# FlyAway Airlines – Platform Redesign Specification

## 1. Background

FlyAway Airlines experienced critical issues with their previous microservice-based architecture. Core complaints included latency, inconsistency, frequent downtime under high load, and poor tooling for airline staff.

This redesign aims to replace the architecture with a modern, scalable, secure, and globally accessible platform for both airline staff and travelers.

## 2. Requirements

### Must Have

- Authentication (OAuth2, JWT) with GDPR compliance
- User and role management (customer, staff, admin)
- Flight search by destination, date, and geo-mapping
- Seat availability checks with class-specific limits
- Secure booking system with locking mechanism
- Payment processing with PCI-DSS compliant gateway
- Audit trails and history tracking
- Staff dashboard for traveler validation
- Mobile-optimized UI and global performance

### Should/Could Have

- Multilingual support and localization
- Loyalty programs, chatbot support
- Offline support for staff interface

## 3. System Architecture

The architecture uses a hybrid model:

- Microservices for Auth, Booking, Payments, Search, and History
- Modular monolith for internal staff tools
- API Gateway for traffic routing
- EKS/ECS for container orchestration
- RDS/Aurora PostgreSQL for consistent, reliable relational storage
- Redis for seat locking and high-speed caching
- S3 for encrypted document storage (e.g., passport scans, invoices)
- ElasticSearch for full-text flight/airport queries

### Security

- JWT-based access, TLS across services
- Secrets managed via AWS Secrets Manager
- IAM roles with fine-grained access per service

### Observability

- Prometheus for metrics
- Grafana dashboards
- CloudWatch for centralized logs
- OpenTelemetry for distributed tracing

### Data Flow Highlights

- **User Onboarding:** Auth service issues JWT, profile data stored in User DB, documents stored in S3.
- **Flight Search:** Query ElasticSearch/indexed DB, augment with airport geo data.
- **Booking:** Seats locked via Redis + Booking table transaction. TTLs ensure expired locks are released.
- **Payment:** Calls external gateway, updates Booking state on success, invoice saved to S3.
- **Audit:** All actions logged to History DB, accessible to users and admins.

## 4. CI/CD and Infrastructure

- CI: GitHub Actions with build/test/deploy stages
- CD: Auto-deploy to staging, approval-based to production
- IAC: Terraform or AWS CDK
- Environment isolation: Dev, staging, production
- Blue/green deployments for zero downtime updates

## 5. Implementation Plan

- Week 1–2: Infra provisioning, CI/CD setup
- Week 3–4: Auth, User, and Profile microservices
- Week 5–6: Search service and map integration
- Week 7–8: Booking system with seat lock logic
- Week 9–10: Payment flow and webhooks
- Week 11–12: Staff portal (monolith) and role features
- Week 13–14: Frontend and mobile UIs
- Week 15–16: Audits, migration, and go-live

## 6. Evaluation Criteria

- P95 latency < 300ms globally
- 99.95% uptime and auto-healing services
- No overbooked flights or payment errors
- Positive user/staff satisfaction surveys (NPS)
- Pass all compliance audits (PCI-DSS, GDPR)