Low Level Design Document

Introduction

This Low Level Design (LLD) document outlines the core implementation details for **SecurePay - Payment Security Simulator**. The platform enables finance professionals to simulate, test, and improve payment system security through real-world attack scenarios, secure authentication, and analytics. The design emphasizes modularity, security, and extensibility using Java, Spring, PostgreSQL, React, and OWASP best practices.

---

1. System Components

Component	Technology	Key Responsibilities
Web Frontend	React	UI, scenario selection, analytics visualization
API Gateway	Spring Boot	Route requests, authentication, rate limiting
Simulation Engine	Java/Spring	Run attack scenarios, manage simulation state
Auth Service	Spring Security	User authentication (2FA, OAuth2), session mgmt
Analytics Service	Java/Spring	Collect, process, and serve simulation analytics
Database	PostgreSQL	Persist users, scenarios, results, logs

---

2. Class/Interface Overview

Class/Interface	Description	Key Methods/Attributes
`UserController`	User endpoints (register, login)	`register()`, `login()`, `logout()`
`SimulationController`	Manage simulations	`startSimulation()`, `getStatus()`
`ScenarioService`	Scenario logic/management	`runScenario()`, `getScenarios()`
`AuthService`	Auth logic (2FA, OAuth2)	`authenticate()`, `generateToken()`
`AnalyticsService`	Analytics aggregation	`recordEvent()`, `getReport()`
`User` (Entity)	User data model	`id`, `username`, `passwordHash`
`Scenario` (Entity)	Attack scenario model	`id`, `name`, `description`, `steps`
`SimulationResult` (Entity)	Simulation result data	`id`, `userId`, `scenarioId`, `score`

Relationships:

- `User` ↔ `SimulationResult` (1:N)
- `Scenario` ↔ `SimulationResult` (1:N)

- Controllers use Services; Services access Entities via Repositories

---

3. Data Structure Overview

Table/Model	Fields
`users`	`id`, `username`, `password_hash`, `email`, `roles`
`scenarios`	`id`, `name`, `description`, `steps` (JSON)
`simulation_results`	`id`, `user_id`, `scenario_id`, `score`, `timestamp`
`events`	`id`, `simulation_id`, `event_type`, `details`, `ts`

---

4. Algorithms/Logic

Simulation Execution Flow (Pseudocode):

```
function runSimulation(userId, scenarioId):
    scenario = scenarioRepo.findById(scenarioId)
    state = initializeState(scenario)
    for step in scenario.steps:
        result = executeStep(step, state)
        if result.isAttackSuccess():
            logEvent(userId, scenarioId, "AttackSuccess", result.details)
            break
    score = calculateScore(state)
    saveSimulationResult(userId, scenarioId, score)
    return score
```

Authentication Flow (Summary):

- User submits credentials
- System validates password hash
- If 2FA enabled, verify OTP
- On success, issue JWT token

---

5. Error Handling

Scenario	Handling Approach
Invalid credentials/auth failure	Return 401 Unauthorized, log attempt
Simulation step error	Log error, abort simulation, return error status
DB connection failure	Retry (max 3), else return 503 Service Unavailable
Invalid scenario input	Return 400 Bad Request with validation message
Analytics/reporting failure	Return partial data, log error

End of Document