Chitta Developer Onboarding Guide

Welcome to Chitta

This document will give you everything you need to understand how Chitta works, from high-level philosophy to low-level implementation details. By the end, you should be able to draw the complete architecture diagram and start contributing to the codebase.

Table of Contents

---

1. Vision & Mission

What is Chitta?

Chitta is an AI-powered child developmental understanding platform. It helps parents and clinicians see children clearly - not through checklists and assessments, but through natural conversation.

The name comes from Sanskrit - *Chitta* (चित्त) means consciousness, awareness. The witnessing presence itself.

The Core Insight

Traditional developmental tools ask: *"Does your child do X? Yes/No. Score: 7/10."*

Chitta asks: *"Tell me about your child. What happened yesterday at the park?"*

Stories are gold. A single story can reveal multiple developmental signals that a checklist would miss entirely.

What Chitta Is NOT

- ❌ A diagnostic tool (can't diagnose)
- ❌ A replacement for professionals (points, doesn't replace)
- ❌ A rigid assessment system (no completeness scores)
- ❌ A chatbot (it's an observing intelligence)

What Chitta IS

- ✅ A developmental companion that holds the complete picture
- ✅ An expert observer that notices patterns across domains
- ✅ A bridge connecting all professionals around a child
- ✅ A system driven by curiosity, not checklists

---

2. Core Philosophy

The Darshan Metaphor

The observing intelligence at the heart of Chitta is called **Darshan** (Sanskrit: दर्शन), meaning "mutual seeing" - to see and be seen.

Darshan does three things: 1. **HOLDS** - Child and Session are its memory 2. **NOTICES** - Extracts observations, stories, and evidence 3. **ACTS** - Responds with guidance and updates curiosities

Four Types of Curiosity

Understanding emerges through **curiosity**, not checklists:

Type	Purpose	Example
discovery	Open receiving	"Who is this child?"
question	Following a thread	"What triggers meltdowns?"
hypothesis	Testing a theory	"Music helps him regulate"
pattern	Connecting dots	"Sensory input affects everything"

Critical: Type and certainty are INDEPENDENT. - Type = what kind of exploration - Certainty = how confident within that type

You can have a weak hypothesis (certainty=0.3) or a strong discovery (certainty=0.8).

Key Principles

1. **Show, Don't Conclude** - Describe, don't diagnose
2. **Follow the Current** - Context-driven, not state-machine
3. **Be Invisible** - Proactive ≠ Pushy
4. **Minimum NECESSARY Complexity** - Not minimum possible

---

3. The 10 Golden Rules

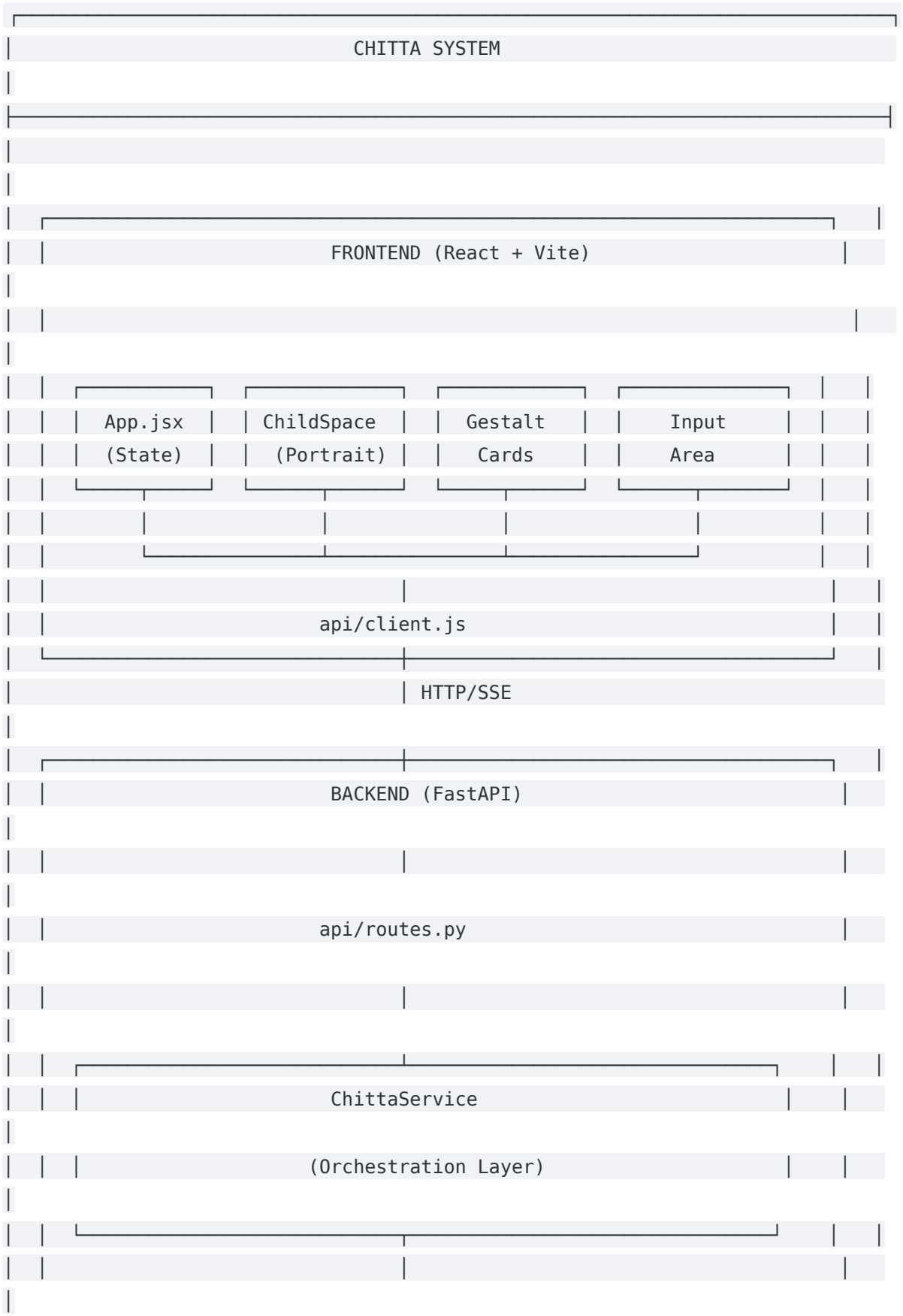
These are inviolable principles when writing code for Chitta:

