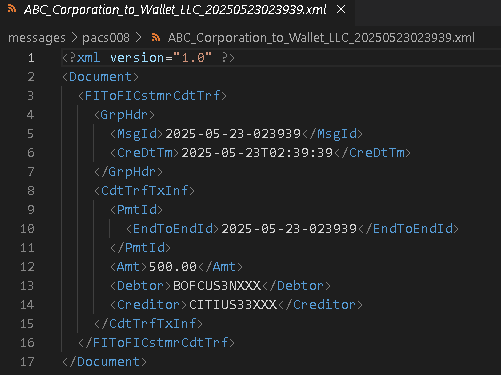
* Debtor Simulator Created PAIN.001 message
* Debtor Simulator Sent PAIN.001 to Debtor Agent
* Debtor Agent Created PACS.008
* Debtor Agent Sent PACS.008 to Exchange
* Exchange validated PACS.008
* Exchange Created PACS.002
* Exchange Sent PACS.002 to Debtor Agent
* Exchange component created PACS.008
* Exchange sent PACS.008 to Creditor Agent
* PACS.002 from Creditor Agent
* Creditor Agent created PACS.002
* Creditor Agent sent PACS.002 to Exchange
* Settlement in progress
* Exchange sent PACS.002 to Debtor/Creditor Agent
* Creditor Agent created CAMT.054
* Creditor Agent sent CAMT.054 to Creditor Simulator



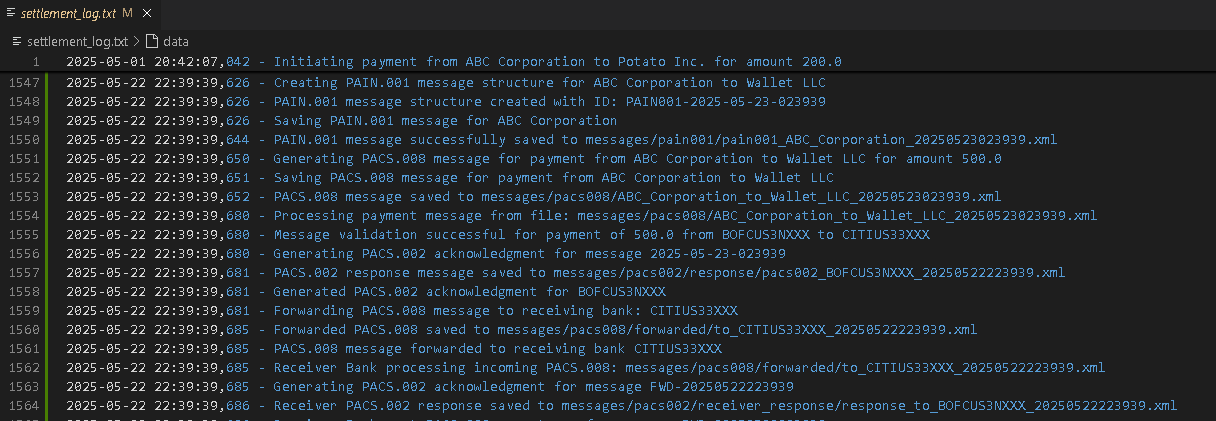
The respective messages can be viewed in the messages folder

****

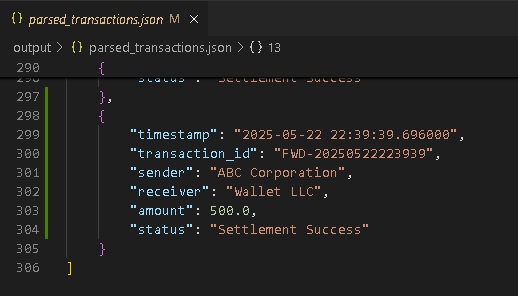
Message Structure Example:

