

## Contents

<b>SECTION 18: THINK TANK CONSUMER PLATFORM (v3.5.0)</b>	<b>1</b>
	1
18.1 Think Tank Overview . . . . .	1
18.2 Think Tank Database Schema . . . . .	1
18.3 Think Tank Engine Service . . . . .	2
	7

## SECTION 18: THINK TANK CONSUMER PLATFORM (v3.5.0)

### 18.1 Think Tank Overview

Complex problem decomposition and multi-step reasoning with chain-of-thought processing.

### 18.2 Think Tank Database Schema

-- migrations/027\_think\_tank.sql

```
CREATE TABLE thinktank_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    problem_summary TEXT,
    domain VARCHAR(50),
    complexity VARCHAR(20),
    total_steps INTEGER DEFAULT 0,
    avg_confidence DECIMAL(3, 2),
    solution_found BOOLEAN DEFAULT false,
    total_tokens INTEGER DEFAULT 0,
    total_cost DECIMAL(10, 6) DEFAULT 0,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMPTZ
);
CREATE TABLE thinktank_steps (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES thinktank_sessions(id) ON DELETE CASCADE,
    step_number INTEGER NOT NULL,
    step_type VARCHAR(50) NOT NULL,
    description TEXT,
    reasoning TEXT,
    result TEXT,
    confidence DECIMAL(3, 2),
    model_used VARCHAR(100),
```

```

tokens_used INTEGER,
duration_ms INTEGER,
created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE thinktank_tools (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tool_name VARCHAR(100) NOT NULL UNIQUE,
    tool_type VARCHAR(50) NOT NULL,
    description TEXT,
    parameters_schema JSONB NOT NULL,
    implementation TEXT,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_thinktank_sessions_tenant ON thinktank_sessions(tenant_id, created_at DESC);
CREATE INDEX idx_thinktank_steps_session ON thinktank_steps(session_id, step_number);

ALTER TABLE thinktank_sessions ENABLE ROW LEVEL SECURITY;
ALTER TABLE thinktank_steps ENABLE ROW LEVEL SECURITY;

CREATE POLICY thinktank_sessions_isolation ON thinktank_sessions USING (tenant_id = current_setting('app.current_tenant'));
CREATE POLICY thinktank_steps_isolation ON thinktank_steps USING (
    session_id IN (SELECT id FROM thinktank_sessions WHERE tenant_id = current_setting('app.current_tenant'))
);

```

### 18.3 Think Tank Engine Service

```

// packages/core/src/services/thinktank-engine.ts

import { Pool } from 'pg';
import { BedrockRuntimeClient, InvokeModelCommand } from '@aws-sdk/client-bedrock-runtime';

interface ThinkTankProblem {
    tenantId: string;
    userId: string;
    problem: string;
    domain?: string;
    maxSteps?: number;
    tools?: string[];
}

interface ThinkTankStep {
    stepNumber: number;
    type: 'decompose' | 'reason' | 'execute' | 'verify' | 'synthesize';
    description: string;
    reasoning: string;
}

```

```

    result: string;
    confidence: number;
}

interface ThinkTankResult {
    sessionId: string;
    solution: string;
    steps: ThinkTankStep[];
    confidence: number;
    totalTokens: number;
    totalCost: number;
}

export class ThinkTankEngine {
    private pool: Pool;
    private bedrock: BedrockRuntimeClient;

    constructor(pool: Pool) {
        this.pool = pool;
        this.bedrock = new BedrockRuntimeClient({});
    }

    async solve(problem: ThinkTankProblem): Promise<ThinkTankResult> {
        // Create session
        const sessionResult = await this.pool.query(`

            INSERT INTO thinktank_sessions (tenant_id, user_id, problem_summary, domain)
            VALUES ($1, $2, $3, $4)
            RETURNING id
        `, [problem.tenantId, problem.userId, problem.problem.substring(0, 500), problem.domain]);

        const sessionId = sessionResult.rows[0].id;
        const steps: ThinkTankStep[] = [];
        let totalTokens = 0;
        let totalCost = 0;

        // Step 1: Decompose problem
        const decomposition = await this.decomposeProblem(problem.problem);
        steps.push(await this.recordStep(sessionId, 1, 'decompose', decomposition));
        totalTokens += decomposition.tokens;

        // Step 2-N: Solve each sub-problem
        for (let i = 0; i < decomposition.subProblems.length && i < (problem.maxSteps || 10); ) {
            const subProblem = decomposition.subProblems[i];

            // Reason about approach
            const reasoning = await this.reason(subProblem, steps);
            steps.push(await this.recordStep(sessionId, steps.length + 1, 'reason', reasoning));
            totalTokens += reasoning.tokens;
        }
    }
}

