MFE Payments System - Executive Summary

Document Version: 1.2 **Date:** December 31, 2025 **Status:** POC-3 Complete with CI Pipeline - Production-Ready Architecture **Prepared For:** Executive Leadership Review **Repository:** GitHub (https://github.com/pateatlau/payments-system-mfe-microservices-fullstack-nx-2026)

Table of Contents

---

1. Executive Overview

Project Summary

The MFE Payments System represents a comprehensive, production-ready platform that demonstrates enterprise-grade architecture patterns for modern payment processing applications. Built upon a microfrontend foundation with microservices backend, this system embodies industry best practices in scalability, security, and operational excellence.

The platform showcases the successful implementation of complex distributed systems architecture while maintaining developer productivity through intelligent tooling and automation.

Key Achievements

Phase	Status	Deliverables
POC-0	Complete	Foundation architecture, monorepo structure, development environment

Phase	Status	Deliverables
POC-1	Complete	Rspack migration, Module Federation v2, Hot Module Replacement optimization
POC-2	Complete	Full-stack integration, JWT authentication, RBAC, design system implementation
POC-3	Complete	Production infrastructure, observability stack, GraphQL API, theme system, CI pipeline

Business Value Proposition

- **Scalability:** Independent deployment cycles for frontend modules and backend services enable teams to operate autonomously without cross-functional dependencies
- **Maintainability:** Domain-driven design with clear bounded contexts reduces cognitive load and accelerates onboarding of new team members
- **Performance:** Sub-second page loads achieved through intelligent code splitting, distributed caching, and optimized build pipelines
- **Security:** Banking-grade authentication framework with JWT tokens, role-based access control, and comprehensive audit logging
- **Observability:** Complete visibility into system behavior through metrics, distributed tracing, and centralized error tracking
- **Developer Velocity:** CI pipeline with Nx Cloud distributed caching reduces build times by 50-65%, enabling rapid iteration

Current Status

Aspect	Status	Details
Development Environment	Operational	Fully functional with HTTPS/TLS, observability, production infrastructure
Local Demonstration	Available	Complete feature set accessible at https://localhost
CI Pipeline	Complete	GitHub Actions with Nx Cloud distributed caching (50-65% faster builds)
CD Pipeline	Planned	Scheduled for next implementation phase
Public Demonstration	Pending	Awaiting deployment pipeline implementation

---

2. Architecture Overview

High-Level Architecture

```
                                   [Internet]
                                       |
                                       v
                             +------------------+
                             |   nginx Proxy    |
                             |  (SSL/TLS, LB)   |
                             +------------------+
                                       |
             +-------------------------+-------------------------+
             |                         |                         |
             v                         v                         v
   +------------------+      +------------------+      +------------------+
   |   Shell App      |      |   API Gateway    |      |   WebSocket      |
   |   (Port 4200)    |      |   (Port 3000)    |      |   Server         |
   +------------------+      +------------------+      +------------------+
             |                         |
   +---------+---------+---------+     +---------+---------+---------+
   |         |         |         |     |         |         |         |
   v         v         v         v     v         v         v         v
+------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+
| Auth | | Pay  | |Admin | |Profile| | Auth | | Pay  | |Admin | |Profile|
| MFE  | | MFE  | | MFE  | | MFE  | | Svc  | | Svc  | | Svc  | | Svc  |
+------+ +------+ +------+ +------+ +------+ +------+ +------+ +------+
                               |         |         |         |
                               v         v         v         v
                          +------+  +------+  +------+  +------+
                          |Auth  |  |Pay   |  |Admin |  |Profile|
                          |  DB  |  |  DB  |  |  DB  |  |  DB  |
                          +------+  +------+  +------+  +------+
```

Architectural Principles

1. **Microfrontend Architecture:** Independently deployable frontend modules with shared runtime dependencies
2. **Microservices Backend:** Domain-driven decomposition with autonomous service boundaries
3. **API Gateway Pattern:** Unified entry point providing authentication, routing, and rate limiting
4. **Event-Driven Communication:** Asynchronous messaging via RabbitMQ for loose coupling
5. **Database per Service:** Complete data isolation enabling independent scaling and evolution

---

3. Technology Stack

Frontend Technologies

Category	Technology	Version	Purpose
Framework	React	18.3.1	Component-based UI development
Build Tool	Rspack	1.6.x	High-performance bundling with Module Federation
Monorepo	Nx	22.1.x	Workspace orchestration and intelligent caching
Styling	Tailwind CSS	4.0.0	Utility-first design system
State	Zustand	4.5.x	Lightweight client-side state management
Server State	TanStack Query	5.x	Server state synchronization and caching
Forms	React Hook Form + Zod	7.52.x / 3.25.x	Type-safe form handling and validation
Design System	shadcn/ui	Latest	Accessible, customizable component library

