# Loan Application Management Service - Project Documentation

This document provides detailed design and implementation notes for the Loan Application Service project. It covers purpose, steps, validation rules, errors, inputs/outputs, and related technical details for each service.

## 1. Project Overview

### 1.1 Project Summary

The Loan Application Service Project is designed to provide a complete end-to-end loan management process.
It consists of multiple IBM App Connect Enterprise (ACE 12) services that manage applicant data, loan requests, validations, approvals, and audit logging.
The solution ensures data consistency, validation, error handling, and integration across database, MQ, and REST APIs.

---

### 1.2 Architecture

The solution follows a service-oriented architecture with the following components:

- Services (REST APIs in ACE 12)
  - Each service has a defined ServiceCode.
  - Services follow a Header + Body input/output pattern with schema validation.
  - Logic is implemented using Compute nodes + Stored Procedures.
- Database (SQL Server)
  - Core business data stored in tables.
  - Stored procedures handle reusable logic.
- Error Handling & Logging
  - Standardized error catalog (ERRxxx codes).
  - Logging of all requests and responses with TransactionId.

---

### 1.3 Goals

The project aims to:

- Provide a loan processing system with reusable services.
- Ensure validation of all inputs using JSON Schema.
- Support end-to-end error handling.
- Maintain auditability of every request/response.
- Integrate with external/internal systems via HTTP APIs and MQ for real-time loan approvals.

---

## 2. Project Standards & Best Practices

### 1. Input/Output Standards

- All services must follow a standard input/output format (Header + Body).
- Ensure schema validation using JSON Schema for all inputs.
- Use TransactionId across logs and flows.

---

### 2. Validation & Error Handling

- Every service should:
  - Validate inputs strictly.
  - Use Try-Catch blocks to handle and classify errors.
  - Return standardized error codes/messages.
- Define a centralized Error Catalog with:
  - ErrorCode
  - ErrorMessage
- All services should catch:
  - Input errors (missing fields, invalid formats).
  - Business logic errors (e.g., applicant not found).
  - Integration errors (DB down, MQ timeout, HTTP failure).

---

### 3. Service Naming & Organization

- Use clear and consistent service names with verbs + entities.

---

### 4. Logging & Security

- Log all incoming requests and outgoing responses:
  - Store in a log table.
  - Include timestamps, request body, response body, and error (if any).

---

### 5. Integration (HTTP, MQ, DB, Files)

- Use HTTP Request nodes to call external/internal REST services.
- Use MQ Input/Output for event-driven operations.
- Use parameterized queries for database safety.
- Use Stored Procedures where business logic is heavy or shared.

---

### 6. Testing & Documentation

- Include both positive and negative test cases for every flow.
- Document example payloads for all scenarios (success + error).

---

## 3. Documentation & Training Work

As part of my training, I created **ISD** (Integration Service Design) and **Unit Testing documentation** for the service: GetAllPendingApplicationsService

- **ISD (Integration Service Design) Document**
  - Defined the service name and code.
  - Wrote a clear description of the business purpose.
  - Documented input and output JSON structures with sample payloads.
  - Specified validation rules and business constraints.
  - Listed possible error codes and messages.

- **Unit Testing Document**
  - Included sample requests and expected responses.
  - Documented results for each test case (pass/fail).
  - Covered edge scenarios (e.g., missing required fields, invalid data types, schema errors).

---

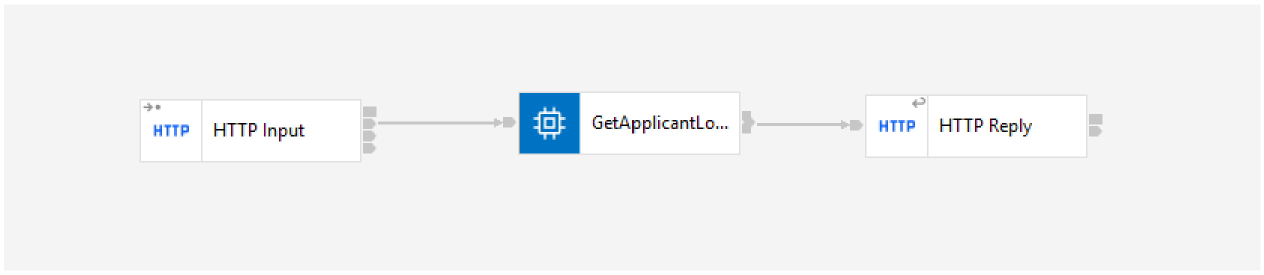
# 4. Services Documentation

## 4.1 GetApplicantService

### Purpose

The purpose of this service is to fetch applicant details from the database using the provided ApplicantId.
It ensures the applicant exists before proceeding with any loan application request.

### Steps / Flow Logic



- Receive request containing ApplicantId.
- Validate that ApplicantId is provided and is not empty.
- Call stored procedure sp_GetApplicant with the ApplicantId.
- If no records are found → return error ERR301 (Applicant not found).
- On success → return full applicant details (Id, Name, Email, Phone, Salary, CreditScore, etc.).

