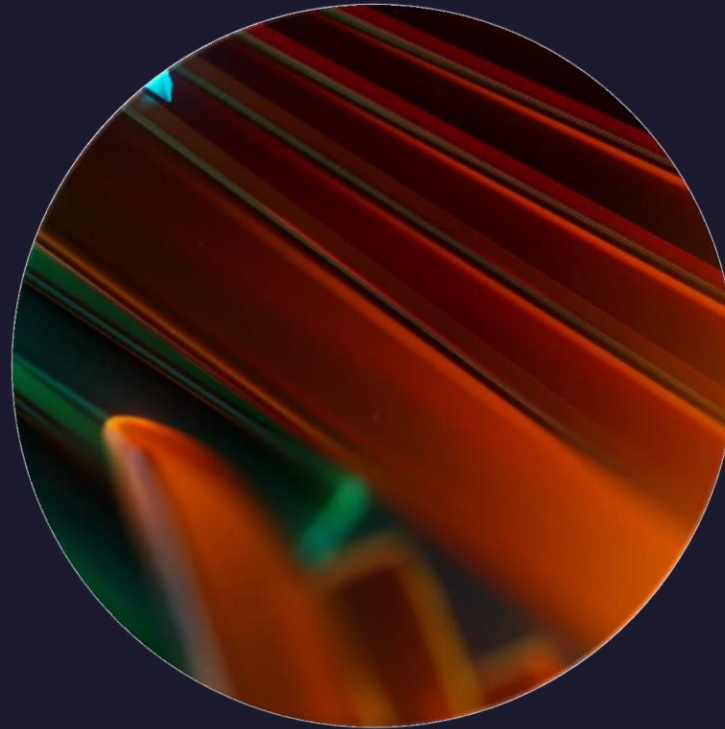


# Number Verification Microservice

Design and Implementation

# Agenda

1. Overview
2. Architecture & Design
3. Technology Stack
4. API Implementation
5. Security
6. Observability & Monitoring
7. Testing
8. Deployment
9. Management
10. Q&A





# 1. Overview

Business context, business value, use cases, requirements



# Overview – Business context

- Digital identity verification is critical for modern applications
- Phone number verification provides a secure authentication layer
- Reduces fraud and ensures legitimate user access
- Seamless integration with existing user journeys

# Overview – Business value

- Reduced Fraud: 60% decrease in account takeovers
- Improved UX: 45% faster authentication vs. traditional SMS OTP
- Cost efficiency: 30% reduction in SMS verification costs
- Regulatory compliance: meets KYC requirements for financial services

# Overview – Use cases

- Account registration: verify phone numbers during sign-up to prevent fraud
- Transaction authentication: add security layer for high-value transactions
- Multi-factor authentication: strengthen security with network-verified identities
- Password recovery: ensure recovery requests come from legitimate device owners

# Overview – Requirements

- Implement Number Verification CAMARA API
- Two key endpoints:
  - POST /verify (validate user's phone number)
  - GET /device-phone-number (retrieve phone number from device)
- Focus on security, logging, observability, and monitoring
- Deployable microservice architecture

