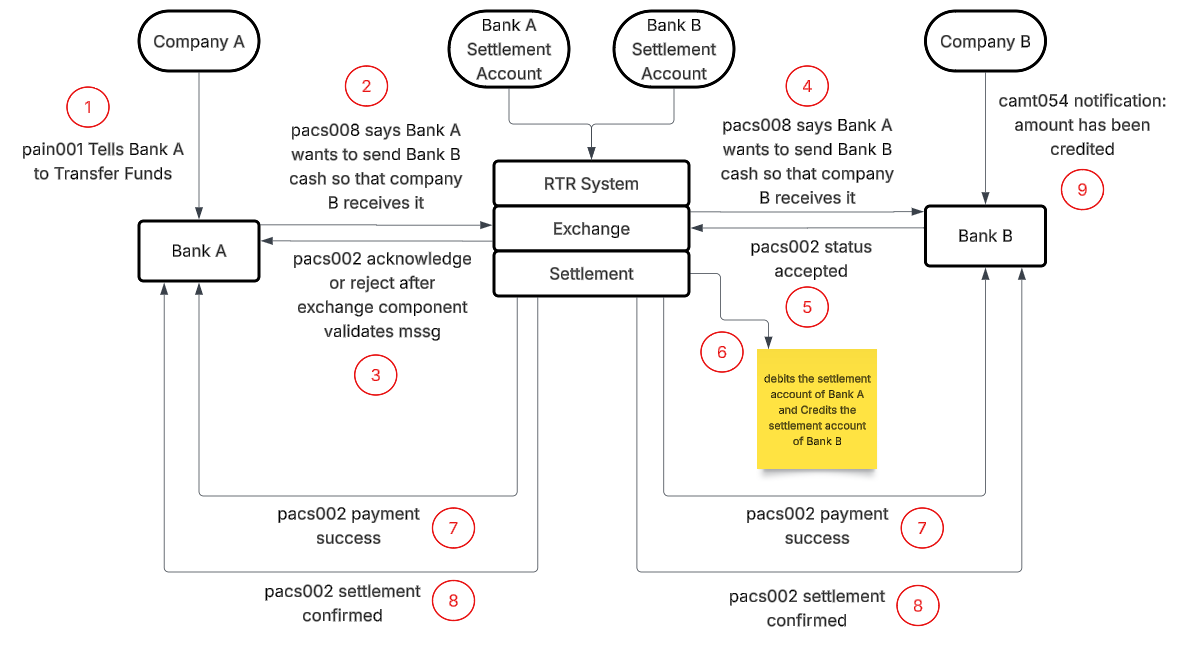
**RTR Payment Simulation Project Documentation**

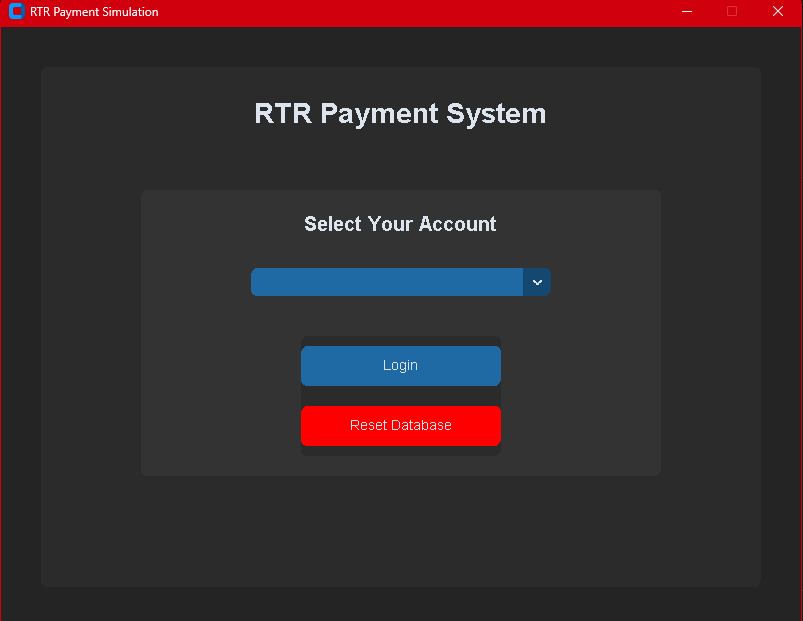
**GitHub:** <https://github.com/Rouvin-rebello/RTR_Payment_Simulation>

