FinSolve Technologies Engineering Document

1. Introduction

1.1 Company Overview

FinSolve Technologies is a leading FinTech company headquartered in Bangalore, India, with operations across North America, Europe, and Asia-Pacific. Founded in 2018, FinSolve provides innovative financial solutions, including digital banking, payment processing, wealth management, and enterprise financial analytics, serving over 2 million individual users and 10,000 businesses globally.

1.2 Purpose

This engineering document outlines the technical architecture, development processes, and operational guidelines for FinSolve's product ecosystem. It serves as a comprehensive guide for engineering teams, stakeholders, and partners to ensure alignment with FinSolve's mission: "To empower financial freedom through secure, scalable, and innovative technology solutions."

1.3 Scope

This document covers:

- System architecture and infrastructure
- Software development lifecycle (SDLC)
- Technology stack
- Security and compliance frameworks
- Testing and quality assurance methodologies
- Deployment and DevOps practices
- Monitoring and maintenance protocols
- Future technology roadmap

1.4 Document Control

Version	Date	Author	Changes
1.0	2025-05-01	Engineering Team	Initial version
1.1	2025-05-14	Tech Architecture Council	Updated diagrams and monitoring section

2. System Architecture

2.1 Overview

FinSolve's architecture is a microservices-based, cloud-native system designed for scalability, resilience, and security. It leverages a modular design to support

rapid feature development and seamless integration with third-party financial systems (e.g., payment gateways, credit bureaus, regulatory reporting systems).

2.2 High-Level Architecture

```
[Client Apps]
     Mobile Apps (iOS, Android)
     Web App (React)
     APIs (REST, GraphQL)

[API Gateway]
     AWS API Gateway (Routing, Authentication, Rate Limiting)

[Microservices Layer]
     Authentication Service (OAuth 2.0, JWT)
     Payment Processing Service
     Wealth Management Service
     Analytics Service
     Notification Service

[Data Layer]
     PostgreSQL (Transactional Data)
     MongoDB (User Profiles, Metadata)
     Redis (Caching, Session Management)
     Amazon S3 (Documents, Backups)

[Infrastructure]
     AWS (EC2, ECS, Lambda)
     Kubernetes (Orchestration)
     Cloudflare (CDN, DDoS Protection)
```

2.3 Key Components

2.3.1 Client Applications

- **Mobile Apps**: Native mobile applications developed using Swift (iOS) and Kotlin (Android), providing a seamless user experience with biometric authentication, push notifications, and offline capabilities.
- **Web Application**: A responsive Single Page Application (SPA) built with React, Redux, and Tailwind CSS, optimized for various screen sizes and compliant with WCAG 2.1 accessibility standards.
- **API Interfaces**: RESTful and GraphQL APIs enabling third-party integrations, partner systems, and future expansions.

2.3.2 API Gateway

- Centralized entry point for all client requests

- Implements authentication, authorization, and rate limiting
- Provides API versioning and documentation via Swagger/OpenAPI
- Handles request logging and basic analytics
- AWS API Gateway with custom Lambda authorizers for sophisticated permission models

2.3.3 Microservices

- **Authentication Service**: Manages user identity, authentication (OAuth 2.0), and authorization using JWT tokens. Supports multi-factor authentication and Single Sign-On (SSO).
- **Payment Processing Service**: Handles domestic and international payment transactions, recurring payments, and reconciliation with multiple payment gateways.
- **Wealth Management Service**: Provides portfolio management, investment recommendations, and financial goal tracking.
- **Analytics Service**: Processes user financial data to deliver insights, spending patterns, and budgeting recommendations.
- **Notification Service**: Manages push notifications, emails, and SMS alerts based on user preferences and system events.

2.3.4 Data Layer

- **PostgreSQL**: Primary relational database for transactional data requiring ACID compliance.
- **MongoDB**: NoSQL database storing user profiles, preferences, and semi-structured data.
- **Redis**: In-memory data store for caching, session management, and pub/sub messaging between services.
- **Amazon S3**: Object storage for documents, statements, user uploads, and encrypted backups.