### Input Validation & Errors

| Error Code | Description | Cause |
|---|---|---|
| E501 | ApplicantId missing/invalid | Input does not contain a valid ApplicantId |
| E502 | Applicant not found | No matching record found in Applicants table |
| E503 | ApplicantId mismatch with IdNumber in the request | Triggered when ApplicantId in the request body does not match IdNumber in the request header. |

### Postman Test Cases

**Success Case**

Input Example:

```
{
 "GetApplicantRq": {
  "Header": {
   "TransactionId": "77889902",
   "IDNumber": "123456777",
   "ServiceCode": "GA"
  },
  "Body": {
   "RequestDetails": {
    "ApplicantId": "123456777"
   }
  }
 }
}
```

Output:

```
{
   "GetApplicantRs": {
      "Header": {
         "TransactionId": "77889902",
         "Status": "Success"
      },
      "Body": {
         "ApplicantDetails": {
            "ApplicantId": "123456777",
            "Name": "Ali",
            "IDNumber": "123456777",
            "NationalIdNumber": "99887766",
            "salary": 5000,
            "CreditScore": 700,
            "Gender": "X",
            "PhoneNumber": "0790000000",
            "Email": "test@example.com",
            "Address": "Amman"
         }
      }
   }
}
```

**Error Case 1**

ApplicantId (IDNumber) required

Input Example:

```
{
 "GetApplicantRq": {
   "Header": {
    "TransactionId": "77889902",
    "IDNumber": "123456777",
    "ServiceCode": "GA"
   },
   "Body": {
    "RequestDetails": {
      }
    }
  }
}
```

Output:

```
{
   "GetApplicantRs": {
      "Header": {
         "TransactionId": "77889902",
         "Status": "Error"
      },
      "Body": {
         "ErrorList": {
            "ErrorItem": {
               "ErrorCode": "E501",
               "ErrorDescription": "ApplicantId (IDNumber) required"
            }
         }
      }
   }
}
```

**Error Case 2**

ApplicantId mismatch with IdNumber in the request

Input Example:

```
{
  "GetApplicantRq": {
    "Header": {
      "TransactionId": "77889902",
      "IDNumber": "123456777",
      "ServiceCode": "GA"
    },
    "Body": {
      "RequestDetails": {
        "ApplicantId": "123456666"
      }
    }
  }
}
```

Output:

```
{
   "GetApplicantRs": {
      "Header": {
         "TransactionId": "77889902",
        "Status": "Error"
      },
      "Body": {
        "ErrorList": {
          "ErrorItem": {
            "ErrorCode": "E503",
            "ErrorDescription": "ApplicantId mismatch with IdNumber in the request"
          }
        }
      }
    }
}
```

**Error Case 3**

No applicant found

Input Example:

```
{
  "GetApplicantRq": {
    "Header": {
      "TransactionId": "77889902",
      "IDNumber": "123456666",
      "ServiceCode": "GA"
    },
    "Body": {
      "RequestDetails": {
        "ApplicantId": "123456666"
      }
    }
  }
}
```

Output:

```
{
    "GetApplicantRs": {
      "Header": {
        "TransactionId": "77889902",
        "Status": "Error"
      },
      "Body": {
        "ErrorList": {
          "ErrorItem": {
            "ErrorCode": "E502",
            "ErrorDescription": "No applicant found"
          }
        }
      }
```

```
  }
}
```

### Database

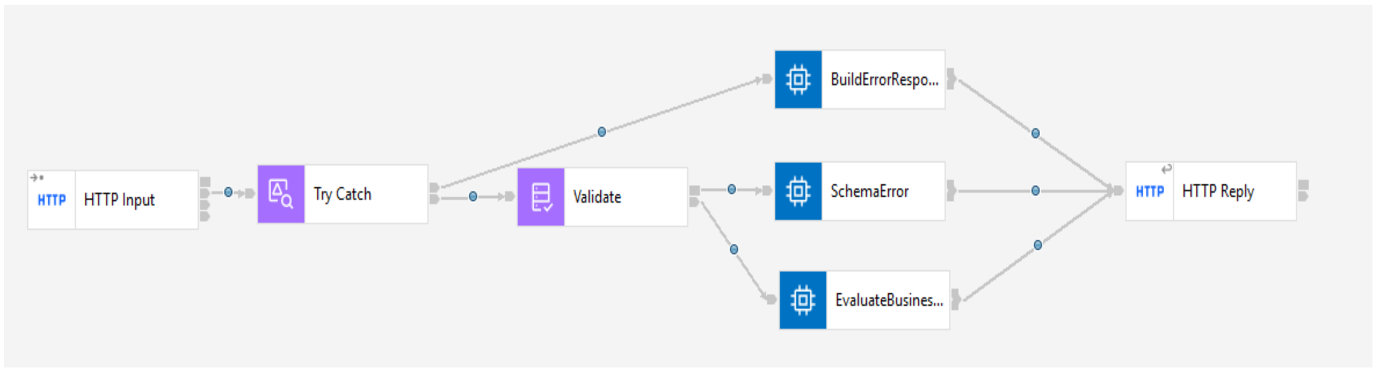
- Database Table: Applicants
- Stored Procedure: sp_GetApplicant @ApplicantId

## 4.2 ValidationService

### Purpose

The ValidationService ensures that loan application requests are complete, consistent, and compliant with business rules.
It validates applicant details, inserts missing applicants into the DB, checks loan rules (salary, loanAmount, age, creditScore), and always records the request/response in the AuditLog.

