Contents

Think Tank Admin Guide

Administration of the Think Tank Consumer AI Platform

Version: 4.18.0 | Last Updated: January 2026

Overview

This guide covers administration of **Think Tank**, the consumer-facing AI assistant application. Think Tank Admin is a **separate application** from the RADIANT Platform Admin Dashboard.

Application Separation

Application	URL	Purpose
RADIANT Admin	`admin.radiant.ai`	Platform infrastructure, tenants, billing, models, compliance
Think Tank Admin	`manage.thinktank.ai`	Consumer AI features, users, conversations, AI behavior

For platform-level administration, see RADIANT-ADMIN-GUIDE.md.

Table of Contents

1. Dashboard

Path: `/`
App File: `apps/thinktank-admin/app/(dashboard)/page.tsx`

Overview

The Think Tank Admin dashboard provides real-time overview of:

- **Active Users**: Users currently engaged with Think Tank
- **Total Conversations**: Conversation count with period comparison
- **User Rules**: Custom AI behavior rules created by users
- **API Requests**: Request volume and trends

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank-admin/dashboard/stats`	Dashboard statistics

Implementation

- **Lambda**: `lambda/thinktank-admin/dashboard.ts`
- **CDK**: `lib/stacks/thinktank-admin-api-stack.ts`

---

2. Users Management

Path: `/users`
App File: `apps/thinktank-admin/app/(dashboard)/users/page.tsx`

Features

- View all Think Tank users for the tenant
- User activity metrics (conversations, messages, tokens)
- User suspension/activation
- Usage statistics per user

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/users`	List users with pagination
GET	`/api/thinktank/users/:id`	Get user details
GET	`/api/thinktank/users/stats`	Aggregate user stats
POST	`/api/thinktank/users/:id/suspend`	Suspend user
POST	`/api/thinktank/users/:id/activate`	Activate user

Implementation

- **Lambda**: `lambda/thinktank/users.ts`
- **Database**: `users` table with RLS on `tenant_id`

---

3. Conversations

Path: `/conversations`
App File: `apps/thinktank-admin/app/(dashboard)/conversations/page.tsx`

Features

- Browse all conversations across users
- Filter by date, user, model
- View conversation messages
- Delete conversations (compliance)
- Conversation statistics

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/conversations`	List conversations
GET	`/api/thinktank/conversations/:id`	Get conversation detail
GET	`/api/thinktank/conversations/:id/messages`	Get messages
DELETE	`/api/thinktank/conversations/:id`	Delete conversation
GET	`/api/thinktank/conversations/stats`	Conversation stats

Implementation

- **Lambda**: `lambda/thinktank/conversations.ts`
- **Database**: `thinktank_conversations`, `thinktank_messages`

---

4. Analytics

Path: `/analytics`
App File: `apps/thinktank-admin/app/(dashboard)/analytics/page.tsx`

Features

- Usage trends over time (7/30/90 days)
- Model usage breakdown
- Cost analysis
- Token consumption metrics
- Average session duration

Metrics Tracked

Metric	Description
Total Users	All registered users
Active Users	Users with activity in period
Total Conversations	Conversation count
Total Messages	Message count
Total Tokens	Token consumption
Total Cost	API cost in dollars
Avg Session Duration	Average conversation length

API Endpoints

Method	Endpoint	Purpose
GET	`/api/admin/thinktank/analytics?days=30`	Usage analytics

Implementation

- **Lambda**: `lambda/thinktank/analytics.ts`
- **CDK**: `lib/stacks/thinktank-admin-api-stack.ts`

---

5. My Rules (User-Defined AI Behavior)

Path: `/my-rules`
App File: `apps/thinktank-admin/app/(dashboard)/my-rules/page.tsx`

Overview

Users can define custom rules that modify AI behavior. Administrators can view, manage, and create preset rules.

Rule Types

Type	Purpose
`style`	Response style (formal, casual, technical)
`behavior`	Behavioral preferences
`constraint`	Restrictions on output
`enhancement`	Additional capabilities

Rule Structure

```
interface UserRule {
  id: string;
  userId: string;
  tenantId: string;
  name: string;
  description: string;
  type: 'style' | 'behavior' | 'constraint' | 'enhancement';
  ruleContent: string;
  priority: number;
  isActive: boolean;
  createdAt: Date;
  updatedAt: Date;
}
```

API Endpoints

Method	Endpoint	Purpose
GET	`/api/admin/my-rules`	List all rules
POST	`/api/admin/my-rules`	Create rule
PUT	`/api/admin/my-rules/:id`	Update rule
DELETE	`/api/admin/my-rules/:id`	Delete rule
GET	`/api/admin/my-rules/presets`	Get preset rules

Implementation

- **Lambda**: `lambda/thinktank/my-rules.ts`
- **Database**: `user_rules` table
- **CDK**: `lib/stacks/thinktank-admin-api-stack.ts`

---

6. Domain Modes

Path: `/domain-modes`
App File: `apps/thinktank-admin/app/(dashboard)/domain-modes/page.tsx`

Overview

Configure domain-specific AI behavior. The system detects query domain and adjusts model selection, prompts, and behavior accordingly.

Domain Hierarchy

```
Field (e.g., Medicine)
    Domain (e.g., Cardiology)
        Subspecialty (e.g., Interventional Cardiology)
```

Features

- View domain taxonomy
- Configure domain-specific prompts
- Set model preferences per domain

- Enable/disable domain detection

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/domain-modes/config`	Get configuration
PUT	`/api/thinktank/domain-modes/config`	Update configuration
GET	`/api/domain-taxonomy`	Get taxonomy
POST	`/api/domain-taxonomy/detect`	Detect domain from prompt

Implementation

- **Lambda**: `lambda/thinktank/domain-modes.ts`
- **Service**: `lambda/shared/services/domain-taxonomy.service.ts`

---

7. Model Categories

Path: `/model-categories`
App File: `apps/thinktank-admin/app/(dashboard)/model-categories/page.tsx`

Overview

Organize AI models into categories for user selection. Categories control which models appear in the Think Tank UI.