2.3.5 Infrastructure

- **AWS**: Primary cloud provider utilizing EC2, ECS, Lambda, RDS, S3, CloudFront, and other managed services.
- **Kubernetes**: Container orchestration platform managing microservices deployment, scaling, and failover.
- **Cloudflare**: Content Delivery Network (CDN) and security layer providing DDoS protection, Web Application Firewall (WAF), and edge caching.

2.4 Scalability Architecture

2.4.1 Horizontal Scaling

- Kubernetes Horizontal Pod Autoscaler (HPA) automatically scales services based on CPU/memory metrics and custom metrics (e.g., queue length).

- Auto-scaling groups for EC2 instances in the underlying infrastructure.
- Microservices designed to be stateless, enabling seamless scaling.

2.4.2 Database Scalability

- PostgreSQL uses range-based sharding for high-volume transactional tables.
- Read replicas for analytics and reporting workloads.
- MongoDB sharding for user data distribution across multiple clusters.
- Database connection pooling via PgBouncer to optimize connection management.

2.4.3 Caching Strategy

- Multi-level caching architecture:
  - Application-level caching with Redis
  - API Gateway response caching
  - CDN caching for static assets
  - Database query result caching
- Cache invalidation using event-based triggers and time-to-live (TTL) policies.

2.5 Resilience and Fault Tolerance

2.5.1 High Availability

- Multi-Availability Zone (AZ) deployments in AWS regions.
- Active-active configurations for critical services.
- Database replication with automated failover capabilities.
- Global load balancing for geographic redundancy.

2.5.2 Circuit Breakers

- Implemented using Istio service mesh to prevent cascading failures.
- Configurable thresholds for error rates and latency.
- Fallback mechanisms for degraded service modes.

2.5.3 Disaster Recovery

- Regular backups to Amazon S3 with versioning enabled.
- Cross-region replication for critical data.
- Recovery Time Objective (RTO) of 4 hours.
- Recovery Point Objective (RPO) of 15 minutes.
- Quarterly disaster recovery drills and documentation.

2.5.4 Data Consistency

- Event sourcing patterns for critical financial transactions.

- Saga pattern for distributed transactions across microservices.
- Eventual consistency with compensation transactions where appropriate.

3. Technology Stack

3.1 Comprehensive Technology Matrix

Layer	Primary Technologies	Supporting Technologies	Testing Tools
Frontend	React 18, Redux Toolkit, Tailwind CSS	TypeScript, React Query, D3.js	Jest, React Testing Library, Cypress
Mobile	Swift 5.5 (iOS), Kotlin 1.6 (Android)	SwiftUI, Jetpack Compose	XCTest, Espresso, Appium
Backend	Node.js 18 LTS, Python 3.11 (FastAPI), Go 1.19	Express.js, Pydantic, Gin	Jest, Pytest, Go test
APIs	REST, GraphQL, gRPC	OpenAPI, Apollo Server, Protocol Buffers	Postman, GraphQL Playground
Database	PostgreSQL 15, MongoDB 6.0, Redis 7.0	TimescaleDB, Mongoose, Jedis	TestContainers, MongoDB Memory Server
Infrastructure	AWS, Kubernetes 1.25+	Terraform, Helm, Kustomize	InSpec, Terratest
CI/CD	Jenkins, GitHub Actions, ArgoCD	SonarQube, Nexus, Harbor	JUnit, pytest
Monitoring	Prometheus, Grafana, ELK Stack	Jaeger, Kiali, Fluentd	Synthetic monitoring, Chaos Monkey
Security	OAuth 2.0, JWT, AWS WAF, Cloudflare	Vault, CertManager, OPA	OWASP ZAP, Snyk

3.2 Technology Selection Criteria

- **Performance**: Technologies that deliver sub-200ms response times for critical paths.
- **Scalability**: Ability to handle projected growth (10x in 3 years).
- **Maturity**: Preference for well-established technologies with active communities.
- **Security**: Strong security models and regular security updates.

