🔌 VCTT-AGI Engine - Complete API Reference

Last Updated: 2025-11-22
Purpose: Prevent accidental API breakages by documenting all endpoint contracts
Stack: Frontend (React/Next.js on Vercel) ↔ Backend (NestJS on Render) + PostgreSQL

📋 Table of Contents

🎯 Core Chat System

Purpose: Main VCTT-AGI conversation interface
Frontend Component: `ChatbotLanding.tsx`
Backend Controller: `session.controller.ts`
Base Path: `/api/v1/session`

POST `/api/v1/session/start`

Create a new conversation session
- **Request:**
`json`
```
{
  "user_id": "anonymous",
  "input": "Hello"
}
```
- **Response:**
`json`
```
{
  "session_id": "550e8400-e29b-41d4-a716-446655440000"
}
```
- **Frontend Usage:** `ChatbotLanding.tsx:62` - Called when user starts new chat
- ⚠️ **CRITICAL:** Must use RFC 4122 UUID, not timestamps!

POST `/api/v1/session/step`

Process a conversation turn (send message and get response)

- **Request:**

```
json
  {
    "session_id": "550e8400-e29b-41d4-a716-446655440000",
    "input": "What is consciousness?"
  }
```

- **Response:**

```
json
  {
    "response": "Based on your question...",
    "internal_state": {
      "vctt": {
        "voice": 0.8,
        "choice": 0.75,
        "transparency": 0.7,
        "trust": 0.75
      },
      "sim": {
        "tension": 0.3,
        "uncertainty": 0.2,
        "emotional_intensity": 0.15
      },
      "regulation": "normal"
    }
  }
```

- **Frontend Usage:** `ChatbotLanding.tsx:140` - Called when user sends message
- ⚠️ **CRITICAL:** Keys are `session_id` (not `sessionId`) and `input` (not `message`)!

GET `/api/v1/session/:id`

Retrieve full session details (resume conversation)

- **Response:**

```
json
  {
    "session_id": "550e8400-e29b-41d4-a716-446655440000",
    "user_id": "anonymous",
    "created_at": "2024-01-15T10:00:00Z",
    "messages": [
      {
        "id": "msg-uuid",
        "role": "user",
        "content": "Hello",
        "timestamp": "2024-01-15T10:00:00Z"
      }
    ],
    "internal_state": { ... }
  }
```

- **Frontend Usage:** `ChatbotLanding.tsx:41` - Called when resuming session from URL param

🖥️ IDE / DeepAgent

Purpose: Autonomous code editor with AI-powered editing, terminal, Git integration
Frontend Component: `DeepAgent.tsx` , `AIChat.tsx`
Backend Controller: `ide.controller.ts` , `deepagent.controller.ts`
Base Path: `/api/ide` (IDE operations), `/api/deep` (terminal execution)

POST `/api/deep/execute`

Execute sandboxed terminal command
- **Request:**
```
json
  {
    "command": "ls -la"
  }
```
- **Frontend Usage:** `DeepAgent.tsx:387` - Terminal execution
- ⚠️ **Security:** Dangerous commands are blocked (rm -rf, wget, curl, chmod 777)

POST `/api/ide/workspace/load`

Load workspace folder context (file tree + recently modified files)
- **Request:**
```
json
  {
    "path": "/workspace"
  }
```
- **Frontend Usage:** `DeepAgent.tsx:205` - Initial IDE load

GET `/api/ide/file-tree`

Get file tree structure for navigation
- **Query Params:** `path` (optional)
- **Frontend Usage:** FileTree components - File browser

POST `/api/ide/file-operation`

Create, read, update, delete files
- **Request:**
```
json
  {
    "operation": "read" | "write" | "delete" | "rename",
    "path": "/workspace/src/main.ts",
    "content": "..." // for write operations
  }
```
- **Frontend Usage:** `FileTree.tsx` , `FileTreeWithIcons.tsx` - File operations