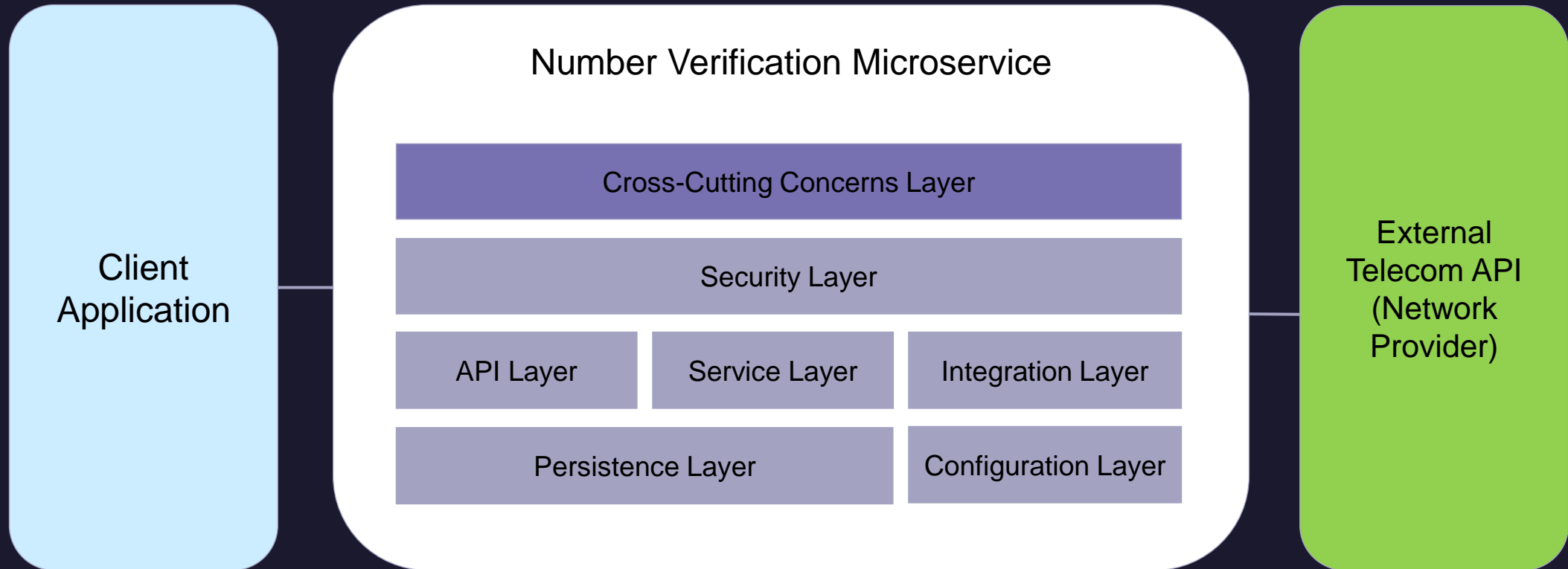


## 2. Architecture & Design

High-level architecture, components breakdown, calling sequence



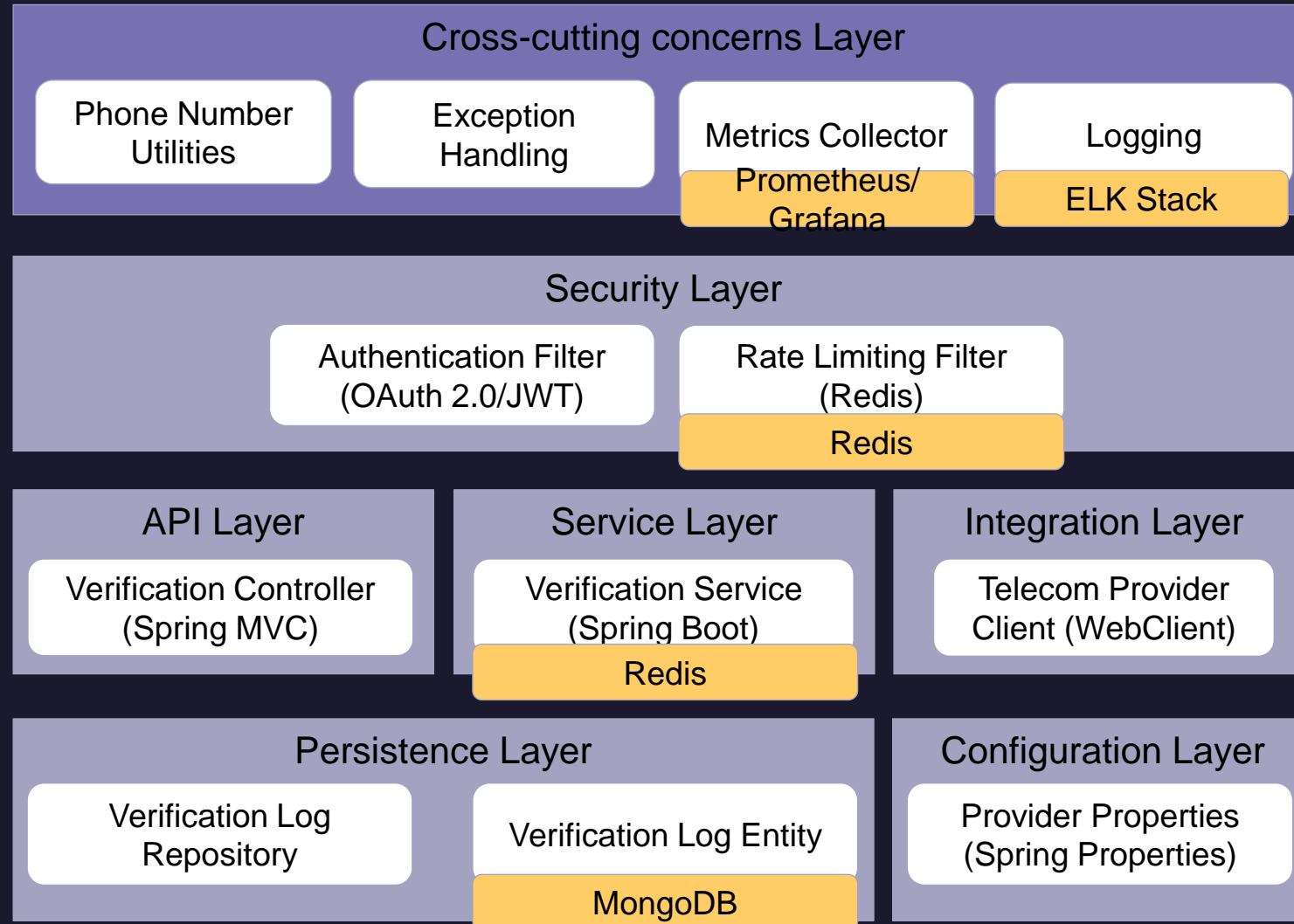
# Architecture & Design – High-level architecture