- **Developer Experience**: Tools that enhance productivity and reduce bugs.
- **Cost Efficiency**: Balance between performance and operational costs.

3.3 Version Control and Management

- All dependencies are locked to specific versions.
- Dependency upgrade schedule: Security patches (immediate), Minor versions (monthly), Major versions (quarterly).
- Automated vulnerability scanning of dependencies using Snyk and Dependabot.

4. Software Development Lifecycle (SDLC)

4.1 Agile Methodology

FinSolve follows a Scrum-based Agile process with 2-week sprints:

4.1.1 Scrum Ceremonies

- **Sprint Planning**: Product owners and engineering leads define sprint goals and prioritize tasks (4 hours).
- **Daily Standups**: 15-minute meetings to track progress and address blockers.
- **Sprint Review**: Demo of completed features to stakeholders (2 hours).
- **Sprint Retrospective**: Team discusses improvements for the next sprint (1.5 hours).

4.1.2 Roles and Responsibilities

- **Product Owner**: Maintains product backlog, sets priorities, accepts stories.
- **Scrum Master**: Facilitates ceremonies, removes impediments, coaches team.
- **Development Team**: Self-organizes to deliver sprint commitments.
- **Technical Lead**: Ensures technical excellence and architectural consistency.

4.2 Development Workflow

4.2.1 Requirements Engineering

- Product managers create user stories in Jira following the format: "As a [user role], I want [feature] so that [benefit]."
- Acceptance criteria defined using Gherkin syntax (Given-When-Then).
- Engineering leads validate technical feasibility and estimate complexity using story points (Fibonacci sequence).

- Definition of Ready (DoR) checklist ensures stories are fully specified before development.

4.2.2 Design Phase

- Architects create technical designs using UML diagrams and C4 model documentation.
- API specifications defined using OpenAPI/Swagger with clear request/response examples.
- UI/UX designs created in Figma with component-based architecture.
- Design reviews conducted with senior engineers and stakeholders.

4.2.3 Coding Standards

- Language-specific style guides enforced via linters:
  - JavaScript/TypeScript: ESLint with Airbnb configuration
  - Python: Black formatter and Flake8
  - Go: gofmt and golint
  - SQL: pgFormatter
- Documentation requirements:
  - Public APIs must have complete documentation
  - Complex algorithms require explanatory comments
  - README.md files for all microservices

4.2.4 Code Review Process

- Pull requests require at least two approvals:
  - One from a peer engineer
  - One from a senior engineer or technical lead
- Automated checks must pass before code review:
  - Linting and style validation
  - Unit test coverage (minimum 85%)
  - No security vulnerabilities (via Snyk)
- Review guidelines focus on:
  - Correctness
  - Performance
  - Security
  - Maintainability
  - Test coverage

4.2.5 Testing Process

- Automated tests run in the following sequence:
  - Unit tests
  - Integration tests
  - End-to-end tests
  - Performance tests

- Test environments:
  - Development (automated deployment of feature branches)
  - Staging (production-like for QA testing)
  - Pre-production (exact replica of production)

4.2.6 Deployment Pipeline

- Continuous integration via Jenkins or GitHub Actions:
  - Build and package
  - Run tests
  - Static code analysis
  - Security scanning
- Continuous deployment to development and staging environments
- Production releases:
  - Scheduled bi-weekly
  - Require manual approval
  - Use blue-green or canary deployment strategies

4.3 Version Control Strategy

4.3.1 Git Workflow

- Tool: Git (hosted on GitHub Enterprise)
- Branch Strategy: Gitflow
  - `main`: Production-ready code
  - `develop`: Integration branch for features
  - `feature/*`: New features and non-emergency fixes
  - `release/*`: Release preparation
  - `hotfix/*`: Emergency production fixes

