SPEC-02-Vault Module

Background

The Vault module is the centralized secret management and credential-brokering service at the core of the Hearthlink ecosystem. It safeguards symmetric and asymmetric keys, API tokens, database credentials, and transient session secrets. Vault's design must address cross-domain consumption by UI components, backend microservices, integration blueprints, and audit pipelines—providing high-availability, strong encryption, and tamper-evident logging without becoming a bottleneck for developer or runtime workflows.

Requirements

Using the MoSCoW method:

Must have

- Secure at-rest storage of all secrets using AES-256-GCM with per-secret random IVs
- Integration with an HSM (or KMS) for root key wrapping and unwrapping (FIPS-140-compliant)
- Role-based Access Control (RBAC) for secret read/write operations, tied to OAuth2 scopes
- Immutable audit log of all vault operations (create, read, update, delete) with timestamp, actor, and context
- **Comprehensive log retention policy:** Store all audit logs for a minimum of 90 days, with automated cycling/archiving of older logs to avoid storage bloat
- High-availability cluster (active-active) with automated failover and data replication

Should have

- Dynamic secret leasing and automatic rotation for database credentials (e.g. RDS-style)
- Credential brokering API for short-lived tokens (JWT or OAuth2 client_credentials grants)
- JSON-schema–validated secret templates to enforce structure (e.g., SSH key pairs, TLS certs)

Could have

- Optional multi-tenant namespace isolation for scoped v2 deployments
- Pluggable storage backends (e.g. AWS S3, Azure Key Vault, on-premise HSM clusters)

Won't have (this increment)

- Client-side encryption libraries (deferred to future client SDK release)
- Automatic secret discovery across external integrations

Method

Architecture Component Diagram

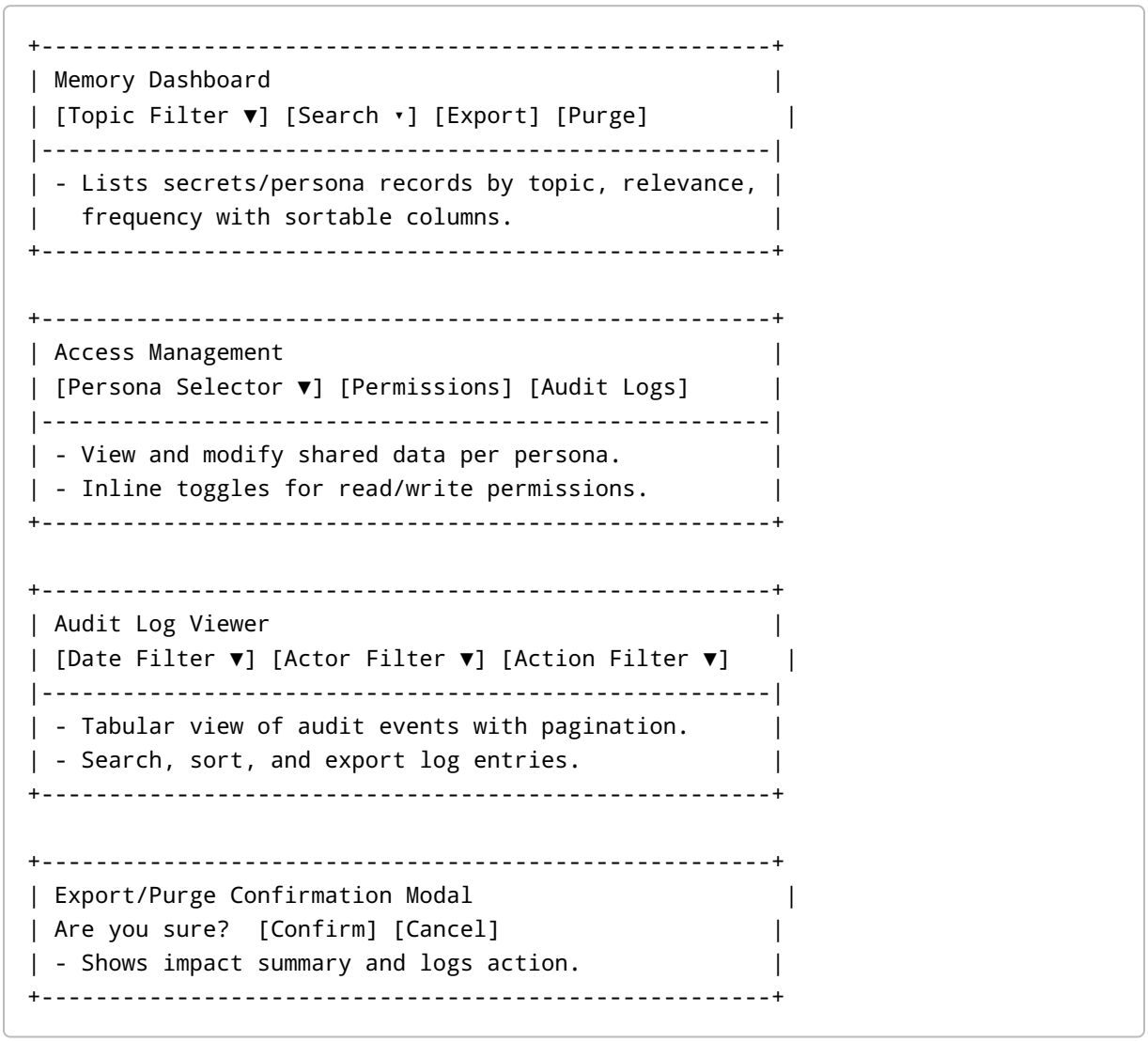
```
@startuml
package "Vault Cluster" {
  [Vault API] --> [Storage Backend]
  [Vault API] --> [HSM/KMS]
  [Vault API] --> [Audit Database]
}
package "Consumers" {
  [UI Components]
  [Backend Services]
  [Integrations]
}
[Consumers] --> [Vault API]
@enduml
```

