SPEC-06: Mimic Module

1. Background

The **Mimic** module encapsulates persona simulation and decision-making logic within Hearthlink. It maintains user-defined persona profiles, evaluates input context (from Alice CBP and Synapse events), and generates adaptive recommendations or automated actions. Mimic drives targeted UI suggestions, automated workflows, and persona-driven content personalization.

2. Requirements (MoSCoW)

Must have

- Persona management: CRUD persona profiles with trait definitions
- Decision engine: rule-based and weighted algorithms for recommendations
- Integration with CBP: ingest Alice CBP data and adjust persona state
- Exposed UI for persona selection and trait editing
- Secure API scopes: `mimic.persona.*`, `mimic.decision.*`

Should have

- Real-time persona switching and context injection into UI panels
- Analytics on persona usage and effectiveness metrics
- Persona versioning and rollback

Could have

- Machine-learned persona profiling from historical data
- Persona marketplace import/export

Won't have (this increment)

- External AI model training pipelines (deferred)

3. Method

3.1 Architecture Diagram

```
@startuml
package "Mimic Core" {
  [Persona API] --> [Persona Store]
  [Decision Engine] --> [Rules Engine]
  [Decision Engine] --> [Weights Store]
  [Decision Engine] --> [Analytics Export]
}
```

```
package "Consumers" {
  [Alden UI]
  [Alice]
  [Project Command]
}
Consumers -> Persona API
Consumers -> Decision Engine
@enduml
```

3.2 Data Schema

```
@startuml
table Persona {
  + persona_id : UUID [PK]
  + name       : VARCHAR
  + traits     : JSON   -- { trait: weight }
  + created_at : TIMESTAMP
  + updated_at : TIMESTAMP
}

table Rule {
  + rule_id    : UUID [PK]
  + persona_id : UUID [FK]
  + condition  : JSON   -- e.g. { cbp.sentiment < 0.2 }
  + action     : VARCHAR
  + priority   : INT
}

table WeightHistory {
  + history_id : BIGINT [PK]
  + persona_id : UUID [FK]
  + trait       : VARCHAR
  + old_weight  : FLOAT
  + new_weight  : FLOAT
  + changed_at  : TIMESTAMP
}
@enduml
```

4. UI Components & Wireframes

4.1 Persona Management Panel

```
+-----------------------------------------------------------+
| Mimic Persona Manager                                     |
| [Create Persona] [Import ▼] [Export ▼] [Refresh]          |
```

```
|-----------------------------------------------------------|
| [Persona List: Name, #Traits, Last Used, Actions]         |
+-----------------------------------------------------------+
```

Component	Function	Data/API Call
PersonaTable	Lists personas	`GET /v1/personas`
CreatePersonaButton	Opens new persona form	N/A
ImportDropdown	Import persona JSON	`POST /v1/personas/import`
ExportDropdown	Export selected or all personas	`GET /v1/personas/export`
RefreshButton	Reload persona list	`GET /v1/personas`
PersonaActionsCell	Edit/Delete persona	`PUT /v1/personas/{id}`, `DELETE`

4.2 Persona Detail & Trait Editor

```
+-----------------------------------------------------------+
| Persona: <Name>                                           |
| [Back] [Save Traits]                                      |
|-----------------------------------------------------------|
| [Trait Name] [Weight Slider] [±] [History ▼]              |
| (Repeat for each trait)                                  |
+-----------------------------------------------------------+
```

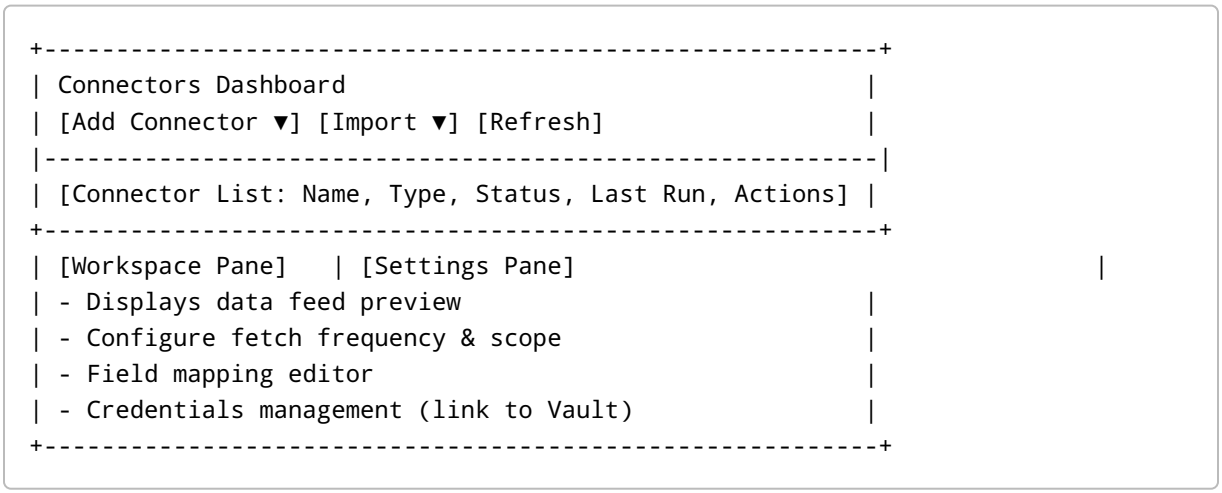
Component	Function	Data/API Call
TraitList	Displays trait names and sliders	`GET /v1/personas/{id}/traits`
WeightSlider	Adjust trait influence weight	Local UI; on change triggers PUT below
IncrementButton [+/-]	Fine-tune weight by step	`PUT /v1/personas/{id}/traits`
HistoryDropdown	Show weight change history	`GET /v1/personas/{id}/history`
SaveTraitsButton	Persists trait updates	`PUT /v1/personas/{id}`