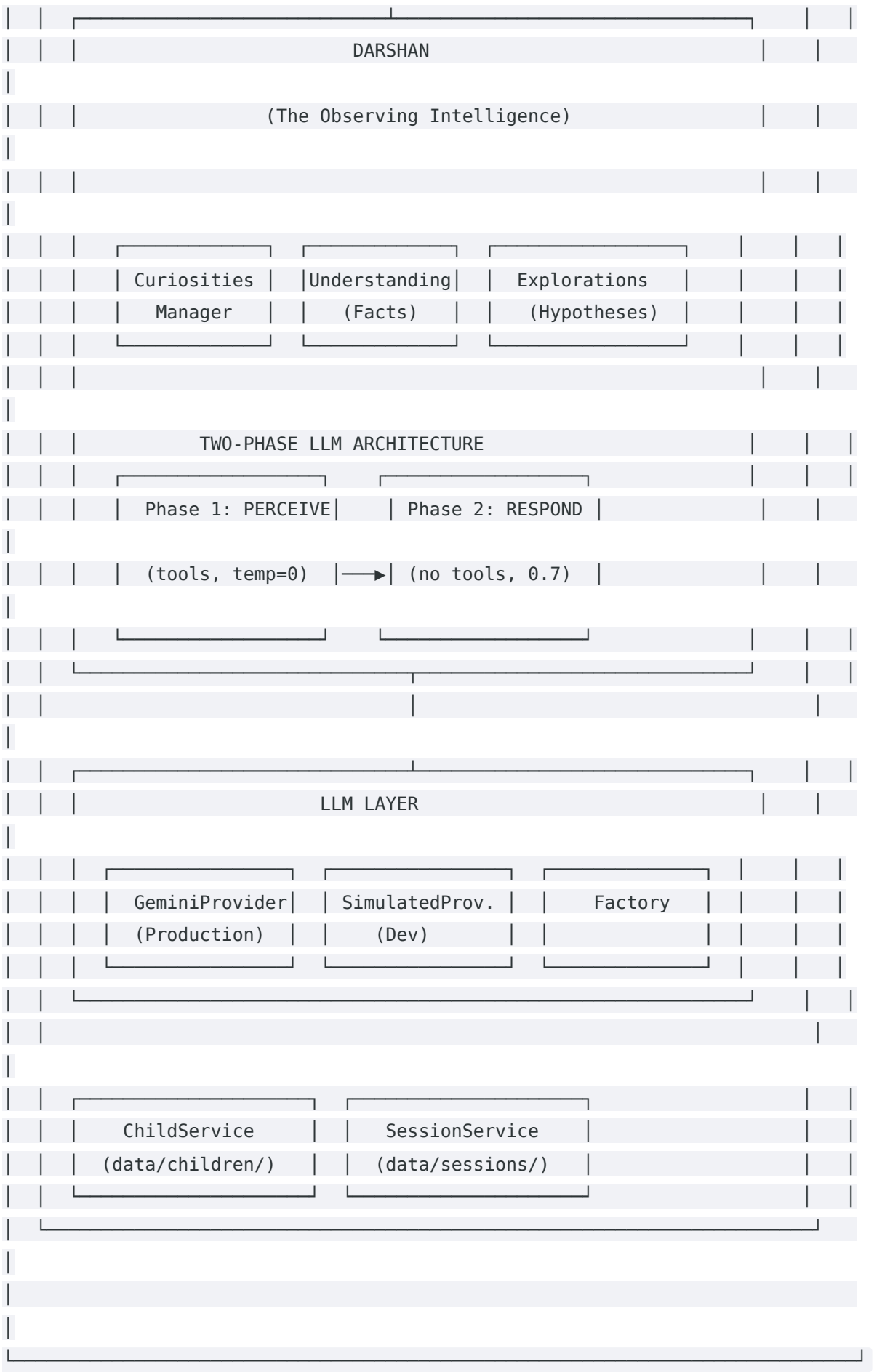
#	Rule	Description
1	Follow the Current	No rigid state machines (`if step == 3`). Use context-driven logic.
2	Start Fresh	Functions accept `context` object for "fresh look" at data.
3	Adapt Like Water	Single adaptive functions, not separate paths for Parent/Clinician.
4	Show, Don't Conclude	`observation: "looked at face"` not `diagnosis: "delayed"`
5	Be Invisible	Only trigger logic when `value > interruption_cost` .
6	Simplicity (פשוט)	Minimum NECESSARY complexity.
7	Honest Uncertainty	Return structured ambiguity, never hallucinate certainty.
8	Emergence	Small composable functions over monolithic managers.
9	Naming Convention	Use `notice` , `wonder` , `capture_story` . Avoid `diagnose` , `assess` .
10	Natural Language	Warm Hebrew ("שמתי לב") not robotic ("המערכת זיהתה").

