Service Interaction Strategy

Part 1: REST-First Strategy (Current Phase)

In the current implementation phase, the system will follow a REST-first microservices communication strategy. All microservices will expose standardized, versioned RESTful APIs over HTTP, with JWT-based authentication and internal service-to-service authorization policies enforced.

Service-to-Service Communication

Each microservice is independently deployed and interacts with other services using REST APIs with stateless HTTP requests. The FastAPI framework is used to expose Swagger/OpenAPI-compliant interfaces, enabling discoverability and auto-documentation of all endpoints.

All REST APIs are:

- Secured via bearer tokens (Authorization: Bearer <JWT>)
- Communicated via HTTPS in production
- Versioned using URI pathing (e.g., /api/v1/employee/{id})

REST API Contracts Between Services

Below is the list of inter-service interactions in Phase 1:

Employee → Auth Service

Endpoint	Method	Description
/auth/verify-token	POST	Internal token verification used by middleware in downstream services

---

Payroll Service → Employee Service

Endpoint	Method	Description
/employee/{id}	GET	Fetches personal and salary base data for payroll processing
/employee/{id}/job-role	GET	Used to determine job role and pay band

---

Payroll Service → Attendance Service

Endpoint	Method	Description
/attendance/summary/{employee_id}?month=YYYY-MM	**GET**	Returns working day count and late entries for payroll computation

---

Leave Service → Employee Service

Endpoint	Method	Description
/employee/{id}/manager	GET	Resolves the reporting manager for leave approval hierarchy
/employee/{id}	GET	Used to fetch current status before approving leave (e.g., active/inactive)

AI Service → Multiple Services

Endpoint	Method	Consumed From	Description
/employee/all	GET	Employee Service	For training career path prediction model
/reviews/all	GET	Performance Review Service	For retention prediction
/attendance/history	GET	Attendance Service	Used as part of behavior analysis
/mood/latest	GET	Mood Index Service	Used to correlate mood with attrition

---

Security Strategy for REST Interactions

- All services include a shared JWT verification module using asymmetric key pairs (RSA256) to validate token claims.
- Roles (admin, hr_manager, employee_manager, employee) are extracted from JWT payload and verified using middleware per route.
- Internal service-to-service authentication is handled via service-level tokens using a microservice identity model.

This REST-first strategy ensures clean API boundaries, testable interfaces, and minimal coupling between services, while still allowing for easy scaling and later replacement with asynchronous or event-driven components.

Part 2: Message Bus Strategy (Pub/Sub – Future Phase)

As the system scales and inter-service complexity increases, adopting a message bus architecture is critical to maintaining loose coupling, improving reliability, and enabling reactive AI microservices. In this future phase, a Pub/Sub (Publisher/Subscriber) pattern will be introduced using tools such as RabbitMQ or NATS, depending on performance and durability requirements.

Rationale for Pub/Sub Integration

The current REST-based synchronous model works well for core interactions. However, certain scenarios require asynchronous, event-driven behaviour where one service needs to "react" to a change that occurred in another, without requiring immediate responses.
Key advantages of using a message bus include:

- Decoupling services: Publishers don't need to know about the subscribers.
- Improved fault tolerance: Services can buffer and retry in case of failure.
- Better scalability: Services can horizontally scale and subscribe to only relevant event queues.
- Enabling AI microservices to passively observe behavioural changes in real time, such as mood logs or performance updates.

Core Event Flows

In the Pub/Sub model, services emit **domain events** that are picked up by interested consumers. These events are lightweight payloads describing an action that just occurred.
Below is a list of planned key events:

Event Name	Publisher Service	Subscriber Services	Purpose
employee_created	Employee Service	Payroll, Attendance, AI Career Path	Trigger salary setup, attendance creation, and career path initialization
leave_approved	Leave Service	Employee Manager Dashboard, HR Reports	Update calendars, notify HR dashboards
payroll_generated	Payroll Service	Payslip Generator, Notification Service	Automatically generate payslips and notify employees
mood_logged	Mood Index Service	Retention Risk Predictor (AI)	Feed mood score for attrition prediction

---

Tools & Implementation Options

Several technologies are under consideration for implementing the event-driven layer:

1. RabbitMQ

- Message Queue-based: Supports durable, ordered message delivery.
- Offers routing, exchange types, and dead-letter queues .
- Excellent for transactional events like payroll_generated.

2. NATS

- Lightweight and high-throughput pub/sub system.
- Ideal for broadcasting events like mood logs or login audits.
- Native support for microservices written in Go, Node.js, Python, etc.

3. Celery + RabbitMQ (Early MVP Option)

- Use Celery workers with FastAPI background tasks to simulate async jobs.
- Minimal setup and aligns well with current FastAPI ecosystem.
- Suitable for tasks like sending notification emails post-event.