POST `/api/ide/code-edit`

🎨 AI-powered code editing (Cmd+K feature) - MIN Autonomous Engine
- **Request:**
```
json
  {
    "filePath": "/workspace/src/main.ts",
```

```
  "instruction": "Add error handling to this function",
  "currentCode": "function foo() { ... }",
  "cursorPosition": { "line": 10, "column": 5 }
}
```

- **Frontend Usage:** `AIChat.tsx:234` - AI code editing panel
- ⚠️ **CRITICAL:** Uses Jazz Team (GPT-4, Claude, Gemini) for multi-perspective edits

POST `/api/ide/search-files`

Search for text across workspace files

- **Request:**

```
json
{
  "query": "function handleSubmit",
  "path": "/workspace"
}
```

- **Frontend Usage:** `SearchModal.tsx:35` - Cmd+Shift+F search

POST `/api/ide/run-tests`

Execute test suite

- **Frontend Usage:** AIChat.tsx - Test execution panel

POST `/api/ide/code-analysis`

Analyze code for issues, complexity, suggestions

- **Frontend Usage:** AIChat.tsx - Code analysis panel

GET `/api/ide/deployment-status`

Get current deployment status

- **Frontend Usage:** `DeploymentPanel.tsx` - Deployment status indicator

POST `/api/ide/deploy`

Trigger deployment to production

- **Frontend Usage:** `DeploymentPanel.tsx` - Deploy button

GET `/api/ide/image-preview`

Get image preview for rendering in editor

- **Query Params:** `path` (image file path)
- **Frontend Usage:** Image preview in editor

🎯 Goals System

Purpose: Goal tracking, progress monitoring, subgoal decomposition
Frontend Component: `GoalsPanel.tsx` (part of AIChat)
Backend Controller: `goal.controller.ts`
Base Path: `/api/goals`

POST `/api/goals`

Create a new goal

- **Request:**

```
json
  {
    "title": "Launch VCTT-AGI",
    "description": "Complete MVP and deploy to production",
    "priority": "high",
    "parent_id": null
  }
```

- **Frontend Usage:** Goals panel - Create goal button

GET `/api/goals`

Get all goals

- **Query Params:** `status` (optional) - Filter by “active”, “completed”, “paused”
- **Frontend Usage:** Goals panel - Initial load

GET `/api/goals/active`

Get only active goals

- **Frontend Usage:** `AIChat.tsx:193` - Active goals display

GET `/api/goals/tree`

Get goal hierarchy tree (goals + subgoals)

- **Response:** Nested tree structure with parent-child relationships
- **Frontend Usage:** Goals panel - Tree visualization

GET `/api/goals/state-awareness`

Get MIN state awareness (self-awareness test) - Returns system’s understanding of its own goals

- **Frontend Usage:** Admin monitoring

GET `/api/goals/:id`

Get single goal details

- **Frontend Usage:** Goals panel - Goal detail view

PUT `/api/goals/:id`

Update a goal

- **Request:**

```
json
  {
    "title": "Updated title",
    "status": "in_progress"
  }
```

- **Frontend Usage:** Goals panel - Edit goal

POST `/api/goals/:id/status`

Change goal status

- **Request:**

```
json
  {
    "status": "completed" | "paused" | "active"
```

```
  }
```
- **Frontend Usage:** Goals panel - Status dropdown

POST `/api/goals/:id/progress`

Add progress update to a goal
- **Request:**
```
json
  {
    "progress_percent": 75,
    "note": "Completed frontend integration"
  }
```
- **Frontend Usage:** Goals panel - Progress tracking

DELETE `/api/goals/:id`

Delete a goal
- **Frontend Usage:** Goals panel - Delete button

🛡️ Safety System

Purpose: Safety controls, kill switch, audit logging, mode regulation
Frontend Component: `AdminSafetyDashboard.tsx`, `AIChat.tsx` (safety status)
Backend Controller: `safety.controller.ts`
Base Path: `/api/safety`

GET `/api/safety/status`

Get current safety system status
- **Response:**
```
json
  {
    "kill_switch_active": false,
    "operation_mode": "normal",
    "memory_enabled": true,
    "last_audit": "2024-01-15T10:00:00Z"
  }
```
- **Frontend Usage:** `AIChat.tsx:194` - Safety status indicator