```

```

    // Execute solution
    const execution = await this.execute(subProblem, reasoning.approach, problem.tools);
    steps.push(await this.recordStep(sessionId, steps.length + 1, 'execute', execution));
    totalTokens += execution.tokens;
}

// Final step: Synthesize solution
const synthesis = await this.synthesize(problem.problem, steps);
steps.push(await this.recordStep(sessionId, steps.length + 1, 'synthesize', synthesis));
totalTokens += synthesis.tokens;

// Calculate cost and update session
totalCost = totalTokens * 0.00001; // Simplified cost calculation
const avgConfidence = steps.reduce((sum, s) => sum + s.confidence, 0) / steps.length;

await this.pool.query(`

    UPDATE thinktank_sessions
    SET total_steps = $2, avg_confidence = $3, solution_found = true,
        total_tokens = $4, total_cost = $5, completed_at = NOW()
    WHERE id = $1
`, [sessionId, steps.length, avgConfidence, totalTokens, totalCost]);

return {
    sessionId,
    solution: synthesis.result,
    steps,
    confidence: avgConfidence,
    totalTokens,
    totalCost
};
}

private async decomposeProblem(problem: string): Promise<any> {
    const response = await this.invokeModel(`

        Decompose this problem into smaller sub-problems:

        Problem: ${problem}

        Return JSON: { "subProblems": ["sub1", "sub2", ...], "complexity": "low|medium|high" }
    `);

    return {
        subProblems: response.subProblems || [problem],
        complexity: response.complexity || 'medium',
        tokens: response.tokens,
        description: 'Problem decomposition',
        reasoning: `Identified ${response.subProblems?.length || 1} sub-problems`,
    };
}

```

```

        result: JSON.stringify(response.subProblems),
        confidence: 0.9
    );
}

private async reason(subProblem: string, previousSteps: ThinkTankStep[]): Promise<any> {
    const context = previousSteps.map(s => s.result).join('\n');

    const response = await this.invokeModel(`

        Given context:
        ${context}

        Reason about how to solve: ${subProblem}

        Return JSON: { "approach": "description", "confidence": 0.0-1.0 }
    `);

    return {
        approach: response.approach,
        tokens: response.tokens,
        description: `Reasoning about: ${subProblem.substring(0, 50)}`,
        reasoning: response.approach,
        result: response.approach,
        confidence: response.confidence || 0.8
    };
}

private async execute(subProblem: string, approach: string, tools?: string[]): Promise<any> {
    const response = await this.invokeModel(`

        Execute this approach to solve the sub-problem:

        Sub-problem: ${subProblem}
        Approach: ${approach}
        Available tools: ${tools?.join(', ') || 'none'}

        Return JSON: { "result": "solution", "confidence": 0.0-1.0 }
    `);

    return {
        tokens: response.tokens,
        description: 'Executing solution',
        reasoning: approach,
        result: response.result,
        confidence: response.confidence || 0.8
    };
}

private async synthesize(originalProblem: string, steps: ThinkTankStep[]): Promise<any> {

```

```

const stepResults = steps.map(s => s.result).join('\n');

const response = await this.invokeModel(`

    Synthesize a final solution from these steps:

Original problem: ${originalProblem}

Step results:
${stepResults}

Return JSON: { "solution": "complete solution", "confidence": 0.0-1.0 }
`);

return {
    tokens: response.tokens,
    description: 'Synthesizing final solution',
    reasoning: 'Combining all step results',
    result: response.solution,
    confidence: response.confidence || 0.85
};
}

private async invokeModel(prompt: string): Promise<any> {
    const response = await this.bedrock.send(new InvokeModelCommand({
        modelId: 'anthropic.claude-3-haiku-20240307-v1:0',
        body: JSON.stringify({
            anthropic_version: 'bedrock-2023-05-31',
            max_tokens: 2048,
            messages: [{ role: 'user', content: prompt }]
        }),
        contentType: 'application/json'
    }));

    const result = JSON.parse(new TextDecoder().decode(response.body));
    const text = result.content[0].text;

    try {
        const jsonMatch = text.match(/\{[\s\S]*\}/);
        const parsed = jsonMatch ? JSON.parse(jsonMatch[0]) : { result: text };
        return { ...parsed, tokens: result.usage?.output_tokens || 100 };
    } catch {
        return { result: text, tokens: result.usage?.output_tokens || 100 };
    }
}

private async recordStep(
    sessionId: string,
    stepNumber: number,

```

```
    stepType: string,
    data: any
  ): Promise<ThinkTankStep> {
  await this.pool.query(`  

    INSERT INTO thinktank_steps (session_id, step_number, step_type, description, reasoning)  

    VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
  `, [sessionId, stepNumber, stepType, data.description, data.reasoning, data.result, data.confidence]
  return {
    stepNumber,
    type: stepType as any,
    description: data.description,
    reasoning: data.reasoning,
    result: data.result,
    confidence: data.confidence
  };
}
}
```