4.3.2 Commit Guidelines

- Semantic commit messages:
  - `feat:` New features
  - `fix:` Bug fixes
  - `docs:` Documentation changes
  - `style:` Code formatting
  - `refactor:` Code restructuring
  - `perf:` Performance improvements
  - `test:` Test additions or corrections
  - `chore:` Maintenance tasks
- Conventional commits linked to Jira tickets (e.g., `feat(AUTH-123): add biometric authentication`)

4.3.3 Release Management

- Semantic versioning (MAJOR.MINOR.PATCH)

- Automated changelog generation from commit messages
- Release notes published to internal documentation portal
- Post-release monitoring period with on-call support

5. Security and Compliance

5.1 Security Architecture

5.1.1 Authentication and Authorization

- **User Authentication**:
  - OAuth 2.0 implementation with JWT tokens
  - Multi-factor authentication (MFA) via SMS, email, or authenticator apps
  - Biometric authentication for mobile devices
  - Session management with configurable timeouts
- **Authorization**:
  - Role-Based Access Control (RBAC) for administrative functions
  - Attribute-Based Access Control (ABAC) for fine-grained permissions
  - Regular permission audits and least-privilege enforcement

5.1.2 Data Protection

- **Encryption**:
  - Data in transit: TLS 1.3 for all communications
  - Data at rest: AES-256 encryption using AWS KMS
  - Field-level encryption for PII and financial data
  - Database column-level encryption for sensitive fields
- **Data Classification**:
  - Level 1: Public data
  - Level 2: Internal use only
  - Level 3: Confidential (PII, account data)
  - Level 4: Restricted (payment credentials, authentication tokens)

5.1.3 Network Security

- **Perimeter Protection**:
  - AWS WAF for web application protection
  - Cloudflare for DDoS mitigation
  - IP whitelisting for administrative endpoints
- **Network Segmentation**:
  - VPC with public, private, and restricted subnets
  - Security groups with least-privilege rules
  - Network ACLs as a secondary defense layer
- **API Security**:
  - Rate limiting to prevent abuse
  - Input validation and sanitization

  - Request signing for partner APIs

5.2 Compliance Frameworks

5.2.1 Regulatory Compliance

- **Digital Personal Data Protection Act, 2023 (DPDP)**:
  - Data localization requirements
  - User consent management
  - Right to access and delete personal data
- **General Data Protection Regulation (GDPR)**:
  - Data subject rights
  - Data Protection Impact Assessments
  - Breach notification procedures
- **Payment Card Industry Data Security Standard (PCI-DSS)**:
  - Level 1 compliance for payment processing
  - Regular penetration testing
  - Cardholder data environment isolation

5.2.2 Industry Standards

- **ISO 27001**: Information security management system
- **OWASP Top 10**: Protection against common web vulnerabilities
- **NIST Cybersecurity Framework**: Security control implementation

5.2.3 Compliance Monitoring

- Quarterly internal audits
- Annual external audits
- Automated compliance checks in CI/CD pipeline
- Continuous control monitoring via AWS Config

5.3 Security Operations

5.3.1 Vulnerability Management

- Regular scanning using:
  - OWASP ZAP for dynamic application security testing
  - Snyk for dependency vulnerabilities
  - Custom scripts for business logic vulnerabilities
- Severity classification:
  - Critical: Immediate remediation (24 hours)
  - High: Remediation within 7 days
  - Medium: Remediation within 30 days
  - Low: Next planned release

5.3.2 Incident Response

- **Security Operations Center (SOC)**:
  - 24/7 monitoring via Splunk
  - Automated alerts based on MITRE ATT&CK framework
  - Threat intelligence integration
- **Incident Classification**:
  - P0: Critical (data breach, service outage)
  - P1: High (potential breach, significant impact)
  - P2: Medium (limited impact)
  - P3: Low (minimal impact)
- **Response Procedure**:
  - Identification and containment
  - Evidence collection
  - Remediation and recovery
  - Post-incident analysis and lessons learned

5.3.3 Security Training

- Mandatory security awareness training for all employees
- Role-specific security training for developers, administrators
- Quarterly phishing simulations
- Security champions program within engineering teams

