**Title : IBAN generator  
  
Description**

As a developer, I want to generate valid random IBANs for supported EU countries, so that I can test transaction flows and validate IBAN-based services with format-compliant test data.  
  
**Why**  
  
Field :25: in an MT103 statement contains the **account identification**, which often includes the **IBAN**. This field shows which account the statement is about. It tells the receiving system or user exactly **which bank account** the transactions in the statement belong to.

**Acceptance Criteria:**

1. **IBAN format compliance**
   * The IBAN must start with the correct 2-letter country code and 2-digit check digits.
   * The total length of the IBAN must match the official length defined per country (e.g., AT = 20, MT = 31).
   * The BBAN (Basic Bank Account Number) section must be randomly generated using alphanumeric characters (A–Z, 0–9) where appropriate.
   * When a random IBAN is generated, the country it was generated for must be logged using SLF4J.
2. **Country-specific support**
   * Only supported countries (as per the table provided) must be used.
   * EUCountryCode enum must include IBAN length and country code metadata.
3. **Random country generation**
   * A utility method (generateRandomIban) must generate an IBAN for a randomly selected EU country.
4. **Definition of done:**
   * Unit tests must verify:
     + Generated IBANs conform to specified lengths.  
       Country code is correctly prefixed.  
       BBAN length is accurate.  
       All supported countries are covered in generation.

### **Title : Transaction reference, field :20:**

### **Description:**

As a developer, I want to generate a compliant and random :20: transaction reference for MT103 messages, so that I can simulate financial transaction data in a valid format for testing and development.

**Why**:  
Field :20: in an MT103 statement is the **Transaction Reference Number**. It uniquely identifies the statement or transaction and is used to track and match messages between banks or systems. This reference helps ensure consistency and avoid duplication in financial records.

### **Acceptance Criteria:**

1. **Fixed format**
   * Each generated reference must begin with the prefix "REF".
   * The remaining 9 characters must be randomly generated, consisting of uppercase letters and digits.
   * Final reference must be exactly 12 characters long.
2. **Random generation**
   * Consecutive calls to the generator should yield different references.
   * The randomness must combine both numeric and alphabetic characters (A–Z, 0–9).
3. **Encapsulation**
   * Code must be part of a Spring @Service-annotated class (Mt103Field20Generator).
   * Random logic must be implemented cleanly in a private helper method.

### **Test Coverage:**

* Unit test to verify that:
  + The generated reference is exactly 12 characters long.
  + The reference always starts with "REF".
  + The suffix matches [A-Z0-9]{9}.

### **Title : Transaction reference, field :28:**

**Description:**

As a developer, I want to generate a valid :28C: field for MT103 bank statement messages, so that I can simulate realistic statement and sequence numbers to test statement processing logic.  
  
**Why:**The :28C: bank statements identify the statement number and sequence number of the account statement. It helps uniquely label each statement and its parts, allowing banks and customers to track multiple statements or segmented statements correctly. The field typically consists of a statement number (usually six digits) followed by a sequence number (such as /001 or /002), indicating the position of the statement or its segment within a reporting period.

### **Acceptance Criteria:**

1. **Field format compliance**
   * The generated :28C: field must be formatted as XXXXXX/YYY.
   * The statement number part (XXXXXX) must be exactly 6 digits, where:
     + The first 5 digits can be any number from 00000 to 99999 (leading zeros allowed).
     + The 6th (last) digit must be either 1 or 2.
   * The sequence number part (YYY) must be either "001" or "002".
2. **Randomness**
   * Statement number and sequence number must be randomly generated each time.
   * Sequence number must strictly be "001" or "002" (formatted with leading zeros).

### **Test Coverage:**

* Unit tests verify the field format is always \d{6}/\d{3}.
* The last digit of the 6-digit statement number is always 1 or 2.
* The sequence number is always "001" or "002".

**Title: Opening Balance Generator – Field :60F:**

**Description**:  
 As a developer, I want to generate a valid and realistic :60F: opening balance field for MT940 messages, so that each statement begins with a consistent and correctly formatted financial snapshot.

**Why**:  
 Field :60F: represents the opening balance of the account statement. It is essential for reconstructing account movement and calculating balances accurately throughout the statement.

**Acceptance Criteria**:

**Fixed format**

* The field must follow the pattern: :60F:<D/C><YYMMDD><Currency><Amount>
* Currency should be fixed to EUR for simulation.
* Amount must use a comma as decimal separator (e.g., 1000,00).
* Debit or credit indicator must be either D or C.

**Dynamic values**

* The date must be dynamically generated, formatted as yyMMdd.
* The amount should be a positive decimal, randomly chosen within a configurable or reasonable range (e.g., 500.00–5000.00).
* The credit/debit flag should be randomly set to C or D.

**Encapsulation**

* Implemented in a dedicated Spring @Service class (Mt940BalanceGenerator or similar).
* Use helper method(s) for date and amount formatting.

**Test Coverage**:

* Unit test verifies that:
  + The generated field starts with :60F:.
  + Date is in yyMMdd format.
  + Currency is EUR.
  + Amount uses a comma, not a dot.
  + The full value matches pattern :60F:[CD]\d{6}EUR\d+,\d{2}.

