Unifying Travels & Viagens

Architecture & Roadmap

Presenter: Ernesto Anaya Ruiz

Thoughtworks | Preparation for Case Study

Summary

Problem & Vision

Problem

- Two customer-facing apps (Travels: Struts/JSP on Tomcat; Viagens: React SPA + BFF)
- Multiple backends across AWS, GCP, Azure
- Low test coverage, batch releases, coordination overhead

Vision

- A **single, resilient, fast web experience** powered by a **frontend platform** (React/Next.js shell) that composes **domain micro-frontends** and brokers requests via **domain BFFs** across existing backends.

Approach & Outcomes

Approach (high level)

- **Micro-frontends (MFE) with a shell** + **domain BFFs** + **API Gateway**
- **Feature flags**, **contract testing**, **progressive delivery**
- **Strangler Fig** to retire Struts/JSP gradually

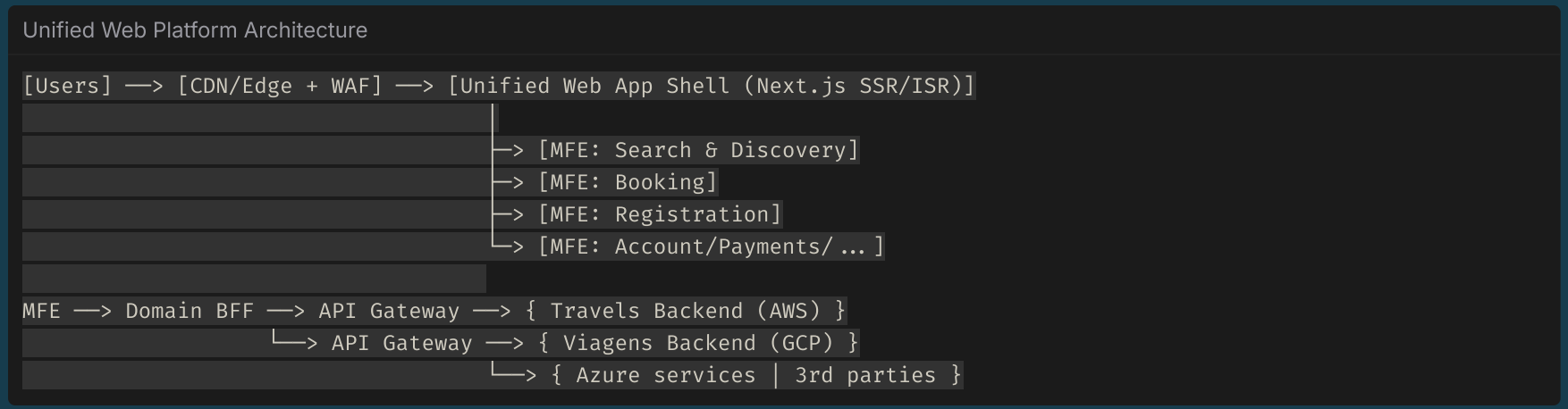
Outcomes

- **Faster delivery:** Independent team deployments, reduced coordination overhead
- **Better user experience:** Unified interface, improved Core Web Vitals, faster page loads
- **Reduced risk:** Progressive rollouts, feature flags, immediate rollback capability
- **Operational efficiency:** Consolidated data aggregation, simplified maintenance
- **Future flexibility:** Backend-agnostic design enables gradual modernization

Target Architecture

High-Level Diagram

📊 View detailed architecture diagram



Why this approach?

- Keeps heterogeneous backends intact; **no backend unification required**
- **Domain BFFs as aggregation layer**: consolidate data from 2 companies (e.g., merge destination lists)
- **Response orchestration**: BFFs handle complex scenarios (merging search results, normalizing data)
- SSR/ISR via **Next.js** → better **SEO/Core Web Vitals** vs SPA-only
- **API Gateway**: routing, rate limiting, authz, cross-cloud abstraction

Key Technology Choices

- **Web shell:** React + Next.js (SSR/ISR) with **Module Federation** for MFEs
- **BFFs (Backend for Frontend):**
  - Node/TypeScript for response aggregation and data transformation
  - **Key responsibilities:** Merge data from Travels + Viagens, normalize formats, handle pagination
  - Example: Search BFF calls both backends, merges destinations, deduplicates, sorts by relevance
- **Contracts:** REST/JSON or GraphQL; **consumer-driven contracts** (Pact) MFE↔BFF & BFF↔Backend
- **Edge/CDN:** Cloudflare/Akamai/Fastly for assets, image optimization

Migration Roadmap

Migration Strategy Overview

Approach: Big bang migration with comprehensive preparation and revenue protection

Key Phases: Discovery → Foundations → Preparation → Migration → Optimization

Revenue Protection: Pre-migration testing, performance benchmarking, immediate rollback capability

Phase 0-2: Preparation & Build

Phase 0 — Discovery & Planning

- Inventory frontends/backends, auth, analytics, SEO, SLAs
- Baseline current metrics (conversion, performance, revenue)

Phase 1 — Foundations

- Stand up Web Shell + first MFE skeleton
- Platform toolchain (CI/CD, flags, telemetry, design system)
- Create API Gateway + Booking BFF (pilot)

Phase 2 — Complete Migration Preparation

- Build all MFEs (Search, Booking, Registration, Account, Payments)
- Feature parity validation: comprehensive testing vs legacy functionality
- Performance benchmarking to ensure ≥100% baseline performance