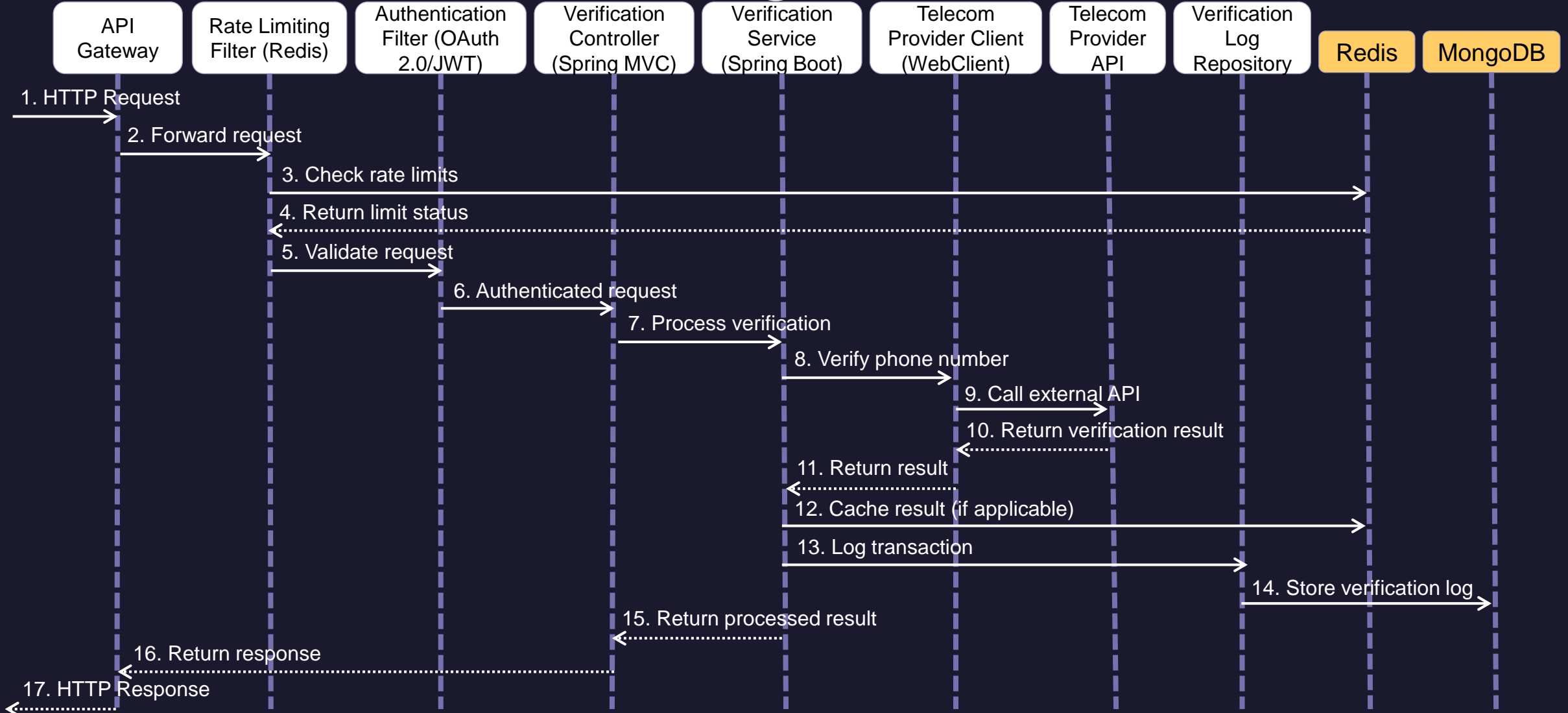
# Architecture & Design – Components breakdown