Backend Technologies

Category	Technology	Version	Purpose
Runtime	Node.js	24.11.x	Server-side JavaScript runtime
Framework	Express	5.x	HTTP server framework
Database	PostgreSQL	16	Relational data persistence
ORM	Prisma	5.x	Type-safe database operations
Cache	Redis	7.x	Session storage and caching
Message Broker	RabbitMQ	3.x	Event-driven messaging
API	REST + GraphQL	-	Dual API paradigm support

Infrastructure Technologies

Category	Technology	Purpose
Reverse Proxy	nginx	SSL termination, load balancing, rate limiting
Containerization	Docker	Service isolation and deployment
Orchestration	Docker Compose	Development environment management
CI Platform	GitHub Actions	Automated build, test, and validation
Build Cache	Nx Cloud	Distributed caching for accelerated builds
Metrics	Prometheus	Time-series metrics collection
Dashboards	Grafana	Operational visualization and monitoring
Tracing	Jaeger	Distributed request tracing
Error Tracking	Sentry	Error aggregation and alerting

---

4. Frontend Architecture

Module Federation Strategy

The frontend employs Webpack Module Federation v2 (via Rspack) to achieve true microfrontend independence, enabling autonomous deployment cycles while maintaining runtime cohesion.

Shell Application (Host)

- Orchestrates dynamic microfrontend loading at runtime
- Manages application-wide routing and navigation
- Provides shared authentication context and global state

Remote Microfrontends

Module	Port	Exposed Components	Responsibility
Auth MFE	4201	SignIn, SignUp	User authentication flows
Payments MFE	4202	PaymentsPage, PaymentReports	Payment processing and reporting
Admin MFE	4203	AdminDashboard	Administrative operations
Profile MFE	4204	ProfilePage	User profile management

Shared Libraries

```
libs/
├── shared-types/          # TypeScript interfaces and domain types
├── shared-utils/          # Common utility functions
├── shared-ui/             # Base UI primitives
├── shared-design-system/  # shadcn/ui component library
├── shared-auth-store/     # Authentication state management (Zustand)
├── shared-api-client/     # HTTP client with authentication interceptors
├── shared-event-bus/      # Cross-MFE communication channel
├── shared-header-ui/      # Unified header component
├── shared-websocket/      # WebSocket client for real-time features
├── shared-session-sync/   # Cross-tab and cross-device session synchronization
└── shared-theme-store/    # Theme management (dark/light mode)
```

State Management Strategy

State Type	Solution	Application
Client State	Zustand	Authentication, UI preferences, theme
Server State	TanStack Query	API data, intelligent caching, background synchronization
Form State	React Hook Form	Form inputs, validation, submission
Cross-MFE	Event Bus	Inter-module communication and coordination

Theme System

The application implements a comprehensive theme system supporting both light and dark modes with intelligent defaults and user preference persistence.

Capabilities:

- System preference detection with automatic theme inheritance

- User preference override with persistent storage
- Cross-tab synchronization via BroadcastChannel API
- Seamless integration with Tailwind CSS v4 design tokens
- Smooth transition animations between themes

Implementation Status: Complete across Shell and all MFE modules with verified cross-tab synchronization.

---

5. Backend Architecture

Service Decomposition

Service	Port	Database	Responsibility
API Gateway	3000	-	Request routing, authentication, rate limiting
Auth Service	3001	auth_db	User management, JWT tokens, RBAC enforcement
Payments Service	3002	payments_db	Payment processing, transaction management
Admin Service	3003	admin_db	Administrative operations, audit logging
Profile Service	3004	profile_db	User profiles, preference management

API Gateway Capabilities

- **Request Routing:** Intelligent path-based routing to backend services
- **Authentication:** JWT validation with automatic user context injection
- **Rate Limiting:** Configurable limits protecting against abuse
- **CORS Management:** Cross-origin request policy enforcement
- **Streaming Proxy:** Zero-buffering request/response streaming for optimal performance
- **GraphQL Support:** Apollo Server with comprehensive schema
- **WebSocket Integration:** Real-time bidirectional communication

Event-Driven Architecture

RabbitMQ provides reliable, persistent messaging enabling loose coupling between services.