6. Testing and Quality Assurance

6.1 Testing Strategy

6.1.1 Test Pyramid

- **Unit Tests**:
  - Cover 90% of code base
  - Focus on business logic and edge cases
  - Implemented using Jest (Node.js), Pytest (Python), Go testing
- **Integration Tests**:
  - Validate microservice interactions
  - Test database operations and external service integrations
  - Implemented using Postman/Newman and custom test harnesses
- **End-to-End Tests**:
  - Simulate complete user journeys
  - Cover critical business flows
  - Implemented with Cypress (web) and Appium (mobile)

6.1.2 Specialized Testing

- **Performance Testing**:
  - Load testing with JMeter (target: 2,000 concurrent users)
  - Stress testing to identify breaking points
  - Endurance testing (24-hour continuous operation)

    - Performance targets:
        * API response time: P95 $<$ 200ms
        * Page load time: $<$ 2 seconds
- **Security Testing**:
    - OWASP ZAP for vulnerability scanning
    - Manual penetration testing quarterly
    - Secure code reviews for critical components
- **Accessibility Testing**:
    - WCAG 2.1 AA compliance
    - Screen reader compatibility
    - Keyboard navigation support

6.1.3 Mobile Testing

- Testing across multiple iOS and Android versions
- Device farm for physical device testing
- Mobile-specific scenarios (offline mode, interruptions)

6.2 Test Automation

6.2.1 CI/CD Integration

- All tests integrated into Jenkins pipelines
- Parallelized test execution for faster feedback
- Automatic retry for flaky tests (maximum 3 attempts)

6.2.2 Test Data Management

- Anonymized production data for realistic testing
- Data generators for edge cases and stress testing
- On-demand test environment provisioning

6.2.3 Quality Gates

- Codecov enforces minimum 85% test coverage
- SonarQube quality gates for code smells and bugs
- Performance regression detection ($<$ 10% degradation)

6.3 Defect Management

6.3.1 Bug Tracking

- Jira for logging and tracking defects
- Required fields: steps to reproduce, expected vs. actual results, environment
- Severity classification:
    - S1 (Critical): System unusable, data corruption, security vulnerability

  - S2 (Major): Major function impacted, no workaround
  - S3 (Minor): Minor impact, workaround available
  - S4 (Cosmetic): UI issues, typos, non-functional issues

6.3.2 Defect SLAs

- S1: Resolution within 24 hours, immediate patch release if needed
- S2: Resolution within 72 hours, included in next scheduled release
- S3: Resolution within 2 weeks
- S4: Prioritized based on business impact

6.3.3 Bug Triage Process

- Daily triage meeting for new bugs
- Weekly bug review for outstanding issues
- Monthly quality metrics review

7. Deployment and DevOps Practices

7.1 CI/CD Pipeline

7.1.1 Continuous Integration

- Every commit triggers:
  - Code compilation and static analysis
  - Unit and integration tests
  - Security scanning
  - Code quality checks
- Feature branches built and deployed to ephemeral environments

7.1.2 Continuous Deployment

- **Staging Environment**:
  - Automatic deployment from the `develop` branch
  - Full test suite execution
  - Performance testing
- **Production Environment**:
  - Scheduled deployments (bi-weekly)
  - Blue-green deployment strategy
  - Automated smoke tests post-deployment
  - Automated rollback on failure

7.1.3 Pipeline Technologies

- Jenkins for build orchestration
- ArgoCD for GitOps-based deployment
- Nexus Repository for artifact storage
- Prometheus and Grafana for deployment monitoring

7.2 Infrastructure as Code (IaC)

7.2.1 Cloud Infrastructure

- **Terraform Modules**:
  - Network infrastructure (VPC, subnets, security groups)
  - Compute resources (EC2, ECS, Lambda)
  - Database services (RDS, DynamoDB)
  - Storage and CDN (S3, CloudFront)
- **Version Control**:
  - Infrastructure code in Git repository
  - PR-based changes with peer review
  - Change approval process for production infrastructure