POST `/api/safety/kill-switch`

🚨 **ACTIVATE KILL SWITCH - Immediately halt all operations (ADMIN ONLY)**
- **Request:**
```
json
  {
    "reason": "Safety concern detected"
  }
```
- **Frontend Usage:** AdminSafetyDashboard.tsx - Emergency stop button
- ⚠️ **ADMIN ONLY:** Requires @BypassRegulation() decorator

POST `/api/safety/kill-switch/deactivate`

Deactivate kill switch and resume operations (ADMIN ONLY)
- **Frontend Usage:** AdminSafetyDashboard.tsx - Resume button

POST `/api/safety/mode`

Change operation mode: “normal” | “clarify” | “slow_down” (ADMIN ONLY)
- **Request:**
```
json
  {
    "mode": "clarify"
  }
```
- **Frontend Usage:** AdminSafetyDashboard.tsx - Mode selector

POST `/api/safety/memory/enable`

Enable memory system for a user (ADMIN ONLY)
- **Request:**
```
json
  {
    "user_id": "user123"
  }
```
- **Frontend Usage:** AdminSafetyDashboard.tsx - Memory controls

POST `/api/safety/memory/disable`

Disable memory system for a user (ADMIN ONLY)
- **Frontend Usage:** AdminSafetyDashboard.tsx - Memory controls

GET `/api/safety/audit`

Get audit logs (ADMIN ONLY)
- **Query Params:** `limit` (optional, default 100)
- **Response:** Array of audit log entries with timestamps, actions, outcomes
- **Frontend Usage:** AdminSafetyDashboard.tsx - Audit log viewer

GET `/api/safety/audit-log`

Get audit logs with limit (ADMIN ONLY)
- **Query Params:** `limit` (required)
- **Frontend Usage:** AdminSafetyDashboard.tsx - Paginated audit logs

POST `/api/safety/config`

Update safety configuration (ADMIN ONLY)
- **Request:**
```
json
  {
    "max_repair_iterations": 3,
    "enable_proactive_safety": true
  }
```
- **Frontend Usage:** AdminSafetyDashboard.tsx - Configuration panel

GET `/api/safety/charter`

Get VCTT AGI Safety Charter (public endpoint)
- **Response:** Full safety charter document
- **Frontend Usage:** Public safety charter page

GET `/api/safety/summary`

Get safety system summary overview
- **Frontend Usage:** Dashboard overview

GET `/api/safety/checks`

Get all safety checks
- **Frontend Usage:** Safety checks dashboard

GET `/api/safety/checks/goal`

Get goal-specific safety checks
- **Frontend Usage:** Goals safety validation

GET `/api/safety/checks/knowledge`

Get knowledge-specific safety checks
- **Frontend Usage:** Knowledge graph safety validation

GET `/api/safety/checks/coach`

Get coach-specific safety checks
- **Frontend Usage:** Coach proposal validation

POST `/api/safety/checks`

Create a new safety check rule
- **Frontend Usage:** Safety check configuration

GET `/api/safety/guardian/status`

Get SafetySteward guardian status (watchdog system health)
- **Frontend Usage:** Admin monitoring

🧠 Knowledge Graph

Purpose: Entity-relationship graph for context-aware learning
Frontend Component: Knowledge graph panels (planned)
Backend Controller: `knowledge.controller.ts`
Base Path: `/api/knowledge`

POST `/api/knowledge/extract`

Extract entities and relationships from text using LLM
- **Request:**

```json
  {
    "text": "Claude is a large language model developed by Anthropic."
  }
```

- **Response:**

```
json
  {
    "entities": [
      { "name": "Claude", "type": "AI_MODEL" },
      { "name": "Anthropic", "type": "ORGANIZATION" }
    ],
    "relationships": [
      { "from": "Claude", "to": "Anthropic", "type": "DEVELOPED_BY" }
    ]
  }
```