****

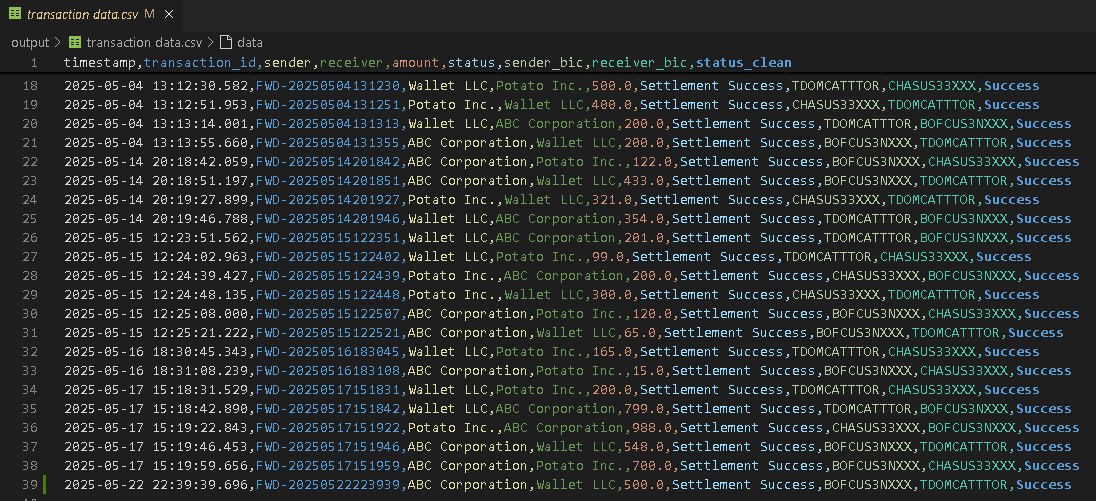
All transactions, messages and steps are logged in settlement\_log.txt

****

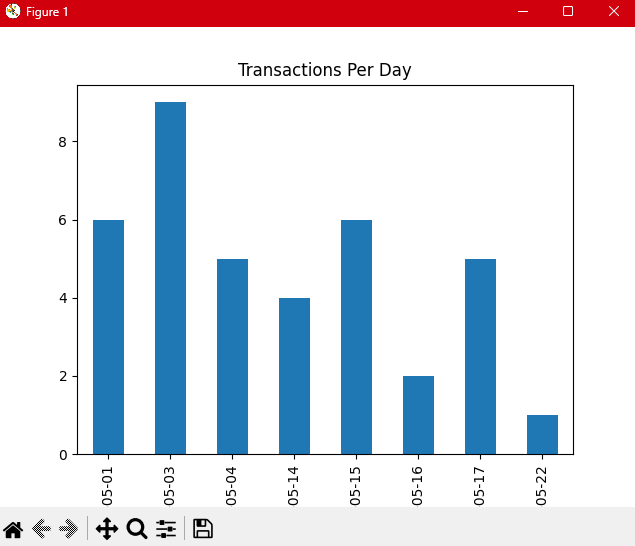
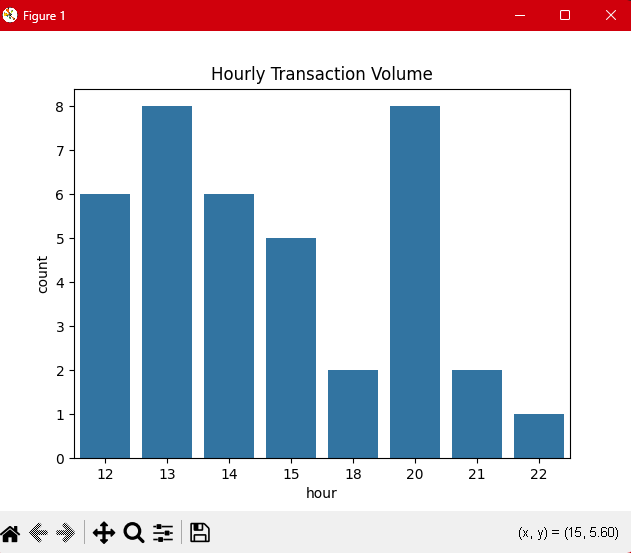
This file is parsed, and relevant information is stored in parsed\_transactions.json

****

parsed\_transactions.json is then converted to a CSV file called transaction data.csv



The Data from transaction\_data.csv is analyzed with the help of Python Pandas and Tableau to provide insights (some of which are below)



**Project Structure**

Folders:

