Air Canada Flights Routes API

TABLE OF CONTENTS

1 Introduction 3

1.1 Document overview *3*

1.2 Abbreviations and Glossary *3*

1.2.1 Abbreviations 3

1.2.2 Glossary 3

1.3 References *3*

1.3.1 Project References 3

1.4 Conventions *3*

2 Architecture 4

2.1 Architecture overview *4*

2.2 Physical architecture overview *4*

2.2.1 Hardware Component 1 description 4

2.3 Logical architecture overview *4*

2.3.1 Software Component 1: Controllers 4

2.3.2 Software Component 2: Services 5

2.3.3 Software Component 3: Provider 5

2.4 Software COTS *5*

3 Dynamic Behavior Of Architecture 6

3.1 Workflow / Sequence 1 /flightRoutes *6*

3.2 Workflow / Sequence 2 /flightRoutesWithHotels *6*

4 Justification Of Architecture 7

4.1 System architecture capabilities *7*

4.2 Network architecture capabilities *7*

4.3 Risk analysis outputs *7*

4.4 SOUP integration *7*

5 Requirements Traceability 8

1 Introduction

1.1 Document overview

This document describes the architecture of the Air Canada Flight Routes API system.

It describes:

- A general description of the system and its purpose.
- The logical architecture of software components.
- The physical deployment architecture.
- The technical choices made and rationale.
- The traceability between the architecture and the system requirements.

1.2 Abbreviations and Glossary

1.2.1 Abbreviations

- API: Application Programming Interface
- REST: Representational State Transfer
- COTS: Components Off The Shelf (used: Express.js, Axios, Winston)

1.2.2 Glossary

- **Flight Routes API:** REST API exposing flight routes and enriched hotel data.
- **Provider:** Abstraction layer integrating with external travel APIs (currently Amadeus).

1.3 References

1.3.1 Project References

#	Document Identifier	Document Title
R1	Amadeus API Docs	https://developers.amadeus.com/self-service/apis-docs
R2	Express.js Docs	https://expressjs.com
R3	Node.js Docs	https://nodejs.org/en/docs

1.4 Conventions

- All requests carry a X-Correlation-Id header for tracing.
- All errors follow a standard JSON structure.
- Architecture diagrams use standard box-arrow notation.

2 Architecture

2.1 Architecture overview

- Environment: Cloud-hosted (Render.com)
- Users: Internal systems, partner airline systems
- Purpose: Provide partner-facing API for retrieving flight offers and hotels near destinations.
- Main functions:
  - /flightRoutes endpoint → Flight offers
  - /flightRoutesWithHotels endpoint → Flight offers + hotels
- Interfaces:
  - Input: RESTful GET requests
  - Output: JSON responses

2.2 Physical architecture overview

- Cloud-hosted → Render.com → Linux-based container deployment.
- No dedicated physical hardware managed by system.

2.2.1 Hardware Component 1 description

- Cloud-hosted on Render platform → auto-scaling container-based architecture.
- No physical servers managed directly.

2.3 Logical architecture overview

Software components:

- Express Application
  - Controllers Layer
  - Services Layer
  - Provider Layer
  - Middleware Layer
  - Utils Layer
- External dependency: Amadeus API

Operating system: Linux-based container on Render.

2.3.1 Software Component 1: Controllers

- Purpose: Expose API endpoints, handle input validation.
- Interfaces: REST API, response headers.
- Network: HTTP over TLS.
- Hardware: Minimal resource usage.

2.3.2 Software Component 2: Services

- Purpose: Orchestrate business logic.
- Interfaces: Controllers → Provider.

2.3.3 Software Component 3: Provider

- Purpose: Abstract external API (Amadeus).
- Interfaces: REST API calls to Amadeus.
- Future-proofed for additional providers.

2.4 Software COTS

Component	Version	Purpose	Maintained?
Express.js	4.x	Web framework	Yes
Axios	1.x	HTTP client	Yes
Winston	3.x	Logging	Yes

3 Dynamic Behavior Of Architecture

3.1 Workflow / Sequence 1 /flightRoutes

- Client sends GET request to /flightRoutes.
- Middleware attaches CorrelationId.
- Controller validates request.
- Service calls Provider.
- Provider retrieves flight data from Amadeus.
- Service builds response.
- Controller sends response → logs.

3.2 Workflow / Sequence 2 /flightRoutesWithHotels

- Client sends GET request to /flightRoutesWithHotels.
- Same flow as above, with Provider making additional hotels API call.

4 Justification Of Architecture

4.1 System architecture capabilities

- **Performance**: Lightweight, optimized for RESTful requests.
- **Safety**: Not applicable — informational API.
- **Protection**: Rate limiting enforced.
- **Scalability**: Cloud-native → can auto-scale.
- **Availability**: Highly available → Render platform + CDN.
- **Security**: TLS enforced; API Gateway can be added.
- **Administration**: CI/CD with GitHub → auto-deploys.
- **Monitoring**: Health check endpoint + structured logging.

4.2 Network architecture capabilities

- No dedicated network hardware.
- Cyber security: TLS required, secure API calls.
- Data loss: Handled by retries on client side (Provider layer uses retry logic).

4.3 Risk analysis outputs

- Rate limiting prevents abuse.
- Request timeout protects against long-hanging requests.
- Standardized error handling improves client UX.

4.4 SOUP integration

COTS components (Express, Axios, Winston) documented in 2.4.

5 Requirements Traceability

Requirement	Component	Comment
REQ-001: Provide flight routes API	Controllers, Services, Provider	Implemented in /flightRoutes
REQ-002: Provide flight routes with hotels API	Controllers, Services, Provider	Implemented in /flightRoutesWithHotels
REQ-003: Implement health check	Middleware, /health endpoint	Implemented
REQ-004: Implement rate limiting	Middleware	Implemented
REQ-005: Implement request timeouts	Middleware, Axios config	Implemented