Section 1: System Design & Problem Solving

1. **Web & Mobile Architecture Design**

You are tasked with designing a multi-platform system where a mobile app and a web dashboard communicate with a shared backend. How would you structure the architecture to ensure:

■ Scalability (handling a 10x traffic increase)

■ Security (preventing unauthorized access & data breaches)

■ Maintainability (allowing multiple developers to collaborate efficiently)

Answer :

Multi-Platform System Design (Web + Mobile with Shared Backend)

1. Architecture Overview

- [Client Layer]
  - Mobile App (Ionic/iOS/Android)
  - Web Dashboard
- [API Layer]
  - API Gateway
  - Load Balancer
- [Service Layer]
  - Auth Service (Laravel)
  - Business Logic Services
  - Async Workers
- [Data Layer]

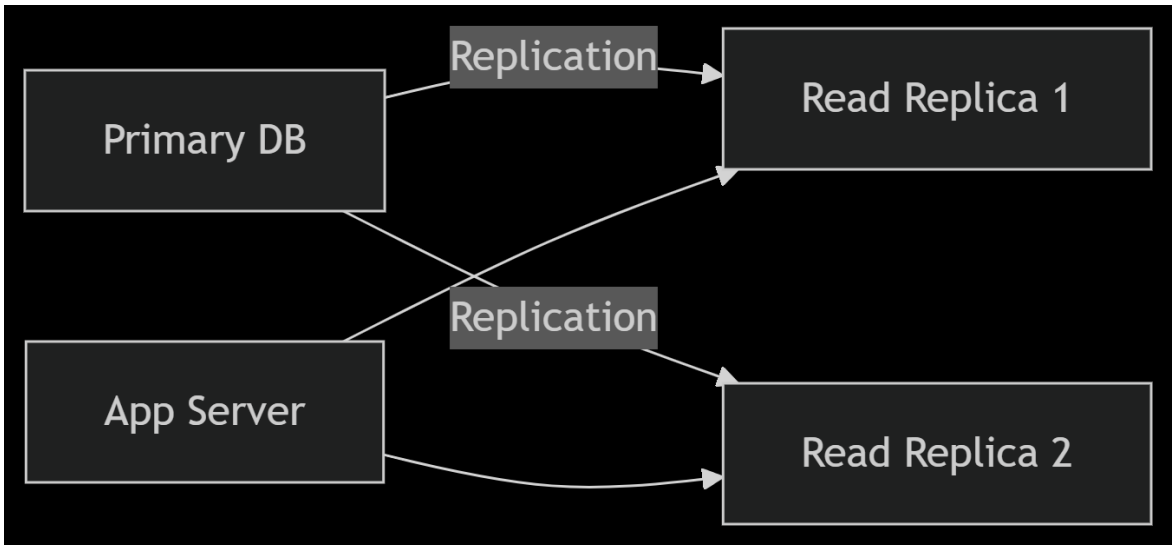
Primary DB (PostgreSQL)

Read Replicas

2. Scalability Design

2.1 Horizontal Scaling

- Stateless API Servers :
    - Containerize Laravel with Docker
    - Use Kubernetes for orchestration
    - Automatically scale based on CPU and RAM metrics
- Database Scaling :