* Messages
  + camt054 – Stores all camt messages generated
  + pacs002 – Stores all pacs002 messages generated
  + pacs008 – Stores all pacs008 messages generated
  + pain001 – Stores all pain001 messages generated
* output
  + parsed\_transactions.json – Stores information parsed from settlement\_log.txt
  + transaction data.csv – Stored data from parsed\_transactions.json as .csv

Files:

|  |  |
| --- | --- |
| **File Name** | **Description** |
| Agent Creditor\_Simulator.py | Simulates the receiving bank by processing incoming PACS.008 messages and generating PACS.002 and CAMT.054. |
| Agent Debtor\_Simulator.py | Simulates the sending bank by converting PAIN.001 messages into interbank PACS.008 transfers. |
| Analytics\_analyze\_transactions.py | Analyzes RTR transaction logs to identify trends, performance metrics, and anomalies. |
| Analytics\_ETL.py | Extracts, transforms, and loads raw settlement logs into structured datasets for analysis. |
| db\_check\_records.py | Retrieves and displays records from key tables in the RTR simulation database for debugging and validation. |
| db\_manager.py | Initializes or resets the RTR simulation database with predefined schema and data. |
| db\_setup.py | Sets up the RTR database by recreating tables and inserting sample users and institutions. |
| ISO20022\_Camt054\_Generator.py | Generates and saves CAMT.054 credit notification messages in ISO 20022 format. |
| ISO20022\_Pacs002\_Generator.py | Creates and stores PACS.002 acknowledgment messages for payment instructions. |
| ISO20022\_Pacs008\_Generator.py | Builds and transmits PACS.008 interbank payment messages from user and BIC data. |
| ISO20022\_Pain001\_Generator.py | Constructs and stores PAIN.001 messages to simulate customer-initiated credit transfers. |
| payment\_interface.py | Provides a GUI for initiating, tracking, and analyzing RTR payments using ISO 20022 messages. |
| payment\_system.db | SQLite database that stores users, BIC codes, and transaction history for RTR simulation. |
| RTR\_Exchange Processor.py | Orchestrates the end-to-end RTR payment flow including validation, routing, and acknowledgment. |
| RTR Settlement\_Processor.py | Executes core settlement logic, balance verification, and transaction logging in RTR. |
| settlement log.txt | Text-based audit log capturing each step of the RTR payment and settlement lifecycle. |
| Transaction Analysis Workbook.twb | Interactive dashboard for analyzing transaction trends, participant performance, and SLA compliance. |

**Limitations**

* The settlement process does not take place through participant settlement accounts held with the RTR system. Instead, the amount owing is deducted from the debtor’s account and is added to the creditor's account
* Admi messages have not been integrated