---

4. High-Level Architecture

System Overview




DARSHAN
(The Observing Intelligence)
Curiosities Manager
Understanding (Facts)
Explorations (Hypotheses)
TWO-PHASE LLM ARCHITECTURE
Phase 1: PERCEIVE
(tools, temp=0)
Phase 2: RESPOND
(no tools, 0.7)
LLM LAYER
GeminiProvider (Production)
SimulatedProv. (Dev)
Factory
ChildService (data/children/)
SessionService (data/sessions/)

Technology Stack

Layer	Technology
Frontend	React 18 + Vite + TailwindCSS
Backend	Python 3.11 + FastAPI
LLM	Google Gemini (2.5-flash for chat, 2.5-pro for synthesis)
Data	JSON files (children, sessions)
Real-time	Server-Sent Events (SSE)

---

5. Backend Deep Dive

Directory Structure

```
backend/
├── app/
│   ├── main.py                    # FastAPI entry point
│   ├── api/
│   │   ├── routes.py              # Main API endpoints
│   │   └── dev_routes.py          # Dev/test endpoints
│   ├── chitta/                    # Core Darshan module
│   │   ├── gestalt.py             # Darshan class (THE BRAIN)
│   │   ├── service.py             # ChittaService (orchestration)
│   │   ├── curiosity.py           # Curiosity model & manager
│   │   ├── models.py              # All data models
│   │   ├── tools.py               # LLM tool definitions
│   │   ├── formatting.py          # Prompt formatting
│   │   ├── synthesis.py           # Portrait/Crystal generation
│   │   ├── portrait_schema.py     # Pydantic schemas
│   │   └── clinical_gaps.py       # Gap detection
│   ├── services/
│   │   ├── llm/                   # LLM abstraction
│   │   │   ├── base.py            # BaseLLMProvider interface
│   │   │   ├── factory.py         # Provider factory
│   │   │   └── gemini_provider.py # Gemini implementation
│   │   ├── child_service.py       # Child persistence
│   │   └── session_service.py     # Session management
│   └── models/
│       └── child.py               # Child entity model
├── data/
│   ├── children/                  # {family_id}.json files
│   ├── sessions/                  # Session files
```

```
│   └── videos/                    # Uploaded videos
└── tests/
```

The Darshan Class (gestalt.py)

This is the heart of Chitta. Key methods:

```python
class Darshan:
    """The observing intelligence."""

    # === THREE PUBLIC METHODS ===

    async def process_message(self, message: str) -> Response:
        """
        Process parent message with TWO-PHASE architecture.
        1. Phase 1: Perceive with tools (temp=0)
        2. Apply learnings from tool calls
        3. Phase 2: Respond without tools (temp=0.7)
        """

    def get_active_curiosities(self) -> List[Curiosity]:
        """Get current curiosities sorted by pull."""

    def synthesize(self) -> Optional[SynthesisReport]:
        """Create synthesis using strongest model."""

    # === INTERNAL METHODS ===

    async def _phase1_perceive(self, turn_context) -> PerceptionResult:
        """LLM call WITH tools, temp=0. Returns tool calls only."""

    async def _phase2_respond(self, turn_context, perception) -> str:
        """LLM call WITHOUT tools, temp=0.7. Returns text only."""

    def _apply_learnings(self, tool_calls: List[ToolCall]):
        """Execute tool calls and update state."""
```

ChittaService (service.py)

Thin orchestration layer:

```python
class ChittaService:
    """
    Orchestrates conversation through Darshan.
```

```
    Responsibilities:
    - Get/create Darshan for family
    - Detect session transitions (>4 hour gap)
    - Trigger memory distillation on transition
    - Persist state after each message
    - Derive action cards
    """

    SESSION_GAP_HOURS = 4  # Hours that define session transition

    async def process_message(self, family_id: str, message: str) -> dict:
        """
        Returns:
        {
            "response": "natural Hebrew text",
            "curiosity_state": {
                "active_curiosities": [...],
                "open_questions": [...]
            },
            "cards": [...]
        }
        """
```

LLM Tools (tools.py)

Six tools for perception:

Tool	Purpose
`notice`	Record observation about child
`wonder`	Spawn new curiosity (4 types)
`capture_story`	Capture meaningful story (GOLD!)
`add_evidence`	Add evidence to exploration
`spawn_exploration`	Start focused exploration
`record_milestone`	Record developmental milestone

Curiosity Manager (curiosity.py)

```
class Curiosities:
    """Manages Darshan's curiosities."""

    # 8 perpetual curiosities (always present)
    PERPETUAL_TEMPLATES = [
        {"focus": "מי הילד הזה?", "type": "discovery", ...},
```

```
        {"focus": "מה הוא אוהב?", "type": "discovery", ...},
        # ... 6 more
    ]

    # Pull calculation factors
    DECAY_RATE_PER_DAY = 0.02      # Time decay
    GAP_BOOST_PER_ITEM = 0.1       # Gap boost
    HIGH_CERTAINTY_DAMPEN = 0.2   # Satisfaction dampening
```

---

6. Frontend Deep Dive

Directory Structure

```
src/
├── App.jsx                          # Main component (state orchestrator)
├── main.jsx                         # React entry point
├── api/
│   └── client.js                    # API client class
├── components/
│   ├── ChildSpace.jsx               # The Living Portrait (4 tabs)
│   ├── ChildSpaceHeader.jsx         # Header with badges
│   ├── GestaltCards.jsx             # Action/context cards
│   ├── ConversationTranscript.jsx # Chat display
│   ├── InputArea.jsx                # Message input
│   ├── CuriosityPanel.jsx           # Curiosity display
│   ├── ProfessionalSummary.jsx     # Summary generation
│   └── deepviews/                   # Modal views
│       ├── VideoUploadView.jsx
│       ├── FilmingInstructionView.jsx
│       ├── ShareView.jsx
│       └── ...
└── services/
```

