**Enrollment Verification Service** **partner-agnostic**,

The architecture should follow key principles like interface-driven development, configuration-based routing, and modularity.

Here’s the **refined architecture design** that enables **seamless onboarding of new partners** with minimal code changes:

**High-Level Architecture Overview**

┌────────────────────────────┐

│ EnrollmentVerificationAPI │ ← Partner-Agnostic Controller

└────────────┬───────────────┘

│

┌────────────────────────────────────┐

│ PartnerRouter (based on config) │ ← Routes request to correct implementation

└────────────────┬───────────────────┘

▼

┌─────────────────────────────────────────────────────┐

│ EnrollmentVerificationPartnerIntegrationService │ ← Interface (Contract) │

└─────────────────────────────────────────────────────┘

▲ ▲ ▲ ▲

│ │ │ │

┌──────────────┐ ┌────────────┐ ┌────────────┐ ┌────────────┐

│AirCanadaImpl │ │AviosImpl │ │IHGImpl │ │MarriottImpl│ ← Partner-specific logic

└──────────────┘ └────────────┘ └────────────┘ └────────────┘

**Key Design Components**

**1. Single Generic Controller (EnrollmentVerificationAPI)**

* Only one controller handles incoming requests.
* Accepts a partnerCode or partnerId in the request.
* No need to create new controller classes per partner.

**2. Partner Router (Factory or Strategy Pattern)**

* Based on partnerCode, dynamically loads the correct implementation.
* Configuration-driven mapping:

yaml

CopyEdit

partner:

aircanada: AirCanadaImpl

avios: AviosImpl

ihg: IHGImpl

marriott: MarriottImpl

* Easily extendable via Spring Boot, reflection, or DI frameworks.

**3. Interface: EnrollmentVerificationPartnerIntegrationService**

* Defines a common contract (e.g., verifyEnrollment(request): response)
* All partner implementations conform to this.

**4. Partner Implementations**

* Each partner has its own package with:
  + Request/response mappers
  + Validation logic
  + API calls, retries, etc.
* Onboarding a new partner = create new class, register in config.

**5. Shared Model & Exception Handling**

* Shared request and response models under model/
* Global exceptions, enums, and utils

| **Feature** | **Benefit** |
| --- | --- |
| One Controller | No more boilerplate duplication |
| Config-Driven Routing | No code change needed to onboard a partner |
| Interface-Driven | Enforces standard behavior across partners |
| Independent Modules | Easy unit/integration testing per partner |
| Spring Boot DI or Reflection | Load new partner beans without restarting app |

**Example: How to Add a New Partner**

1. Create new service class: HyattEnrollmentVerificationServiceImpl.java
2. Implement the EnrollmentVerificationPartnerIntegrationService interface
3. Add entry to config:

yaml

CopyEdit

partner:

hyatt: HyattEnrollmentVerificationServiceImpl

1. No controller/service wiring changes needed.