### Steps / Flow Logic



- Receive request with Header (TransactionId, IdNumber, ServiceCode) and Body (ApplicantId, Loan details, Salary, Age, CreditScore).
- Set response header: TransactionId, IdNumber, ServiceCode, with default Status = "Success."
- Check Applicant in DB:
  - Query Applicants table using IdNumber.
  - If Applicant does not exist → insert Applicant with basic details (Salary, Name, CreditScore, etc.).
- Validate ApplicantId consistency:
  - If Header.IdNumber ≠ Body.ApplicantId → reject with error E999 ("Unauthorized access: mismatch").
- Apply business rules:
  - Salary ≥ 3000.
  - LoanAmount ≤ 10 × Salary.
  - TermMonths ≤ 60.
  - Age between 21–55.
  - CreditScore ≥ 500.
  - If any rule fails → mark isValid = FALSE and collect error codes.

- Return result:
  - If valid → isValid = TRUE.
  - If invalid → detailed list of ErrorItems with codes E101–E105.

- Insert Audit Log:
  - Call sp_InsertAuditLog_appconnect with TransactionId, RequestPayload, ResponsePayload, and ServiceType = "Validation."

## Input Validation & Errors

| Error Code | Description | Cause |
|---|---|---|
| E101 | Salary must be at least 3000 | When salary < 3000 |
| E102 | Loan amount exceeds salary limit | When loanAmount > (salary × 10). |
| E103 | Term cannot exceed 60 months | When termMonths > 60. |
| E104 | Applicant age must be between 21 and 55 | When age < 21 OR age > 55. |
| E105 | Credit score must be 500 or higher | When creditScore < 500 |
| E999 | Unauthorized access: ApplicantId mismatch | When Header.IDNumber ≠ Body.ApplicantId. |

## Postman Test Cases

**Success Case**

Input Example:

```
{
 "ValidateLoanApplicationsRq": {
  "Header": {
   "TransactionId": "TX-001",
   "IDNumber": "123456789",
   "ServiceCode": "VLA00001"

  },
  "Body": {
   "RequestDetails": {
    "ApplicantId": "123456789",
    "NationalIdNumber": "99887766",
    "loanAmount": 10000,
    "name": "Ali",
    "salary": 5000,
    "termMonths": 12,
    "creditScore": 700,
    "MonthlyPayment": 400,
    "age": 30
   }
  }
 }
}
```

Output:

```
{
  "ValidateLoanApplicationsRs": {
    "Header": {
      "TransactionId": "TX-001",
      "IDNumber": "123456789",
      "ServiceCode": "VLA00001",
      "Status": "Success"
    },
    "Body": {
      "RequestDetails": {
        "ApplicantId": "123456789",
        "NationalIdNumber": "99887766",
        "loanAmount": 10000,
        "name": "Ali",
        "salary": 5000,
        "termMonths": 12,
        "creditScore": 700,
        "MonthlyPayment": 400,
        "age": 30,
        "isValid": true
      }
    }
  }
}
```

**Error Case 1**

SchemaError

Input Example:

```
{
 "ValidateLoanApplicationsRq": {
  "Header": {
   "IDNumber": "123456789",
   "ServiceCode": "VLA00001"
```

```
    },
    "Body": {
      "RequestDetails": {
        "ApplicantId": "123456789",
        "NationalIdNumber": "99887766",
        "loanAmount": 10000,
        "name": "Ali",
        "salary": 5000,
        "termMonths": 12,
        "creditScore": 700,
        "MonthlyPayment": 400,
        "age": 30
      }
    }
  }
}
```

Output:

```
{
    "ValidateLoanApplicationsRs": {
        "Error": "SchemaError"
    }
}
```

**Error Case 2**

Business rule

Input Example:

```
{
  "ValidateLoanApplicationsRq": {
    "Header": {
      "TransactionId": "TX-001",
      "IDNumber": "123456789",
      "ServiceCode": "VLA00001"
    },
    "Body": {
      "RequestDetails": {
        "ApplicantId": "123456789",
        "NationalIdNumber": "99887766",
        "loanAmount": 10000,
        "name": "Ali",
        "salary": 5000,
        "termMonths": 66,
        "creditScore": 300,
        "MonthlyPayment": 400,
        "age": 80
      }
    }
  }
}
```

Output:

```
{
   "ValidateLoanApplicationsRs": {
      "Header": {
         "TransactionId": "TX-001",
         "IDNumber": "123456789",
         "ServiceCode": "VLA00001",
         "Status": "Success"
      },
      "Body": {
         "RequestDetails": {
            "ApplicantId": "123456789",
            "NationalIdNumber": "99887766",
            "loanAmount": 10000,
            "name": "Ali",
            "salary": 5000,
            "termMonths": 66,
            "creditScore": 300,
            "MonthlyPayment": 400,
            "age": 80,
            "isValid": false
         },
         "errors": {
            "ErrorItem": {
               "ErrorCode": "E105",
               "ErrorDescription": "Credit score must be 500 or higher"
            }
         }
      }
   }
}
```