Main App State (App.jsx)

```
// Core state
const [familyId, setFamilyId] = useState(INITIAL_FAMILY_ID);
const [messages, setMessages] = useState([]);
const [cards, setCards] = useState([]);
const [curiosityState, setCuriosityState] = useState({
  active_curiosities: [],
  open_questions: []
});
```

```
const [childSpace, setChildSpace] = useState(null);
const [isChildSpaceOpen, setIsChildSpaceOpen] = useState(false);

// Message sending
const handleSend = async (message) => {
  const response = await api.sendMessage(familyId, message);
  setMessages(prev => [...prev,
    { role: 'user', content: message },
    { role: 'assistant', content: response.response }
  ]);
  setCuriosityState(response.curiosity_state);
  setCards(response.cards);
};
```

ChildSpace Tabs

The Living Portrait has 4 tabs:

Tab	ID	Icon	Hebrew	Content
Essence	`essence`	Heart	הדיוקן	Who this child IS (portrait sections)
Discoveries	`discoveries`	Lightbulb	המסע	Exploration journey (curiosities, explorations)
Observations	`observations`	Video	מה ראינו	Stories, videos, observations
Share	`share`	Share2	שיתוף	Generate professional summaries

Card Types

Cards appear below the chat to prompt user actions:

Card Type	Purpose	Actions
`video_suggestion`	Hypothesis formed, video would help	Accept / Decline
`baseline_video_suggestion`	Early discovery, want to see child	Accept / Later
`video_guidelines_generating`	Creating filming instructions	(loading)
`video_guidelines_ready`	Instructions ready	View / Upload
`video_uploaded`	Video ready for analysis	Analyze
`video_analyzing`	Analysis in progress	(loading)
`video_analyzed`	Analysis complete	Dismiss

Card Type	Purpose	Actions
`video_validation_failed`	Video doesn't match request	View guidelines

API Client (api/client.js)

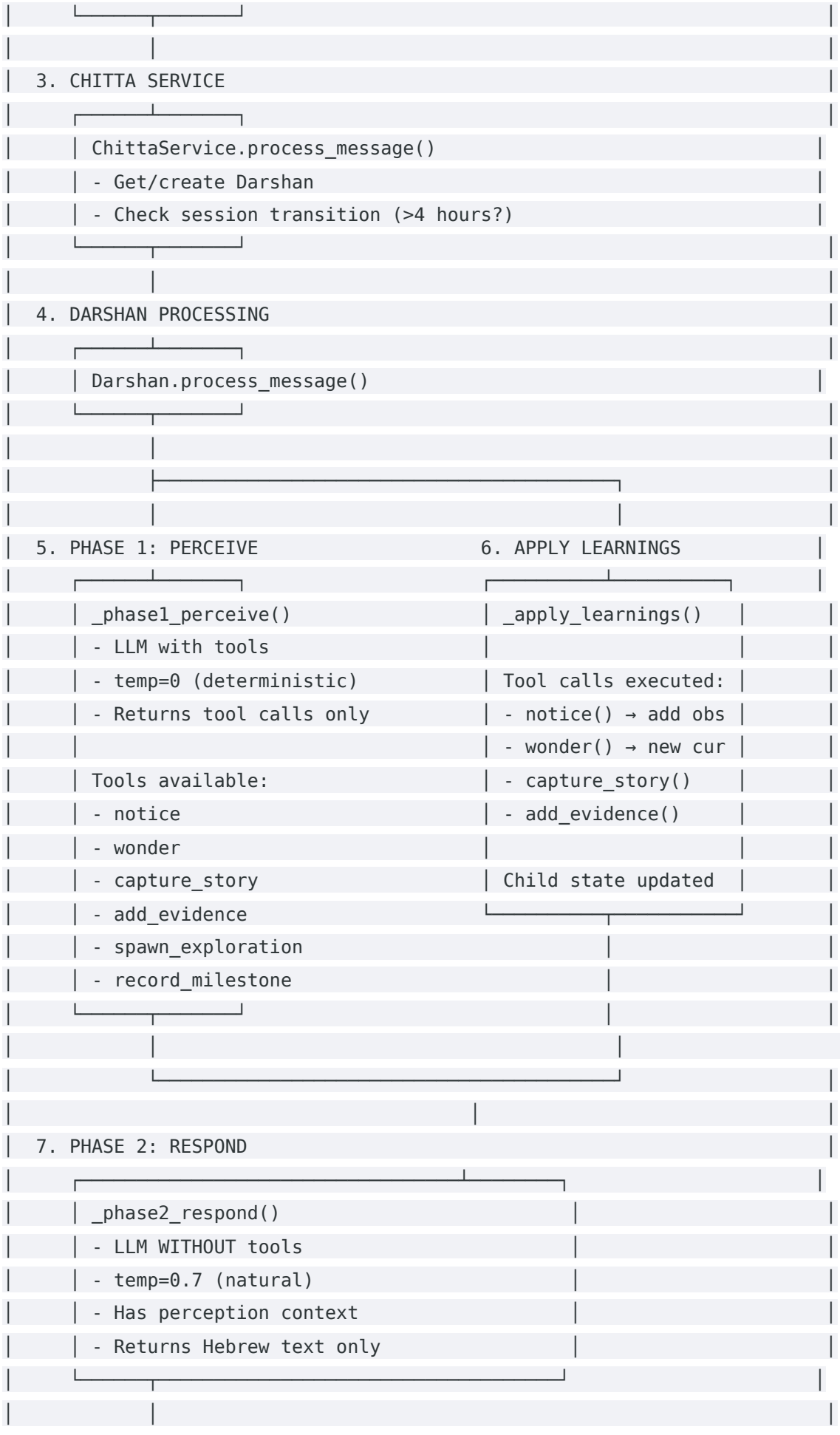
```
class ChittaAPIClient {
  constructor(baseUrl) {
    this.baseUrl = baseUrl || '/api';
  }

  // Core methods
  async sendMessage(familyId, message, parentName)
  async getState(familyId)
  async getChildSpace(familyId)

  // Video methods
  async uploadVideo(familyId, videoId, scenario, duration, file, onProgress)
  async analyzeVideos(familyId)
  async acceptVideoSuggestion(familyId, cycleId)
  async declineVideoSuggestion(familyId, cycleId)

  // Summary methods
  async generateProfessionalSummary(familyId, recipientType, recipientInfo)
}
```