4.3 Decision Simulation Panel

```
+-----------------------------------------------------------+
| Decision Engine Simulator                                 |
| [Select Persona ▼] [Input Context ▼] [Simulate] [Reset]   |
|-----------------------------------------------------------|
| [Result Table: Condition, Action, Score]                 |
+-----------------------------------------------------------+
```

Component	Function	Data/API Call
PersonaSelector	Choose persona for simulation	`GET /v1/personas`
ContextDropdown	Predefined context scenarios	Local list
SimulateButton	Run decision engine with selected context	`POST /v1/simulate`
ResetButton	Clear results	N/A
SimulationResultTable	Show rule outcomes and scores	Response from `/v1/simulate`

4.4 Connectors Dashboard



Component	Function	Data/API Call
ConnectorListTable	Lists configured connectors with status and actions	`GET /v1/connectors`
AddConnectorButton	Opens modal to select connector type (Browser/REST/WS)	N/A
ImportDropdown	Bulk import connector definitions	`POST /v1/connectors/import`
RefreshButton	Reload connectors list	`GET /v1/connectors`
WorkspacePane	Shows live data preview and statistics for selected connector	`GET /v1/connectors/{id}/workspace/preview`
SettingsPane	Configure scheduling, scope, and field mappings	`GET/PUT /v1/connectors/{id}/settings`
CredentialLinkButton	Opens Vault UI for stored credentials	`GET /v1/connectors/{id}/credentials`
TestConnectorButton	Run one-off connector fetch and display results	`POST /v1/connectors/{id}/test`

5. API Endpoints

Method	Path	Description	Auth Scope
GET	/v1/connectors	List all connectors	`mimic.connector.read`
POST	/v1/connectors	Create a new connector configuration	`mimic.connector.write`
GET	/v1/connectors/{id}	Retrieve connector details	`mimic.connector.read`
PUT	/v1/connectors/{id}	Update connector settings	`mimic.connector.write`
DELETE	/v1/connectors/{id}	Delete a connector	`mimic.connector.write`
POST	/v1/connectors/import	Bulk import connectors	`mimic.connector.write`
GET	/v1/connectors/export	Export all connectors	`mimic.connector.read`
POST	/v1/connectors/{id}/test	Trigger a test run of connector	`mimic.connector.execute`
GET	/v1/connectors/{id}/workspace/preview	Fetch preview data for connector workspace	`mimic.connector.read`

6. Implementation

1. Extend Persona API service to support connector CRUD.
2. Develop Browser Connector microservice using Puppeteer headless Chrome for web scraping.
3. Implement generic Connector Framework: load connector plugins via dynamic import and mapping schemas.
4. Build Connectors Dashboard UI: React components for list, workspace preview, settings.
5. Integrate Vault credential retrieval: fetch and inject secrets at runtime.
6. Schedule data harvesting: use Quartz/cron in backend for periodic runs.
7. Stream fetched data into Persona Store and Analysis pipeline via Synapse.
8. Add unit/integration tests for connectors, including test endpoint.

7. Milestones

Milestone	Timeline	Owner
Connector CRUD API & DB Schema	Week 1	Backend Team
Browser Connector MVP	Week 2	Backend/ML Team
Connector Framework & Plugins	Week 3	Integration Team
Connectors Dashboard UI	Week 4	Frontend Team
Vault Credential Integration	Week 5	Security Team

Milestone	Timeline	Owner
Scheduling & Workspace Preview	Week 6	Backend Team
E2E Connector Test & Validation	Week 7	QA Team

8. Gathering Results

- Connector creation and updates ≥99.9% success rate
- Browser Connector test run latency <2 s for Discord fetch use-case
- Scheduled harvests complete within configured windows
- Workspace preview accuracy meets 95% correctness on data mapping

9. References & Dependencies

- **Vault Module** (SPEC-02) for credential storage and audit
- **Synapse Module** (SPEC-05) for event routing of harvested data
- **Alice Module** (SPEC-04) for CBP-driven triggers

10. Additional Enhancement Suggestions

To empower Mimic's advanced research and specialist-agent capabilities—especially for tasks like web data ingestion and trend analysis—consider these enhancements:

... [existing suggestions] ...

