UNPLUGG GROUP

PASSION MEETS TECHNOLOGY

Easypay Online Bill Payment Change Request Document

Author: Kholofelo Nkadimeng

Date: 07 October 2025

Version: 1.0

1. Introduction

1.1 Purpose

This Business Requirements Specification (BRS) document serves as a comprehensive blueprint for transitioning Unplugg's current payment integration with Easypay from a batch-based Secure File Transfer Protocol (SFTP) process to a more efficient, real-time online bill payment API integration. The purpose is to clearly define the business needs, challenges, and proposed solutions to guide the development team in implementing a robust, scalable, and accurate payment update system. This document aims to bridge the communication gap between stakeholders and developers by outlining functional and non-functional requirements, as well as integration considerations that ensure seamless data exchange and improved customer experience.

1.2 Scope

This document covers the end-to-end integration of payment transaction data from Easypay into Unplugg's policy management system, specifically:

- Replacing manual SFTP batch file exchanges with automated API-based real-time data transfer.
- Handling payment validations, including multiple payments and payment month.
- Enforcing business rules around policy statuses and payment amounts.
- Ensuring instant notification to policyholders upon payment updates. Excluded are unrelated internal policy system functionalities not impacted by payment updates or third-party systems outside Easypay.

1.3 Project Background

Unplugg currently manages payment statuses by sending payment updates in bulk batches through SFTP at scheduled intervals. This method has significant drawbacks, including delays in reflecting payment statuses, difficulties in handling failed transactions promptly, and manual intervention requirements to reconcile discrepancies. These issues not only affect operational efficiency but also impact customer experience negatively due to outdated payment statuses.

1.4 Definitions and Acronyms

SFTP: Secure File Transfer Protocol used in the current batch file transfer system.
API: Application Programming Interface facilitating real-time interaction between systems.
Payment Job: Automated task that downloads, processes, and uploads payment files.
Policy Status: Indicates whether a policy is active, on trial, lapsed, or cancelled.
SMS Notifications: Text messages sent to policyholders to confirm payment updates.

2. Business Objectives

Eliminate Latency in Payment Updates: Moving from batch updates that occur at intervals (hours or days) to near-instantaneous real-time updates, closing the gap between payment initiation and status reflection.
Increase Transaction Accuracy: Ensure that each payment status update is reliably delivered and recorded, reducing reconciliation errors.
Scale System Capacity: Build a solution capable of handling growing transaction volumes, accommodating business growth without performance degradation.
Enhance Customer Satisfaction: Provide customers and downstream systems with immediate access to their payment statuses, improving their overall experience and reducing support queries.
Reduce Operational Overhead: Automate status updates and error detection, minimizing manual reconciliations and monitoring efforts.

3. Current Integration Overview

The existing payment integration relies on batch processing using SFTP, which sends periodic files containing payments. This creates several problems:

- **Delayed Updates**: Policyholders and internal systems must wait until the next batch transmission to see payment results.
- **Error Handling Challenges**: Failures or anomalies are not detected promptly, causing delayed resolutions.
- **Scalability Issues**: As transaction numbers grow, batch files become larger and more cumbersome to process.
- **Manual Monitoring**: The system requires constant manual oversight to ensure batches are transmitted and received correctly.
- **Communication**: When payments are updated on policies, SMS is not automatically triggered to be sent to policyholders.

4. Proposed Solution

To address the above issues, the project proposes a real-time API integration that:

- Enables asynchronous, event-driven payment status updates initiated by Easypay's system.
- Reduces reliance on batch transmission, improving freshness of data.
- Implements secure, authenticated RESTful API calls for payment submissions and status queries.
- Supports callbacks from Easypay notifying Unplugg immediately about payment state changes.
- Incorporates retry and error handling mechanisms that automatically handle transient failures.
- Uses a modular architecture supporting future scalability and integration with additional payment services.

5. Stakeholders

- **Business Owners**: Responsible for defining business needs and prioritizing deliverables.
- **Development Team**: Tasked with designing, coding, testing, and deploying the API integration.
- **Operations Team**: Monitor system health, manage deployment, and support maintenance.
- **Customers**: End-users who will benefit from faster and more accurate payment information.
- **Security Officers**: Ensure compliance with payment security standards and privacy laws.

6. Business Requirements

6.1 Real-Time Payment Status Updates

The system must deliver payment status updates in real time, ensuring that all stakeholders, including customers and internal systems, receive immediate information about payment outcomes. This replaces the existing batch update method which causes delays.