**Project Overview**

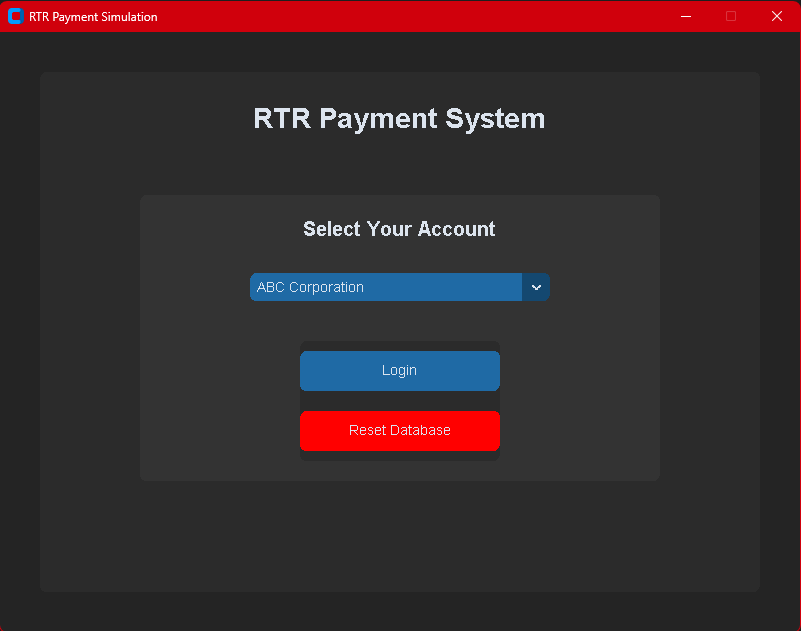
**Push Transaction with pacs008**

****

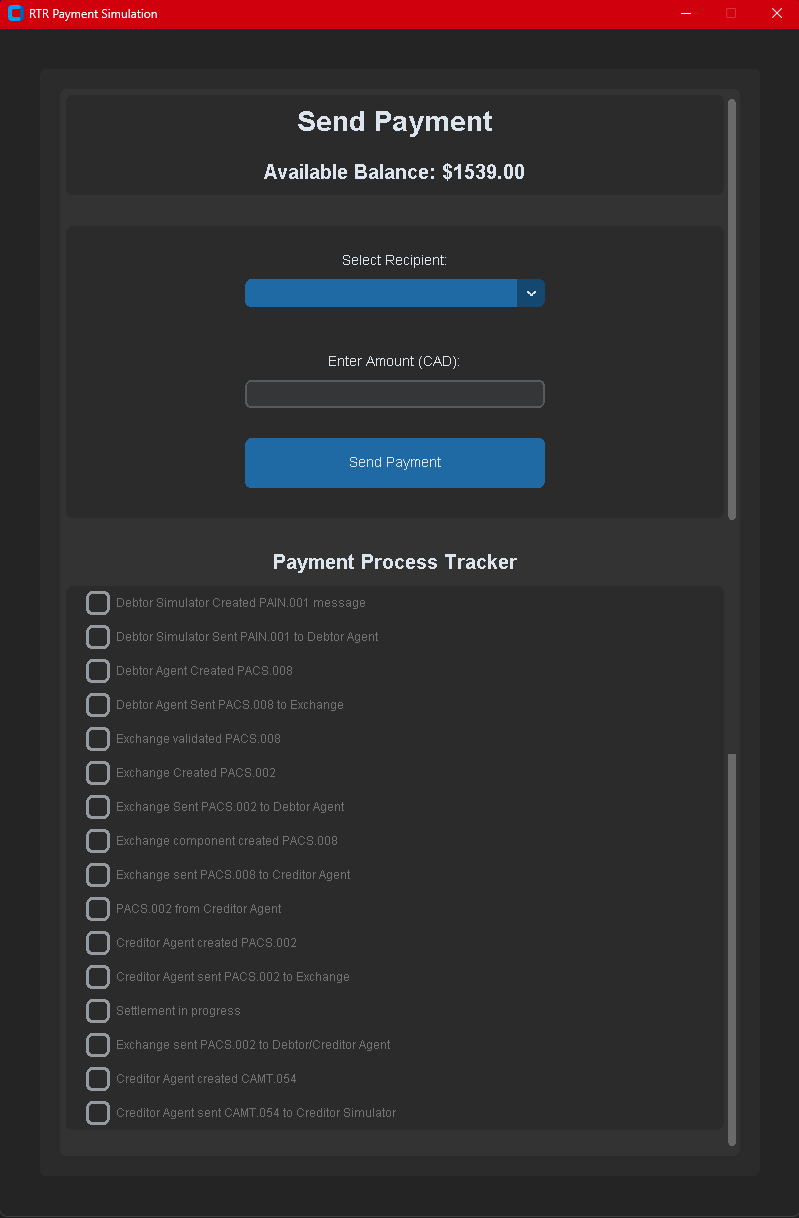
**Project Demonstration:**

****

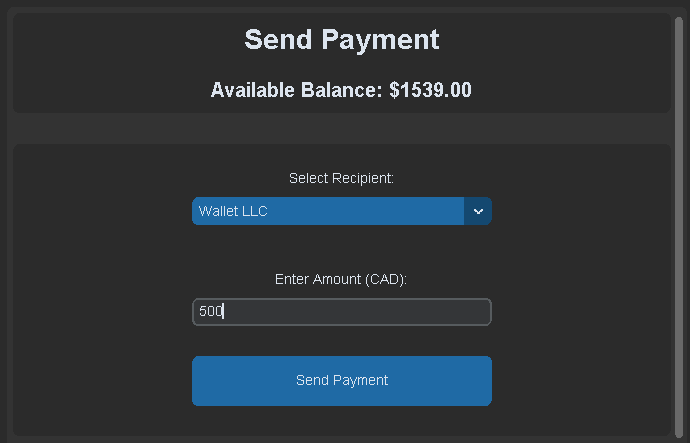
A user (Company) is prompted with a simple login page



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

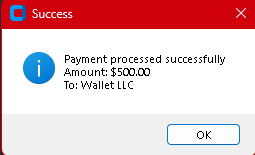


In this case, ABC Corporation will send Wallet LLC $500

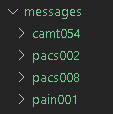
