📄 FSD_07 – Agent Memory Construction & Retrieval

🧠 Purpose:

This module builds and manages **agent memory** — a structured representation of the SMB's knowledge, preferences, and operational context — for use during live interactions. It allows the agent to:

- **Recall relevant information** from onboarding and documents
- **Answer follow-up questions** with consistency
- **Maintain state** across interactions
- **Retrieve tagged business logic** on-demand

It is the core of Audira's long-term intelligence layer — acting like the agent's internal brain.

🧩 FSD_07 – Section Breakdown

Section	Description
1. Scope	What memory means in Audira, and what it's used for
2. Input Requirements	What data feeds the memory structure
3. Memory Construction Logic	How memory is initialized and organized
4. Memory Retrieval API	How memory is accessed in runtime
5. Memory Update & Expansion	How new data is added over time
6. Memory Conflict Resolution	How contradictory inputs are handled
7. Integration with LLMs	How memory is injected into prompts
8. Future Enhancements	Vector memory, retention strategies, access logs

---

🔷 Section 1: Scope

🎯 Purpose:

The **Agent Memory** is a persistent, queryable knowledge layer that allows an Audira agent to **remember and retrieve** verified information from the client's onboarding journey, uploaded files, qualification answers, and validated tag data.

Unlike the Prompt Chain (FSD_06), which is **assembled per session**, the Agent Memory acts as a **long-term storage and reasoning layer** that lives with the agent across its lifecycle.

✅ **Responsibilities:**

Function	Description
Memory Initialization	Constructs a structured memory object from validated inputs during onboarding
Tag-Linked Recall	Stores all confirmed tag values and associated source content
Segment Indexing	Maps important segments (from FSD_02) to retrievable memory blocks
Answer Injection	Preserves high-confidence onboarding and follow-up answers
Retrieval Interface	Provides structured access to memory by tag, intent, or query category
Session Recall	Maintains transient state (user identity, last question asked, agent goal)

❌ **Not in Scope:**

Excluded	Why
LLM inference or generation	That's handled in FSD_08
Scoring readiness for launch	Already validated in FSD_05
Prompt compilation or styling	Handled in FSD_06
Unstructured file parsing	Happens earlier in FSD_01 + FSD_02

📌 **Use Cases Enabled:**

- "What did the user say their pricing model was?"
- "Retrieve vendor payout policy segment."
- "Is KYC handled manually or automatically?"
- "Summarize known facts about their support process."

These memory queries enable consistent, auditable, and context-rich answers across all LLMs, plugins, and integrations.

📎 **References**:

- 📘 *AUDIRA PRODUCT BLUEPRINT* – defines agent memory as a persistent layer
- 📘 *AUDIRA FILE & DATA UPLOAD SCHEMA* – provides anchor IDs for segment recall
- 📘 *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW* – shows memory-to-prompt injection

- 📘 *AUDIRA PRE-LAUNCH VALIDATOR SPEC* – confirms readiness for memory construction

🔷 Section 2: Input Requirements

This section defines all **upstream data sources** needed to construct and maintain an accurate, queryable agent memory.

📄 A. Validated Discovery Tag Map

Source: FSD_03 → FSD_05
Every tag marked as `covered`, `clarified`, or `inferred` is included, along with confidence, source(s), and priority.

```
{
  "tag_id": "pricing_model",
  "value": "three-tier (Basic, Pro, Enterprise)",
  "confidence": 0.94,
  "source_type": "onboarding_answer",
  "linked_segments": ["seg_12", "seg_14"]
}
```

📄 B. Enriched Segment Anchors

Source: FSD_02
All document blocks used to support tag coverage or qualification are indexed and stored.

```
{
  "segment_id": "seg_12",
  "text": "Our pricing is structured in three tiers: Basic, Pro, and
Enterprise...",
  "tag_links": ["pricing_model"],
  "file_source": "company_profile.pdf",
  "date_uploaded": "2025-06-03"
}
```