7.2.2 Application Configuration

- **Kubernetes Resources**:
  - Helm charts for all microservices
  - Kustomize for environment-specific configurations
  - ConfigMaps and Secrets for application settings
- **Config Management**:
  - Environment variables for non-sensitive configuration
  - AWS Parameter Store for sensitive configuration
  - Feature flags via LaunchDarkly

7.2.3 IaC Security

- Terraform scanning with Checkov
- IAM permissions audit with CloudTracker
- Kubernetes security scanning with Kubesec

7.3 Containerization Strategy

7.3.1 Docker Standards

- Minimal base images (Alpine where possible)
- Multi-stage builds to minimize image size
- Non-root user execution
- Image scanning with Trivy

7.3.2 Kubernetes Configuration

- Resource limits and requests for all containers
- Pod security policies enforced
- Network policies controlling pod-to-pod communication
- Horizontal Pod Autoscalers based on custom metrics

7.3.3 Registry and Artifact Management

- Private Docker registry with vulnerability scanning
- Image promotion process across environments
- Immutable tags with git commit hashes
- Image retention policies

7.4 Release Management

7.4.1 Release Planning

- Bi-weekly release schedule
- Release planning meeting at sprint start
- Release readiness review before deployment

7.4.2 Release Process

- Release branch created from `develop`
- Regression testing on release branch
- Release notes compiled from Jira tickets
- Change Advisory Board approval for production deployment

7.4.3 Hotfix Process

- Critical issues patched directly from `main`
- Abbreviated testing focused on the specific issue
- Immediate deployment with post-deployment verification
- Patch merged back to `develop` branch

8. Monitoring and Maintenance

8.1 Monitoring Strategy

8.1.1 Metrics and Dashboards

- **Infrastructure Metrics**:
  - CPU, memory, disk usage
  - Network throughput and latency
  - Container health and resource utilization
- **Application Metrics**:
  - Request rate, errors, duration (RED)
  - Business KPIs (transactions, user signups)
  - Database performance (query times, connection counts)
- **Dashboards**:
  - Executive summary
  - Service health
  - User experience
  - Business metrics

8.1.2 Key Performance Indicators

- **Technical KPIs**:
  - API latency (P95 < 200ms)
  - Error rate (< 0.1% of requests)
  - Uptime (99.99%)
  - CPU/Memory utilization (< 80%)
- **Business KPIs**:
  - Transaction success rate (> 99.9%)
  - User session duration
  - Feature adoption rates
  - Conversion funnel metrics

8.1.3 Alerting Strategy

- **Alert Channels**:
  - PagerDuty for critical incidents (24/7 response)
  - Slack for non-critical notifications
  - Email for informational alerts
- **Alert Configuration**:
  - Avoid alert fatigue through tuned thresholds
  - Multi-stage alerts (warning → critical)
  - Auto-remediation where possible
  - Clear ownership and escalation paths

8.2 Logging Framework

8.2.1 Log Architecture

- **Centralized Logging**:
  - ELK Stack (Elasticsearch, Logstash, Kibana)
  - Structured logging format (JSON)
  - Consistent correlation IDs across services
- **Log Categories**:
  - Application logs
  - Access logs
  - Audit logs
  - System logs

8.2.2 Log Management

- **Retention Policies**:
  - Hot storage: 30 days (full resolution)
  - Warm storage: 90 days (aggregated)
  - Cold storage: 1 year (archival in S3)
- **Log Security**:
  - PII redaction in logs
  - Encrypted transport and storage

  - Access control on log viewing

8.2.3 Log Analysis

- Automated pattern detection
- Anomaly detection using machine learning
- Business insights extraction

8.3 Maintenance Procedures

8.3.1 Routine Maintenance

- **Patching Schedule**:
  - OS updates: Monthly
  - Dependency updates: Bi-weekly
  - Critical security patches: Within 48 hours
- **Database Maintenance**:
  - Index optimization: Weekly
  - Vacuum and analyze: Daily
  - Statistics update: Daily