---

7. Data Flow & Message Processing

Complete Message Flow

```
┌─────────────────────────────────────────────────────────────────────────┐
│                        MESSAGE PROCESSING FLOW                          │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  1. USER INPUT                                                          │
│     ┌──────────────┐                                                    │
│     │  InputArea   │ User types: "הבן שלי מתקשה במעברים"                │
│     └──────┬───────┘                                                    │
│            │                                                            │
│  2. API CALL                                                            │
│     ┌──────┴───────┐                                                    │
│     │ api.sendMessage(familyId, message)                                │
│     │ POST /api/chat/v2/send                                            │
```


3. CHITTA SERVICE
ChittaService.process_message()
- Get/create Darshan
- Check session transition (>4 hours?)
4. DARSHAN PROCESSING
Darshan.process_message()
5. PHASE 1: PERCEIVE
_phase1_perceive()
- LLM with tools
- temp=0 (deterministic)
- Returns tool calls only
Tools available:
- notice
- wonder
- capture_story
- add_evidence
- spawn_exploration
- record_milestone
6. APPLY LEARNINGS
_apply_learnings()
Tool calls executed:
- notice() → add obs
- wonder() → new cur
- capture_story()
- add_evidence()
Child state updated
7. PHASE 2: RESPOND
_phase2_respond()
- LLM WITHOUT tools
- temp=0.7 (natural)
- Has perception context
- Returns Hebrew text only

```
│  8. PERSIST & RETURN                                                    │
│     ┌──────┴──────┐                                                     │
│     │ - Save to data/children/{family_id}.json                          │
│     │ - Derive cards                                                    │
│     │ - Return response + curiosity_state + cards                       │
│     └──────┬──────┘                                                     │
│            │                                                            │
│  9. FRONTEND UPDATE                                                     │
│     ┌──────┴──────┐                                                     │
│     │ - Add messages to transcript                                      │
│     │ - Update curiosity panel                                          │
│     │ - Show/update cards                                               │
│     └─────────────┘                                                     │
│                                                                         │
└─────────────────────────────────────────────────────────────────────────┘
```

Two-Phase Architecture (CRITICAL)

Why two phases? Tool calls and text responses CANNOT be reliably combined in a single LLM call.

```
# Phase 1: Perception
config = {
    "temperature": 0.0,              # Deterministic
    "tool_config": FunctionCallingMode.ANY,  # Force tool calls
    "automatic_function_calling": DISABLED,
    "max_output_tokens": 4000
}
# Returns: [ToolCall(name="notice", args={...}), ToolCall(name="wonder", args=
  {...})]

# Phase 2: Response
config = {
    "temperature": 0.7,              # Natural variation
    "functions": None,               # NO TOOLS
    "max_output_tokens": 4000        # Prevent Hebrew truncation
}
# Returns: "שמתי לב שמעברים מרגישים גדולים עבורו..."
```

---

8. Data Models

Core Models (models.py)

```python
@dataclass
class Understanding:
    """Everything known about a child."""
    observations: List[TemporalFact]    # Individual observations
    milestones: List[DevelopmentalMilestone]
    stories: List[Story]                # Captured stories (GOLD)
    patterns: List[Pattern]             # Cross-domain patterns
    essence: Optional[str]              # Who this child IS
    strengths: List[str]

@dataclass
class TemporalFact:
    """Observation with temporal information."""
    content: str
    domain: str                         # motor, social, emotional, etc.
    confidence: float
    when_type: Optional[str]            # now, days_ago, age_months, etc.
    when_value: Optional[int]
    created_at: datetime

@dataclass
class Curiosity:
    """Something Darshan wants to understand."""
    focus: str                          # What we're curious about
    type: str                           # discovery|question|hypothesis|pattern
    pull: float                         # How strongly it draws attention (0-1)
    certainty: float                    # Confidence within type (0-1)
    theory: Optional[str]               # For hypothesis
    video_appropriate: bool             # Can video test this?
    domain: Optional[str]

@dataclass
class Exploration:
    """Focused investigation following curiosity."""
    id: str
    focus: str
    exploration_type: str               # discovery|question|hypothesis|pattern
    theory: Optional[str]
```

```
    confidence: float
    evidence: List[Evidence]
    video_scenarios: List[VideoScenario]
    status: str                          # active|complete|abandoned

@dataclass
class Story:
    """Captured story with developmental signals."""
    id: str
    summary: str
    reveals: List[str]                   # What the story reveals
    domains: List[str]
    significance: float
    captured_at: datetime

@dataclass
class Crystal:
    """Cached synthesis - the crystallized understanding."""
    essence_narrative: str
    temperament: List[str]
    core_qualities: List[str]
    patterns: List[Pattern]
    intervention_pathways: List[InterventionPathway]
    portrait_sections: List[PortraitSection]
    expert_recommendations: List[ExpertRecommendation]
    open_questions: List[str]
    created_at: datetime
    last_updated: datetime
```