****

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

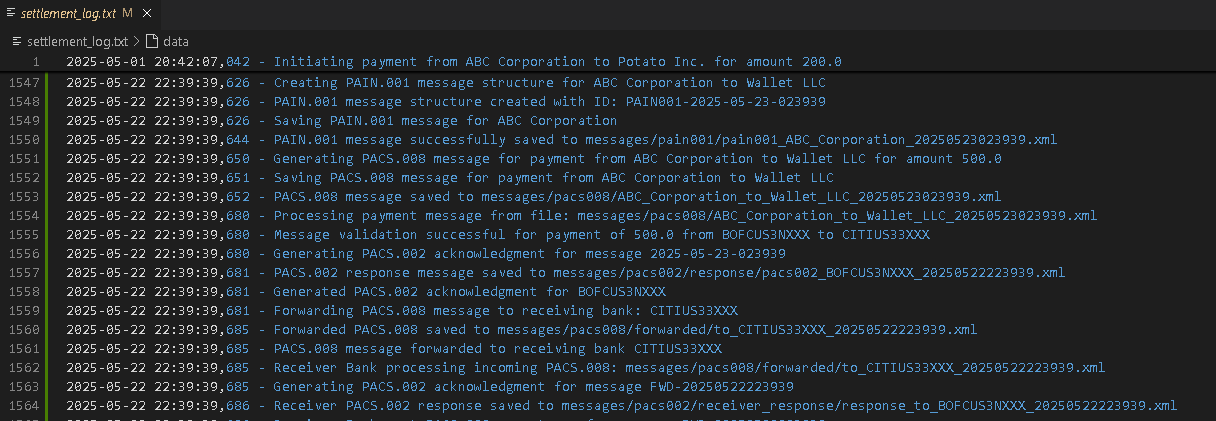
* Debtor Simulator Created PAIN.001 message
* Debtor Simulator Sent PAIN.001 to Debtor Agent
* Debtor Agent Created PACS.008
* Debtor Agent Sent PACS.008 to Exchange
* Exchange validated PACS.008
* Exchange Created PACS.002
* Exchange Sent PACS.002 to Debtor Agent
* Exchange component created PACS.008
* 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