### **Title: Transaction Line Generator – Field :61:**

**Description**:  
 As a developer, I want to generate realistic :61: transaction lines for M103 files, so I can simulate financial movements and reflect them correctly in the bank statement structure.

**Why**:  
 The :61: field represents individual transactions within the statement. It includes dates, amount, direction, transaction type, and unique references. This is key to simulating the core purpose of an MT103 message: showing account activity.

**Acceptance Criteria**:

**Fixed format**

* Must follow SWIFT MT940 standard structure:  
   :61:<ValueDate><EntryDate><D/C><Amount>NTRFNONREF//<Timestamp>
* Dates must follow yyMMdd format.
* The amount must be in decimal format, using a comma as a separator.
* Debit or credit indicator must be D or C.

**Dynamic values**

* ValueDate and EntryDate must reflect the current system date.
* Amount should be random within a realistic range (e.g., 1.00–500.00).
* Timestamp should be an ISO 8601 formatted date-time string (e.g., 2025-05-27T16:00:04).

**Encapsulation**

* Logic implemented inside a Spring @Service class.
* Internal helper methods handle formatting for dates, amount, and timestamp.

**Test Coverage**:

* Unit test confirms:
  + Output starts with :61:.
  + Dates match yyMMdd pattern.
  + Debit/credit character is valid.
  + Amount uses comma separator.
  + Ends with a valid ISO timestamp.

### **Title: Closing Balance Generator – Field :62F:**

**Description**:  
 As a developer, I want to generate a valid :62F: closing balance field for each MT940 statement, so the statement includes a final summary of account status after all transactions.

**Why**:  
 Field :62F: represents the closing balance at the end of the statement period. It must be accurate and consistent with simulated transactions, providing the closing financial state of the account.

**Acceptance Criteria**:

**Fixed format**

* Must match the format: :62F:<D/C><YYMMDD><Currency><Amount>
* Currency must be EUR.
* Amount must include two decimals, using a comma separator.

**Logical consistency**

* Closing balance should reflect the opening balance minus/plus the transaction amount(s) if tracking is implemented.
* For now, may be statically defined or randomized independently within a range.

**Encapsulation**

* Implemented in a Spring @Service class (e.g., Mt940BalanceGenerator).
* Shared formatting method with :60F: may be reused.

**Test Coverage**:

* Unit test checks:
  + Field starts with :62F:.
  + Date formatting is valid.
  + Debit/Credit indicator is valid.
  + Amount format is compliant.
  + Matches pattern :62F:[CD]\d{6}EUR\d+,\d{2}.

### **Title: Narrative Information Generator – Field :86:**

**Description**:  
 As a developer, I want to generate structured :86: narrative fields containing remittance information and merchant details, so that payment context is clearly provided in simulated MT103 files.

**Why**:  
 The :86: field in MT103 carries structured information used for reconciliation, including invoice reference, payer name, and remittance information. It is crucial for reproducing real-world bank statement narratives.

**Acceptance Criteria**:

**Fixed format**

* Format must be:  
   :86:/EREF/<eref>/NAME/<merchantName>/REMI/<remittanceInfo>
* All tokens must be included and properly prefixed.

**Dynamic values**

* eref (end-to-end reference), merchantName, and remittanceInfo must be randomly selected or composed from predefined lists.
* Characters used must be valid within MT940 (uppercase, numbers, basic punctuation).

**Encapsulation**

* Logic resides in a Spring @Service-annotated class (e.g., Mt940NarrativeGenerator).
* Separate methods for composing eref, name, and remittanceInfo.

**Test Coverage**:

* Unit test verifies:
  + Field starts with :86: and includes all required sections.
  + Each section begins with the correct tag (/EREF/, /NAME/, /REMI/).
  + Values are not null or empty.
  + Matches pattern :86:/EREF/.+/NAME/.+/REMI/.+.

### 

### 

### **Title : Scheduled job to generate MT103 files.**

### **Description:**

As a developer, I want to automatically generate MT103 statement files at a fixed interval with realistic randomized data, write these files to disk, and publish related transaction messages to Kafka, so that the system can simulate real-time transaction data for downstream processing and testing.

### **Acceptance Criteria:**

1. **Scheduled generation**
   * The system triggers the generation of an MT103 statement file every 100 milliseconds (configurable fixed rate).
   * Each generated file includes all required MT103 fields populated with valid and randomized test data.
2. **Data generation**
   * Use Mt103Field20Generator for transaction reference (:20:).
   * Use IbanGeneratorByCountry for account number (:25:).
   * Use Mt103Field28CGenerator for statement number (:28C:).
3. **File writing**
   * MT103 content is saved as a .sta file in src/main/resources/mt103\_files/.
   * Filename format: <timestamp>\_<accountNumber>.sta, where timestamp is yyyyMMddHHmmss.

### **Test Coverage:**

* Unit and integration tests to verify:  
  + The scheduled method runs at expected fixed intervals (mock scheduler or timing control).
  + Generated MT103 content meets formatting and field requirements.