ADR-9

Use of a Shared Database Across All Microservices

Date: 15-04-2025

Status

Accepted

Author

System Architecture Team

Context

Our system is decomposed into multiple microservices (User, Facility, Booking, and Payment). While designing the persistence layer, a key architectural decision had to be made: whether each microservice should maintain its own dedicated database (Database-per-service pattern) or whether a shared database should be used across services.

Decision

We decided to use a **single shared relational database** for all the microservices in our system.

Alternatives

Alternative	Pros	Cons
Shared Database (Chosen)	Simple setup, easy to query across services, low maintenance	Breaks microservice independence, potential coupling
Database per Microservice	High modularity, better scalability and autonomy	Overkill for current system scale, complex setup, more tables needed

Rationale

Given our current project scope:

- The system has relatively **simple schema requirements**, with not many tables.
- Introducing **separate databases** per microservice adds operational and development overhead, which is not justified at this stage.
- A **shared database enables faster development**, easier coordination, and meets all our current stakeholder needs (as identified in the Stakeholder Table).

This decision is revisitable if data volume, complexity, or cross-service interference increases significantly.

Consequences

Positive

- Easier data inspection and debugging during development.
- Simpler DevOps pipeline and lower infrastructure cost.
- No need for complex data replication or synchronization strategies.

Negative

- Potential for tight coupling between services.
- Difficult to enforce strict ownership over tables across services.
- Limits flexibility for future scaling or independent evolution of services.