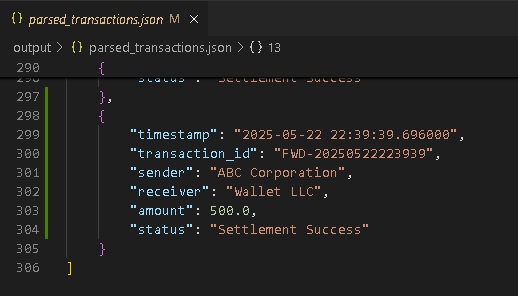
The respective messages can be viewed in the messages folder

****

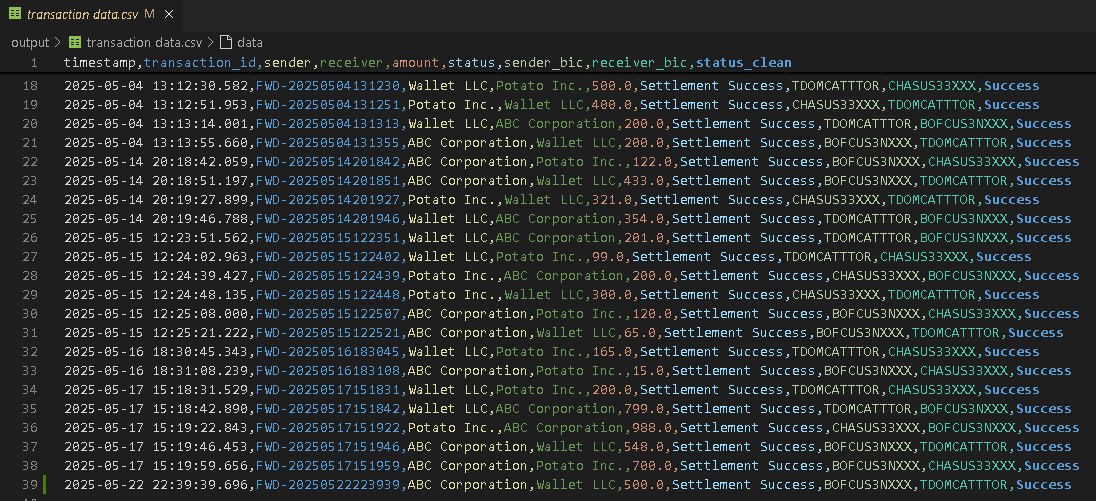
All transactions, messages and steps are logged in settlement\_log.txt

****

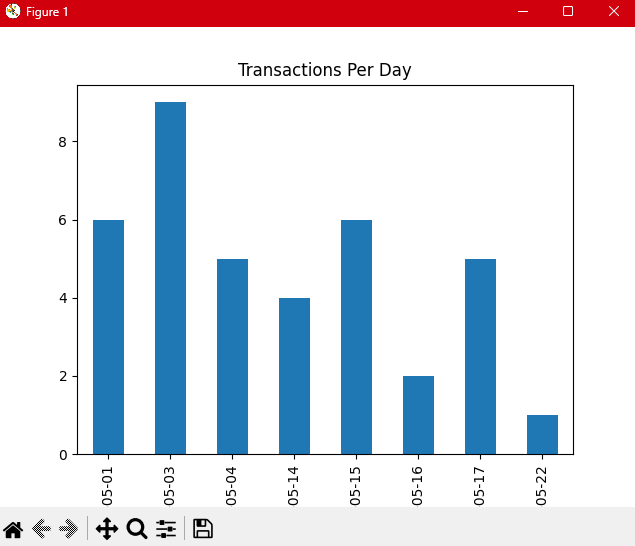
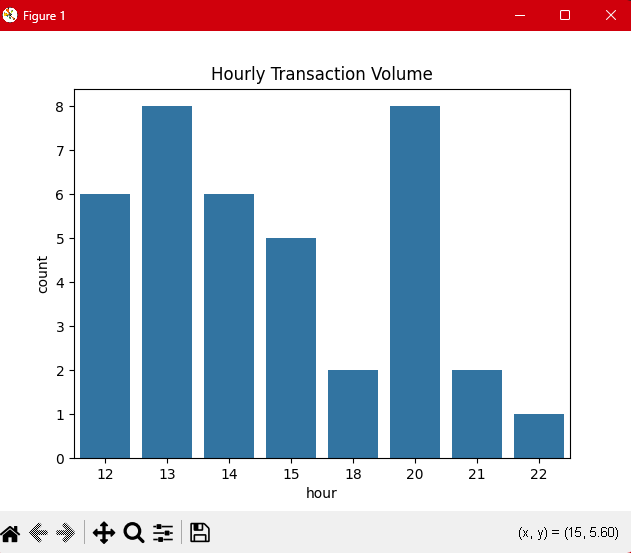
This file is parsed, and relevant information is stored in parsed\_transactions.json