**Additional details**

|  |  |
| --- | --- |
| **File Name** | **Description** |
| Agent Creditor\_Simulator.py | **Purpose:** Simulates the receiver-side bank (agent creditor) in the RTR payment system, responsible for acknowledging and finalizing received payments.  **Key Functionalities:**   * **PACS.008 Processing:**   + Parses incoming ISO 20022 PACS.008 payment messages.   + Extracts metadata such as message ID and debtor details.   + Generates and stores a PACS.002 (ACCP status) acceptance message. * **Settlement Completion:**   + Generates CAMT.054 credit notification messages post-settlement.   + Encodes and saves creditor-side transaction details in XML format. * **File Handling & Logging:**   + Organizes messages in timestamped directories (pacs002/receiver\_response, camt054/).   + Logs key operations and errors for traceability.   **Use Case:** Enables end-to-end testing of ISO 20022 message flow from receipt to settlement in the RTR system. |
| Agent Debtor\_Simulator.py | **Purpose:** Simulates the sending bank (agent debtor) handling customer-initiated payments.  **Key Functionalities:**   * Parses ISO 20022 PAIN.001 messages for credit transfer initiation. * Extracts payer/payee details and transaction amount. * Constructs PACS.008 interbank transfer messages. * Saves outbound PACS.008 messages to simulate interbank instruction.   **Use Case:** Bridges the customer-to-bank initiation flow with interbank processing in the RTR ecosystem. |
| Analytics\_analyze\_transactions.py | **Purpose:** Analyzes RTR transaction logs to uncover trends, performance insights, and anomalies.  **Key Functionalities:**   1. **Trend & Volume Analysis**    * Daily/hourly volume, totals, averages. 2. **Status Breakdown**    * Success vs. failure rates. 3. **Top Participants**    * Sender/receiver rankings by volume and amount. 4. **Net Flow Analysis**    * Sent vs. received balances. 5. **Behavioral Patterns**    * Recurring pairs, transaction gaps. 6. **Anomaly Detection**    * Outliers and off-hour activity. 7. **Exploratory Insights**    * Identifies busiest days, top failures, etc.   **Outputs:** Console reports and Seaborn/Matplotlib visualizations. |
| Analytics\_ETL.py | **Purpose:** Performs ETL operations on raw settlement logs, preparing structured datasets for analysis.  **Key Functionalities:**   * **Extract:** Parses settlement\_log.txt for timestamp, sender, receiver, amount, and status. * **Transform:** Adds BIC codes, cleans statuses, standardizes timestamps. * **Load:** Saves output as JSON and CSV (output/parsed\_transactions.json, transaction data.csv).   **Key Components:**   * parse\_log\_file(), enrich\_transaction(), etl\_pipeline(), run\_etl() |
| db\_check\_records.py | **Purpose:** Utility for inspecting key tables in the payment\_system.db SQLite database.  **Key Functionalities:**   * Connects to database and retrieves records from:   + bic\_codes (financial institutions)   + users (participants) * Prints records for manual verification.   **Use Case:** Debugging and data validation during development. |
| db\_manager.py | **Purpose:** Provides functionality to initialize or reset the RTR simulation database.  **Key Functionalities:**   * **init\_db():** Creates bic\_codes, users, and transactions tables with schema constraints and default entries. * **reset\_db():** Clears and reinitializes the database for clean test runs. |
| db\_setup.py | **Purpose:** Initial setup script for payment\_system.db, used in testing and simulations.  **Key Functionalities:**   * Drops existing tables and recreates:   + bic\_codes, users, payments * Inserts sample financial institutions and users. * Verifies successful data insertion via console output. |
| ISO20022\_Camt054\_Generator.py | **Purpose:** Generates ISO 20022 CAMT.054 XML credit notification messages for settled transactions.  **Key Functionalities:**   * generate\_camt054\_message(): Builds XML with message metadata. * save\_camt054\_message(): Stores formatted XML in messages/camt054/.   **Use Case:** Simulates account credit notifications to receiving institutions. |
| ISO20022\_Pacs002\_Generator.py | **Purpose:** Creates ISO 20022 PACS.002 XML acknowledgment messages for payment instructions.  **Key Functionalities:**   * generate\_pacs002\_message(): Builds response message (ACCP/RJCT). * save\_pacs002\_message(): Saves output in messages/pacs002/.   **Use Case:** Enables simulation of payment acceptance or rejection acknowledgment. |
| ISO20022\_Pacs008\_Generator.py | Purpose: Generates ISO 20022 PACS.008 messages and forwards them through the RTR exchange.  **Key Functionalities:**   * Accesses user/BIC data from payment\_system.db. * Builds, logs, and saves PACS.008 XML messages. * Sends messages to RTR processor via process\_through\_rtr().   **Use Case:** Simulates the interbank credit transfer flow. |
| ISO20022\_Pain001\_Generator.py | **Purpose:** Creates ISO 20022 PAIN.001 XML messages for initiating customer credit transfers.  **Key Functionalities:**   * generate\_pain001\_message(): Constructs single transaction request. * save\_pain001\_message(): Stores readable XML in messages/pain001/.   **Output:** Well-formed XML files that serve as input to debtor agents. |