Default Categories

Category	Description	Example Models
Fast	Quick responses	GPT-4o-mini, Claude Haiku
Balanced	Default choice	GPT-4o, Claude Sonnet
Powerful	Complex tasks	Claude Opus, GPT-4
Specialized	Domain-specific	Codex, DALL-E

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/model-categories`	List categories
PUT	`/api/thinktank/model-categories/:id`	Update category
POST	`/api/thinktank/model-categories/reorder`	Reorder models

Implementation

- **Lambda**: `lambda/thinktank/model-categories.ts`

---

8. Artifacts (GenUI)

Path: `/artifacts`
App File: `apps/thinktank-admin/app/(dashboard)/artifacts/page.tsx`

Overview

The Artifact Engine generates interactive UI components from natural language. Administrators configure generation rules, dependency allowlists, and monitor generation metrics.

Features

- Generation dashboard with metrics
- Dependency allowlist management
- Code pattern library
- Validation rules
- Escalation workflow

API Endpoints

Method	Endpoint	Purpose
GET	`/api/v2/admin/artifact-engine/dashboard`	Dashboard data
GET	`/api/v2/admin/artifact-engine/metrics`	Generation metrics
GET	`/api/v2/admin/artifact-engine/validation-rules`	Validation rules
PUT	`/api/v2/admin/artifact-engine/validation-rules/:id`	Update rule
POST	`/api/v2/admin/artifact-engine/allowlist`	Add to allowlist
DELETE	`/api/v2/admin/artifact-engine/allowlist/:pkg`	Remove from allowlist

Implementation

- **Lambda**: `lambda/thinktank/artifact-engine.ts`
- **Service**: `lambda/shared/services/generative-ui.service.ts`
- **Database**: `artifact_sessions`, `artifact_generations`

---

9. Collaboration

Path: `/collaborate`, `/collaborate/enhanced`
App Files: - `apps/thinktank-admin/app/(dashboard)/collaborate/page.tsx` - `apps/thinktank-admin/app/`

Overview

Real-time collaboration features allowing multiple users to work on shared conversations.

Features

- **Basic Collaboration**: Shared conversation links
- **Enhanced Collaboration**: Real-time cursors, typing indicators, presence

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/collaboration/sessions`	List sessions
POST	`/api/thinktank/collaboration/sessions`	Create session
GET	`/api/thinktank/collaboration/settings`	Get settings
PUT	`/api/thinktank/collaboration/settings`	Update settings

Implementation

- **Lambda**: `lambda/thinktank/enhanced-collaboration.ts`
- **WebSocket**: `lambda/collaboration/` handlers
- **Service**: `lambda/shared/services/enhanced-collaboration.service.ts`

---

10. Delight System

Path: `/delight`, `/delight/statistics`
App Files: - `apps/thinktank-admin/app/(dashboard)/delight/page.tsx` - `apps/thinktank-admin/app/(das`

Overview

The Delight System provides achievement notifications, progress tracking, and gamification features to enhance user engagement.

Features

- Achievement configuration
- Delight message templates
- User progress tracking
- Statistics dashboard

Delight Types

Type	Description
`achievement`	Milestone completions
`streak`	Consecutive usage
`discovery`	Feature exploration
`mastery`	Skill development

API Endpoints

Method	Endpoint	Purpose
GET	`/api/delight/messages`	Get delight messages
GET	`/api/delight/achievements`	List achievements

Method	Endpoint	Purpose
GET	`/api/delight/progress/:userId`	User progress
PUT	`/api/delight/preferences`	Update preferences

Implementation

- **Lambda**: `lambda/delight/handler.ts`
- **Service**: `lambda/shared/services/delight.service.ts`

---

11. Governor (Economic Optimization)

Path: `/governor`
App File: `apps/thinktank-admin/app/(dashboard)/governor/page.tsx`

Overview

The Economic Governor optimizes AI costs by learning routing patterns and selecting cost-effective models without sacrificing quality.

Features

- Real-time cost dashboard
- Savings metrics
- Mode switching (aggressive, balanced, quality)
- Budget alerts

Modes

Mode	Description	Savings Target
`aggressive`	Maximum savings	70%+
`balanced`	Balance cost/quality	50%
`quality`	Quality priority	20%

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/economic-governor/dashboard`	Dashboard
GET	`/api/thinktank/economic-governor/config`	Get config
PUT	`/api/thinktank/economic-governor/config`	Update config
POST	`/api/thinktank/economic-governor/mode`	Quick mode switch

Implementation

- **Lambda**: `lambda/thinktank/economic-governor.ts`

- **Service**: `lambda/shared/services/economic-governor.service.ts`

---

12. Shadow Testing (A/B Testing)

Path: `/shadow-testing`
App File: `apps/thinktank-admin/app/(dashboard)/shadow-testing/page.tsx`

Overview

Test pre-prompt optimizations and model configurations in production without affecting users.

