**Assignment Submission – Internship Case Study**

**Applicant: Vanraj Jhala**
**Date: 18 September 2025**
**Position: Software Development Engineering (Web) Intern**

---

## 1. Introduction

The given case study requires evaluating different backend system architectures and identifying viable options that balance scalability, cost-efficiency, and maintainability. The goal is to propose solutions that can handle user growth, ensure reliability, and support future system expansion.

---

## 2. Viable Options

### Option 1: Monolithic Backend (Node.js + MySQL)

A single backend service handling all APIs, business logic, and database operations.

- Pros: Easy to build, simple deployment, low cost.
- Cons: Limited scalability, single point of failure.
- Feasibility: Works well for small systems and MVPs.

---

### Option 2: Microservices Architecture

Independent services for users, rewards, pricing, and ledger.

- Pros: Scalable, modular, fault-tolerant.
- Cons: Complex setup, requires DevOps, higher initial effort.
- Feasibility: Suitable for large-scale platforms with high traffic.

---

**Option 3: Serverless Backend**

Cloud functions (AWS Lambda, Firebase Functions) with managed databases.

- Pros: Auto-scaling, pay-per-use, no server management.
- Cons: Cold starts, vendor lock-in, limited control.
- Feasibility: Best for quick prototypes and low-maintenance systems.

---

**Option 4: Hybrid Approach (Recommended)**

Start with a modular monolith but design it so components can evolve into microservices later.

- Pros: Fast development, cost-effective, clear migration path.
- Cons: Requires careful modular design.
- Feasibility: Balanced choice for startups—launch quickly, scale later.

---

**3. Recommendation**

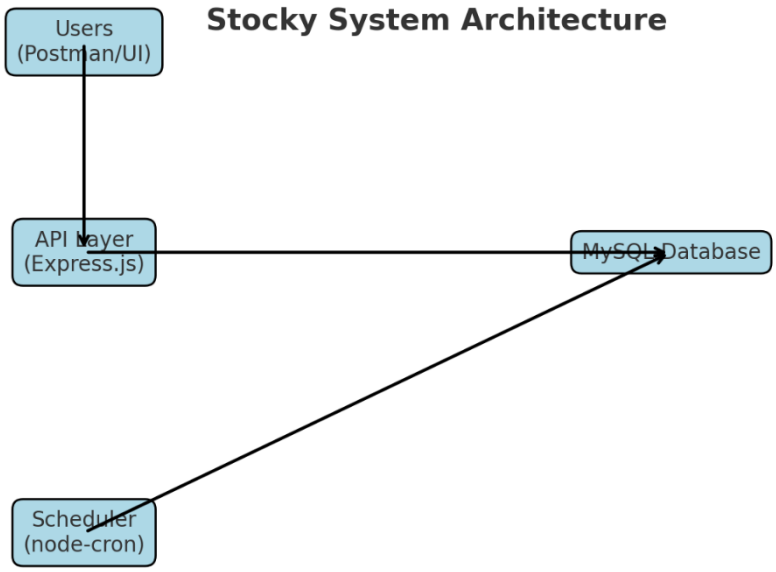
After comparing the options, the Hybrid Approach (Option 4) is the most practical. It allows for quick delivery with low complexity at the beginning while ensuring the system can transition smoothly into microservices as user load grows. This ensures time-to-market advantage without sacrificing future scalability.

---

**4. Conclusion**

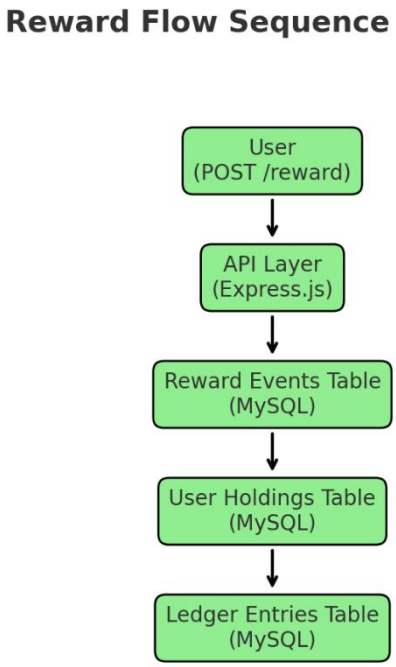
In conclusion, the recommended solution balances simplicity, performance, and scalability. By starting with a modular monolith and preparing for eventual service separation, the system will remain efficient in the short term while being adaptable for future growth. This approach offers the best mix of reliability and flexibility for the case study requirements.

**System architecture diagram**




- **Users (Postman/UI)** → interact with your APIs
- **API Layer (Express.js)** → processes requests & responses
- **MySQL Database** → stores users, rewards, holdings, price history, ledger
- **Scheduler (node-cron)** → auto-updates stock prices every hour

**Sequence diagram for reward flow**




1. **User** calls POST /reward
2. **API Layer (Express.js)** handles the request
3. Record is added to **Reward Events**
4. Update **User Holdings** (increase stock count)
5. Insert **Ledger Entries** (double-entry: StockAssets, Expenses, Cash)