# **End-to-End Context Testing System for Financial Applications**

## **1. Overview**

This document provides a comprehensive explanation of the End-to-End Context Testing System for Financial Applications. The system is designed to automate test case generation and execution for a financial transaction processing application. It integrates a T5-based AI model, JIRA for feature descriptions, Flask for test case generation, Excel for test management, and Selenium for automated execution.

### **Business Use Case**

Financial applications, such as transaction fee calculators, require extensive testing to ensure accuracy, compliance, and security. Manually writing test cases for different scenarios is time-consuming and prone to human errors. This system automates the process, ensuring efficient and context-aware test case generation and execution, reducing manual effort, and improving software reliability.

## **2. System Architecture**

The system consists of the following key components:

### **1. Why T5 Model?**

* The **T5 (Text-to-Text Transfer Transformer)** model is a powerful NLP model that treats every problem as a text generation task. It is highly effective for generating structured output from text input.
* For this use case, T5 is ideal because it can learn the structure of financial test cases and generate diverse, context-aware test cases from feature descriptions.
* It ensures that test cases align with real-world financial scenarios, improving test coverage and accuracy.

### 

### 

### **2. Training the T5 Model**

* A T5-based model is trained on sample financial test cases.
* The model learns to generate context-aware test cases from feature descriptions.

### **3. Test Case Generation**

* A Flask application hosts the trained T5 model.
* Feature descriptions are provided in two ways:  
  + **Manual Input** – Users enter the feature description directly.
  + **JIRA Integration** – The system fetches feature descriptions from JIRA issues.
* The model generates unique test cases along with expected outcomes.

### **4. Exporting Test Cases to Excel**

* The generated test cases are exported to an Excel file.
* The Excel sheet stores test cases, expected outcomes.

### **5. Automated Test Execution with Selenium**

* A Selenium-based automation script monitors the Excel file for updates.
* The script performs the following:  
  + Logs into the sample financial application (Transaction Fee Calculator).
  + Executes test cases by interacting with the UI and API.
  + Validates responses against expected outcomes.
  + Updates test results (Pass/Fail) in the Excel file.

### 

### 

### **6. Sample Financial Application (Transaction Fee Calculator)**

* A Flask-based application that:  
  + Handles user authentication.
  + Processes currency exchange transactions.
  + Calculates transaction fees, exchange rates, and regulatory fees.
  + Provides an API (/calculate\_fee) for automated testing.

## **3. System Flow**

### **Step 1: Training the T5 Model**

* Train the T5-based model using sample test cases.
* The model learns patterns in test case generation.

### **Step 2: Generating Test Cases**

* The trained model is integrated into a Flask web app.
* Users enter feature descriptions manually or fetch them from JIRA.
* The model generates test cases automatically.

### **Step 3: Exporting Test Cases**

* Generated test cases are downloaded in an Excel file.

### **Step 4: Automating Test Execution**

* A Selenium script observes changes in the Excel file.
* It logs into the sample application and executes test cases.
* The script updates the test case status (Pass/Fail) in the Excel file.

## 

## **4. Technologies Used**

| **Component** | **Technology Used** |
| --- | --- |
| AI Model | T5 Transformer (Hugging Face) |
| Backend | Flask |
| Frontend | HTML, CSS |
| Test Case Management | Excel (Pandas) |
| JIRA Integration | JIRA API |
| Automation | Selenium, Watchdog |
| Sample App | Flask (Transaction Fee Calculator) |

## **5. Key Benefits**

✅ **Automated Test Case Generation** – AI-based test case generation reduces manual effort.

✅ **JIRA Integration** – Ensures real-time synchronization with product features.

✅ **Excel Export & Monitoring** – Simplifies test case management.

✅ **Automated Execution** – Selenium automated test case execution and validation.

✅ **Scalability** – The system can be extended to other financial applications.

## **6. Conclusion**

This End-to-End Context Testing System for Financial Applications streamlines the entire test automation lifecycle, from test case generation to execution. It ensures accuracy, efficiency, and seamless integration with financial application workflows.