📄 C. Onboarding Answer Matrix

Source: FSD_01
Confirmed answers from fixed onboarding questions — includes tag links, confidence scoring, and timestamps.

```
{
  "question_id": "q_4",
  "answer_text": "We serve SMEs across MENA and offer monthly billing.",
```

```
  "linked_tags": ["target_customer", "billing_frequency"],
  "timestamp": "2025-06-01T10:15:00Z"
}
```

📄 D. Dynamic Qualification Responses

Source: FSD_04
Only answers marked as `status: answered` and validated by the Readiness Validator (FSD_05) are injected.

```
{
  "fq_id": "fq_008",
  "tag_id": "support_hours",
  "answer_text": "We provide 24/7 live support via Zendesk.",
  "linked_segments": [],
  "source": "follow_up"
}
```

📄 E. Agent Intent Profile

Source: AGENT ONBOARDING
Used to determine which tags, blocks, and memory keys are considered high-priority for this specific agent.

```
{
  "agent_id": "AGENT_00112",
  "capabilities": ["Q&A", "Sales Enablement"],
  "memory_priority_tags": ["revenue_model", "customer_type",
"unique_selling_point"]
}
```

🧩 Summary Input Table

Input	Purpose
`tag_map`	Core memory key-value pairs
`segments`	Anchor content for reference and citation
`answers`	Natural language memory content
`follow_ups`	Clarification and precision improvement
`intent_profile`	Prioritization and layout logic

📎 **References**:

- 📘 *AUDIRA FILE & DATA UPLOAD SCHEMA* – segment metadata and timestamping
- 📘 *AUDIRA DISCOVERY TAGS DICTIONARY* – tag-to-category mapping

- 📘 *AUDIRA AGENT ONBOARDING FRAMEWORK* – fixed answer source and priority logic
- 📘 *AUDIRA PRE-LAUNCH VALIDATOR SPEC* – filters and confirms memory-worthy input

🔷 Section 3: Memory Construction Logic

This section defines **how memory is structured**, populated, and stored once onboarding is complete and validation has passed.

🧠 Core Architecture

Agent memory is organized as a structured key-value store, where each **key** is a `discovery_tag_id`, and each **value** is an object containing:

- Cleaned natural language value
- Source references (segments, onboarding answers)
- Confidence metadata
- Priority and usage context

📠 *Memory Block Structure*

```
{
  "tag_id": "pricing_model",
  "value_text": "Three-tier pricing: Basic, Pro, and Enterprise",
  "sources": {
    "segments": ["seg_12", "seg_14"],
    "answers": ["q_4"]
  },
  "confidence": 0.94,
  "priority": "core",
  "last_updated": "2025-06-10T18:22:00Z"
}
```

⚙️ Construction Steps

Step 1: Ingest Confirmed Tags

From the validator-approved `tag_map`, select all tags with:

- `status == covered` OR `status == clarified`
- `confidence ≥ 0.75`

Step 2: Assemble Source Context

For each tag:

- Pull text from linked `segments` and `answers`
- Select the most concise and accurate natural language phrasing
- Normalize and clean (remove redundancy, fix grammar)

Step 3: Build Memory Block

Each tag becomes a memory block with:

- `value_text` (summarized)
- `sources` (cross-linked)
- `priority` (from dictionary)
- `confidence` (weighted average if multiple inputs)

Step 4: Store in Indexed Object

Memory is saved as:

```
"agent_memory": {
  "pricing_model": {...},
  "target_customer": {...},
  "support_hours": {...}
}
```

Safety & Validation:

Rule	Enforcement
Conflicting values	Flagged (sent to FSD_07 Section 6)
Missing source	Not stored
Confidence < 0.75	Excluded by default
Duplicate tags	Merged if values match; else flagged for admin

---

📎 **References**:

- 📘 *AUDIRA DISCOVERY TAGS DICTIONARY* – source of tag priority and descriptions
- 📘 *AUDIRA FILE & DATA UPLOAD SCHEMA* – provides canonical segment text
- 📘 *AUDIRA AGENT ONBOARDING FRAMEWORK* – input answer structure
- 📘 *AUDIRA PRE-LAUNCH VALIDATOR SPEC* – enforces minimum inclusion thresholds

---

Section 4: Memory Retrieval API

This section defines the **interface and query structure** for accessing agent memory during live interactions. It is designed for both **LLM agent logic (FSD_08)** and **admin tools (FSD_09)**.