Features

- Create A/B tests for prompts
- Traffic allocation (0-100%)
- Statistical significance tracking
- Promote winning variant

Test Structure

```
interface ShadowTest {
  id: string;
  name: string;
  description: string;
  controlPrompt: string;
  testPrompt: string;
  trafficPercent: number;
  status: 'draft' | 'running' | 'paused' | 'completed';
  metrics: {
    controlSamples: number;
    testSamples: number;
    controlScore: number;
    testScore: number;
    pValue: number;
  };
}
```

API Endpoints

Method	Endpoint	Purpose
GET	`/api/admin/shadow-tests`	List tests
POST	`/api/admin/shadow-tests`	Create test
POST	`/api/admin/shadow-tests/:id/start`	Start test
POST	`/api/admin/shadow-tests/:id/stop`	Stop test
POST	`/api/admin/shadow-tests/:id/promote`	Promote winner
GET	`/api/admin/shadow-tests/settings`	Get settings
PUT	`/api/admin/shadow-tests/settings`	Update settings

Implementation

- **Lambda**: `lambda/thinktank/shadow-testing.ts`
- **Database**: `shadow_tests`, `shadow_test_samples`
- **CDK**: `lib/stacks/thinktank-admin-api-stack.ts`

---

13. Concurrent Execution

Path: `/concurrent-execution`
App File: `apps/thinktank-admin/app/(dashboard)/concurrent-execution/page.tsx`

Overview

Execute multiple AI model calls in parallel and synthesize results.

Features

- Configure concurrent model pools
- Set merge strategies
- Monitor execution metrics
- Task queue management

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/concurrent-execution/config`	Get config
PUT	`/api/thinktank/concurrent-execution/config`	Update config
GET	`/api/thinktank/concurrent-execution/queue`	Queue status
GET	`/api/thinktank/concurrent-execution/metrics`	Metrics

Implementation

- **Lambda**: `lambda/thinktank/concurrent-execution.ts`
- **Service**: `lambda/shared/services/concurrent-execution.service.ts`

---

14. Structure from Chaos

Path: `/structure-from-chaos`
App File: `apps/thinktank-admin/app/(dashboard)/structure-from-chaos/page.tsx`

Overview

Transform unstructured input (whiteboards, notes, voice transcripts) into structured documents.

Features

- Input type configuration
- Output template management
- Extraction pipeline settings
- Processing metrics

Supported Inputs

Input Type	Description
`whiteboard`	Whiteboard images
`notes`	Unstructured notes
`transcript`	Voice/meeting transcripts
`brainstorm`	Brainstorming sessions

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/structure-from-chaos/config`	Get config
PUT	`/api/thinktank/structure-from-chaos/config`	Update config
POST	`/api/thinktank/structure-from-chaos/synthesize`	Process input
GET	`/api/thinktank/structure-from-chaos/metrics`	Metrics

Implementation

- **Lambda**: `lambda/thinktank/structure-from-chaos.ts`
- **Service**: `lambda/shared/services/structure-from-chaos.service.ts`

---

15. Grimoire (Procedural Memory)

Path: `/grimoire`
App File: `apps/thinktank-admin/app/(dashboard)/grimoire/page.tsx`

Overview

The Grimoire is the system's procedural memory—storing learned patterns, corrections, and "spells" that improve AI responses over time.

Features

- View learned spells
- Create custom spells
- Spell effectiveness metrics
- Automatic spell generation

Spell Types

Type	Description
`correction`	Error correction patterns
`enhancement`	Response improvements
`procedure`	Multi-step processes
`template`	Reusable response templates

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/grimoire/spells`	List spells
GET	`/api/thinktank/grimoire/spells/:id`	Get spell
POST	`/api/thinktank/grimoire/spells`	Create spell

Implementation

- **Lambda**: `lambda/thinktank/grimoire.ts`
- **Service**: `lambda/shared/services/grimoire.service.ts`

---

16. Magic Carpet (Adaptive Flows)

Path: `/magic-carpet`
App File: `apps/thinktank-admin/app/(dashboard)/magic-carpet/page.tsx`

Overview

Adaptive conversation flows that adjust based on user expertise and context. Includes demo components for Reality Scrubber, Quantum Split View, and Pre-Cognition suggestions.

Features

- **Reality Scrubber Timeline**: Video-editor style navigation through state snapshots
  - Bookmark creation with toast notifications
  - Play/pause timeline controls
- **Quantum Split View**: Side-by-side comparison of parallel realities
  - Branch selection with state management
  - Branch collapse with confirmation
- **Pre-Cognition Suggestions**: Predicted actions that are pre-computed
  - Prediction selection with execution feedback
  - Prediction dismissal
- **Magic Carpet Navigator**: Intent-based navigation
  - Flying to destinations with toast feedback
  - Landing confirmation

Implementation

- **App**: `apps/thinktank-admin/app/(dashboard)/magic-carpet/page.tsx`
- **Components**: `apps/thinktank-admin/components/magic-carpet/`

---

17. Workflow Templates

Path: `/workflow-templates`
App File: `apps/thinktank-admin/app/(dashboard)/workflow-templates/page.tsx`

Overview

Pre-defined workflow templates users can invoke for common tasks.

Features

- Template creation
- Variable configuration
- Usage analytics
- Template versioning

Implementation

- **App**: `apps/thinktank-admin/app/(dashboard)/workflow-templates/page.tsx`

---

18. Polymorphic UI

Path: `/polymorphic`
App File: `apps/thinktank-admin/app/(dashboard)/polymorphic/page.tsx`

Overview

Configure the polymorphic UI system that adapts interface complexity based on user needs.

Views

View	Description
`simple`	Basic chat interface
`standard`	Full-featured interface
`advanced`	Power user tools

Implementation

- **App**: `apps/thinktank-admin/app/(dashboard)/polymorphic/page.tsx`

---

19. Ego System

Path: `/ego`
App File: `apps/thinktank-admin/app/(dashboard)/ego/page.tsx`

Overview

The Zero-Cost Ego System provides persistent AI personality through database state injection.