**Error Case 3**
ApplicantId mismatch
Input Example:

```
{
  "ValidateLoanApplicationsRq": {
    "Header": {
      "TransactionId": "TX-001",
      "IDNumber": "123456777",
        "ServiceCode": "VLA00001"
  },
  "Body": {
   "RequestDetails": {
    "ApplicantId": "123456789",
    "NationalIdNumber": "99887766",
    "loanAmount": 10000,
    "name": "Ali",
    "salary": 5000,
    "termMonths": 12,
    "creditScore": 700,
    "MonthlyPayment": 400,
    "age": 30
   }
  }
 }
}
```

Output:

```
{
  "ValidateLoanApplicationsRs": {
    "Header": {
      "TransactionId": "TX-001",
      "IDNumber": "123456777",
      "ServiceCode": "VLA00001",
      "Status": "Success"
    },
    "Body": {
      "RequestDetails": {
        "ApplicantId": "123456789",
        "NationalIdNumber": "99887766",
        "loanAmount": 10000,
        "name": "Ali",
        "salary": 5000,
        "termMonths": 12,
        "creditScore": 700,
        "MonthlyPayment": 400,
        "age": 30,
        "isValid": false
      },
      "errors": {
        "ErrorItem": {
          "ErrorCode": "E999",
          "ErrorDescription": "Unauthorized access: ApplicantId mismatch"
        }
      }
    }
  }
}
```

### Database

- Reads from Applicants table.
- Inserts into Applicants table if missing.
- Calls stored procedure sp_InsertAuditLog_appconnect.

## 4.3 SubmitLoanService

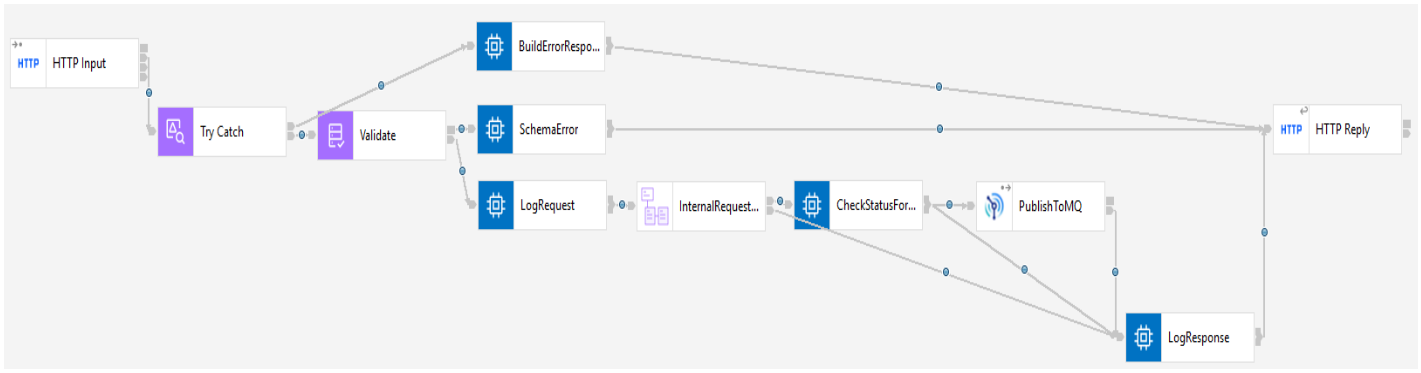
### Purpose

The SubmitLoanApplications service is the entry point for loan application submission.
It arrange calls to:

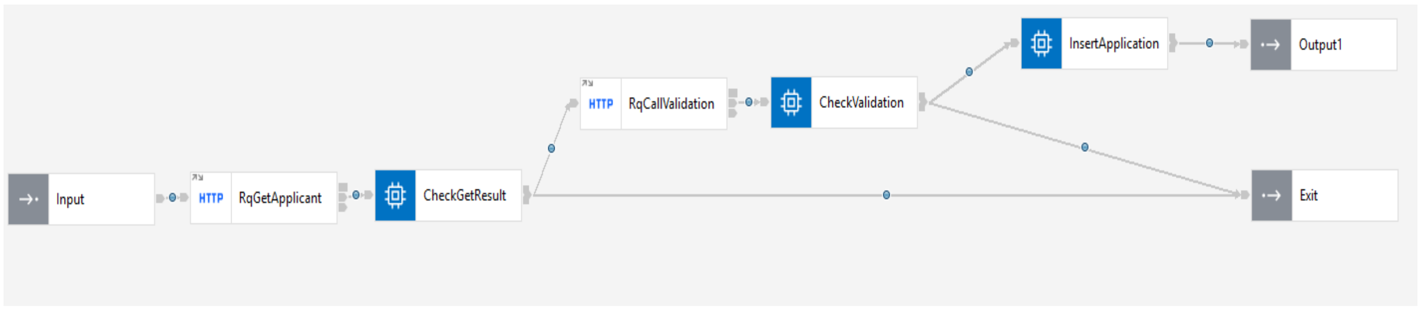
- GetApplicant Service (to check applicant existence).
- Validation Service (to apply business rules).
- Database Insert + Audit Logging.
- MQ Publish (if loan is pending approval).

It ensures only valid applications move forward, while invalid or pending ones are either logged or queued for later approval.