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :). API Endpoints

Method	Path	Description	Auth Scope
GET	/v1/personas	List all personas	`mimic.persona.read`
POST	/v1/personas	Create new persona	`mimic.persona.write`
GET	/v1/personas/{id}	Get persona details	`mimic.persona.read`
PUT	/v1/personas/{id}	Update persona traits	`mimic.persona.write`
DELETE	/v1/personas/{id}	Delete persona	`mimic.persona.write`
GET	/v1/personas/{id}/history	Fetch trait weight change history	`mimic.persona.read`
POST	/v1/personas/import	Bulk import personas	`mimic.persona.write`
GET	/v1/personas/export	Export personas	`mimic.persona.read`

Method	Path	Description	Auth Scope
POST	/v1/simulate	Simulate decision engine	`mimic.decision.execute`

6. Implementation

1. **Persona API Service**
2. Implement CRUD endpoints with JSON Schema validation
3. Persist in PostgreSQL with audit triggers into Vault AuditLog
4. **Trait Editor UI**
5. Build React sliders and history dropdown
6. Debounce weight updates for efficiency
7. **Decision Engine**
8. Develop rule engine in Go/Python applying conditions and weights
9. Expose `/v1/simulate` for synchronous evaluation
10. **Integration with CBP**
11. Subscribe to CBP events via Synapse to adjust real-time weights
12. **Analytics Export**
13. Stream decision outcomes to Core Services metrics API
14. **Testing & Validation**
15. Unit tests for rules evaluation, schema validation
16. E2E simulation tests with varied persona/context

7. Milestones

Milestone	Timeline	Owner
Persona API & DB	Week 1	Backend Team
Trait Editor & History UI	Week 2	Frontend Team
Decision Engine MVP	Week 3	ML/Backend Team
CBP Integration & Analytics	Week 4	Integration Team
Simulation Panel & E2E Tests	Week 5	QA Team

8. Gathering Results

- Persona CRUD success rate ≥99.9% under load
- Decision simulation latency <100 ms per request
- CBP-driven weight adjustments reflect <10 ms from event ingestion
- Slider and UI interactions maintain >60 FPS responsiveness

10. Additional Enhancement Suggestions

To empower Mimic's advanced research and specialist-agent capabilities—especially for tasks like web data ingestion and trend analysis—consider these enhancements:

- **Browser Connector Toolkit**\ Provide a secure, headless-browser plugin or microservice that can authenticate to user sites (e.g., Discord web), traverse page structures, and extract structured conversation logs or metrics. Expose this via a `/v1/connectors/browser` API.

- **Custom Connector Framework**\ Define a pluggable connector interface allowing devs to write adapters for other web services (REST, GraphQL, WebSocket) with schema-driven mapping of external data into Mimic's context pipeline.

- **Scheduled Data Harvesting**\ Allow users to schedule recurring data pulls (e.g., daily Discord sync) with configuration in a **Connectors Dashboard**, managing frequency, scope, and transformation rules.

- **Data Transformation & Enrichment**\ Integrate lightweight ETL capabilities: field mapping, sentiment normalization, keyword tagging, and anomaly detection on imported datasets, all configurable via the Connector UI.

- **Secure Credential Vaulting**\ Store connector credentials and tokens securely in Vault, with scoped access via `mimic.connector.read/write` scopes and audit logging of each crawl/ingest operation.

- **Context-Aware Workspaces**\ Introduce per-connector workspaces in the UI where users can review incoming data feeds, apply filters, and launch context-driven simulations in the Decision Simulation Panel.

- **Result Export & Integration**\ Enable exporting analysis results (e.g., conversation trend charts) directly to external tools or into Alden-managed tasks, providing seamless workflow handoff.

- **AI-Assisted Connector Setup**\ Leverage Alice's CBP to suggest connector templates based on user preferences and past data sources (e.g., "I see you often analyze Discord chats—shall I set up a Discord connector?").

9. References & Dependencies

- **Vault Module** (SPEC-02) for audit logging and secret storage
- **Synapse Module** (SPEC-05) for event routing and CBP subscription
- **Alice Module** (SPEC-04) for CBP data source
- **Core Services** (SPEC-07) for metrics ingestion and auth

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)