| payment\_interface.py | **Purpose:** Provides a graphical user interface (GUI) for interacting with the RTR payment simulator.  **Key Functionalities:**   * **User Interaction:**   + Login, initiate payments, specify amount. * **Message Processing Flow:**   + Generates PAIN.001 → PACS.008 → PACS.002/CAMT.054. * **Visualization:**   + Displays a 16-stage payment tracker. * **Database & Analytics Integration:**   + Triggers ETL post-transaction and logs activity.   **Use Case:** Enables user-friendly simulations of ISO 20022-compliant payment flows. |
| payment\_system.db | **Purpose:** Primary SQLite database for the RTR simulation system.  **Key Responsibilities:**   * Stores user credentials and metadata. * Tracks all transactions and message logs. * Links payments to financial institutions via BIC codes.   **Use Case:** Persistent storage for payment lifecycle simulation. |
| RTR\_Exchange Processor.py | **Purpose:** Serves as the central orchestrator in the Real-Time Rail (RTR) simulation environment. It manages the full lifecycle of a real-time payment message—covering validation, acknowledgment, forwarding, and settlement—according to ISO 20022 standards.  **Key Responsibilities:**   * **Process Incoming Payments:** Parses PACS.008 XML messages. * **Validation:** Verifies mandatory fields such as debtor, creditor, amount, and message ID. * **Acknowledgment:** Generates PACS.002 messages to confirm receipt. * **Message Forwarding:** Directs the PACS.008 message to the appropriate receiving bank. * **Receiver Simulation:** Waits for a simulated PACS.002 response from the receiver bank. * **Routing & Settlement:**   + Routes payments based on BIC codes.   + Settles transactions via the RTRSettlementProcessor. * **Notification:** Sends settlement confirmations (PACS.002) to both sender and receiver; notifies receiver bank of settlement completion. * **Error Handling:** Logs issues and generates rejection messages for invalid or incomplete inputs.   **Collaborates With:**   * RTRSettlementProcessor: Manages balance and settlement logic. * ISO20022\_Pacs002\_Generator: Generates acknowledgment messages. * Agent\_Creditor\_Simulator: Simulates receiver bank behavior.   **Use Case:** When run as a standalone script, processes a sample payment\_request.xml to simulate a full RTR payment flow in a test environment. |
| RTR Settlement\_Processor.py | **Purpose:** Implements the core settlement logic for the Real-Time Rail (RTR) system, ensuring transactional integrity, data consistency, and auditability throughout the payment lifecycle.  **Key Responsibilities:**   * **Participant Validation:** Confirms debtor and creditor BIC codes match valid users. * **Balance Verification:** Checks for sufficient debtor funds before and during processing. * **Transactional Integrity:** Uses exclusive locking to avoid race conditions during balance updates. * **Atomic Balance Updates:** Adjusts sender and receiver account balances within a secure transaction. * **Payment Recording:** Logs each successful transaction with a timestamp. * **Error Handling:** Rolls back changes in the event of processing errors. * **Logging:** Tracks transaction details and exceptions for audit and debugging.   **Dependencies:**   * sqlite3: Manages user and payment data. * logging: Captures operational events and errors. * datetime: Generates timestamps for transaction logs.   **🧩 Role in System:** Ensures that payments are settled accurately, securely, and consistently, forming the backbone of trust in the RTR system. |
| settlement log.txt | **Purpose:** Provides a chronological audit trail of payment processing and settlement activities in the RTR simulation environment. Used for operational monitoring, debugging, and compliance verification.  **Key Activities Captured:**   * **Payment Initiation:** Tracks new payment requests between entities (e.g., ABC Corp → Potato Inc.). * **Message Generation:** Records creation of PAIN.001, PACS.008, and PACS.002 messages. * **Persistence:** Logs file paths of saved XML messages. * **Validation & Routing:** Confirms routing and message validation steps. * **Settlement:** Captures balance updates and transaction completion between banks. * **Notifications:** Logs dispatch of PACS.002 acknowledgments and CAMT.054 credit notifications.   **Use Case:** Enables developers, testers, and operations teams to trace payment flows and identify anomalies. Supports system audits, root-cause analysis, and SLA verification. |
| Transaction Analysis Workbook.twb | **Purpose:** Provides visual analytics and reporting on transaction data within the RTR system. Enables stakeholders to explore trends, monitor performance, and detect anomalies in real-time and historical data.  **Key Features:**   * **Volume & Value Analysis:** Displays transaction counts and values over time, segmented by participant or type. * **Timeliness Monitoring:** Highlights processing delays and SLA breaches. * **Participant Performance:** Compares throughput and success rates of financial institutions. * **Anomaly Detection:** Visual cues for identifying irregularities or outliers. * **Interactive Filters:** Allows users to explore data by date, participant, or message type.   **Use Case:** Used by business analysts, compliance officers, and operational teams to derive insights and ensure adherence to RTR standards. Enhances visibility and supports data-driven decision-making. |