Architecture Plan

Initially, services can simulate event-based communication via POST /event calls to internal endpoints:

```
POST /internal/event/employee_created
 {
  "employee_id": 101,
  "name": "Shobhon Sengupta",
  "role": "ML Engineer",
  "doj": "2025-06-01"
 }
```

This allows for gradual adoption of event-based flows without tightly binding to a queue infrastructure.

In future production environments, this will be replaced by a message broker:

```
[Employee Service] --(event)---> [RabbitMQ Exchange] ---> [Payroll, Attendance, AI Services]
```

Events will be categorized into:

- **Domain Events**: e.g., employee_created, leave_approved
- **System Events**: e.g., service_started, error_occurred
- **AI/Analytics Events**: e.g., mood_logged, login_audit

Event Security & Reliability

- Messages will be digitally signed and schema-validated (via Pydantic or Avro) before publishing.
- Queues will implement acknowledgment strategies to ensure message delivery.
- Dead-letter queues (DLQ) will be used to capture failed or malformed events for further inspection.

Summary

The integration of a message bus will provide long-term benefits in performance, maintainability, and modularity of the system. It unlocks advanced features like real-time analytics, AI reaction loops, and plug-and-play microservice addition, which are core to the product's innovation goals.
Initial implementation will emulate this behavior via RESTful internal event endpoints and background tasks. In later phases, the system will transition to a full-fledged event-driven microservices architecture using RabbitMQ or NATS depending on observed load and reliability needs.

Part 3: External APIs (3rd Party Integrations)

To enhance the core functionality of the AI-driven HRMS platform and provide enriched user experiences, the system will integrate with multiple third-party APIs. These APIs will extend system intelligence, automate verifications, support personalized learning recommendations, and facilitate financial wellness for employees.
All external API interactions will be abstracted behind service-specific proxy endpoints and will be secured using API keys, OAuth2 tokens, or signed payloads. Error handling, rate-limiting, and caching strategies will be implemented to ensure high availability and reliability of integrations.

Planned External Integrations

External API	Integrated Module	Purpose
Glassdoor / LinkedIn Salary APIs	Payroll AI Module	To benchmark employee compensation based on role, experience, and geography. Helps dynamically recommend salary revisions for retention.
Coursera / LMS APIs	Career Path AI Module	To fetch recommended upskilling courses tailored to employee performance and growth predictions. Supports personalized development journeys.
GST/PAN/Bank Verification APIs	Employee Onboarding / Profile Management	Enables automatic validation of government-issued IDs and bank accounts during employee onboarding, reducing HR workload and fraud risk.
Fintech APIs (e.g., RazorpayX, Slice, Jupiter)	Financial Health Module	Allow employees to view credit scores, set savings goals, and apply for salary advances or insurance through trusted partners via API.

Security and Governance

All external integrations will follow the principle of least privilege and operate via encrypted channels (HTTPS/TLS). API credentials will be securely managed via:

- Environment variables or secret managers (e.g., AWS Secrets Manager)
- Token rotation and expiration policies
- Webhooks (where applicable) with HMAC or JWT signing for validation

---

Standardization Approach

- External APIs will be wrapped by internal service adapters that provide a uniform contract to the rest of the system.
- Responses will be validated using Pydantic schemas to handle API drift or schema mismatches.
- Retry logic and fallback caching will be implemented using tools like httpx, tenacity, or custom middlewares.

Conclusion

To build a smart and reliable HR and Payroll system, we've laid out a step-by-step communication plan between all the moving parts. Think of it like designing how different departments in an office talk to each other — sometimes through direct conversation (REST APIs), sometimes through notices or announcements (message events), and sometimes by reaching out to experts outside the company (external APIs).

First, we focused on simple, direct connections between each service using web links (called REST APIs). For example, the Payroll service can ask the Attendance service how many days someone worked, or the Leave service can check who the employee's manager is before approving leave. This approach is easy to understand, simple to build, and great for getting the system up and running quickly.

Next, we planned for the future — where services won't always need to talk directly. Instead, they'll send out event messages, like "a new employee has joined" or "payroll is completed," and any other part of the system that cares about that information will get notified. This makes the system faster, cleaner, and more flexible — like putting up a company-wide announcement instead of making dozens of individual calls.

Finally, we included ways to connect with trusted outside sources — like salary data from Glassdoor, courses from Coursera, or financial help from fintech companies. These external services help the system get smarter — recommending better career paths, validating identity faster, or helping employees with loans and savings — all automatically.

Together, these three strategies form a powerful communication plan that allows the HRMS to grow over time. It starts simple but is built with the future in mind — where everything works smoothly, smart features are triggered automatically, and employees get a personalized experience without extra work from HR.