8.3.2 Capacity Planning

- Quarterly infrastructure review
- Growth projections and scaling recommendations
- Cost optimization analysis

8.3.3 Technical Debt Management

- Dedicated 20% of sprint capacity to technical debt
- Quarterly architectural review
- Deprecation strategy for legacy components

9. Future Roadmap

9.1 Short-Term Initiatives (Q2–Q4 2025)

9.1.1 AI and Machine Learning Integration

- **Personalized Financial Insights**:
  - Spending pattern recognition
  - Anomaly detection for fraud prevention
  - Budget recommendations based on user behavior
- **Chatbot Implementation**:
  - Natural language processing for customer support
  - Financial advisor virtual assistant
  - Multi-language support

9.1.2 Blockchain and Cryptocurrency

- **Digital Assets Support**:
  - Cryptocurrency wallet integration
  - Support for major cryptocurrencies (Bitcoin, Ethereum)
  - Blockchain-based transaction verification
- **Smart Contracts**:
  - Automated lending agreements
  - Programmatic escrow services
  - Transparent audit trails

9.1.3 Localization and Internationalization

- **Language Support Expansion**:
  - Hindi, Spanish, Mandarin, Arabic
  - Right-to-left (RTL) language support
  - Culturally sensitive financial terminology
- **Regional Compliance**:
  - Regulatory adapters for new markets
  - Regional payment method integration

9.2 Long-Term Strategic Direction (2026–2027)

9.2.1 Global Market Expansion

- **Geographic Targets**:
  - Latin America (Brazil, Mexico)
  - Africa (Nigeria, Kenya)
  - Southeast Asia (Vietnam, Philippines)
- **Infrastructure**:
  - Regional data centers
  - Edge computing for lower latency
  - Multi-region high availability

9.2.2 Open Banking Ecosystem

- **API Marketplace**:
  - Developer portal and SDK
  - Partner integration platform
  - Revenue sharing model
- **Regulatory Compliance**:
  - PSD2 and equivalent standards
  - Strong Customer Authentication (SCA)
  - Consent management framework

9.2.3 Next-Generation Infrastructure

- **Serverless Architecture**:

  - Function-as-a-Service for suitable workloads
  - Event-driven processing
  - Pay-per-use cost model
- **Zero-Downtime Operations**:
  - Advanced canary deployments with Istio
  - Self-healing infrastructure
  - Chaos engineering practice

10. Appendices

10.1 Glossary of Terms

Term	Definition
ACID	Atomicity, Consistency, Isolation, Durability - properties of database transactions
API	Application Programming Interface
CIDR	Classless Inter-Domain Routing - IP address allocation method
FinTech	Financial Technology
JWT	JSON Web Token - compact, URL-safe means of representing claims between two parties
Microservices	Architectural style structuring an application as a collection of loosely coupled services
OAuth 2.0	Industry-standard protocol for authorization
PII	Personally Identifiable Information
REST	Representational State Transfer - architectural style for distributed systems
SPA	Single Page Application

10.2 Reference Documents

Document	Location	Purpose
AWS Well-Architected Framework	Internal Wiki	Cloud architecture best practices
PCI-DSS Guidelines	Security Portal	Payment security compliance
Kubernetes Documentation	kubernetes.io	Container orchestration reference
OWASP Security Standards	Internal Wiki	Web application security
Data Protection Policy	Legal Repository	Data handling requirements

10.3 Contact Information

Team	Email	Response SLA
Engineering Lead	engineering@finsolve.com	4 hours
Security Team	security@finsolve.com	1 hour
DevOps Support	devops@finsolve.com	2 hours
Data Protection Officer	dpo@finsolve.com	24 hours
API Support	api-support@finsolve.com	8 hours

Note: This document is a living artifact and will be updated quarterly to reflect changes in architecture, processes, or technologies. For clarifications, contact the Engineering Lead at engineering@finsolve.com.

Last Updated: May 14, 2025