Features

- Identity configuration (name, narrative, values)
- Personality trait sliders
- Emotional state monitoring
- Working memory management
- Goal tracking

Configuration

Setting	Description
`enabled`	Enable ego injection
`name`	AI personality name
`narrative`	Background narrative
`traits`	Personality traits (0-1)

API Endpoints

Method	Endpoint	Purpose
GET	`/api/admin/ego/dashboard`	Dashboard data
GET	`/api/admin/ego/config`	Get config
PUT	`/api/admin/ego/config`	Update config
GET	`/api/admin/ego/identity`	Get identity
PUT	`/api/admin/ego/identity`	Update identity
GET	`/api/admin/ego/preview`	Preview injection

Implementation

- **Lambda**: `lambda/admin/ego.ts`
- **Service**: `lambda/shared/services/ego-context.service.ts`
- **Database**: `ego_config`, `ego_identity`, `ego_affect`

---

20. Compliance

Path: `/compliance`
App File: `apps/thinktank-admin/app/(dashboard)/compliance/page.tsx`

Overview

Compliance settings specific to Think Tank consumer features.

Features

- Data retention policies
- Content filtering rules
- Audit log access
- Privacy settings

Implementation

- **App**: `apps/thinktank-admin/app/(dashboard)/compliance/page.tsx`

20A. Sovereign Mesh (v5.52.0)

Path: `/sovereign-mesh/*`

Overview

Sovereign Mesh provides decentralized AI agent management and transparency for Think Tank.

Sub-Pages

Page	Path	Purpose
Overview	`/sovereign-mesh`	Dashboard with agent/app stats, health score
Agents	`/sovereign-mesh/agents`	Manage AI agents, view status and performance
Apps	`/sovereign-mesh/apps`	View deployed apps, instances, users
Transparency	`/sovereign-mesh/transparency`	Audit logs, decision trails
AI Helper	`/sovereign-mesh/ai-helper`	AI assistance requests, ratings
Approvals	`/sovereign-mesh/approvals`	Pending/processed approval requests

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank-admin/sovereign-mesh/overview`	Dashboard stats
GET	`/api/thinktank-admin/sovereign-mesh/agents`	List agents
GET	`/api/thinktank-admin/sovereign-mesh/apps`	List apps
GET	`/api/thinktank-admin/sovereign-mesh/audit-logs`	Audit log entries
GET	`/api/thinktank-admin/sovereign-mesh/decision-trails`	Decision trails
GET	`/api/thinktank-admin/sovereign-mesh/ai-helper/requests`	AI helper requests
GET	`/api/thinktank-admin/sovereign-mesh/approvals`	Approval requests
POST	`/api/thinktank-admin/sovereign-mesh/approvals/:id`	Process approval

Implementation

- **App Files**: `apps/thinktank-admin/app/(dashboard)/sovereign-mesh/*.tsx`
- **Navigation**: All pages linked in sidebar under "Sovereign Mesh" section

---

21. Settings

Path: `/settings`
App File: `apps/thinktank-admin/app/(dashboard)/settings/page.tsx`

Overview

Global Think Tank configuration settings.

Settings Categories

Category	Description
General	Basic configuration
Features	Enable/disable features
Limits	Usage limits
Notifications	Alert settings

API Endpoints

Method	Endpoint	Purpose
GET	`/api/admin/thinktank/status`	Service status
GET	`/api/admin/thinktank/config`	Get config
PATCH	`/api/admin/thinktank/config`	Update config

Implementation

- **Lambda**: `lambda/thinktank/settings.ts`
- **CDK**: `lib/stacks/thinktank-admin-api-stack.ts`

---

22. Cato Persistent Memory System

Overview

Cato operates as the cognitive core behind Think Tank, implementing a **three-tier hierarchical memory system** that fundamentally differentiates it from competitors suffering from session amnesia. Unlike ChatGPT or Claude standalone—where closing a tab erases all context—Cato maintains persistent memory that survives sessions, employee turnover, and time.

Why This Matters for Think Tank Users

Competitor Problem	Think Tank Solution
Close tab = lose context	Memory persists across sessions
Every conversation starts fresh	AI "remembers" your preferences
Generic responses for everyone	Personalized to your communication style
Static model behavior	AI gets smarter over time

The Three Memory Tiers

1. Tenant-Level Memory (Institutional Intelligence) Your organization's accumulated learning:

Feature	User Benefit
Smart Model Routing	Legal queries go to citation-accurate models; creative requests go to generative models—automatically
Department Preferences	Legal teams get formal briefs; marketing gets conversational copy—no configuration needed
Cost Optimization	System learns cheaper approaches that maintain quality, reducing your costs over time
Compliance Ready	7-year audit trails for FDA, HIPAA, SOC 2 requirements

2. User-Level Memory (Relationship Continuity) Your personal AI relationship through **Ghost Vectors**—4096-dimensional representations of your interaction style:

Feature	User Benefit
Remembers Your Style	Formal vs. casual, verbose vs. concise
Tracks Expertise	Beginner explanations vs. expert shorthand
Persona Selection	Choose your preferred AI mood: Balanced, Scout, Sage, Spark, Guide
No Personality Discontinuity	Model upgrades don't break the relationship feel

Persona Options:

Persona	Behavior	Best For
Balanced	Default equilibrium	General queries
Scout	Exploratory, information-gathering	Research, discovery
Sage	Deep expertise, authoritative	Complex analysis
Spark	Creative, generative	Brainstorming, ideation
Guide	Teaching, step-by-step	Learning, onboarding

3. Session-Level Memory (Real-Time Context) Active interaction state (expires after session ends):

- Current conversation context
- Temporary persona overrides
- Real-time safety evaluations
- Epistemic uncertainty tracking

Twilight Dreaming (How Think Tank Gets Smarter)

During low-traffic periods (4 AM your local time), Think Tank consolidates patterns through **LoRA fine-tuning**:

1. **Pattern Collection**: Gathers learning from your daily interactions
2. **Intelligence Consolidation**: Transforms individual patterns into organizational intelligence
3. **Cost Optimization**: Trains smarter, cheaper routing decisions
4. **Result**: Your Think Tank deployment gets measurably smarter over time—automatically

Outcome: 60%+ cost reduction while maintaining accuracy for healthcare and financial queries.