POST `/api/knowledge/entity`

Create or update an entity in the knowledge graph

- **Request:**

```
json
  {
    "name": "VCTT-AGI",
    "type": "SYSTEM",
    "properties": { "version": "1.0", "status": "production" }
  }
```

GET `/api/knowledge/entity/:id`

Get entity with all relationships

POST `/api/knowledge/relationship`

Create a relationship between two entities

- **Request:**

```
json
  {
    "from_entity_id": "entity-1",
    "to_entity_id": "entity-2",
    "relationship_type": "DEPENDS_ON",
    "properties": { "weight": 0.9 }
  }
```

GET `/api/knowledge/query`

Query knowledge graph using natural language

- **Query Params:** `q` (query string)

GET `/api/knowledge/subgraph/:entityId`

Get subgraph around an entity (N-hop neighborhood)

- **Query Params:** `depth` (default 2)

GET `/api/knowledge/concepts`

Get concept hierarchy (taxonomy tree)

GET `/api/knowledge/concepts/:conceptId/entities`

Get all entities tagged with a concept

DELETE `/api/knowledge/entity/:id`

Delete an entity and its relationships

GET `/api/knowledge/graph`

Get full knowledge graph visualization data (nodes + edges)

GET `/api/knowledge/search`

Search knowledge graph by query string
- **Query Params:** `q` (search query)

💾 Memory System

Purpose: GDPR-compliant user memory with consent management
Frontend Component: Memory consent dialogs
Backend Controller: `memory.controller.ts`
Base Path: `/api/memory`

POST `/api/memory/consent/grant`

User grants consent for memory persistence
- **Request:**

```
json
  {
    "user_id": "user123"
  }
```

POST `/api/memory/consent/revoke`

User revokes consent and deletes all memories (GDPR right to deletion)
- **Request:**

```
json
  {
    "user_id": "user123"
  }
```

GET `/api/memory/consent/:userId`

Get consent status for a user

POST `/api/memory/store`

Store a memory entry
- **Request:**

```
json
  {
    "user_id": "user123",
    "content": "User prefers dark mode",
    "type": "preference",
    "importance": 0.8
  }
```

GET `/api/memory/retrieve`

Retrieve memories for a user
- **Query Params:** `userId` (required), `type` (optional), `limit` (optional)

DELETE `/api/memory/:memoryId`

Delete a specific memory
- **Query Params:** `userId` (required for authorization)

DELETE `/api/memory/all/:userId`

Delete all memories for a user (GDPR right to deletion)

GET `/api/memory/export/:userId`

Export all memories for a user (GDPR data portability)
- **Response:** JSON export of all user memories

📅 Scheduler

Purpose: Task scheduling with human-in-the-loop approval
Frontend Component: Scheduler panel (planned)
Backend Controller: `scheduler.controller.ts`
Base Path: `/api/scheduler`

POST `/api/scheduler/schedule`

Schedule a new task
- **Request:**

```
json
  {
    "goal_id": "goal-uuid",
    "task_description": "Run daily health check",
    "scheduled_time": "2024-01-15T10:00:00Z",
    "requires_approval": true
  }
```

PUT `/api/scheduler/approve/:id`

Approve a pending scheduled task (ADMIN ONLY)

DELETE `/api/scheduler/cancel/:id`

Cancel a scheduled task (ADMIN ONLY)

GET `/api/scheduler/pending`

Get all pending tasks awaiting approval

GET `/api/scheduler/goal/:goalId`

Get all scheduled tasks for a specific goal

GET `/api/scheduler/status/:id`

Get status of a scheduled task

🎓 Coach & Skills

Purpose: Self-improvement proposals for skills and strategies
Frontend Component: `CoachDashboard.tsx`
Backend Controllers: `coach.controller.ts`, `skills.controller.ts` (NOT YET IMPLEMENTED)
Base Path: `/api/coach`, `/api/skills`

⚠️ NOTE: These endpoints are called by frontend but controllers don't exist yet!