Data Persistence

```
data/
├── children/
│   └── {family_id}.json          # Full child state
│       {
│         "child_id": "family_abc123",
│         "child_name": "יואב",
│         "birth_date": "2021-03-15",
│         "understanding": {...},
│         "explorations": [...],
│         "stories": [...],
│         "journal": [...],
│         "curiosities": {...},
```

```
│         "crystal": {...}
│       }
├── sessions/
│   └── {family_id}_{session_id}.json
└── videos/
    └── {video_id}.mp4
```

---

9. API Reference

Main Endpoints

Method	Endpoint	Purpose
POST	`/api/chat/v2/send`	Send message (main)
GET	`/api/state/{family_id}`	Get family state
GET	`/api/family/{family_id}/space`	Get child space data

Message Endpoint

```
POST /api/chat/v2/send
```

```
Request:
{
  "family_id": "family_abc123",
  "message": "הבן שלי אוהב דינוזאורים",
  "parent_name": "שרה"
}

Response:
{
  "response": "איזה כיף! ספרי לי עוד על האהבה הזו לדינוזאורים...",
  "curiosity_state": {
    "active_curiosities": [
      {
        "focus": "מי הילד הזה?",
        "type": "discovery",
        "pull": 0.8,
        "certainty": 0.3,
        "domain": "essence"
      }
    ],
    "open_questions": ["מה מביא אותם לכאן?"]
  },
```

```
  "cards": [
    {
      "type": "baseline_video_suggestion",
      "title": "אשמח לראות את הילד/ה",
      "description": "סרטון קצר יעזור לי להכיר...",
      "actions": [...]
    }
  ]
}
```

Video Endpoints

Method	Endpoint	Purpose
POST	`/api/video/upload`	Upload video file
POST	`/api/video/analyze`	Analyze uploaded videos
POST	`/api/exploration/{cycle_id}/accept-video`	Accept video suggestion
POST	`/api/exploration/{cycle_id}/decline-video`	Decline video suggestion
GET	`/api/exploration/{cycle_id}/scenarios`	Get video scenarios

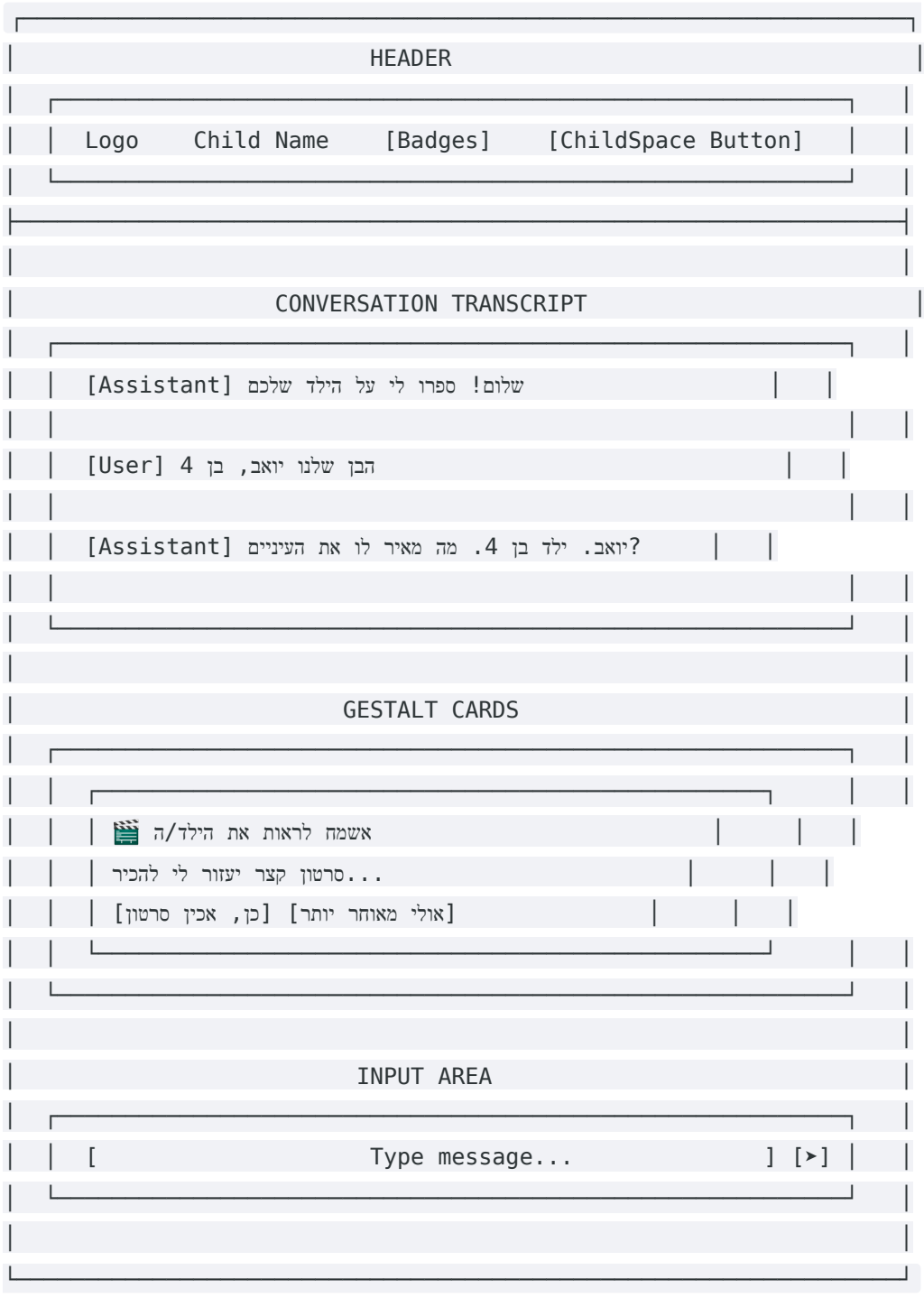
Child Space Endpoint

```
GET /api/family/{family_id}/space

Response:
{
  "crystal": {
    "essence_narrative": "יואב הוא ילד סקרן...",
    "portrait_sections": [...],
    "patterns": [...],
    "intervention_pathways": [...],
    "open_questions": [...]
  },
  "explorations": [...],
  "stories": [...],
  "observations": [...],
  "badges": {
    "stories_count": 5,
    "videos_count": 2,
    "patterns_count": 3,
    "exploration_count": 4
  }
}
```