- API layer: REST controllers, request handling, validation
- Service layer: business logic, data transformation, coordination
- Integration layer: telecom API communication, resilience
- Persistence layer: logging, audit trails, data access
- Configuration layer: properties, environment settings
- Security layer: authentication, authorization, rate limiting
- Cross-cutting concerns layer: logging, metrics, common utilities

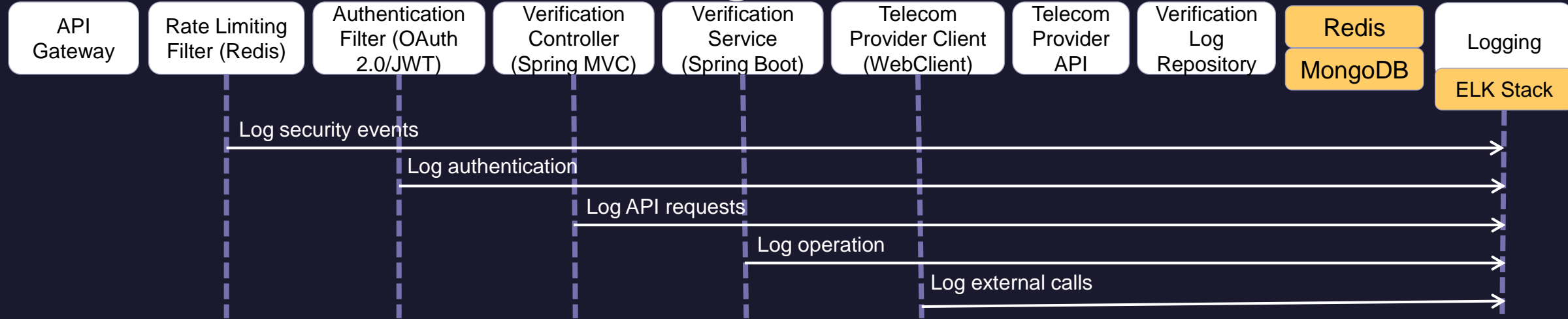
# Architecture & Design – Components architecture



# Architecture & Design – Calling sequence (POST)



# Architecture & Design – Calling sequence (logs)







# 3. Technology Stack





Core technologies & supporting technologies

# Technology Stack – Core technologies

- Language: Java 17
- Framework: Spring Boot 3.2
- API Documentation: OpenAPI 3.0
- Build Tool: Gradle 8.x
- Container: Docker
- Testing Framework: JUnit 5 + Mockito + JMeter 5.6



# Technology Stack – Supporting Technologies

- Database: MongoDB 6.0 (log repository) 
- Cache: Redis 7.0 (rate limiting filter, caching in service layer) 
- Authentication: OAuth 2.0 / JWT (authentication filter, token validation)
- Metrics: Micrometer + Prometheus (metrics collection, performance monitoring)
- Logging: Logback + ELK Stack (centralized log) 
- Monitoring: Grafana (dashboard, alerting) 
- Security: Spring Security 6.x (security filter chain)
- Rate Limiting: Bucket4j (rate limiting filter)



# 4. API Implementation

POST /verify & GET /device-phone-number, provider integration

# API Implementation – POST /verify

Validate if provided phone number matches user's device

## Request:

```
{  
  "phoneNumber": "+12345678901",  
  "correlationId": "abc-123-def-456"  
}
```

## Response:

```
{  
  "verificationId": "ver-12345",  
  "status": "MATCH",  
  "verificationTime": "2025-05-15T08:10:56Z"  
}
```

## Error Handling:

- 400: Invalid phone number format
- 401: Unauthorized request
- 429: Rate limit exceeded
- 500: Internal service error
- 503: External provider unavailable



# API Implementation – GET /device-phone-number

Retrieve phone number associated with user's device

Request:

`GET /device-phone-number`

Response:

```
{  
  "phoneNumber": "+34698765432",  
  "retrievalTime": "2023-06-30T12:34:56Z"  
}
```

Error Handling:

- 400: Invalid phone number format
- 401: Unauthorized request
- 500: Internal service error
- 503: External provider unavailable

# API Implementation – Provider integration

- Adapter pattern: standardized interface supporting multiple telecom providers
- Failover strategy: primary/secondary provider configuration
- Circuit breaker: prevents cascading failures during provider outages
- Retry policy: configurable exponential backoff for transient failures