Data Schema

```
@startuml
table SecretRecord {
  + id : UUID [PK]
  + namespace : VARCHAR
  + type : VARCHAR
  + payload_ciphertext : VARBINARY
  + iv : VARBINARY(16)
  + created_at : TIMESTAMP
  + version : INT
}

table AuditLog {
  + log_id : BIGINT [PK, auto]
  + secret_id : UUID [FK]
  + action : ENUM('CREATE','READ','UPDATE','DELETE')
  + actor : VARCHAR
  + timestamp : TIMESTAMP
  + metadata : JSON
}
@enduml
```

4. UI Components & Wireframes



Component	Function	Data/API Call
MemoryDashboardTable	Displays secret records with filters and sort	`GET /v1/secrets?namespace={ns}&type={type}`
TopicFilterDropdown	Filters records by namespace	Client-side filtering
SearchInput	Searches records by metadata	`GET /v1/secrets?search={query}`
ExportButton	Exports current view to CSV, JSON	`POST /v1/secrets/export`
PurgeButton	Initiates purge modal	N/A

Component	Function	Data/API Call
PersonaSelectorDropdown	Chooses persona context for access management	`GET /v1/personas`
PermissionsToggle	Grants or revokes read/write access	`PUT /v1/secrets/{id}/permissions`
AuditLogsButton	Opens audit log pane	`GET /v1/audit/logs?secret_id={id}`
AuditLogViewer	Renders paginated audit logs with filters and export functionality	`GET /v1/audit/logs?start={date}&actor={actor}&action={action}&limit=50`
ConfirmationModal	Confirms destructive actions; shows summary	N/A

9. References & Dependencies

- **Integration Blueprints**: appendix_b_integration_blueprints.md (authentication flows, error handling patterns)
- **UI Blueprints**: appendix_c_ui_blueprints.md (component library mapping)
- **SOP for Vault Management**: SOP_Role_Assignment.md (policy for RBAC roles)
- **DevOps Guide**: _DEVELOPMENT_OPERATIONS_GUIDE.md (CI/CD, failover playbooks)

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)

- **Master Key:** Stored in HSM/KMS; never leaves secure boundary
- **Data Encryption Key (DEK):** Generated per secret, wrapped by Master Key
- **Cipher:** AES-256-GCM with 128-bit IV, 128-bit authentication tag
- **Compliance:** HSM integration for FIPS-140; optional HIPAA-safe logging (PII redaction)

Implementation

1. **Provision infrastructure:**
2. Deploy Vault nodes in Kubernetes with StatefulSets, attached PVs encrypted via CSI
3. Configure Vault Operator to manage unseal keys via HSM integration
4. **Define RBAC policies:**
5. Write HCL policies for read/write/lease/audit roles
6. Map OAuth2 scopes to policies in ingress proxy
7. **Database & Audit:**
8. Create `SecretRecord` and `AuditLog` tables in PostgreSQL with TDE enabled
9. Deploy metrics exporter and log forwarder for audit streams
10. **Implement log cycling job:** configure a scheduled cleanup/archival process that moves audit records older than 90 days to cold storage and purges from primary DB to control table size

11. **API & SDK: API & SDK:**
12. Implement endpoints in Go (Gin framework) using native Vault SDK patterns
13. Provide OpenAPI spec and generated client libraries
14. **Testing & Hardening:**
15. Integration tests for all endpoints, secret rotation, failover scenarios
16. Penetration testing, static code analysis, dependency scanning

Milestones

Milestone	Timeline	Owner
Infra & HSM integration	Week 1–2	DevOps Lead
Core CRUD API & storage tests	Week 3–4	Backend Team
RBAC & Audit logging	Week 5	Security Team
Dynamic leasing & rotation	Week 6–7	Backend Team
SDK & documentation release	Week 8	Tech Writing
Performance & fault testing	Week 9–10	QA Team

Gathering Results

- Verify >99.9% uptime in HA cluster under simulated failures
- Audit logs retention and immutable verification every 24 hrs
- Encryption key integrity checks via HSM health probes
- Successful end-to-end secret issuance and rotation within SLA (<100 ms)

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)