### Steps / Flow Logic



### Below the internal request flow:



- Log Request
  - Extracts TransactionId and stores original request payload in Environment.
  - Builds a GetApplicantRq and sends it to the GetApplicant Service.

- Check GetApplicant Response
  - If applicant not found (Status=Error) → Build error response immediately and return.
  - If success → Build ValidateLoanApplicationsRq and continue.

- Check Validation Response
  - If Validation Service down/unavailable → Insert record as Pending (StatusId=1).
  - If isValid=false → Return error response immediately (with detailed validation errors).
  - If isValid=true → Mark application as Validated (StatusId=2) and continue.

- Insert into Database
  - Store loan application details (ApplicantId, LoanAmount, TermMonths, etc.) with appropriate StatusId.

- Check Status for MQ
  - If StatusId=1 (Pending) → Publish an ApprovalRequest message to Loan.Failed.Queue.
  - If validated → Skip MQ and continue.

- Log Response
  - Build SubmitLoanApplicationsRs for Postman response.
  - Insert audit log (TransactionId, request payload, response payload) using stored procedure sp_InsertAuditLog.

## Input Validation & Errors

| Error Code | Description | Cause |
|---|---|---|
| E502 | No applicant found | GetApplicantService returned Applicant not found |
| E503 | ApplicantId mismatch with IdNumber in the request | GetApplicantService returned that the applicantid in body doesn't match IDnumber in header |
| E101-E105 | Validation failed (invalid business rules ) | ValidationService returned isValid = false |
| 500 | Internal server error | Unexpected system/DB/MQ failure |
| 400 | Schema validation failed | |

## Postman Test Cases

**Success Case**

Input Example:

```
{
 "SubmitLoanApplicationsRq": {
   "Header": {
    "TransactionId": "TX-001",
    "IDNumber": "123456789",
      "ServiceCode": "VLA00001"
  },
  "Body": {
    "RequestDetails": {
      "ApplicantId": "123456789",
      "NationalIdNumber": "99887766",
      "loanAmount": 10000,
      "name": "Ali",
      "salary": 5000,
      "termMonths": 12,
      "creditScore": 700,
      "MonthlyPayment": 400,
      "age": 30
    }
  }
 }
}
```

Output:

```
{
  "SubmitLoanApplicationsRs": {
    "Header": {
      "TransactionId": "TX-001",
      "IDNumber": "123456789",
      "ServiceCode": "VLA00001",
      "ResponseStatus": "Success"
    },
    "Body": {
      "RequestDetails": {
        "ApplicationId": 1031,
        "statusId": 2,
        "Description": "Passed validation, ready for review"
      }
    }
  }
}
```

**Error Case 1**
SchemaError   (Loan amount provided as string not number)
Input Example:

```
{
  "SubmitLoanApplicationsRq": {
    "Header": {
      "TransactionId": "1221212",
      "IDNumber": "2589475793",
      "ServiceCode" : "SLA00001"
  },
  "Body": {
    "RequestDetails":
      {
      "ApplicantId": "2589475793 ",
      "NationalIdNumber": "992101031 ",
      "loanAmount": "15000",
      "MonthlyPayment" : 200
      }
  }
}
}
```

Output:

```
{
  "SubmitLoanApplicationsRs": {
    "Header": {
      "ResponseStatus": "Error"
    },
    "Body": {
      "ErrorDescription": "Schema validation failed"
    }
  }
}
```

**Error Case 2**
IDNumber doesn’t exist in DB
Input Example:

```
{
 "SubmitLoanApplicationsRq": {
  "Header": {
   "TransactionId": "TX-1001",
   "IDNumber": "2589475793",
   "ServiceCode": "SLA00001"
  },
  "Body": {
   "RequestDetails": {
      "ApplicantId": "2589475793",
      "NationalIdNumber": "992101031",
      "loanAmount": 15000,
      "MonthlyPayment": 200
    }
   }
 }
}
```

Output:

```
{
  "SubmitLoanApplicationsRs": {
    "Header": {
      "TransactionId": "TX-1001",
      "ResponseStatus": "Error"
    },
    "Body": {
      "ErrorList": {
        "ErrorItem": {
          "ErrorCode": "E502",
          "ErrorDescription": "No applicant found"
        }
      }
    }
  }
}
```

**Error Case 3**
applicantid in body doesn't match IDnumber in header
Input Example:

```
{
 "SubmitLoanApplicationsRq": {
   "Header": {
     "TransactionId": "77889902",
     "IDNumber": "123456777",
     "ServiceCode": "GA"
   },
   "Body": {
     "RequestDetails": {
       "ApplicantId": "123456666",
       "NationalIdNumber": "NID1234567890",
       "loanAmount": 15000,
       "MonthlyPayment": 450
     }
   }
 }
}
```

Output:

```
{
  "SubmitLoanApplicationsRs": {
   "Header": {
     "TransactionId": "77889902",
     "ResponseStatus": "Error"
   },
   "Body": {
     "ErrorList": {
       "ErrorItem": {
         "ErrorCode": "E503",
         "ErrorDescription": "ApplicantId mismatch with IdNumber in the request"
       }
     }
   }
  }
}
```

