MockPay — Payment Gateway Simulator Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

MockPay is a developer-focused payment gateway simulation platform that allows developers to test payment flows, webhooks, and failure scenarios without using real money. This document specifies the functional and non-functional requirements of the system.

1.2 Scope

The system will allow merchants to create accounts and projects, generate fake API keys, create and simulate payments, simulate success/failure/timeout scenarios, send webhooks, and provide an analytics dashboard.

1.3 Users

Target users include developers, students, and startup teams testing payment integrations.

2. Overall Description

MockPay consists of a web dashboard for merchants and a backend API that simulates a payment gateway. It supports project management, API key authentication, payment creation, webhook dispatching, and analytics.

3. Functional Requirements

3.1 Authentication System

FR-1: The system shall allow user signup and login.
FR-2: The system shall support JWT-based authentication.
FR-3: The system shall allow a user to create and manage multiple projects.

3.2 Project & API Key Management

FR-4: The user shall be able to create projects.
FR-5: The system shall generate a unique API key per project.
FR-6: The API key shall be required for all API calls and stored securely (hashed).

3.3 Payment Creation API

FR-7: The system shall expose an endpoint POST /api/payments/create.
FR-8: The API shall accept amount, currency, customer_email, and redirect_url.
FR-9: The system shall return payment_url and payment_id.

3.4 Hosted Payment Page

FR-10: The payment URL shall open a hosted payment page.
FR-11: The user shall be able to choose the outcome: success, failure, or timeout.

3.5 Payment Processing

FR-12: The system shall update and persist the payment status.
FR-13: The system shall store all payments in the database.

3.6 Webhook System

FR-14: The merchant shall be able to configure a webhook URL.
FR-15: The system shall send a POST request to the webhook on payment status change.
FR-16: The webhook shall include a signature header.
FR-17: The system shall retry failed webhook deliveries.

3.7 Dashboard & Analytics

FR-18: The user shall see total volume, success rate, and failure rate.
FR-19: The user shall be able to filter analytics by date range.

4. Non-Functional Requirements

NFR-1: The system shall respond to API calls within 300 ms under normal load.
NFR-2: The system shall securely store API keys.
NFR-3: The system shall support at least 1000 concurrent payments.
NFR-4: The system shall log all significant events.

5. Data Models

User(id, email, password_hash)
Project(id, user_id, name)
ApiKey(id, project_id, hash)
Payment(id, project_id, amount, status, created_at)
Webhook(id, project_id, url, secret)

6. System Architecture

The system consists of a React-based dashboard, a Node.js API server, a database, and a webhook dispatcher module. The frontend communicates with the backend API, which stores data in the database and dispatches webhooks to merchant servers.