API Overview

The **Agent Memory Retrieval API** supports:

- **Direct Tag Queries**
- **Category Queries** (e.g. all pricing-related tags)
- **Similarity or Phrase-Based Lookups**
- **Segment Recall by Anchor ID**

A. Get Tag Memory

```
GET /agent_memory/{agent_id}/tag/{tag_id}
```

Returns:

```
{
  "tag_id": "pricing_model",
  "value_text": "Three-tier pricing: Basic, Pro, and Enterprise",
  "sources": {
    "segments": ["seg_12", "seg_14"],
    "answers": ["q_4"]
  },
  "confidence": 0.94,
  "priority": "core"
}
```

B. Get Category Memory

```
GET /agent_memory/{agent_id}/category/pricing
```

Returns all memory blocks whose tag falls under the `pricing` category from the Tag Dictionary.

C. Phrase-Based Semantic Lookup

```
POST /agent_memory/{agent_id}/search
{
  "query": "How do they handle vendor payouts?"
}
```

Returns memory blocks that semantically match the phrase using embedding comparison with `value_text`.

📄 *D. Segment Recall by Anchor ID*

```
GET /agent_memory/{agent_id}/segment/seg_12
```

Returns the original uploaded segment linked to the tag.

🛠️ Control Flags

Flag	Description
`confidence_min`	Filters out low-confidence blocks
`include_sources`	Whether to include segment/answer links
`as_prompt_block`	Formats memory as inject-ready prompt for LLM

🔐 Access Layers

Role	Access Type
LLM Agent (FSD_08)	Reads memory in real-time for prompt injection
Admin Dashboard (FSD_09)	Retrieves memory for editing, override, or simulation
Audit System	Logs memory usage for compliance and error traceability

---

📎 **References**:

- 📘 *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW* – defines how memory is injected into live prompts
- 📘 *AUDIRA AGENT ONBOARDING FRAMEWORK* – supports user role access levels
- 📘 *AUDIRA FILE & DATA UPLOAD SCHEMA* – source of segment anchor IDs
- 📘 *AUDIRA INTEGRATION SCAFFOLDS GUIDE* – maps memory API to runtime deployment targets

---

🔷 Section 5: Memory Update & Expansion

This section defines how agent memory evolves after launch — allowing new information from the user, documents, or system observations to be added, replaced, or expanded intelligently.

🔁 Update Triggers

Agent memory is updated when:

Trigger	Description
New document upload	Triggers segment parsing and tag re-evaluation (FSD_01 → FSD_03 → FSD_05)
Follow-up questions answered	Adds new value for previously uncovered tags
Admin override	Manually injects or corrects memory via dashboard
Live interaction feedback	(Planned) Based on flagged incorrect answers or clarifications from user
Scheduled retraining	Periodic memory refresh if data is stale (e.g., over 90 days old)

Update Modes

Mode	Description	Example
Replace	Tag already exists and new value is more recent/confident	New document shows updated support policy
Merge	Additional data extends an existing tag's value	Adds "Live chat now available" to support_methods
Ignore	Incoming value is duplicate or too weak	File says "monthly billing" but already confirmed in onboarding
Flag for Review	Conflict detected between new and existing value	One doc says "no returns", another says "30-day return policy"

Versioning + Timestamps

Each memory block includes:

```
{
  "version": 2,
  "last_updated": "2025-06-11T02:48:00Z",
  "source_type": "document_upload",
  "change_type": "merge"
}
```

Memory history is stored for audit and retraining purposes.