Exchange Topology

Exchange	Type	Purpose
user.events	topic	User lifecycle events
payment.events	topic	Payment processing events
admin.events	topic	Administrative events
system.events	fanout	System-wide notifications

Event Flow Pattern

```
[Auth Service] ---> [user.events] ---> [Payments Service]
     |                  |                    |
     |                  |                    v
     |                  |            Update user context
     |                  |
     |                  +-----------> [Admin Service]
     |                                      |
     v                                      v
 user.created                        Log audit event
```

---

6. Infrastructure

nginx Configuration

nginx serves as the primary ingress point with enterprise-grade capabilities:

Feature	Configuration
SSL/TLS	TLS 1.2+ with modern cipher suites
HTTP/2	Enabled for improved connection efficiency
Rate Limiting	API: 100 req/min, Auth: 10 req/min
Compression	gzip for text, JSON, JavaScript
Static Asset Caching	Immutable assets cached for 1 year
WebSocket Support	Upgrade handling for real-time features
Load Balancing	least_conn for HTTP, ip_hash for persistent connections

Database Architecture

Each service maintains complete data sovereignty through dedicated PostgreSQL instances:

Database	Port	Service	Schema
auth_db	5432	Auth Service	users, sessions, tokens
payments_db	5433	Payments Service	payments, transactions
admin_db	5434	Admin Service	audit_logs, settings
profile_db	5435	Profile Service	profiles, preferences

Caching Strategy

Layer	Technology	TTL	Purpose
Browser	Service Worker	Variable	Static assets, offline capability
CDN	nginx	1 year	Immutable static assets
API	Redis	5-60 min	Query results, session data
Database	Prisma	-	Connection pooling

---

7. Security

Authentication Flow

```
[Client] --> [API Gateway] --> [Auth Service]
    |              |                 |
    |              |                 v
    |              |          Validate credentials
    |              |                 |
    |              |                 v
    |              |          Generate JWT tokens
    |              |                 |
    |              <-----------------+
    |              |
    |              v
    |        Set HTTP-only cookies
    |              |
    <--------------+
```

JWT Token Strategy

Token	Lifetime	Storage	Purpose
Access Token	15 minutes	Memory only	API authentication
Refresh Token	7 days	HTTP-only cookie	Secure token renewal

Role-Based Access Control

Role	Permissions
ADMIN	Full system access, user management, audit logs
CUSTOMER	View/create payments, manage own profile
VENDOR	Initiate payments, view reports, limited admin

Security Headers

```
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Referrer-Policy: strict-origin-when-cross-origin
Content-Security-Policy: default-src 'self'; ...
```

8. Observability

Metrics Collection (Prometheus)

Key Metrics

Metric	Type	Description
http_requests_total	Counter	Total requests by method, route, status
http_request_duration_seconds	Histogram	Request latency distribution
http_active_connections	Gauge	Current active connections
http_errors_total	Counter	Error count by type

All backend services expose metrics at the `/metrics` endpoint with collection intervals of 10-15 seconds.

Operational Dashboards (Grafana)

Services Overview Dashboard

- Service health status indicators
- Cross-service request rate comparison
- P95 latency comparison

API Gateway Dashboard

- Request throughput (requests/second)
- Response time percentiles (p50, p90, p95, p99)
- Error rate trends
- Active connection monitoring
- Request breakdown by method, status, route

Distributed Tracing (Jaeger)

OpenTelemetry instrumentation provides comprehensive request tracing:

- Automatic span creation for HTTP requests
- Database query visibility
- Cross-service request correlation
- Latency breakdown analysis

Error Tracking (Sentry)

Centralized error management with:

- Automatic exception capture
- User context correlation
- Release tracking

- Performance monitoring integration

---

9. Continuous Integration

CI Pipeline Overview

The project employs GitHub Actions with Nx Cloud distributed caching, delivering significant performance improvements while maintaining comprehensive quality gates.

Pipeline Performance

Metric	Without Nx Cloud	With Nx Cloud	Improvement
Average Build Duration	15-20 minutes	5-10 minutes	50-65%
Cache Hit Rate	N/A	70-90%	-
Parallel Task Execution	Limited	Full	Significant

Pipeline Architecture

```
[Push/PR] --> [Install Dependencies] --> [Nx Cloud Restore Cache]
                                                     |
                     +-------------------------------+-------------------------------+
                     |               |               |               |               |
                     v               v               v               v               v
                [Lint All]    [Test All]     [Build Backend]  [Build Frontend]   [Typecheck]
                     |               |               |               |               |
                     +-------------------------------+-------------------------------+
                                                     |
                                                     v
                                           [E2E Tests (on merge)]
                                                     |
                                                     v
                                           [Nx Cloud Save Cache]
```

Quality Gates

Stage	Scope	Requirements
Lint	All Projects	ESLint compliance, no errors
Unit Tests	All Projects	Jest tests pass, 70%+ coverage target
Type Check	All Projects	TypeScript compilation success
Build	All Projects	Production build completion
E2E Tests	Critical Paths	Playwright tests pass

Nx Cloud Integration

Nx Cloud provides distributed caching and task execution, enabling:

- **Remote Cache Sharing:** Build artifacts cached and shared across CI runs
- **Distributed Execution:** Parallel task execution across multiple agents
- **Cache Transparency:** Full visibility into cache utilization and performance
- **Developer Experience:** Local builds benefit from CI-generated cache