****

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
  + pacs002
  + pacs008
  + pain001
* output
  + parsed\_transactions.json
  + transaction data.csv

Files:

|  |  |
| --- | --- |
| File Name | Description |
| Agent Creditor\_Simulator.py |  |
| Agent Debtor\_Simulator.py |  |
| Analytics\_analyze\_transactions.py |  |
| Analytics\_ETL.py |  |
| db\_check\_records.py |  |
| db\_manager.py |  |
| db\_setup.py |  |
| ISO20022\_Camt054\_Generator.py |  |
| ISO20022\_Pacs002\_Generator.py |  |
| ISO20022\_Pacs008\_Generator.py |  |
| ISO20022\_Pain001\_Generator.py |  |
| payment\_interface.py |  |
| payment\_system.db |  |
| RTR\_Exchange Processor.py |  |
| RTR Settlement\_Processor.py |  |
| settlement log.txt |  |
| Transaction Analysis Workbook.twb |  |

**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

**Additional details**

Agent\_Creditor\_Simulator.py

This script simulates the behavior of a **receiver bank** (agent creditor) in the RTR (Real-Time Rail) payment ecosystem. It handles the following key responsibilities:

#### 🔄 **Processing Incoming PACS.008 Payments**

* Accepts and parses incoming ISO 20022 PACS.008 payment messages.
* Extracts transaction metadata such as message ID and debtor information.
* Generates a corresponding PACS.002 acceptance message (ACCP status) indicating successful receipt of the payment.
* Saves this response as an XML file with proper formatting.

#### 💳 **Handling Settlement Completion**

* Upon completion of settlement, generates a CAMT.054 credit notification message for the creditor.
* Encodes transaction details such as amount, message ID, and creditor information.
* Saves the CAMT.054 message to a dedicated folder for audit and communication purposes.

#### 📁 **File Handling & Logging**

* Stores all generated messages in organized directories (pacs002/receiver\_response, camt054) with timestamped filenames.
* Implements logging to track the flow of events and capture errors during message processing.

This module acts as a mock receiver FI, enabling end-to-end simulation of message flows in payment processing and settlement scenarios compliant with ISO 20022 standards.

Agent\_Debtor\_Simulator.py

This module simulates the behavior of a **debtor-side financial institution (FI)** in processing incoming payment initiation messages (PAIN.001) as part of the RTR transaction flow. It performs the following core functions:

* **Parses a PAIN.001 message** (an XML-based ISO 20022 standard message for customer credit transfer initiation).
* **Extracts key payment details**, including debtor/creditor names, their respective financial institutions (BIC codes), and the transaction amount.
* **Constructs a PACS.008 message** using the extracted data. PACS.008 messages are used in interbank credit transfers and represent the instruction to move funds from the debtor to the creditor.
* **Saves the PACS.008 message to a file**, simulating the generation of an outbound message by the debtor FI.

This script acts as a bridge between the customer payment initiation (PAIN.001) and the interbank settlement layer (PACS.008), emulating how a sending FI would handle RTR transactions.

Analytics\_analyze\_transactions.py

This script performs a comprehensive analysis of transaction data generated by the RTR (Real-Time Rail) system. It is designed to uncover operational patterns, participant behaviors, performance trends, and anomalies in payment processing.