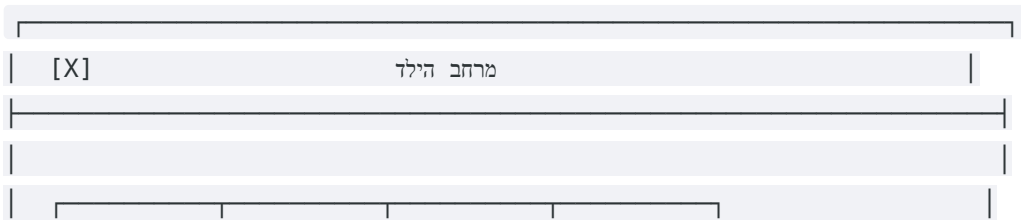
---

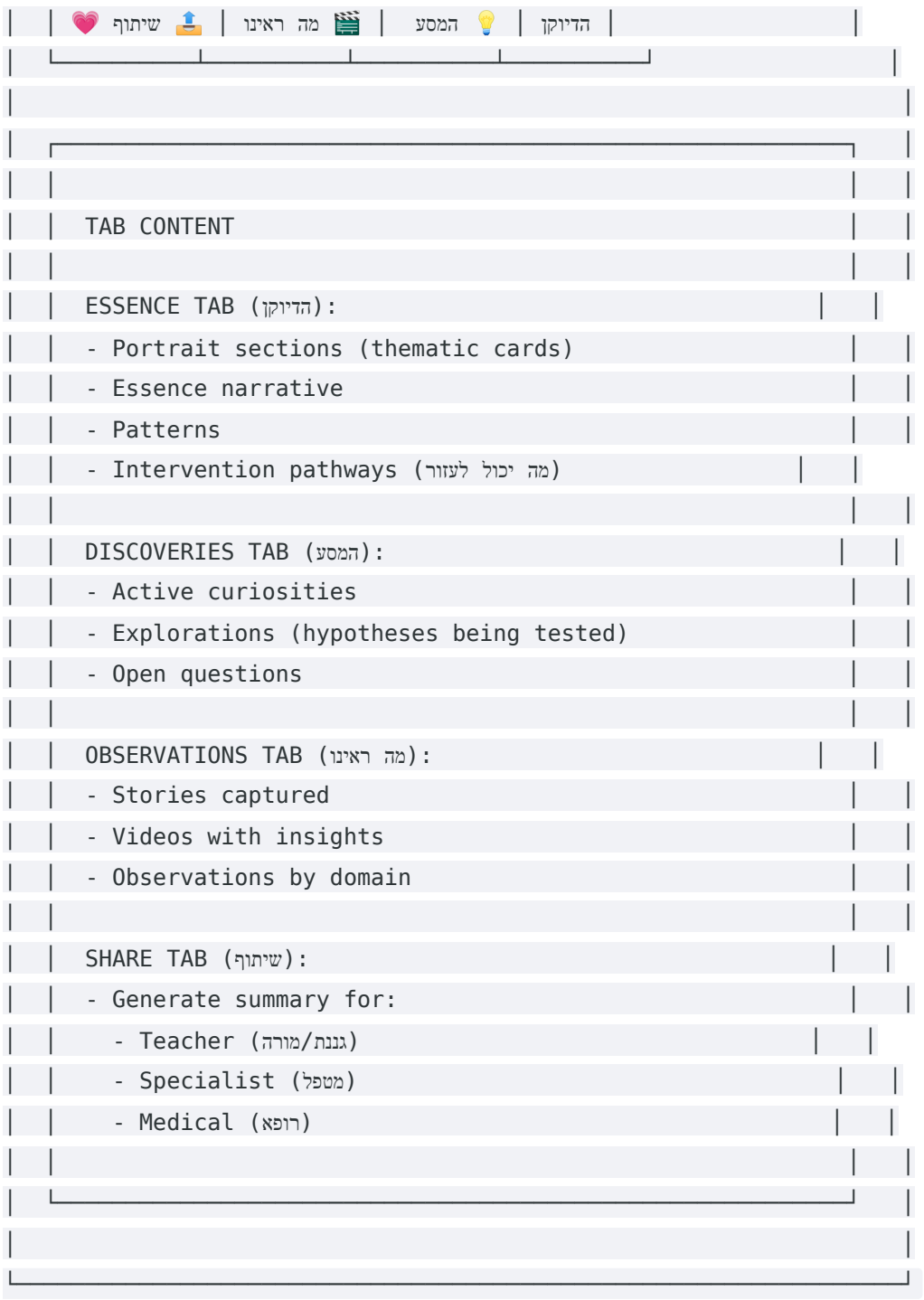
10. UI Components & Screens

Main Screen Layout



ChildSpace (Modal)