Claude as the Conductor

Claude serves as the conductor maintaining this persistent memory layer—not just another model in the rotation, but the intelligence coordinating **105+ other specialized models**, interpreting user intent, selecting workflows, and ensuring responses meet accuracy and safety standards.

Configuration

Cato memory settings are configured through: - **Ego System** (Section 19): Identity and personality settings - **Governor** (Section 11): Cost optimization modes

Why Think Tank Beats Standalone AI (Competitive Moats)

Persistent Memory Creates "Contextual Gravity" Cato's hierarchical memory architecture creates compounding switching costs that deepen with every interaction. When you consider alternatives, here's what you'd lose:

What You'd Lose	Impact
Learned Routing Patterns	Months of optimization showing which models work best for your query types
Department Preferences	System knowledge that legal wants citations, marketing wants conversational tone
Ghost Vectors	The "feel" of thousands of individual user relationships
Compliance Audit Trails	7-year Merkle-hashed records for HIPAA, SOC 2, FDA—cannot be migrated

Competitor Comparison:

Alternative	Problem You'd Face
ChatGPT/Claude Standalone	When an employee quits, their entire AI context walks out the door—zero institutional learning
Flowise/Dify	Same expensive rate regardless of query complexity—no cost optimization learning
CrewAI	Agents don't share memory—five agents needing the same data make five duplicate API calls

Twilight Dreaming: Your Deployment Gets Smarter Over Time Think Tank isn't just a service—it's an **appreciating asset**. During low-traffic periods (4 AM your time), the system "dreams" about the day's learnings:

What Happens	Your Benefit
Routing patterns consolidate	Future queries route to optimal models automatically
Cost patterns identified	Expensive queries get cheaper alternatives
Domain improvements embed	Your deployment becomes more accurate in your specific domains

The Result: A customer who has used Think Tank for two years has a **fundamentally more capable deployment** than a new customer—with routing decisions reflecting thousands of hours of optimization.

When New Models Launch (GPT-5, Claude 5, Gemini 3): Think Tank learns how to optimally route to new capabilities while preserving all your accumulated institutional knowledge. Model improvements compound on top of existing optimization rather than resetting the learning curve.

> **Infrastructure**: See RADIANT-ADMIN-GUIDE.md Section 31A.7 for architecture, database schema, and infrastructure configuration.

---

23. Think Tank Consumer App (End-User Interface)

Path: `apps/thinktank/`
URL: `app.thinktank.ai`

Overview

The Think Tank Consumer App is the primary end-user interface for interacting with Cato. It provides a streamlined chat experience with intelligent defaults while offering advanced controls for power users.

Auto Mode vs Advanced Mode

Think Tank operates in two modes, controlled by the **Advanced Mode Toggle** (`+Shift+A`):

Feature	Auto Mode	Advanced Mode
Model Selection	Cato decides automatically	User can select specific model
Brain Plan Display	Hidden	Shows execution plan with steps
Token/Cost Display	Hidden	Shows per-message metrics
Domain Override	Auto-detected	Manual selection available
Model Routing Visibility	Hidden	Shows routing decisions

Philosophy: Most users should use Auto Mode. Advanced Mode is for developers, researchers, and power users who want granular control.

Consumer App Pages

Page	Path	Purpose
Chat	`/`	Main conversation interface
Settings	`/settings`	User preferences and personality
My Rules	`/rules`	User-defined AI behavior rules
History	`/history`	Conversation history with search
Artifacts	`/artifacts`	Generated code and documents
Profile	`/profile`	User account and statistics

Key Components

Chat Components (`apps/thinktank/components/chat/`)

Component	Purpose
`AdvancedModeToggle`	Toggle between Auto/Advanced modes
`MessageBubble`	Message display with optional metadata
`ChatInput`	Smart auto-resizing input with attachments
`Sidebar`	Conversation list with date grouping
`BrainPlanViewer`	Execution plan display (Advanced Mode)
`ModelSelector`	Full model picker dialog (Advanced Mode)

State Management (`apps/thinktank/lib/stores/`)

Store	Purpose
`ui-store.ts`	UI state (sidebar, advanced mode, sound)
`settings-store.ts`	User preferences with persistence

API Services (`apps/thinktank/lib/api/`)

Service	Endpoints
`chat.ts`	Conversations, messages, streaming
`models.ts`	Model listing, recommendations
`rules.ts`	User rules CRUD, presets
`settings.ts`	User settings, personality
`brain-plan.ts`	Plan generation and display
`analytics.ts`	User stats, achievements
`governor.ts`	Economic optimization status

Settings Page Features

The Settings page (`/settings`) provides:

- **Personality Mode**: Auto, Professional, Subtle, Expressive, Playful
- **Notifications**: Push notification preferences
- **Appearance**: Compact mode, token/cost display toggles
- **Keyboard Shortcuts**: Enable/disable, shortcut reference
- **Sound Effects**: Toggle audio feedback
- **Privacy**: Data export, account deletion

My Rules Page Features

The My Rules page (`/rules`) allows users to:

- **Create custom rules** for AI behavior
- **Browse preset rules** by category
- **Toggle rules** on/off without deleting
- **View rule application stats**

