SPEC-09: SuperClaude Integration into Hearthlink CORE

1. Background

```
@startuml background
title Hearthlink CORE & SuperClaude Integration Context
package "Hearthlink CORE" {
  node Alden
  node Mimic
  node Sentry
  node Alice
}
package "SuperClaude" {
  node Commands ["18 Commands"]
  node Personas ["9 Personas"]
  node MCPs ["Context7, Sequential, Magic, Puppeteer"]
}
Alden --> Commands
Mimic --> Personas
Sentry --> MCPs
Alice --> Commands
@enduml
```

Hearthlink CORE currently provides slash-command workflows, MCP routing, and persona orchestration via Alden, Mimic, Sentry, and Alice. SuperClaude adds a proven configuration framework with 18 structured commands, 9 specialized personas, and 4 MCPs. Integrating SuperClaude will accelerate development, standardize cross-team workflows, and empower Vibe coders by embedding best-practice prompts, flag suggestions, and optimization patterns directly into Mimic.

Objectives:

- Leverage SuperClaude's workflow definitions without rebuilding existing CLI/MCP infrastructure.
- Enrich Mimic with persona-driven routing, inline flag coaching, and auto-activation triggers.
- Ensure UI wireframes for all new command panels, suggestion prompts, and mobile views.

Source Repository:

- SuperClaude GitHub: https://github.com/NomenAK/SuperClaude

2. Requirements (MoSCoW)

```
@startuml requirements
title MoSCoW Prioritization
skinparam rectangle {
  BackgroundColor LightYellow
  BorderColor Black
}
rectangle "Must-have" as M {
  "Register 18 Commands"
  "Extend Mimic Personas"
  "Super Mode Toggle"
  "Integrate MCP Flags"
  "Preserve Existing Connectors"
}
rectangle "Should-have" as S #LightBlue {
  "Toast/Dropdown Suggestions"
  "YAML Workflow Loading"
  "--uc & --profile Flags"
  "Auto-activate Persona"
  "MCP TTL Caching"
}
rectangle "Could-have" as C #LightGreen {
  "Mobile UI Wireframes"
  "Separate Console App"
}
rectangle "Won't-have (v1)" as W #LightCoral {
  "SuperCloud Installer Logic"
}
M --> S --> C --> W
@enduml
```

3. Method

3.1 Component Diagram

```
@startuml componentDiagram
title SuperClaude Integration Components
package CORE {
  [Command Registry]
  [MCP Router]
}
package Mimic {
  [Persona Engine]
  [Suggestion Engine]
```

```
}
package Vault {
  [YAML Store]
}
[Command Registry] --> [Persona Engine]
[Persona Engine] --> [MCP Router]
[MCP Router] --> [YAML Store]
@enduml
```

3.2 Data Flow Sequence

```
@startuml dataFlow
title Data Flow for Slash-Command Execution
actor User
autonumber
User -> CommandRegistry: /command --flags
CommandRegistry -> MimicEngine: parse & annotate
MimicEngine -> SuggestionEngine: recommend flags
MimicEngine -> MCPRouter: dispatch with flags
MCPRouter -> Vault: fetch workflow YAML
Vault --> MCPRouter: return workflow
MCPRouter --> COREHandlers: invoke handlers
COREHandlers --> User: return response
@enduml
```

3.3 UI Wireframes

```
@startuml uiWireframes
title UI Wireframes Overview
skinparam handwritten true
actor User
rectangle "Command Console" as Console {
  [Super Mode Toggle]
  [Advanced Commands List]
  [Input Field] --> [Persona Badge]
}
rectangle "Flag Coach Popover" as Popover {
  "Consider --magic for UI comps"
}
rectangle "Mobile Panel" as Mobile {
  "Linear Workflow Stepper"
}
Console --> Popover
```

```
desnote over Console: Toggle reveals advanced UI
@enduml
```

3.4 Optimization Patterns

- **UltraCompressed Mode**: auto-switch to `--uc` above 75% context
- **MCP TTL Caching**: Context7 (1h), Sequential (session), Magic (2h), Puppeteer (action)
- **Keyword Triggers**: `.tsx` →frontend, `security` →security

4. Implementation

4.1 Sequence Diagram for Implementation

```
@startuml implementationSequence
title Implementation Workflow
participant Developer
participant CORE as C
participant Mimic
participant MCPRouter
database Vault
Developer -> C: register commands + flags
C -> Vault: load YAML workflows
Developer -> Mimic: extend persona registry
Mimic -> SuggestionEngine: load keyword rules
Developer -> MCPRouter: register new flags
MCPRouter -> Vault: configure TTL
Developer -> UI: implement toggle, suggestions
Developer -> Tests: write E2E & unit tests
@enduml
```

Implementation Steps

1. **Registry Extension**: create `superclaude-commands.ts`, load YAML from Vault.
2. **Mimic Enhancements**: add personas, auto-activation, suggestion engine.
3. **MCP Router**: register `--c7`, `--seq`, `--magic`, `--pup`; add TTL caching.
4. **UI Development**: high-fidelity React/Tailwind components; integrate into Storybook.
5. **Testing**: E2E via Puppeteer; unit tests for suggestion logic.

5. Milestones

```
@startuml milestones
title Project Timeline
skinparam rectangle {
  RoundCorner 15
  BackgroundColor #FFEEAA
```

```
}
rectangle Sprint1 as "Sprint 1: Cmd Registry & YAML (1wk)"
rectangle Sprint2 as "Sprint 2: Persona Mapping & Suggestions (1wk)"
rectangle Sprint3 as "Sprint 3: MCP Flags & Caching (1wk)"
rectangle Sprint4 as "Sprint 4: UI Wireframes & Impl (2wks)"
rectangle Sprint5 as "Sprint 5: Testing & Audit (1wk)"
Sprint1 --> Sprint2 --> Sprint3 --> Sprint4 --> Sprint5
@enduml
```

6. Post-integration Audit

```
@startuml auditFlow
title Post-Integration Audit Workflow
actor Claude
Claude -> CORE: /analyze --all-mcp --think-hard --uc
Claude -> CORE: /scan --security --owasp --seq
Claude -> CORE: /troubleshoot --investigate --seq
@enduml
```

At completion, instruct Claude using advanced SuperClaude commands to ensure robust integration and leverage full capabilities.