**Error Case 4**
invalid business rule
Input Example:

```
 {
 "SubmitLoanApplicationsRq": {
  "Header": {
     "TransactionId": "TX-001",
    "IDNumber": "123456789",
    "ServiceCode": "VLA00001"
  },
  "Body": {
    "RequestDetails": {
     "ApplicantId": "123456789",
      "NationalIdNumber": "99887766",
      "loanAmount": 10000.0,
      "name": "Ali",
      "salary": 5000.0,
      "termMonths": 66,
      "creditScore": 300,
      "MonthlyPayment": 400.0,
      "age": 80
    }
  }
 }
}
```

Output:

```
{
  "SubmitLoanApplicationsRs": {
   "Header": {
    "TransactionId": "TX-001",
    "ResponseStatus": "Error"
   },
   "Body": {
    "ErrorList": {
     "ErrorItem": [
      {
       "ErrorCode": "E103",
       "ErrorDescription": "Term cannot exceed 60 months"
      },
      {
       "ErrorCode": "E104",
       "ErrorDescription": "Applicant age must be between 21 and 54"
      },
      {
       "ErrorCode": "E105",
       "ErrorDescription": "Credit score must be 500 or higher"
      }
     ]
    }
   }
  }
 }
```

## Database / MQ Notes

- **Database**
  - Inserts into LoanApplications with StatusId = 1 (Pending) or 2 (Validated).
  - Calls sp_InsertAuditLog to log both request and response payloads.
- **MQ**
  - Queue: Loan.Failed.Queue
  - Message published only when StatusId=1.

## 4.4 ApprovalService

### Purpose

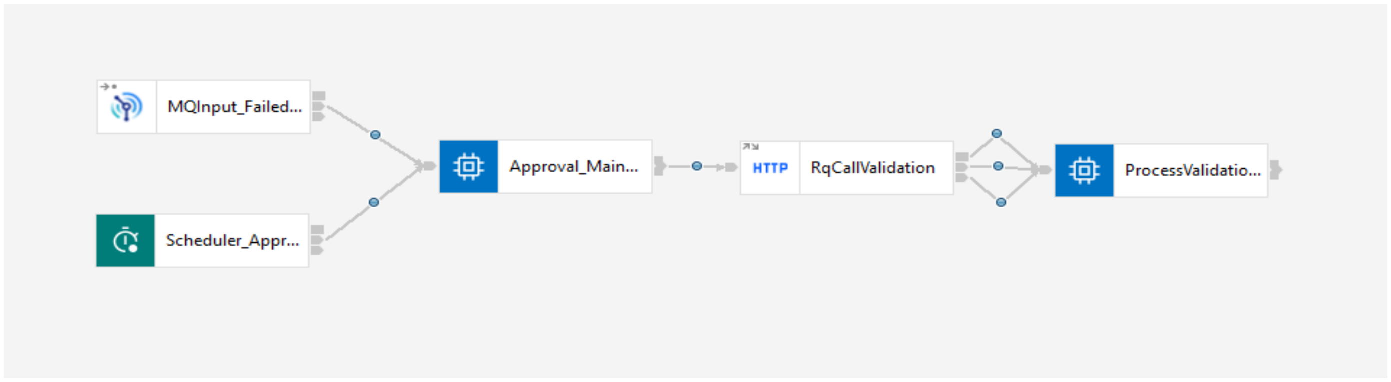
The ApprovalService is responsible for processing pending loan applications and deciding whether they should be approved (validated) or rejected (failed).

It works in two modes:

- Triggered by MQ (when a pending application is published).
- Triggered by Scheduler (e.g., every 3 hours to re-check DB for pending records).

It ensures all pending loans are revalidated against the ValidationService, and the loan status is updated in the database accordingly.

### Steps / Flow Logic



- Read Input
    - Service listens to Loan.Failed.Queue (MQ input).
- Fetch Pending Applications
    - If ApplicationId is in MQ input → fetch that record from DB.
    - If no ApplicationId → fetch top 10 pending applications (StatusId=1) using stored procedure:sp_GetPendingApplicationList.
- Loop Through Records
    - For each pending record, build a ValidateLoanApplicationsRq payload.
    - Fill header values with TransactionId, ServiceCode, and IDs.
    - Fill body with loan details (Salary, LoanAmount, Term, CreditScore, etc.).
    - Send each message to ValidationService.
- Process Validation Result
    - If isValid = true → update DB with StatusId = 2 (Validated).
    - If isValid = false → update DB with StatusId = 5 (Failed).
    - If ValidationService call fails (timeout/system error) → update DB with StatusId = 5 (Failed).

### Database / MQ Notes

- Database Procedures
  - sp_GetPendingApplicationList(Limit, Offset, ApplicationId) → fetches pending applications.
  - sp_ApproveRejectLoan(ApplicationId, NewStatusId) → updates status.
- MQ Queue
  - Input: Loan.Failed.Queue.
  - Trigger:
    -When SubmitLoanApplications inserts Pending record.
    -via Scheduler (every 3 hours).

