SPEC-05: Synapse Module

1. Background

The **Synapse** module is the integration broker and orchestration layer in the Hearthlink ecosystem. It ingests events from UI modules (Alden, Alice), decision engines (Mimic), monitoring systems (Sentry), and external integrations; routes, transforms, and prioritizes them to downstream services (Vault, Core Services, analytics); and ensures reliable, scalable message flows with schema validation, backpressure handling, and retries.

2. Requirements (MoSCoW)

Must have

- Event bus abstraction supporting pub/sub and request/response patterns (Kafka/NATS)
- Schema registry integration for JSON/Avro validation on each message
- At-least-once delivery with configurable retry/backoff policies
- Dynamic routing rules based on metadata (topic, headers, payload content)
- Health-check and metrics endpoints for observability (Prometheus)

Should have

- Dead-letter queue (DLQ) handling with reprocessing dashboard
- Circuit-breaker patterns to pause routing to failing services
- Operational dashboard showing throughput, lag, error rates per stream

Could have

- On-the-fly transformation scripts (JavaScript/SQL) loaded from secure store
- Multi-cluster federation for geo-redundant replication

Won’t have (this increment)

- Native mobile SDK for event consumption
- Event replay from archival storage (deferred)

3. Method

3.1 Architecture Diagram

```
@startuml
package "Synapse Cluster" {
  [Ingress API] --> [Message Broker]
  [Router Service] <-- [Message Broker]
  [Router Service] --> [Workers]
  [Workers] --> [Outbound API]
  [Workers] --> [Dead-Letter Queue]
  [Metrics Exporter] --> [Prometheus]
}
package "Schema Registry" {
  [Registry] <-- [Ingress API]
  [Registry] <-- [Router Service]
}
@enduml
```

3.2 Data Schema & Flow

```
@startuml
table MessageMeta {
  + message_id   : UUID [PK]
  + topic        : VARCHAR
  + headers      : JSON
  + payload_ref  : UUID
  + status       : ENUM('PENDING','PROCESSING','FAILED','ACKED')
  + created_at   : TIMESTAMP
  + updated_at   : TIMESTAMP
}

actor Ingress
actor Router
Ingress -> MessageMeta : write(PENDING)
Ingress -> MessageBroker : publish(topic, message_id)
Router -> MessageBroker : subscribe(topic)
Router -> MessageMeta : update(PROCESSING)
Router -> Workers : invoke(message)
Workers -> OutboundAPI : request(message)
Workers -> MessageMeta : update(ACKED)
Workers --> DeadLetter : onFailure(message)
@enduml
```

4. UI Components & Wireframes

4.1 Synapse Dashboard Overview

```
+--------------------------------------------------------------+
| Synapse Dashboard                                            |
| [Throughput Chart] [Error Rate Chart] [Lag Gauge]            |
| [Topic Selector ▼] [Service Filter ▼] [Refresh]              |
+--------------------------------------------------------------+
```

Component	Function	Data/API Call
ThroughputChart	Displays messages/sec over time	`GET /v1/metrics?metric=throughput`
ErrorRateChart	Shows count and rate of errors	`GET /v1/metrics?metric=errors`
LagGauge	Indicates consumer lag for each topic	`GET /v1/metrics?metric=lag`
TopicSelectorDropdown	Filters across topics	`GET /v1/routes?topic={topic}`
ServiceFilterDropdown	Filters by source/destination service	`GET /v1/routes?service={service}`
RefreshButton	Manual data refresh trigger	Reload charts via client-side API calls

4.2 Dead-Letter Queue (DLQ) Panel

```
+--------------------------------------------------------------+
| Dead-Letter Queue                                            |
| [Topic Filter ▼] [Reprocess All] [Export]                    |
|--------------------------------------------------------------|
| [List of messages with columns: ID, Topic, Error, Date]      |
+--------------------------------------------------------------+
```

Component	Function	Data/API Call
DLQTable	Displays failed messages	`GET /v1/deadletter?limit=50`
ReprocessButton	Reprocess selected or all messages	`POST /v1/deadletter/reprocess/{id}`
ExportButton	Export DLQ entries to CSV	`POST /v1/deadletter/export`