Rule types: Restriction, Preference, Format, Source, Tone, Topic, Privacy

Consumer App Tech Stack

Technology	Version	Purpose
Next.js	14.x	React framework
TypeScript	5.x	Type safety
Tailwind CSS	3.x	Styling
shadcn/ui	latest	Component library
Zustand	latest	Client state management
TanStack Query	5.x	Server state management
Framer Motion	latest	Animations
Lucide	latest	Icons

Implementation Files

```
apps/thinktank/
  app/
    (chat)/
```

```
        page.tsx        # Main chat interface
        layout.tsx      # Chat layout wrapper
      settings/page.tsx  # Settings page
      rules/page.tsx     # My Rules page
      history/page.tsx   # History page
      artifacts/page.tsx # Artifacts page
      profile/page.tsx   # Profile page
      page.tsx           # Root redirect
   components/
      chat/              # Chat-specific components
      ui/                # Base UI components
   lib/
      api/               # API services
      stores/            # Zustand stores
      utils.ts           # Utility functions
```

API Integration

The consumer app connects to the Think Tank API at `/api/thinktank/*`:

Category	Handler	Endpoints
Chat	`chat.ts`	conversations, messages, stream
Rules	`my-rules.ts`	CRUD, presets, toggle
Settings	`settings.ts`	get/update preferences
Models	`models.ts`	list, recommend
Brain Plan	`brain-plan.ts`	generate, execute, status
Analytics	`analytics.ts`	stats, achievements
Governor	`economic-governor.ts`	status, savings
Localization	`localization.ts`	languages, bundles, translations

---

24. Localization (i18n)

Path: `apps/thinktank/lib/i18n/`
API Base: `/api/localization`

Overview

Think Tank supports 18 languages through the Radiant localization registry. **ALL UI text strings MUST be localized** - no hardcoded strings are allowed in components.

Architecture

```
User App   LocalizationProvider   Localization API   Radiant Registry   PostgreSQL



              Translation Bundle (cached 1hr)
```

Key Principle: The consumer app accesses ALL data through the API. Never direct database access.

Supported Languages (18)

Code	Language	Direction
`en`	English	LTR
`es`	Spanish	LTR
`fr`	French	LTR
`de`	German	LTR
`pt`	Portuguese	LTR
`it`	Italian	LTR
`nl`	Dutch	LTR
`pl`	Polish	LTR
`ru`	Russian	LTR
`tr`	Turkish	LTR
`ja`	Japanese	LTR
`ko`	Korean	LTR
`zh-CN`	Chinese (Simplified)	LTR
`zh-TW`	Chinese (Traditional)	LTR
`ar`	Arabic	RTL
`hi`	Hindi	LTR
`th`	Thai	LTR
`vi`	Vietnamese	LTR

Implementation Files

```
apps/thinktank/lib/i18n/
  index.ts                    # Module exports
  types.ts                    # Type definitions
  localization-service.ts     # API client for localization
  localization-context.tsx    # React context provider
  translation-keys.ts         # All translation key constants
  default-translations.ts     # English fallback translations
```

API Endpoints

Method	Endpoint	Purpose
GET	`/api/localization/languages`	Get supported languages
GET	`/api/localization/bundle`	Get translation bundle for language
GET	`/api/localization/translate`	Get single translation

Usage in Components

```
import { useTranslation, T } from '@/lib/i18n';
```

```
function MyComponent() {
  const { t } = useTranslation();

  return (
    <div>
      <h1>{t(T.common.appName)}</h1>
      <p>{t(T.chat.welcomeMessage)}</p>
      <button>{t(T.common.save)}</button>
    </div>
  );
}
```

Language Selection

Users select their language in Settings → Language:

```
import { LanguageSelector } from '@/components/ui/language-selector';

<LanguageSelector variant="list" />
```

Translation Key Categories

Category	Prefix	Example
Common	`thinktank.common.`	`thinktank.common.save`
Chat	`thinktank.chat.`	`thinktank.chat.send`
Settings	`thinktank.settings.`	`thinktank.settings.language`
Rules	`thinktank.rules.`	`thinktank.rules.addRule`
History	`thinktank.history.`	`thinktank.history.title`
Artifacts	`thinktank.artifacts.`	`thinktank.artifacts.download`
Profile	`thinktank.profile.`	`thinktank.profile.achievements`
Errors	`thinktank.errors.`	`thinktank.errors.network`
Notifications	`thinktank.notifications.`	`thinktank.notifications.saved`

Adding New Translations

1. Add key to `translation-keys.ts`
2. Add default English text to `default-translations.ts`
3. Use key in component: `t(T.category.key)`
4. Register in Radiant localization registry for AI translation

Parameter Interpolation

```
// Key: "Delete {{count}} items"
t(T.history.deleteSelectedConfirm, { count: 5 })
// Output: "Delete 5 items"
```

Section 25: 2026+ Consumer App UI/UX

Overview

The Think Tank consumer app features a modern 2026+ interface with glassmorphism effects, animated transitions, and polymorphic UI morphing. Advanced features are hidden until the user activates Advanced Mode.