Phase 3-4: Migration & Optimization

Phase 3 — Big Bang Migration

- Complete cutover from legacy to new platform during maintenance window (minimal impact)
- Immediate monitoring: revenue, conversion, performance dashboards
- Fast rollback capability if any KPIs drop below acceptable thresholds

Phase 4 — Stabilization & Optimization

- Performance tuning and optimization post-migration
- Legacy system decommission once stability confirmed
- Platform improvements and feature development

Critical Success Factors for Revenue Protection: Comprehensive testing, performance benchmarking, immediate rollback capability, intensive monitoring

Team Topology & Responsibilities

Core Teams Structure

Platform Team (Enablement)

- **Core responsibilities:** Shell architecture, design system
- **Developer experience:** Linting, tooling
- **Skills mix:** Frontend architects, UX system designers

Domain Product Squads *(Cross-functional)*

- **Ownership:** Complete MFE + Domain BFF lifecycle (design → development → production)
- **Domains:** Booking, Search & Discovery, Registration, Account/Payments
- **Skills mix:** Frontend, backend, UX, product owner

Supporting Teams & Integration

SRE Team

- **Infrastructure:** IaC, capacity planning, cost optimization, security baselines
- **CI/CD:** Pipeline templates, deployment automation, build optimization
- **Operations:** Incident management, monitoring, performance optimization
- **Skills mix:** Site reliability engineers, DevOps engineers, infrastructure specialists

Quality Engineering & Analytics

- **Quality:** Contract testing, E2E automation, accessibility, performance testing
- **Analytics:** Business dashboards, conversion analysis, A/B testing framework
- **Integration:** Embedded specialists in each domain squad
- **Skills mix:** Test automation engineers, data analysts, accessibility specialists

Processes and Best Practices

Engineering Excellence & Delivery

Development Standards

- **Trunk-based development** with short-lived feature branches and feature flags for safe integration
- **Consumer-driven contracts** (Pact) to prevent breaking changes
- **Test strategy:** Unit-first approach with focused integration and E2E tests
- **Code reviews:** Pair programming encouraged, automated quality gates

Quality & Performance

- **Performance budgets:** Bundle size limits, Core Web Vitals thresholds
- **Preview environments:** Per-PR deployments for stakeholder validation
- **Automated testing:** Visual regression, accessibility, cross-browser compatibility

Delivery & Reliability

- **Infrastructure as Code:** Version-controlled, repeatable deployments
- **Incident response:** Clear runbooks, blameless postmortems, continuous improvement

KPIs for Team Performance

Team Performance & Value Delivery

Delivery Velocity (DORA Metrics)

- **Lead Time:** Feature conception to production deployment
- **Deployment Frequency:** Multiple releases per day per team
- **Change Failure Rate:** Percentage of deployments requiring immediate fix
- **Recovery Time:** Time to restore service after incidents

Engineering Productivity

- **Cycle Time:** Code commit to production deployment
- **Code Review Efficiency:** Time from PR creation to merge
- **Build & Test Performance:** Pipeline execution time and reliability
- **Quality Metrics:** Test coverage, defect escape rate, technical debt

Solution Success Metrics

Business Value Creation

- **Revenue Growth:** Increased bookings and session value through improved UX
- **Conversion Optimization:** Enhanced funnel performance across all touchpoints
- **Customer Experience:** Unified journey reducing friction and abandonment

Technical Performance

- **User Experience:** Core Web Vitals meeting Google’s recommended thresholds
- **System Reliability:** High availability for critical booking and search flows
- **API Performance:** Consistent response times across all micro-frontends

Platform Maturity & Efficiency

- **Migration Success:** Complete transition from legacy systems
- **Cost Optimization:** Lower infrastructure costs through efficient architecture
- **Development Velocity:** Accelerated feature delivery through improved tooling

Thank you

Ernesto Anaya Ruiz

Questions welcome — looking forward to the discussion.

Additional Reference Materials

For Q&A and deeper discussion

Observability & Monitoring

Distributed Tracing & APM

- **OpenTelemetry** → centralized APM/logs across all MFEs and BFFs
- **Performance monitoring** for Core Web Vitals and API response times

SLOs

- **Service Level Objectives (SLOs)** for availability, latency, and error rates
- **Automated alerting** based on SLO violations

Dashboards & Insights

- **Real-time dashboards** for business metrics (conversion, revenue)
- **Technical dashboards** for system health and performance

Security & Compliance

Security Architecture

- **WAF (Web Application Firewall)** at CDN/Edge layer for DDoS and attack protection
- **CSP (Content Security Policy)** and Trusted Types to prevent XSS attacks
- **Secrets management** with centralized vault and rotation policies

Security Engineering

- **SAST/DAST** (Static/Dynamic Application Security Testing) in CI/CD pipeline
- **Dependency scanning** for vulnerable packages and libraries

Common Questions & Talking Points

- **Why micro-frontends vs one SPA?** Org alignment, independent deploys, safer migration
- **Why Next.js SSR/ISR?** Better SEO & Core Web Vitals; edge cacheability; partial static regen
- **Avoiding a distributed monolith?** Clear contracts, BFF boundaries, consumer tests, ownership, observability
- **Backend unification later?** Design is backend-agnostic; unify behind BFFs/gateway when timing is right
- **Keeping quality high?** Shift-left: unit/component/contract; few critical E2Es; PR previews; a11y checks
- **Release strategy?** Progressive delivery (flags/canary), error budgets, auto-rollback
- **Data privacy?** Consent mgmt; minimize PII; policy-as-code; auditing