Update Safety Rules

Rule	Enforcement
Confidence below threshold	Ignored unless flagged
Segment lacks anchor ID	Rejected for audit reasons
Admin-edited blocks	Locked from overwrite by AI

Tags marked "fixed"	Not allowed to change after initial onboarding (e.g., `legal_structure`) unless override enabled

---

📎 **References**:

- 📘 *AUDIRA PRE-LAUNCH VALIDATOR SPEC* – enforces update rules and freeze conditions
- 📘 *AUDIRA FILE & DATA UPLOAD SCHEMA* – defines time and source of updates
- 📘 *AUDIRA AGENT ONBOARDING FRAMEWORK* – lists fixed and override-able tag types
- 📘 *AUDIRA AGENT BLUEPRINT TEMPLATE* – maps update permissions to agent role

---

🔷 Section 6: Memory Conflict Resolution

This section defines how the system detects, flags, and optionally resolves conflicting information within the agent memory — ensuring **accuracy**, **transparency**, and **LLM safety**.

⚠️ Conflict Types

Type	Description	Example
Direct Contradiction	Two sources give incompatible answers	One file says "manual payouts"; another says "automated"
Version Mismatch	New doc has updated but undated info	Segment from 2022 vs. newer onboarding answer
Segment vs. Answer	User declared one thing; document says another	User said "Stripe only", doc lists "Stripe + PayPal"
Low-confidence disagreement	Fuzzy segment mentions differ subtly	"Support hours: 9–5" vs. "around the clock support"

🧭 Conflict Detection Logic

Conflicts are flagged when:

- Same `tag_id` receives multiple values with `confidence ≥ 0.75`
- Values differ semantically (OpenAI embedding or cosine dissimilarity > 0.25)
- Time of input suggests newer info might be valid but unconfirmed

---

🛠️ Conflict Resolution Modes

Mode	Description	Triggered When
Prefer Newer	Accept newer value if it's higher confidence	On document upload
Manual Review Required	Flags block for admin inspection	On contradiction from two authoritative sources
Ask User for Clarification	Creates a follow-up question	If agent is still in configuration phase
Dual-Memory Marking	Stores both values with warning	For non-critical tags (e.g., design_theme, tone_preference)
Suppress Temporarily	Excludes from memory until conflict resolved	For compliance or legal tags

---

🧪 Conflict Block Structure

```
{
  "tag_id": "payout_policy",
  "conflict_detected": true,
  "conflicting_values": [
    {
      "value_text": "Automated via Stripe every 14 days",
      "source": "onboarding_answer",
      "timestamp": "2025-06-03"
    },
    {
      "value_text": "Manual batch payouts by finance team",
      "source": "vendor_terms.pdf",
      "timestamp": "2025-06-10"
    }
  ],
  "status": "pending_resolution",
  "escalation_target": "admin_review"
}
```

📤 Conflict Resolution Routing

Outcome	Routed To
Admin resolution available	✅ FSD_09 – Admin Review Dashboard
User clarification possible	✅ FSD_04 – Dynamic Qualification Generator
AI auto-resolution applied	Logged and versioned in audit trail

---

📎 **References**:

- 📘 *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW* – must avoid injecting unresolved conflicts
- 📘 *AUDIRA PRE-LAUNCH VALIDATOR SPEC* – lists critical tags requiring resolution

- 📘 *AUDIRA AGENT ONBOARDING FRAMEWORK* – contains override permissions for admins
- 📘 *AUDIRA FILE & DATA UPLOAD SCHEMA* – provides timestamp and source authority

---

🔷 Section 7: Integration with LLMs

This section defines how agent memory is integrated into **live language model prompts** at runtime (FSD_08), ensuring continuity, personalization, and factual accuracy.

🧠 Memory-to-Prompt Injection Flow

1. **Query Received**
   → From user or system
2. **Intent Recognized**
   → Maps to tag(s) or category
3. **Memory Retrieved**
   → via Memory API (see Section 4)
4. **Prompt Chain Adapted**
   → Injects memory into proper block (FSD_06)
5. **Prompt Sent to LLM**
   → Live inference begins

🧩 Injection Techniques

Technique	Description	Use Case
Direct Recall Injection	Inserts memory value into prompt directly	"What is their pricing model?"
Contextual Framing	Memory is transformed into prompt background context	“Act as a rep for a company with…”
Guardrail Filtering	Memory enforces constraints (e.g., “Only use this info”)	Legal, policy, compliance modules
Fallback Handling	If no memory found, insert soft fallback prompt	“This business has not confirmed a vendor policy.”