4.3 Routing Rules Management Panel

```
+--------------------------------------------------------------+
| Routing Rules                                                 |
| [Create Rule] [Import/Export Rules] [Refresh]                |
|--------------------------------------------------------------|
| [List of rules: ID, Source Topic, Conditions, Destinations]|
|--------------------------------------------------------------|
| [Rule Detail Pane: Conditions Editor, Actions Editor]        |
+--------------------------------------------------------------+
```

Component	Function	Data/API Call
RulesListTable	Lists existing routing rules with filters	`GET /v1/routes`
CreateRuleButton	Opens modal to define a new routing rule	N/A
ImportExportRulesBtn	Bulk import/export routing definitions (JSON/CSV)	`POST /v1/routes/import`, `GET /v1/routes/export`
RuleDetailPane	Edit selected rule’s conditions and destinations	`GET /v1/routes/{ruleId}`, `PUT /v1/routes/{ruleId}`
DeleteRuleButton	Deletes a selected routing rule	`DELETE /v1/routes/{ruleId}`
RefreshButton	Reloads the routing rules list	`GET /v1/routes`

5. API Endpoints. API Endpoints

Method	Path	Description	Auth Scope
POST	/v1/events	Ingest or publish a new event	`synapse.ingest`
GET	/v1/events/{id}/status	Retrieve processing status	`synapse.read`
POST	/v1/routes	Create or update routing rules	`synapse.write`
GET	/v1/routes	List active routing configurations	`synapse.read`
GET	/v1/events	Query event metadata and history	`synapse.read`
POST	/v1/deadletter/reprocess/{id}`	Reprocess a failed message from DLQ	`synapse.write`
GET	/v1/deadletter	List messages in dead-letter queue	`synapse.read`

6. Implementation

1. **Broker & Registry Setup**
2. Deploy Kafka/NATS cluster with TLS and replication
3. Configure Schema Registry (Confluent/Apicurio) for topic enforcement
4. **Ingress API**
5. Implement Go(Fiber) `/v1/events` endpoint
6. Validate payloads via Schema Registry before publish
7. **Router Service**
8. Build using Kafka Streams/Flink; dynamic routing based on metadata
9. Integrate retry/backoff and circuit-breaker (Resilience4j)
10. **Worker Services**
11. Containerized microservices for domain routing (Vault, Sentry, Core)
12. Emit metrics and handle DLQ logic

13. **Dashboard UI**
14. React/Tailwind components for charts and tables; integrate Recharts
15. Bind data via REST API clients and WebSocket for live updates
16. **Observability & Security**
17. Expose Prometheus metrics; secure mTLS between services
18. Audit routing rule changes via ADR store
19. **Testing & Validation**
20. E2E tests simulating high-throughput (>20k msg/s)
21. Fault-injection tests (broker down, schema errors)

7. Milestones

Milestone	Timeline	Owner
Broker & Registry Provisioning	Week 1	DevOps Lead
Ingress API & Validation	Week 2	Backend Team
Router Service MVP	Week 3–4	Integration Team
Worker Implementation & DLQ	Week 5	Backend Team
Dashboard & DLQ UI	Week 6	Frontend Team
Security Hardening & mTLS	Week 7	Security Team
Performance & Fault Testing	Week 8–9	QA Team

8. Gathering Results

- End-to-end latency <50 ms under nominal load
- At-least-once delivery with zero data loss in fault tests
- DLQ reprocessing success >95%
- Dashboard metrics accuracy >99%
- Circuit-breaker effectiveness validated in fault scenarios

9. References & Dependencies

- **Integration Blueprints**: appendix_b_integration_blueprints.md
- **UI Blueprints**: appendix_c_ui_blueprints.md
- **Prometheus & Grafana Guides**: _DEVELOPMENT_OPERATIONS_GUIDE.md
- **Security Policies**: VOICE_ACCESS_POLICY.md sections on mTLS

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)