**Key Features:**

1. **Volume & Trend Analysis**
   * Analyzes daily and hourly transaction volumes.
   * Calculates total and average transaction amounts per day.
   * Visualizes transaction trends over time.
2. **Status-Based Analysis**
   * Breaks down transaction outcomes (success vs. failure).
   * Computes daily failure rates and identifies recurring failure messages.
3. **Top Participants & FI Performance**
   * Identifies top senders and receivers based on transaction volume.
   * Ranks financial institutions (by BIC) based on the amount sent.
4. **Net Flow Analysis**
   * Calculates net flow (sent vs. received) per entity.
   * Tracks daily net balance changes for participants.
5. **Correlation & Behavior Patterns**
   * Explores average transaction values by sender and status.
   * Detects recurring sender-receiver pairs and evaluates transaction timing gaps.
6. **Basic Anomaly Detection**
   * Flags unusually large or small transactions.
   * Highlights off-hour transactions and failure rate spikes.
7. **Exploratory Questions**
   * Answers key performance questions like busiest day, top FI, and failed transactions list.

**Output:**  
The script prints summary tables to the console and generates basic visualizations (hourly and daily transaction volume charts) using Seaborn and Matplotlib.

Analytics\_ETL.py

### **Analytics\_ETL.py – High-Level Overview**

The Analytics\_ETL.py script is responsible for extracting, transforming, and loading (ETL) transactional data from a raw log file (settlement\_log.txt) into a structured CSV format (transaction data.csv). This enables downstream analytics and dashboard generation for the RTR (Real-Time Rail) project.

#### ✅ Key Functionalities:

1. **Extract**:
   * Parses the raw settlement log file to extract key transaction details such as:
     + Timestamp
     + Sender
     + Receiver
     + Amount
     + Transaction ID
     + Settlement Status
2. **Transform**:
   * Enriches each transaction with synthetic BIC codes for senders and receivers.
   * Cleans the status field into simplified categories: Success or Failure.
   * Converts timestamps to standardized datetime objects.
3. **Load**:
   * Saves parsed transactions to an intermediate JSON file.
   * Converts the enriched data into a pandas DataFrame and writes it to a CSV file for analysis.

#### ⚙️ Key Components:

* parse\_log\_file: Reads and extracts transaction info from unstructured logs using regular expressions.
* enrich\_transaction: Adds metadata (e.g., BIC codes) and formats data fields.
* etl\_pipeline: Executes the transformation and loads the final CSV.
* run\_etl: Orchestrates the full ETL process from log parsing to CSV output.

#### 🗂 Output:

* output/parsed\_transactions.json: Parsed raw data.
* output/transaction data.csv: Cleaned and enriched data ready for analytics.

db\_check\_records.py  
This script is a utility tool designed to verify the contents of the local SQLite database (payment\_system.db) used in the RTR (Real-Time Rail) project. Specifically, it performs the following actions:

* **Connects to the database** using SQLite.
* **Fetches and prints all BIC codes** from the bic\_codes table — typically used to validate which financial institutions are registered in the system.
* **Fetches and prints all user records** from the users table — helping developers confirm that user data has been correctly inserted or migrated.
* Closes the database connection cleanly once operations are complete.

This script is primarily used for debugging or validation during development and testing phases to ensure that key database records are present and accurate.

db\_manager.py

The db\_manager.py module is responsible for setting up and managing the SQLite database (payment\_system.db) used in the RTR (Real-Time Rail) simulation environment. It performs the following key functions:

* **Database Initialization (init\_db)**:  
  This function creates the necessary tables (bic\_codes, users, and transactions) with predefined schemas. It ensures data integrity through foreign key constraints and initializes the database with default financial institutions (via BIC codes) and user entities, each with a starting balance.
* **Database Reset (reset\_db)**:  
  Provides a safe method to reset the entire database back to its initial state, which is useful during testing or development to ensure a clean starting point.