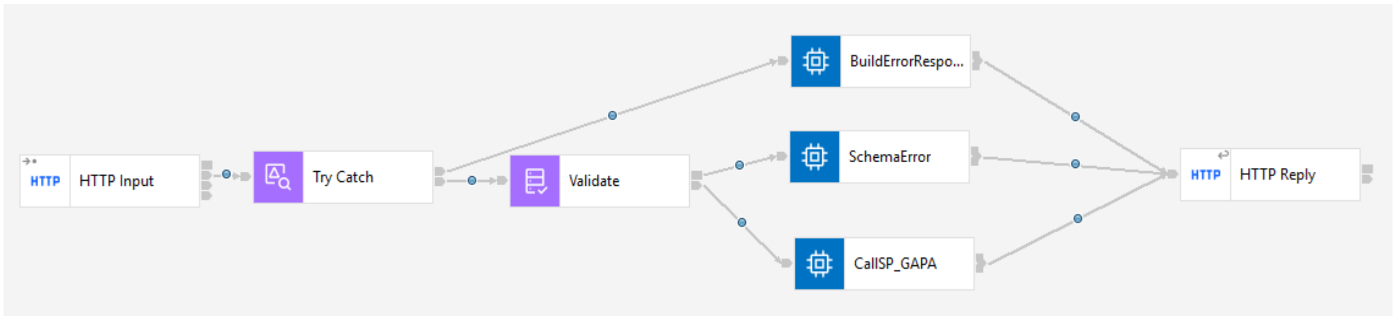
**Important Note:**
No Postman testing (success or failure cases) is applicable here because processing is handled entirely via MQ (WebSphere MQ) and the scheduler.

## 4.5 GetAllPendingApplications

### Purpose

The GetAllPendingApplicationsService provides a paginated list of all loan applications that are still in Pending status.
It allows external systems to query pending applications with limit/offset pagination to avoid returning too many records in one call.

### Steps / Flow Logic



- Read Input
  - Request payload:
```
{
  "GetAllPendingApplicationsRq": {
    "Limit": 10,
    "Offset": 0
  }
}
```
  - Extract Limit and Offset values.
- Validate Pagination Parameters
  - Limit must be >0 and ≤100.
  - Offset must be ≥0.
  - If invalid → return ERR001.

- Call Stored Procedure
  - Uses sp_GetPendingApplicationList with parameters (Limit, Offset, NULL).
  - Fetches only pending applications from LoanApplications table.
- Check for Empty Results
  - If no pending applications → return ERR002.
- Build Response
  - Return a JSON list of applications.
  - Each application contains: ApplicationId, ApplicantId, LoanAmount, StatusId.

## Input Validation & Errors

| Error Code | Description | Cause |
|---|---|---|
| ERR001 | Invalid pagination parameters | Limit ≤ 0, > 100, or Offset < 0 |
| ERR002 | No pending applications found | DB query returned empty result set |
| ERR999 | Internal service error | Unexpected DB/system failure. |

## Postman Test Cases

**Success Case**

Input Example:

```
{
 "GetAllPendingApplicationsRq": {
  "Limit": 10,
  "Offset": 0
 }
}
```

Output:

```
{
  "GetAllPendingApplicationsRs": {
    "Applications": {
      "ApplicationId": 1028,
      "ApplicantId": "A1004",
      "LoanAmount": 15000,
      "StatusId": 1
    }
  }
}
```

**Error Case 1**
Invalid pagination parameters
Input Example:

```
{
 "GetAllPendingApplicationsRq": {
  "Limit": -5,
  "Offset": 0
 }
}
```

Output:

```
{
  "GetAllPendingApplicationsRs": {
    "Error": {
      "ErrorType": "SchemaError",
      "ErrorCode": "ERR001",
      "ErrorDescription": "Invalid pagination parameters, Limit or Offset is missing or invalid"
    }
  }
}
```

**Error Case 2**
No pending applications
Input Example:

```
{
 "GetAllPendingApplicationsRq": {
  "Limit": 10,
  "Offset": 2
 }
}
```

Output:

```
{
  "Error": {
    "ErrorCode": "ERR002",
    "ErrorDescription": "No pending applications found"
  }
}
```

## Database

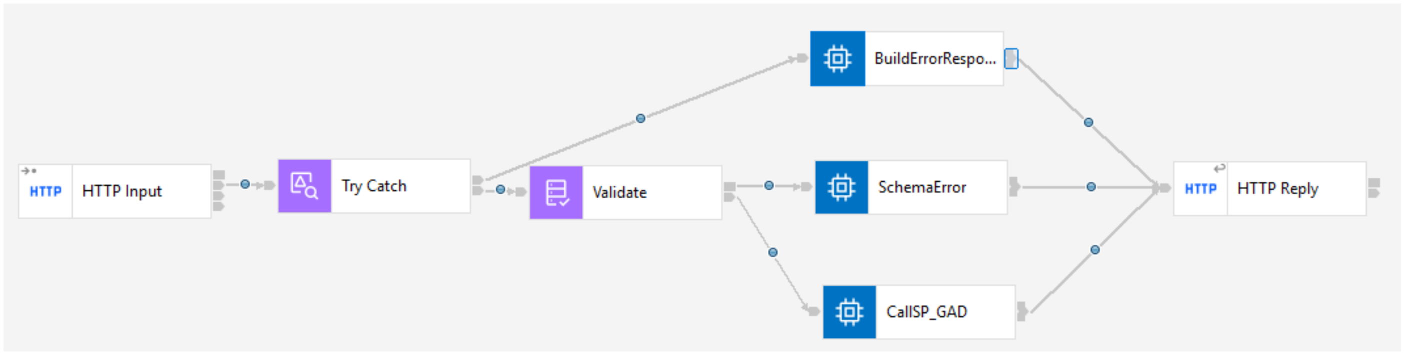
- **Stored Procedure**
  - sp_GetPendingApplicationList(Limit, Offset, ApplicationId)
  - Returns list of applications where StatusId = 1 (Pending).