Why Replicas Solve Scaling Issues

1. Handling Read Traffic (The 80/20 Rule)

2. Performance Boost

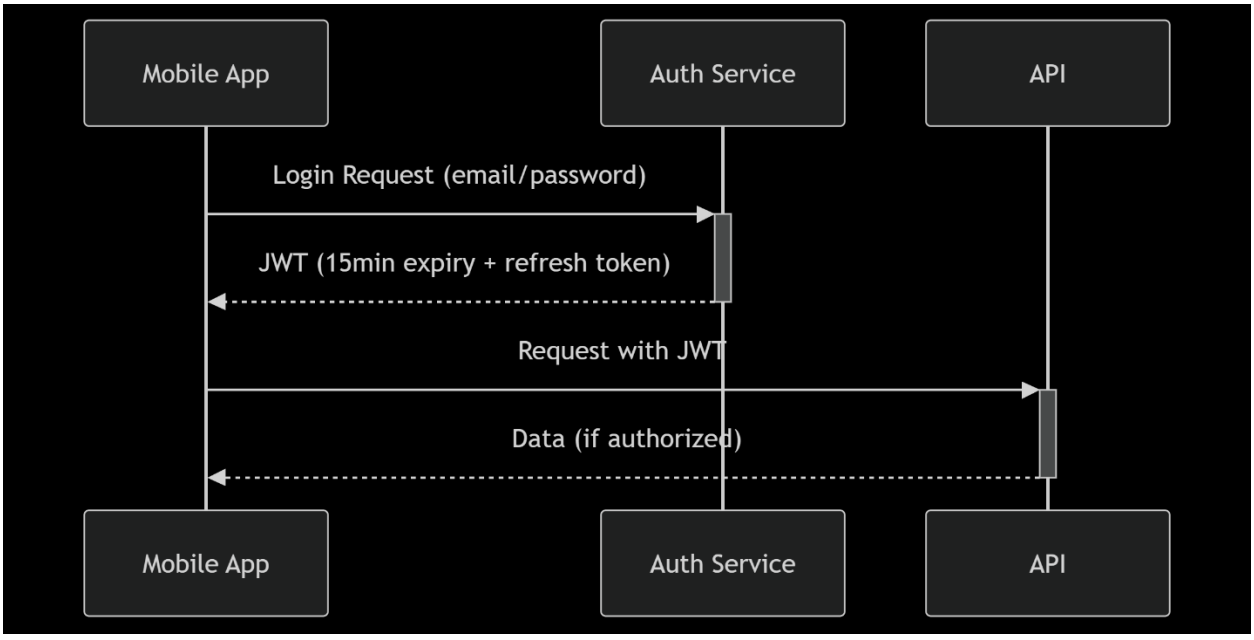
3. High Availability

2.2 Performance Optimization

- Caching Strategy :
    - Redis for :
        - Session storage
        - API response cache (60s TTL)
        - Frequent queries
- CDN Setup :
    - Cloudflare for static assets
    - Edge caching for media files

3. Security Implementation

3.1 Authentication Flow



3.2 Key Security Measures

- Data Protection :
    - TLS 1.3 everywhere
    - PII encryption at application level
    - Regular penetration testing
- API Security :
    - Rate limiting (100 requests/min per IP)
    - Request signing for critical endpoints
    - HSTS headers

4. Maintainability Approach

Create Development Workflow

1. Feature branches (Git Flow)
2. PR reviews required
3. Automated testing :
    - Unit tests (PHPUnit/Jest)
4. CI/CD pipeline :

2. **API Design & Optimization**

Design a REST API for securely processing NFC-based transactions. Your design must address:

■ Authentication using JWT tokens.

■ Data validation to prevent injection attacks.

■ Optimized querying for large transaction data sets.

Answer :

Implemented in project

1. **JWT Works?** ✅
    - Blocks unauthorized access.
2. **Validation Secure?** ✅
    - Blocks SQL/XSS, enforces correct types.
3. **Queries Optimized?** ✅
    - Uses paginate(), indexes, and avoids SELECT *.

3. **Performance Optimization Challenge**

You notice that an application's response time slows down significantly when handling large data queries. What steps would you take to:

■ Identify the root cause?

■ Optimize database queries?

■ Reduce load on the server while maintaining fast response times?

Answer :

1. Identify the Root Cause

- Laravel Telescope : Inspect slow queries in the "Queries" tab.
- Query Logging

2. Optimize Database Queries

- Eager Loading (Laravel Specific)
- Avoid "SELECT *", select only needed column
- Avoid Subqery, prefers JOIN or EXISTS
- Cursor for Memory Efficiency

2. Reduce Server Load

- Use Database Replica
- Queue Heavy Operations

4. **The system needs to process high-volume NFC transactions in real time. How would you:**

○ Optimize the Laravel backend for performance?

○ Minimize API response times in the mobile app?

Answer :

1. **Backend (Laravel)**

- Use queues (Redis) for async processing
- Database: Read replicas + time-series partitioning
- Cache NFC tags/balances in Redis
- JWT with short expiry + refresh tokens

2. **System Design**

- Eventual consistency for transaction status
- Rate limiting + circuit breakers
- Distributed tracing for latency monitoring
- Idempotency keys for duplicate requests
- Offline-first with local transaction queue (Mobile Implementation)
- Predictive pre-fetching of user data (Mobile Implementation)

5. **DevOps & CI/CD Strategy**

Your team is pushing updates frequently. How would you set up a secure, automated deployment pipeline (CI/CD) that ensures:

■ Zero downtime

■ No breaking changes in production

■ Easy rollback in case of issues

Answer :

Pipeline Architecture

Git Flow + Trunk-Based Development:

- main branch = production (protected)
- Feature flags for new functionality
- Automated versioning and tagging

Zero Downtime

Setup Deployment Script (using Laravel Forge) :

- php artisan down --render="maintenance"
- git pull origin main
- composer install --no-dev --optimize-autoloader
- php artisan migrate --force
- npm run production
- php artisan up
- php artisan opcache:clear

Or using Kubernetes

```
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 25%
    maxUnavailable: 0  # Ensures 100% uptime
```

Breaking Change Prevention

- API Contract Tests: Validate OpenAPI/Swagger specs in CI
- Database Check: php artisan migrate:status --env=testing in pre-deploy
- Laravel Pulse: Real-time monitoring post-deploy

Instant Rollback Plan

- Keep previous release folder
- Use Laravel Forge : forge rollback my-app --server=123