BRIGHTGROVE

Fast, safe, reliable and adaptable
business partner to help your business evolve

USE CASE

API-First Architecture Implementation for a Global Hospitality Tech Platform

CLIENT PROFILE

Europe based global hospitality platform:

- 500+ active hospitality partners worldwide
- Processing 20M+ bookings annually
- Revenue: €170M+ annually
- Tech Stack: Java-based monolithic backend, legacy PHP services
- Previous Infrastructure: On-premises data centers with partial cloud presence
- Peak Season Transaction Increase: 300–400%
- Current System Maintenance Cost: €1.5M – €2M annually

BUSINESS CONTEXT

The client faced critical scalability challenges during peak travel seasons, potentially risking up to €800K – €1M in lost bookings during high-demand periods. Their legacy booking system, built over 10 years ago, struggled to handle modern integration requirements and threatened their market position against more agile competitors. With maintenance costs reaching €1.5M – €2M annually and increasing technical debt, the organization needed a strategic transformation to maintain competitiveness and enable future growth.

PROBLEM STATEMENT

The goal was to modernize a legacy booking platform using an API-First approach. The current system suffered from slow integration of new services, difficulty in scaling, inconsistent API designs, long development cycles, limited visibility into system performance, and challenges in maintaining backward compatibility. The main objective was to create a more flexible, scalable, and developer-friendly ecosystem while minimizing disruption to existing services.

CHALLENGES

- **Legacy Integration:** Existing services and partners rely on outdated API structures, making it difficult to implement new standards without breaking functionality
- **Scalability Issues:** The current system struggled to handle traffic spikes during peak travel seasons
- **Inconsistent Design:** Different teams have implemented APIs with varying standards, leading to a fragmented ecosystem
- **Slow Development Cycles:** The lack of standardization and proper documentation slowed down the development of new features
- **Limited Observability:** There was insufficient visibility into system performance and health across different services
- **Versioning and Compatibility:** Maintaining backward compatibility while introducing new API versions was challenging

IMPLEMENTATION TIMELINE

Phase 1: **Foundation**
(Months 1–4)

- API design standardization
- OpenAPI contract implementation
- Initial security framework setup
- Quick Win: 15% reduction in API-related errors within first month

Phase 2: **Migration**
(Months 5–8)

- Legacy system parallel running
- Partner migration in batches
- Observability implementation
- Quick Win: First partner migrations completed 40% faster than projected

Phase 3: **Optimization**
(Months 9–12)

- Performance tuning
- Advanced monitoring setup
- Partner tooling enhancement
- Quick Win: Peak load handling improved by 200%

SOLUTIONS IMPLEMENTED BY BRIGHTGROVE

Standardized API Design:

- URL Structure: Implemented a consistent URL pattern: https://{componentName}.domain.com/{stage}/{version}/{service}
- Error Handling: Adopted RFC7807 with custom attributes for standardized error reporting across all services.
- Versioning: Ensured backward compatibility between versions and implemented a clear versioning strategy.

OpenAPI Contract Management:

- Centralized Storage: Stored OpenAPI contracts within source code repositories at api/description.yaml.
- Version Control: Used git tags (e.g., contract-2.2.0) for contract versioning.
- Accessibility: Provided easy access to contracts via URL, supporting retrieval by version number, branch name and git hash.

Security and Authentication:

- Implemented HTTPS-only access with IPv6 support.
- Standardized authentication mechanisms, using JWT for external calls and flexible options for internal services.

Observability and Monitoring:

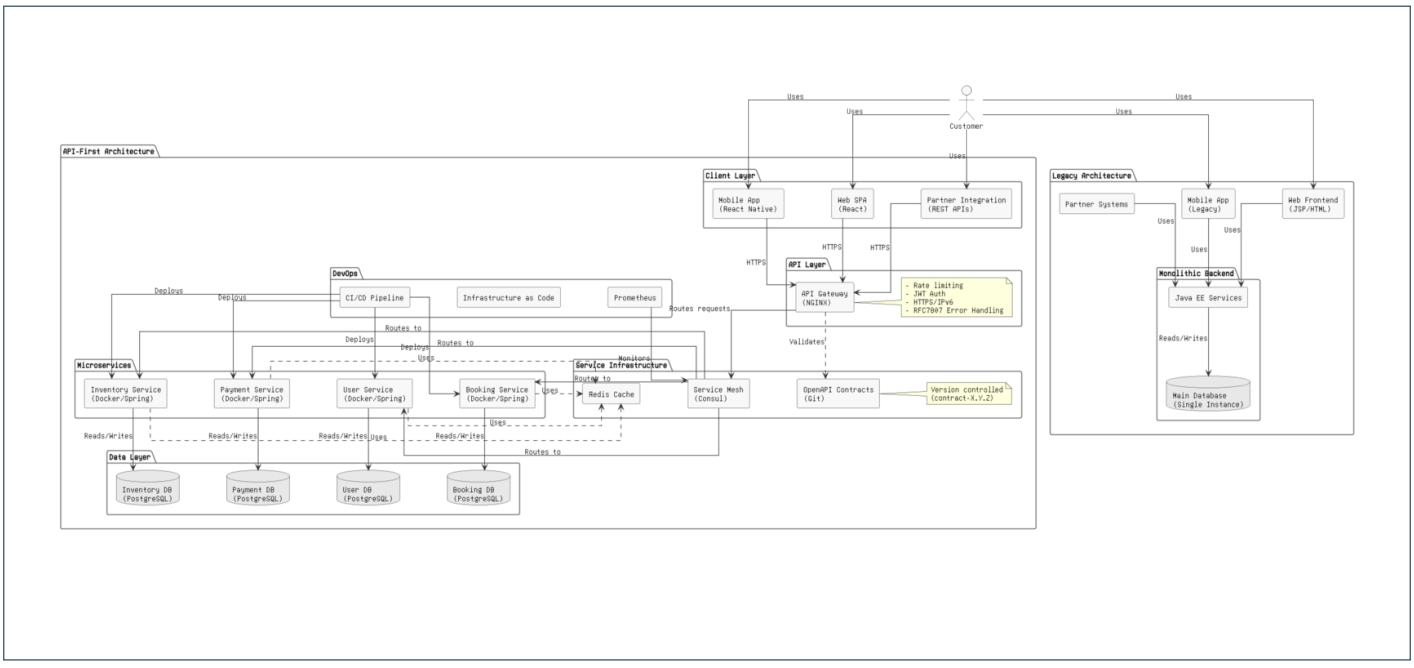
- Introduced endpoints for business and performance metrics (actuator/prometheus).
- Integrated with service discovery (Consul) and metric collection (Prometheus) tools.