6.2 Accuracy and Integrity of Payment Data

Payment data exchanged between Unplugg and Easypay must be accurate, complete, and protected from tampering during transmission and storage to ensure trustworthiness and regulatory compliance.

6.3 Scalability to Handle Growing Volumes

The solution must accommodate increasing transaction volumes without performance degradation, supporting business growth for the foreseeable future.

6.4 Security and Compliance Requirements

The system must comply with security standards such as PCI DSS and relevant data privacy laws, including end-to-end encryption, role-based access control, and audit logging capabilities.

6.5 Operational Efficiency

Reduction of manual reconciliation and monitoring by automating real-time updates, error detection, and notification processes.

6.6 Enhanced Customer Experience

Provide customers with up-to-date payment status information promptly, enabling improved transparency and support responsiveness.

6.7 Auditability and Reporting

Maintain comprehensive logs and reporting functions to track all payment transactions, status updates, and errors for audit and operational purposes.

7. Functional Requirements

- Payment Submission API
  - Provide a secure API endpoint for submitting payment details to Easypay in real time.
  - Validate incoming payment data before transmission to ensure integrity and completeness.
- Payment Status Query API
  - Allow Unplugg to query current payment statuses at any time through a dedicated API endpoint.
- Callback Notification Mechanism
  - Receive asynchronous real-time notifications from Easypay on payment status changes, including success, failure, and pending states.
  - Implement an endpoint to accept and process these callbacks reliably.
- Error and Exception Handling
  - Automatically retry failed transactions based on predefined rules.
  - Generate alerts and notifications for persistent errors.
  - Provide detailed error messages for diagnostic purposes.
- User Roles and Access Control
  - Define and implement role-based access levels for system administrators, developers, operations staff, and auditors.
  - Restrict API access and dashboard functions based on role.
- Reporting and Monitoring
  - Provide dashboards and reports detailing payment transactions, status trends, error logs, and system performance metrics.
  - Support export of reports for operational and compliance reviews.
- Integration with Existing Systems
  - Seamlessly integrate the new API with Unplugg backend systems including customer databases and payment reconciliation modules.

8. Non-Functional Requirements

- System Availability and Reliability
  - The system must provide 99.9% uptime, with planned maintenance windows structured to minimize disruption.
- Performance Metrics
  - API response times should be within 2 seconds under normal operational loads.
  - The system must be capable of processing thousands of concurrent transactions efficiently.
- Data Security
  - Encryption of data in transit using TLS 1.2 or higher.
  - Encryption of sensitive data at rest following best practices.
  - Logging and monitoring of all access and changes for audit trails.
- Backup and Disaster Recovery
  - Regular backups of critical data and system states.
  - Defined disaster recovery objectives (e.g., Recovery Time Objective and Recovery Point Objective) to restore service in the event of failure.
- Compliance
  - Adherence to relevant financial and data protection regulations such as PCI DSS, GDPR, and local payment laws.
- Scalability
  - Architecture designed to horizontally scale with transaction volumes and user load demands.
- Maintainability and Extensibility
  - Codebase and architecture must support easy maintenance and future enhancement including support for additional payment providers.

9. Assumptions and Constraints

- Easypay API and infrastructure will remain stable and supported throughout the project lifecycle.
- Network conditions must support reliable, low-latency communication between systems.
- Unplugg's existing infrastructure must be compatible or upgraded to consume the new API.
- Constraints include budget restrictions, project timelines, and regulatory compliance deadlines.

10. Integration Considerations

- The API will use RESTful design patterns with JSON message formats for interoperability.
- Secure OAuth2 or API key-based authentication methods will be implemented.
- Compatibility testing will be conducted to validate that existing systems properly handle new API responses.
- Rollback strategies will be established to revert to batch processing temporarily in case of critical failures.

11. Risk and Mitigation Plans

- **API Downtime**: Implement fallback batch processing and monitor API health continuously.
- **Data Mismatch**: Employ rigorous validation and logging for all transactions to detect discrepancies early.
- **Security Threats**: Apply comprehensive security audits, penetration testing, and data encryption.
- **Delays in Deployment**: Establish phased rollout and sufficient resources for parallel development and testing.

12. Document Approval

Title/Role	Names	Signature	Approval Date
Project Manager	Vusi Mngomezulu		
Business Analyst	Kholofelo Nkadimeng		
Managing Director	Jennifer Mngomezulu		
Software Developer	Charles Mafarachisi		
Software Developer	Tinashe Chiwaye		