Component Hierarchy

```
App.jsx
├── ChildSpaceHeader
│   └── Badges (stories, videos, patterns, explorations)
├── ConversationTranscript
│   └── Message (user/assistant)
├── GestaltCards
│   └── Card (video_suggestion, guidelines_ready, etc.)
├── InputArea
```

```
│   └── SuggestionsPopup
├── ChildSpace (modal)
│   ├── EssenceTab
│   │   ├── PortraitSections
│   │   ├── Patterns
│   │   └── InterventionPathways
│   ├── DiscoveriesTab
│   │   ├── Curiosities
│   │   └── Explorations
│   ├── ObservationsTab
│   │   ├── Stories
│   │   └── Videos
│   └── ShareTab
│       └── ProfessionalSummary
└── DeepViews (modals)
    ├── VideoUploadView
    ├── FilmingInstructionView
    └── ...
```

---

11. Development Setup

Prerequisites

- Python 3.11+
- Node.js 18+
- Google Gemini API key

Quick Start

```
# Clone and setup
cd chitta-advanced

# Backend setup
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Create .env file
echo "GEMINI_API_KEY=your_key_here" > .env

# Frontend setup
```

```
cd ..
npm install

# Run everything
./start.sh

# Or run separately:
# Terminal 1: cd backend && python -m app.main
# Terminal 2: npm run dev
```

Environment Variables

```
# Required
GEMINI_API_KEY=your_api_key

# Optional
LLM_PROVIDER=gemini           # gemini|simulated
LLM_MODEL=gemini-2.5-flash    # Conversation model
STRONG_LLM_MODEL=gemini-2.5-pro  # Synthesis model
LOG_LEVEL=INFO
```

Running Tests

```
# Unit tests
pytest backend/tests/test_curiosity.py

# Integration tests
pytest backend/tests/test_gestalt_integration.py

# All tests
pytest backend/tests/
```

---

12. Common Development Tasks

Adding a New LLM Tool

1. **Define tool in `tools.py`:**

```
TOOL_MY_NEW_TOOL = {
    "name": "my_new_tool",
    "description": "What it does...",
    "parameters": {
        "type": "object",
        "properties": {...},
```

```
        "required": [...]
    }
}
```

2. **Add to PERCEPTION_TOOLS list:**

```
PERCEPTION_TOOLS = [
    TOOL_NOTICE,
    TOOL_WONDER,
    # ...
    TOOL_MY_NEW_TOOL,  # Add here
]
```

3. **Handle in `gestalt.py` `_apply_learnings()`:**

```
def _apply_learnings(self, tool_calls):
    for call in tool_calls:
        if call.name == "my_new_tool":
            self._handle_my_new_tool(call.args)
```

Adding a New Card Type

1. **Generate card in `service.py` `_derive_cards()`:**

```
cards.append({
    "type": "my_new_card",
    "title": "כותרת בעברית",
    "description": "תיאור...",
    "actions": [
        {"label": "פעולה", "action": "do_something", "primary": True}
    ]
})
```

2. **Add config in `GestaltCards.jsx`:**

```
const CARD_CONFIGS = {
  // ...
  my_new_card: {
    icon: SomeIcon,
    accentColor: 'blue',
    // ...
  }
};
```

3. **Handle action in `App.jsx`:**

```
const handleCardAction = async (card, action) => {
  if (action === 'do_something') {
```

```
    await api.doSomething(familyId);
  }
};
```

Adding a New API Endpoint

1. **Add route in `routes.py`:**

```
@router.post("/my-endpoint")
async def my_endpoint(request: MyRequest):
    service = ChittaService()
    result = await service.my_method(request.family_id)
    return result
```

2. **Add client method in `client.js`:**

```
async myMethod(familyId) {
  const response = await fetch(`${this.baseUrl}/my-endpoint`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ family_id: familyId })
  });
  return response.json();
}
```

Modifying the Portrait/Crystal

1. **Update schema in `portrait_schema.py`:**

```
class PortraitOutput(BaseModel):
    my_new_field: str = Field(description="...")
```

2. **Update synthesis prompt in `synthesis.py`**
3. **Update Crystal dataclass in `models.py`**
4. **Update ChildSpace.jsx to display new field**

Quick Reference Card

Files You'll Touch Most Often

Task	Files
Modify LLM behavior	`gestalt.py`, `formatting.py`
Add new tool	`tools.py`, `gestalt.py`
Change cards	`service.py`, `GestaltCards.jsx`

Task	Files
Modify portrait	`synthesis.py`, `portrait_schema.py`, `ChildSpace.jsx`
Add API endpoint	`routes.py`, `client.js`
Change data model	`models.py`, `service.py`

Key Constants

Constant	Location	Value
Session gap	`service.py`	4 hours
Curiosity decay	`curiosity.py`	2% per day
Max curiosities shown	`service.py`	5
LLM temperature (perceive)	`gestalt.py`	0.0
LLM temperature (respond)	`gestalt.py`	0.7
Max output tokens	`gestalt.py`	4000+

The Two Rules You Must Never Break

1. **Two-Phase Architecture**: Never try to get tool calls and text in one LLM call
2. **No String Matching for Content**: Check domains, not content strings

---

Welcome to the team! Read `CLAUDE.md` for the full coding constitution.