## 4.6 GetApplicationDetailsService

### Purpose

The GetApplicationDetailsService retrieves the full details of a specific loan application, including applicant information.
It validates that the IdNumber provided in the request matches the applicant tied to the ApplicationId.

### Steps / Flow Logic



- Receive Input
  - Request includes ApplicationId in the body and IdNumber in the header.
- Validate Input
  - Ensure ApplicationId is provided.
  - Ensure IdNumber is provided.
  - If invalid → return ERR101.
- Query Database
  - Call stored procedure sp_GetApplicationDetails with ApplicationId.
  - Fetch application + applicant data.
- Check for Existence
  - If no record found → return ERR102.
- Validate Ownership
  - Compare DB ApplicantId with input IdNumber.
  - If mismatch → return ERR101.
- Return Details
  - Return loan information (ApplicationId, LoanAmount, TermMonths, CreditScore, MonthlyPayment, StatusId, etc.).
  - Return applicant information (Name, Phone, Email, Salary, NationalIdNumber, Gender, Address, etc.).

## Input Validation & Errors

| Error Code | Description | Cause |
|---|---|---|
| ERR101 | Input JSON does not match schema or contains invalid values | ApplicationId not provided / invalid OR IdNumber not linked to applicant |
| ERR102 | Application not found | ApplicationId does not exist in DB |
| ERR999 | Internal service error | Unexpected DB/system failure |

## Postman Test Cases

**Success Case**

Input Example:

```
{
 "GetApplicationDetailsRq": {
  "Header": {
   "IdNumber": "A1004"
  },
  "Body": {
   "ApplicationId": 1028
  }
 }
}
```

Output:

```
{
  "GetApplicationDetailsRs": {
    "ApplicationId": 1028,
    "ApplicantId": "A1004",
    "LoanAmount": 15000,
    "TermMonths": 66,
    "CreditScore": 720,
    "StatusId": 1,
    "MonthlyPayment": 1600,
    "LoanCreatedAt": "2025-08-19 10:15:13.553",
    "Applicant": {
      "Name": "Yousef Sami",
      "Phone": "0797654321",
      "Email": "youusef@example.com",
      "Salary": 6000,
      "CreditScore": 720,
      "NationalIdNumber": "11223344",
      "Gender": "F",
      "Address": "Irbid",
      "CreatedAt": "2025-08-19 10:15:02.990"
    }
  }
}
```

**Error Case 1**
missing ApplicationId
Input Example:

```
{
 "GetApplicationDetailsRq": {
  "Header": {
   "IdNumber": "99887766"
  },
  "Body": { }
 }
}
```

Output:

```
{
  "GetApplicationDetailsRs": {
    "Error": {
      "ErrorType": "SchemaError",
      "ErrorCode": "ERR101",
      "ErrorDescription": "Input JSON does not match schema or contains invalid values"
    }
  }
}
```

**Error Case 2**
Application not found
Input Example:

```
{
 "GetApplicationDetailsRq": {
  "Header": {
   "IdNumber": "A1004"
  },
  "Body": {
   "ApplicationId": 1050
  }
 }
}
```

Output:

```
{
  "GetApplicationDetailsRs": {
    "Error": {
      "ErrorCode": "ERR102",
      "ErrorDescription": "Application not found"
    }
  }
}
```

### Database

- **Stored Procedure**
  - sp_GetApplicationDetails(ApplicationId)
  - Returns full loan and applicant details.

---

## 5. Database Design

### Tables

- **Applicants**
  - ApplicantId (PK)
  - Name
  - Phone
  - Email
  - Salary
  - CreditScore
  - NationalIdNumber
  - Gender
  - Address
  - CreatedAt (default = GETDATE)
- **LoanApplications**
  - ApplicationId (PK, Identity)
  - ApplicantId (FK → Applicants.ApplicantId)
  - Salary
  - LoanAmount
  - TermMonths
  - CreditScore
  - StatusId (FK → LoanApplicationStatus.StatusId)
  - MonthlyPayment
  - CreatedAt (default = GETDATE)

- **AuditLog**
  - LogId (PK, Identity)
  - TransactionId
  - RequestPayload (JSON)
  - ResponsePayload (JSON)
  - LogType
  - CreatedAt (default = GETDATE)
- **LoanApplicationStatus**
  - StatusId (PK)
  - StatusName (e.g., Pending, Validated, Rejected)

---

## Stored Procedures (SPs)

- **sp_InsertLoanApplication –** Inserts new loan application and returns ApplicationId.
- **sp_ApproveRejectLoan –** Updates application status (approve or reject).
- **sp_InsertAuditLog –** Logs request/response payloads for auditing.
- **sp_GetApplicationDetails –** Retrieves full details of a loan application with applicant information.