Design System

The design system is defined in `apps/thinktank/lib/design-system/tokens.ts`:

Token Category	Purpose
Colors	Glass effects, aurora gradients, glow colors
Spacing	Responsive spacing scale (0.5-24 rem)
Radius	Border radius from `sm` to `full`
Shadows	Glass shadows, inner glows
Animation	Timing functions, spring configs
Blur	Backdrop blur values

Glassmorphism Components

GlassCard

```
import { GlassCard } from '@/components/ui/glass-card';

<GlassCard
  variant="glow"           // default | elevated | inset | glow
  intensity="medium"       // light | medium | strong
  glowColor="violet"       // violet | fuchsia | cyan | emerald | none
  hoverEffect              // Enable hover animations
  padding="md"             // none | sm | md | lg
>
  Content
</GlassCard>
```

GlassPanel

```
import { GlassPanel } from '@/components/ui/glass-card';

<GlassPanel blur="lg">    // sm | md | lg | xl
  Content
</GlassPanel>
```

Interactive Timeline

The history page features an interactive timeline for browsing conversation history:

```
import { InteractiveTimeline, HorizontalTimeline } from '@/components/ui/timeline';
```

```
<InteractiveTimeline
  items={timelineItems}
  onSelect={(item) => navigateToConversation(item.id)}
  selectedId={currentId}
/>

<HorizontalTimeline
  items={recentItems}
  onSelect={handleSelect}
/>
```

Features: - **Grouping**: Today, Yesterday, This Week, This Month, Older - **Animations**: Entrance animations, hover effects, selection glow - **Indicators**: Favorite stars, mode badges, domain hints - **Navigation**: Horizontal scroll with arrow buttons

Polymorphic UI (ViewRouter)

The UI can morph based on task complexity and user mode:

```
import { ViewRouter } from '@/components/polymorphic';

<ViewRouter
  initialView="chat"
  initialMode="sniper"
  onViewChange={(state) => console.log('View changed:', state)}
  onEscalate={(reason) => console.log('Escalated:', reason)}
>
  {children}
</ViewRouter>
```

Execution Modes: - **Sniper**: Fast, single-model execution (~$0.01/run) - **War Room**: Deep, multi-agent analysis (~$0.50+/run)

View Types: | View | Purpose | |——|———| | `chat` | Standard conversation | | `terminal` | Command center | | `canvas` | Infinite canvas/mindmap | | `dashboard` | Analytics view | | `diff_editor` | Verification split-screen | | `decision_cards` | Human-in-the-loop |

Advanced Mode Features

Features only visible when Advanced Mode is enabled:

Feature	Location	Description
Mode selector (Sniper/War Room)	Header bar	Switch execution modes
Model selector	Input area	Choose specific AI model
Message metadata	Message bubbles	Tokens, latency, cost
Voice input	Input area	Voice-to-text
File attachments	Input area	Upload files
Escalation button	Header bar	Upgrade to War Room

Modern Polish Components (2026+)

Additional UI polish components for super-modern consumer experience:

Component	File	Purpose
`PageTransition`	`page-transition.tsx`	Fade/slide page animations
`StaggerContainer/Item`	`page-transition.tsx`	List entrance animations
`Skeleton` variants	`skeleton.tsx`	Shimmer loading states
`GradientText`	`gradient-text.tsx`	Animated gradient text
`GlowText`	`gradient-text.tsx`	Drop shadow glow effects
`AnimatedNumber`	`gradient-text.tsx`	Counter animations
`Typewriter`	`gradient-text.tsx`	Typing text effect
`TypingIndicator`	`typing-indicator.tsx`	AI thinking states
`EmptyState`	`empty-state.tsx`	Beautiful empty states
`WelcomeHero`	`empty-state.tsx`	First-time user welcome
`ModernButton`	`modern-button.tsx`	Glow/gradient buttons
`IconButton`	`modern-button.tsx`	Icon-only buttons
`PillButton`	`modern-button.tsx`	Filter pill buttons

Voice Input

Whisper-based speech-to-text for consistent cross-browser experience.

```
import { VoiceInput } from '@/components/chat';

<VoiceInput
  isOpen={isVoiceOpen}
  onClose={() => setVoiceOpen(false)}
  onTranscript={(text) => handleVoiceTranscript(text)}
/>
```

Features: - Uses app's localization language setting - Whisper API for 99+ language support - Audio level visualization - Works on all browsers (no Web Speech API dependency)

API Endpoint: `POST /api/thinktank/speech/transcribe`

File Attachments

Drag-and-drop file attachment for chat messages.

```
import { FileAttachment } from '@/components/chat';

<FileAttachment
  isOpen={isAttachOpen}
  onClose={() => setAttachOpen(false)}
  onAttach={(files) => handleFiles(files)}
  maxFiles={5}
  maxSizeMB={10}
/>
```

Supported Types: Images, PDFs, text files, JSON, code files

Liquid Interface (v5.52.7)

Morphing UI components for adaptive chat experiences. **"Don't Build the Tool. BE the Tool."**

The chat interface can morph into specialized tools when users need them. In Advanced Mode, trigger buttons appear in the header.

Morphed View Types

View	Icon	Description
`datagrid`	Table	Interactive spreadsheet with inline editing
`chart`	BarChart3	Bar, line, pie, area charts
`kanban`	Kanban	Drag-and-drop task board
`calculator`	Calculator	Full calculator with memory
`code_editor`	Code	Code editor with run capability
`document`	FileText	Rich text editor

Usage in Chat Page

```
import { LiquidMorphPanel, type MorphedViewType } from '@/components/liquid';

// State
const [morphedView, setMorphedView] = useState<MorphedViewType | null>(null);
const [isMorphFullscreen, setIsMorphFullscreen] = useState(false);
const [showMorphChat, setShowMorphChat] = useState(false);

// Trigger buttons (in header, Advanced Mode only)
<Button onClick={() => setMorphedView('datagrid')}>
  <Table className="h-4 w-4" />
</Button>

// Render panel when view is selected
{morphedView && (
  <LiquidMorphPanel
    viewType={morphedView}
    isFullscreen={isMorphFullscreen}
    onClose={() => setMorphedView(null)}
    onToggleFullscreen={() => setIsMorphFullscreen(!isMorphFullscreen)}
    onChatToggle={() => setShowMorphChat(!showMorphChat)}
    showChat={showMorphChat}
  />
)}
```