Infrastructure as Code (IaC):

- Separated repositories for source code and infrastructure definitions.
- Adopted containerization and cloud-native deployment strategies using Docker and orchestration tools.

Continuous Integration and Deployment:

- Implemented automated testing as part of the deployment process, including E2E tests.
- Integrated with CI/CD tools for release management and transparency.



METRICS	Before	After	Improvement
System Performance			
API Response Time	800ms-1.2s	100-150ms	85% faster
System Availability	97.2%	99.95%	+2.75%
Peak Transaction Capacity	1,000/sec	5,000/sec	400% increase
Error Rate (Peak Load)	5%	0.1%	98% reduction
Critical Incidents (Monthly)	8	2	75% reduction
System Recovery Time	4-6 hours	15-30 min	85% faster
Development & Integration			
Partner Integration Time	8-12 weeks	3-4 weeks	65% faster
Feature Development Cycle	6-8 weeks	3-4 weeks	50% faster
Development Time on Maintenance	40%	15%	62.5% reduction
Partner Onboarding Success Rate	65%	95%	+30%
New Service Integration Time	4-6 weeks	2-3 weeks	50% faster
Developer Onboarding Time	4 weeks	2 weeks	50% faster
Operational Efficiency			
Infrastructure Costs	€1.5M-2M/year	€1.2M-1.6M/year	20% reduction
Deployment Frequency	weekly	daily	5x increase
Testing Coverage	60%	95%	+35%
Documentation Coverage	45%	95%	+50%
Partner Support Tickets	120/Month	45/Month	62.5% reduction
Release Rollback Rate	15%	3%	80% reduction
Business Impact			
Peak Season Booking Success	70%	99.9%	+29.9%
Partner Integration Capacity	4/months	10/month	150% increase
Time to Market (New Features)	3 months	1 month	66% faster
Partner Satisfaction Score	7.2/10	9.1/10	+26%
System Scalability (Peak Load)	300%	750%	150% increase
Revenue Loss Due to Downtime	€800K-1M/year	<€100K/year	88% reduction
Security & Compliance			
Security Incident Response	3 hours	30 min	80% faster
Authentication Methods	15+	2 standardized	—
Security Scan Coverage	70%	100%	+30%
Compliance Audit Time	4 weeks	1.5 weeks	—
Security Vulnerabilities	25/quarter	5/quarter	80% reduction
Data Encryption Coverage	85%	100%	+15%

RESULTS

- Integration Efficiency: Reduced time to integrate new services by 42%, with a 25% increase in third-party integrations within the second year.
- Time-to-Market: New features launched 35% faster on average, with the ability to prototype and test new services 25% quicker.
- Developer Productivity: Onboarding time for new developers reduced by 40%, and API-related bugs decreased by 20%.
- System Resilience: Achieved 99.95% uptime for core booking services and reduced critical incidents related to service integration by 50+%.
- Scalability: Successfully handled a 250% increase in booking traffic during peak seasons, with 60% faster deployment of new service instances.
- Improved Visibility: Comprehensive test coverage with easily accessible results and a real-time service status dashboard providing immediate insights across environments.

Timeline and Budget Range

Implementation: €2.9M (12 months)

Additional first-year support: €590K

Total first-year investment: € 3.5M

Break-even: 20–24 months

3-year ROI: 170% (base case)

Year 1: Implementation and initial returns (-€2.15M net)
Year 2: Full benefit realization (€1.87M)
Year 3: Optimized returns (€2.2M)

Team Structure

Technical Architect	1
Senior Developers	4
DevOps Engineers	2
Technical Project Manager	1
Business Analyst	1

VALUE PROPOSITION

- **Accelerated Innovation:** The API-First approach enabled faster development and integration of new features and services, keeping the platform competitive in the fast-paced hospitality market.
- **Cost Efficiency:** Standardization and improved scalability lead to better resource utilization, reducing infrastructure costs by 20%.
- **Enhanced Partner Ecosystem:** Easier integration and consistent APIs attracted more partners, expanding the platform's offerings and market reach.
- **Improved Developer Experience:** Standardized practices and clear documentation reduced friction in the development process, leading to higher productivity.
- **Futureproofing:** The modular, API-driven architecture made it easier to adopt new technologies and adapt to changing market needs without overhauling the entire system.
- **Data-Driven Decision Making:** Improved observability and standardized metrics across services enabled better insights and more informed business decisions by main stakeholders.

Ready to take the next step?
Reach us today at info@brightgrove.com