GET `/api/coach/proposals?status=pending`

Get coach improvement proposals awaiting approval
- **Frontend Usage:** `CoachDashboard.tsx:27`
- **Status:** ❌ 404 - Controller not implemented yet

GET `/api/skills/candidates?minTau=0.85&minCount=3`

Get skill candidates meeting criteria (high confidence, sufficient usage)
- **Frontend Usage:** `CoachDashboard.tsx:40`
- **Status:** ❌ 404 - Controller not implemented yet

POST `/api/coach/proposals/:proposalId/approve`

Approve a coach proposal (ADMIN ONLY)
- **Frontend Usage:** `CoachDashboard.tsx:58`
- **Status:** ❌ 404 - Controller not implemented yet

POST `/api/coach/proposals/:proposalId/reject`

Reject a coach proposal (ADMIN ONLY)
- **Frontend Usage:** `CoachDashboard.tsx:71`
- **Status:** ❌ 404 - Controller not implemented yet

📊 Analytics

Purpose: Session analytics, trust metrics, cost tracking, performance monitoring
Frontend Component: Analytics dashboard (planned)
Backend Controller: `analytics.controller.ts`
Base Path: `/api/v1/analytics`

GET `/api/v1/analytics/sessions`

Get all sessions for a user
- **Query Params:** `user_id` (optional)

GET `/api/v1/analytics/sessions/:sessionId/history`

Get session history with messages

GET `/api/v1/analytics/trust-metrics`

Get trust (τ) metrics over time
- **Query Params:** `user_id` (optional)

GET `/api/v1/analytics/aggregate`

Get aggregate analytics across all sessions
- **Query Params:** `user_id` (optional)

GET `/api/v1/analytics/cross-session-patterns`

Get cross-session learning patterns

GET `/api/v1/analytics/cost`

Get cost analytics with detailed breakdown
- **Query Params:** `user_id`, `start_date`, `end_date`

GET `/api/v1/analytics/performance`

Get performance metrics (latency, throughput, error rates)

GET `/api/v1/analytics/export`

Export analytics data (CSV/JSON)
- **Query Params:** `format` (“csv” | “json”)

🤖 LLM Committee

Purpose: Multi-LLM consensus system analytics (Jazz Team)
Frontend Component: `LLMCommitteePanel.tsx`
Backend Controller: `llm-committee.controller.ts`
Base Path: `/api/v1/analytics/llm-committee`

GET `/api/v1/analytics/llm-committee/session/:sessionId`

Get LLM Committee statistics for a specific session
- **Response:**

```json
{
  "models_used": ["gpt-4", "claude-3", "gemini-pro"],
  "consensus_rate": 0.87,
  "disagreement_count": 3,
  "avg_response_time_ms": 1240
}
```

GET `/api/v1/analytics/llm-committee/global`

Get global LLM Committee statistics across all sessions

🍄 Truth Mycelium

Purpose: Verified fact network with confidence scores
Backend Controller: `truth-mycelium.controller.ts`
Base Path: `/truth-mycelium`

GET `/truth-mycelium/`

Get all verified facts from the mycelium
- **Response:** Array of facts with confidence scores and verification timestamps

GET `/truth-mycelium/stats`

Get Truth Mycelium statistics
- **Response:**

```
json
  {
    "total_facts": 1247,
    "avg_confidence": 0.89,
    "last_update": "2024-01-15T10:00:00Z"
  }
```

GET `/truth-mycelium/health`

Check Truth Mycelium health status

🏥 Health Check

Purpose: System health monitoring
Backend Controller: `health.controller.ts`
Base Path: `/health`

GET `/health`

Health check endpoint - Returns 200 if system is operational
- **Frontend Usage:** `DeepAgent.tsx:159` - Startup health check
- **Response:**

```
json
  {
    "status": "ok",
    "timestamp": "2024-01-15T10:00:00Z",
    "database": "connected",
    "version": "1.0.0"
  }
```

🔗 Frontend → Backend Connection Map

ChatbotLanding.tsx

```
POST   /api/v1/session/start          → Create session
POST   /api/v1/session/step           → Send message
GET    /api/v1/session/:id            → Resume session
```