📜 Injection Template Example

```
Use the following verified information about the client:
- Pricing Model: Three-tier (Basic, Pro, Enterprise)
- Payment Method: Stripe, monthly recurring
```

```
- Support: 24/7 via Zendesk

Based on this, answer the following user query:
"What happens if a vendor wants a refund?"
```

LLM Runtime Controls

Guardrail	Applied From
`max_tokens_per_memory_block`	Memory API → prevents prompt overload
`reject_low_confidence_blocks`	From Readiness Validator / Memory Filter
`model_routing_hint`	Agent config suggests model variant (e.g., GPT-4 for compliance, Mixtral for summaries)

Multi-turn Memory Usage

For multi-turn chat agents:

- Memory is persisted across turns using a session identifier
- Only **delta blocks** are re-injected after first prompt
- Temporary memory (e.g., "User selected 2023 data") is stored in `session_memory`, not `agent_memory`

References:

- *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW* – runtime agent architecture and memory-to-prompt path
- *AUDIRA AGENT BLUEPRINT TEMPLATE* – defines agent roles and injection preferences
- *AUDIRA INTEGRATION SCAFFOLDS GUIDE* – maps agent types to LLM backends and injection strategies

---

Section 8: Future Enhancements

This section outlines planned upgrades and strategic innovations that will enhance memory flexibility, depth, safety, and responsiveness as Audira scales across thousands of agents and business types.

Upcoming Features

Feature	Description	Value

Vectorized Memory Embeddings	Stores each memory block as a vector (OpenCLIP or BGE), enabling semantic lookup across fuzzy phrasing	Enables smarter answers, even with user variation
Temporal Memory Layers	Supports "as of date" memory slicing (e.g., 2022 vs. 2025 statements)	Improves version tracking and audit history
Memory Snapshot Replay	View, compare, or regenerate agent behavior from historical memory states	Powers audit, version control, and testing
Multi-profile Memory	Supports different views of memory depending on end user (e.g., internal team vs. investor)	Enables fine-grained access + response control
Auto-Clarification Triggers	If memory contains low-confidence or contradictory blocks, trigger dynamic follow-ups	Keeps memory high-quality over time
Agent Memory Sync API	Allow external tools to update or inject into memory securely	Enables CRMs or ERPs to write directly to Audira
Explainable Memory Responses	Let users ask "why did the agent say that?" and return memory trace explanation	Boosts trust and traceability
Memory Freshness Score	Tracks staleness of each block (based on timestamp, source type, and segment age)	Enables alerting or retraining triggers
Privacy-Aware Memory Modes	Tag blocks with sensitivity (e.g., financial, HR) and restrict access or injection scope	Enables GDPR / SOC2 alignment for audits

---

🧰 Compatible OSS Tools

Tool	Usage
LlamaIndex / Haystack	Memory indexing + semantic lookup
LangGraph / LangChain	Multi-path memory routing based on use case
TruLens / DeepEval	Memory-based output evaluation and correction loops
Weaviate / Chroma	Optional vector memory backends for hybrid semantic + key-value recall
Unstructured.io	Used for advanced memory-block construction from multi-format inputs

---

📎 **Linked Modules**:

- 📘 *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW* – memory-prompt handshake and LLM injection architecture
- 📘 *AUDIRA INTEGRATION SCAFFOLDS GUIDE* – memory-to-agent sync endpoints
- 📘 *AUDIRA AGENT SIMULATION TEST KIT* – used to benchmark behavior from different memory states

- 📘 *AUDIRA PRODUCT BLUEPRINT* – defines long-term memory as a platform differentiator

---