This module ensures a consistent and structured data layer for managing transaction simulations between users affiliated with different financial institutions.

db\_setup.py

This script is responsible for initializing the **SQLite database** used in the Real-Time Rail (RTR) payment simulation system. It performs three main functions:

1. **Database Initialization & Table Creation**
   * Connects to (or creates) a local SQLite database file named payment\_system.db.
   * Drops any existing tables (payments, users, bic\_codes) to ensure a clean setup.
   * Defines and creates the following tables:
     + bic\_codes: Maps financial institution (FI) codes to their BIC identifiers.
     + users: Stores user information, linking each user to a specific FI via fi\_code.
     + payments: Records all transaction data between users, including sender, recipient, amount, and timestamp.
2. **Sample Data Insertion**
   * Populates the bic\_codes table with 4 predefined FIs and their BIC codes.
   * Inserts 4 sample users with initial account balances and associated FI codes.
3. **Verification and Debugging**
   * Reconnects to the database at the end of the script.
   * Retrieves and prints all user records to verify successful setup and data insertion.

This setup file is typically run once to prepare the environment for testing and development of RTR transaction functionality.

ISO20022\_Camt054\_Generator.py

This script is responsible for **generating and saving ISO 20022 CAMT.054 credit notification messages** in XML format. These messages are used in real-time payment systems to notify financial institutions of account credits.

**Key functionalities include:**

* **generate\_camt054\_message()**: Creates a CAMT.054 XML structure with standard fields like GrpHdr, Ntfctn, and associated metadata including timestamp, creditor BIC, transaction amount, and a related message ID.
* **save\_camt054\_message()**: Saves the generated XML message to a timestamped file under the messages/camt054/ directory, ensuring the content is properly formatted using minidom.

This module plays a critical role in simulating payment credit notifications within the Real-Time Rail (RTR) system by producing standardized, machine-readable messages.

ISO20022\_Pacs002\_Generator.py

This module is responsible for generating and saving **ISO 20022-compliant PACS.002 messages**, which are acknowledgment responses to payment instructions such as PACS.008. These messages are essential in the Real-Time Rail (RTR) payment ecosystem to indicate whether a transaction was accepted, rejected, or requires further attention.

#### 🔧 Key Functions:

* **generate\_pacs002\_message(original\_message\_id, status, reason=None)**:  
  Creates an XML PACS.002 message with the provided original message ID and transaction status (ACCP, RJCT, etc.). An optional reason code can be included for rejected messages.
* **save\_pacs002\_message(tree, bank\_bic, message\_type="response")**:  
  Saves the generated XML message to the appropriate directory, with a filename that includes the BIC code and timestamp. It also ensures the output is formatted for readability.

#### 📦 Outputs:

* Well-formed PACS.002 XML files saved under messages/pacs002/, ready for transmission to financial institutions or logging for audit purposes.

#### 🔍 Use Case in RTR:

This script automates the creation of response messages for inbound payment instructions, ensuring that acknowledgment flows comply with ISO 20022 standards and are correctly stored and timestamped.

ISO20022\_Pacs008\_Generator.py

This module is responsible for generating and managing **ISO 20022 PACS.008 payment messages** within the RTR (Real-Time Rail) simulation environment. It enables seamless end-to-end creation, formatting, and routing of interbank credit transfer messages.

#### 🔧 Key Responsibilities:

* **Database Access:** Retrieves user and BIC code information from a SQLite database to populate payment message fields accurately.
* **Message Generation:** Constructs ISO 20022-compliant PACS.008 XML messages for credit transfers using payer and payee details.
* **Logging:** Tracks transaction activity in a log file (settlement\_log.txt) for audit and debugging purposes.
* **File Management:** Saves messages in a human-readable XML format with timestamped filenames for traceability.
* **RTR Integration:** Passes the generated messages to the RTR Exchange Processor for validation and settlement simulation.

#### 🔍 Main Functions:

* generate\_iso20022\_message(payer, payee, amount): Builds a minimal PACS.008 message using sender/receiver BICs and transaction details.
* save\_message(tree, payer\_name, payee\_name): Saves the generated XML message to the file system under messages/pacs008/.
* process\_through\_rtr(filename): Forwards the XML file to the RTR exchange processor for downstream processing and response generation.
* get\_user\_by\_name(name), get\_all\_users(): Utility functions to fetch user records and associated BIC codes.

ISO20022\_Pain001\_Generator.py

payment\_interface.py

payment\_system.db

RTR\_Exchange Processor.py

RTR Settlement\_Processor.py

settlement log.txt

Transaction Analysis Workbook.twb