DeepAgent.tsx

```
GET    /health                          → Health check
POST   /api/ide/workspace/load          → Load workspace
POST   /api/deep/execute                → Execute terminal command
```

AIChat.tsx

```
GET    /api/safety/status               → Safety status
GET    /api/goals/active                → Active goals
POST   /api/ide/code-edit               → AI code editing (Cmd+K)
```

CoachDashboard.tsx

```
GET    /api/coach/proposals?status=pending    → ❌ NOT IMPLEMENTED
GET    /api/skills/candidates?...             → ❌ NOT IMPLEMENTED
POST   /api/coach/proposals/:id/approve       → ❌ NOT IMPLEMENTED
POST   /api/coach/proposals/:id/reject        → ❌ NOT IMPLEMENTED
```

FileTree Components

```
POST   /api/ide/file-operation          → Read/write/delete files
```

SearchModal.tsx

```
POST   /api/ide/search-files            → Search in files
```

🚨 Common Mistakes to Avoid

1. Endpoint Path Mismatches ❌

- ❌ **WRONG:** `POST /api/sessions` (doesn't exist)
- ✅ **CORRECT:** `POST /api/v1/session/start`

2. Request Body Key Mismatches ❌

- ❌ **WRONG:** `{ "sessionId": "...", "message": "..." }`
- ✅ **CORRECT:** `{ "session_id": "...", "input": "..." }`

3. UUID Format Errors ❌

- ❌ **WRONG:** `Date.now().toString()` → `"1763813667364"` (timestamp)
- ✅ **CORRECT:** `uuidv4()` → `"550e8400-e29b-41d4-a716-446655440000"` (RFC 4122)

4. CORS Configuration ❌

- ❌ **WRONG:** Hardcoded domain list
- ✅ **CORRECT:** Regex pattern `/^https:\/\/vctt-agi-[a-z0-9-]+\.vercel\.app$/`

5. Missing `/api` Prefix ❌

- ❌ **WRONG:** `@Controller('goals')`

- ✅ **CORRECT:** `@Controller('api/goals')`

🧪 Testing Checklist

Before deploying changes, verify:
- [ ] All frontend API calls match backend endpoint paths
- [ ] Request body keys match backend DTOs
- [ ] Response handling maps backend fields correctly
- [ ] UUIDs are generated with `uuidv4()`, not timestamps
- [ ] CORS allows the deployed frontend domain
- [ ] All controllers have `/api` prefix in `@Controller()` decorator
- [ ] Swagger documentation is updated at `/api-docs`

📚 Quick Reference

Backend URL (Production): https://vctt-agi-backend.onrender.com
Frontend URL (Production): https://vctt-agi-ui.vercel.app
API Documentation: https://vctt-agi-backend.onrender.com/api-docs
Health Check: https://vctt-agi-backend.onrender.com/health

Database: PostgreSQL (Render managed)
ORM: Prisma
Schema: `/nodejs_space/prisma/schema.prisma`

🔧 How to Add New Endpoints

1. **Create Backend Controller:**
   ```
   typescript
   @Controller('api/my-feature')  // ✅ Include /api prefix!
   export class MyFeatureController {
     @Get('list')
     async getList() { ... }
   }
   ```

2. **Add Swagger Documentation:**
   ```
   typescript
   @ApiOperation({ summary: 'Get list of items' })
   @ApiResponse({ status: 200, description: 'Items retrieved' })
   ```

3. **Update Frontend:**
   ```
   typescript
   const response = await fetch(`${BACKEND_URL}/api/my-feature/list`);
   ```

4. **Update This Documentation:**
   - Add endpoint to relevant section
   - Document request/response format

- Note frontend usage locations
- Add to testing checklist

5. **Test End-to-End:**
   - Test locally
   - Deploy backend → Save checkpoint
   - Deploy frontend → Commit & push
   - Verify production deployment

🎉 **You now have a complete API reference! Keep this updated when adding new endpoints.**