# 5. Security

Authentication & authorization, data protection, threat mitigation

# Security – Authentication & authorization

- Client authentication via API keys or OAuth 2.0
- Rate limiting to prevent abuse
- Role-based access control for different API operations

# Security – Data protection

- TLS/SSL encryption for all communications
- Phone number hashing/tokenization for storage
- PII redaction for compliance
- Zero retention policy for sensitive data
- GDPR compliance measures



# Security – Threat mitigation

- Input validation and sanitization
- Protection against common attacks (SQL injection, XSS, CSRF)
- Regular security scanning and penetration testing



# 6. Observability & Monitoring

Logging strategy, metrics collection

# Observability & Monitoring – Logging strategy

- Structured JSON logging
- Correlation IDs across service boundaries
- Log levels (DEBUG, INFO, WARN, ERROR)

# Observability & Monitoring – Metrics collection

- Request count, latency, and error rates
- System metrics (CPU, memory, disk)
- Business metrics (verification success rate)
- Grafana dashboards for visualization
- Alerting thresholds for critical metrics
- On-call rotation and escalation policies



# 7. Testing

Process & metrics



# Testing – Processes

- Dev → QA → Staging → Pre-Production
- Automated tests on every commit: JUnit 5
- Nightly performance testing: JMeter
- Weekly security scans: OWASP Dependency-Check, Snyk, SonarQube
- Mock services for development and testing: Mockito
- End-to-end tests with real sandbox: Postman/Newman
- Chaos testing to validate provider disruptions: Resilience4j

# Testing – Metrics

- Response time: <200ms (p95) under normal load
- Throughput: 500+ requests/second peak handling
- Availability: 99.9% uptime target



# 8. Deployment

CI/CD pipeline, infrastructure as code, scaling & resilience

# Deployment – CI/CD pipeline

- Automated testing (unit, integration, contract, performance)
- Continuous integration with GitHub actions
- Automated deployment

# Deployment – Infrastructure as code & scaling

- Docker containerization
- Kubernetes for orchestration (future scaling)
- Terraform for infrastructure provisioning
- Horizontal scaling capabilities



# 9. Management

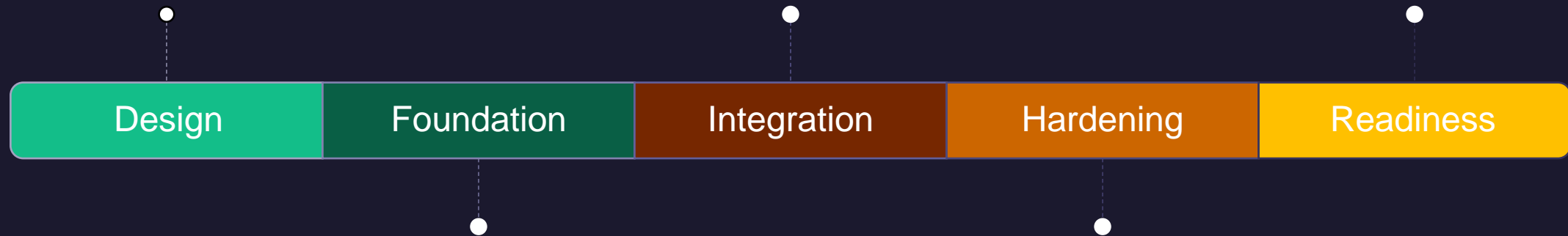
Schedule & tasks, risks

# Management – Schedule & tasks

Week 1: technical blueprint: architecture, API contracts, security planning, and technology validated through stakeholder review.

Weeks 4-5: complete functionality through parallel development and testing, connecting all components including provider integration.

Week 7: deploy to production with monitoring, controlled rollout, and post-deployment verification.



Weeks 2-3: technical infrastructure: environments, pipelines, project structure, and core components to enable efficient development.

Week 6: system quality validation through testing, optimization, and documentation to ensure production readiness.

# Management – Risks

- High Impact, High Likelihood
  - Telecom API Integration
  - Security Vulnerabilities
- High Impact, Medium Likelihood
  - Performance Bottlenecks
  - Regulatory Compliance
  - Timeline Pressure
- Low Impact, High Likelihood
  - Scaling Challenges
- High Impact, Low Likelihood
  - Resilience Failures
  - Deployment Issues
- Medium Impact, Medium Likelihood
  - Technology Stack
  - Monitoring Gaps
  - Cost Management
- Low Impact, Medium Likelihood
  - Security Requirements





# Q&A

May 15th, 2025



# Thank You

Sergio Saraiva

[sergio.saraiva@gmail.com](mailto:sergio.saraiva@gmail.com)