---

10. API Documentation

REST API

Interactive API documentation is available via Swagger UI:

Endpoint	Description
/api-docs	Swagger UI interface
/api-docs.json	OpenAPI 3.0 specification (JSON)
/api-docs.yaml	OpenAPI 3.0 specification (YAML)

API Endpoints Summary

Category	Endpoints	Authentication
Auth	/api/auth/*	Public (login, register)
Payments	/api/payments/*	JWT required
Admin	/api/admin/*	JWT + ADMIN role
Profile	/api/profile/*	JWT required
Health	/health/*	Public

GraphQL API

Apollo Server provides a comprehensive GraphQL API:

Endpoint	Method	Description
/graphql	POST	GraphQL queries and mutations
/graphql	GET	API introspection

Schema Features

- Custom authorization directives (@auth, @admin)
- Type-safe resolvers with TypeScript
- Automatic schema documentation

---

11. Development Workflow

Quick Start

```
# 1. Install dependencies
pnpm install

# 2. Generate SSL certificates (first time only)
pnpm ssl:generate

# 3. Start infrastructure
pnpm infra:start

# 4. Start backend services
pnpm dev:backend

# 5. Start frontend
pnpm dev:all

# 6. Access application
open https://localhost
```

Available Commands

Command	Description
pnpm dev:all	Start all frontend modules
pnpm dev:backend	Start all backend services
pnpm infra:start	Start Docker infrastructure
pnpm test	Execute all tests
pnpm build	Build all projects
pnpm observability:start	Start Prometheus, Grafana, Jaeger

Access Points

Service	URL	Credentials
Application	https://localhost	-
Swagger UI	https://localhost/api-docs	-
GraphQL	https://localhost/graphql	-
Prometheus	http://localhost:9090	-
Grafana	http://localhost:3010	admin/admin
Jaeger	http://localhost:16686	-
RabbitMQ	http://localhost:15672	admin/admin

Testing Strategy

Test Type	Framework	Coverage Target

Test Type	Framework	Coverage Target
Unit	Jest + RTL	70%+
Integration	Jest	Key business flows
E2E	Playwright	Critical user paths
Load	Custom scripts	Performance baselines

---

12. Future Roadmap

Next Phase: Continuous Deployment

CD Pipeline Implementation

- Automated deployment workflows to cloud infrastructure
- Multi-environment promotion (development, staging, production)
- Infrastructure as Code (IaC) with Terraform or Pulumi
- Container registry integration with versioned artifacts
- Blue-green or canary deployment strategies

Cloud Infrastructure

- Cloud provider configuration (AWS, GCP, or Azure)
- Kubernetes orchestration for container management
- Auto-scaling policies based on demand
- CDN integration for global content delivery
- Production-grade SSL certificates with automated renewal

Subsequent Enhancements

Public Demonstration Environment

- Internet-accessible demo instance
- Sample data and demonstration accounts
- Interactive feature showcase
- Published performance benchmarks

User Experience Refinements

- Profile-based theme preference persistence
- Enhanced accessibility compliance (WCAG 2.1 AA)
- Internationalization framework

Appendix: Project Structure

```
payments-system-mfe/
├── apps/
│   ├── shell/                 # Host application
│   ├── auth-mfe/              # Authentication microfrontend
│   ├── payments-mfe/          # Payments microfrontend
│   ├── admin-mfe/             # Admin microfrontend
│   ├── profile-mfe/           # Profile microfrontend
│   ├── api-gateway/           # API Gateway service
│   ├── auth-service/          # Authentication service
│   ├── payments-service/      # Payments service
│   ├── admin-service/         # Admin service
│   └── profile-service/       # Profile service
├── libs/
│   ├── shared-*/              # Shared frontend libraries
│   └── backend/               # Shared backend libraries
├── nginx/                     # nginx configuration
├── prometheus/                # Prometheus configuration
├── grafana/                   # Grafana dashboards
├── rabbitmq/                  # RabbitMQ definitions
└── docs/                      # Documentation
```

Document Conclusion

This document provides a comprehensive overview of the MFE Payments System architecture, current implementation status, and strategic roadmap. The platform represents a mature, production-ready architecture that demonstrates enterprise-grade patterns for building scalable payment processing applications.

The successful implementation of the CI pipeline with Nx Cloud distributed caching marks a significant milestone, establishing the foundation for rapid, reliable development iterations. With the architecture fully operational and the continuous integration infrastructure in place, the system is well-positioned for the next phase: continuous deployment and cloud infrastructure provisioning.

For detailed implementation guides and technical specifications, please refer to the comprehensive documentation available in the `docs/` directory.

Last Updated: December 31, 2025