Morphed View Components Location: `apps/thinktank/components/liquid/morphed-views/`

Component	Features
`DataGridView`	Add/delete rows, inline cell editing, import/export

Component	Features
`ChartView`	Type switching (bar/line/pie/area), SVG rendering
`KanbanView`	**Multi-variant** - see Kanban Variants below
`CalculatorView`	Memory, operations, percentage, delete
`CodeEditorView`	Run code, copy, export, output panel
`DocumentView`	Bold/italic/underline, lists, alignment, export

Kanban Variants (v5.52.8) The Kanban morphed view supports 5 modern frameworks:

Variant	Description	Key Features
Standard	Traditional Kanban	Columns, cards, drag-and-drop
Scrumban	Scrum + Kanban	Sprint header, velocity, story points, WIP limits
Enterprise	Portfolio mgmt	Multi-lane hierarchical boards (Strategic/Ops/Support)
Personal	Individual productivity	Simple 3-column, strict WIP limits
Pomodoro	Timer-integrated	25-min focus, breaks, counts per task

Usage:

```
import { KanbanView, type KanbanVariant } from '@/components/liquid/morphed-views';

<KanbanView initialVariant="scrumban" />
```

Features across all variants: - Variant selector dropdown in toolbar - Analytics panel (toggle): total tasks, completed, cycle time, throughput - Card customization: priority colors, tags, subtasks, assignees - WIP limit indicators (green/amber/red)

Pomodoro-specific: - 25-minute focus timer with 5-minute breaks - Play/pause/reset controls - Completed pomodoro counter - Per-task pomodoro estimates and tracking

File Structure

```
apps/thinktank/
  lib/
     design-system/
        tokens.ts          # Design token definitions
        index.ts           # Exports
     services/
         speech-recognition.ts  # Whisper speech service
  components/
     ui/
        glass-card.tsx     # GlassCard, GlassPanel
        aurora-background.tsx  # Aurora effects
        timeline.tsx       # Timeline components
        page-transition.tsx    # Page transitions
```

```
    skeleton.tsx       # Skeleton loaders
    gradient-text.tsx  # Gradient/glow text
    typing-indicator.tsx   # Typing indicators
    empty-state.tsx    # Empty states
    modern-button.tsx  # Modern buttons
    index.ts           # UI exports
polymorphic/
    view-router.tsx    # Polymorphic router
    index.ts           # Exports
liquid/
    LiquidMorphPanel.tsx   # Morphing panel
    EjectDialog.tsx    # Export dialog
    morphed-views/     # View components (v5.52.7)
       DataGridView.tsx
       ChartView.tsx
       KanbanView.tsx
       CalculatorView.tsx
       CodeEditorView.tsx
       DocumentView.tsx
       index.ts
    index.ts           # Exports
chat/
    ModernChatInterface.tsx  # Main chat UI
    VoiceInput.tsx     # Voice input modal
    FileAttachment.tsx # File attachment modal
    index.ts           # Chat exports
```

---

25. GDPR & Compliance APIs

Path: `/compliance`
App File: `apps/admin-dashboard/app/(dashboard)/thinktank/compliance/page.tsx`

Features

- **Consent Management**: Track and manage user consents for data processing
- **GDPR Requests**: Handle data export, deletion, access, and rectification requests
- **Security Configuration**: Per-tenant security settings

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/consent`	List consent records
POST	`/api/thinktank/consent`	Record new consent
DELETE	`/api/thinktank/consent`	Withdraw consent
GET	`/api/thinktank/gdpr`	List GDPR requests
POST	`/api/thinktank/gdpr`	Create GDPR request

Method	Endpoint	Purpose
PATCH	`/api/thinktank/gdpr`	Update request status
GET	`/api/thinktank/security-config`	Get security config
PUT	`/api/thinktank/security-config`	Update security config

Implementation

- **Lambda**: `lambda/thinktank/consent.ts`, `gdpr.ts`, `security-config.ts`
- **Migration**: `174_thinktank_missing_features.sql`

---

26. User Preferences & Notifications

User Preferences

Method	Endpoint	Purpose
GET	`/api/thinktank/preferences`	Get user preferences
PUT	`/api/thinktank/preferences`	Update preferences
GET	`/api/thinktank/preferences/models`	Get model preferences
POST	`/api/thinktank/preferences/models/favorite`	Toggle favorite model

Rejection Notifications

Method	Endpoint	Purpose
GET	`/api/thinktank/rejections`	Get rejection notifications
POST	`/api/thinktank/rejections`	Create rejection
PATCH	`/api/thinktank/rejections/:id/read`	Mark as read
DELETE	`/api/thinktank/rejections/:id/dismiss`	Dismiss notification

Implementation

- **Lambda**: `lambda/thinktank/preferences.ts`, `rejections.ts`

---

27. UI Feedback & Improvement

UI Feedback

Collect user feedback on UI components and experiences.

Method	Endpoint	Purpose
GET	`/api/thinktank/ui-feedback`	List feedback
POST	`/api/thinktank/ui-feedback`	Submit feedback
PUT	`/api/thinktank/ui-feedback`	Update feedback status

UI Improvement Sessions

AI-assisted UI improvement sessions.

Method	Endpoint	Purpose
GET	`/api/thinktank/ui-improvement`	List sessions
POST	`/api/thinktank/ui-improvement/start`	Start session
POST	`/api/thinktank/ui-improvement/request`	Request improvement
POST	`/api/thinktank/ui-improvement/apply`	Apply improvement
POST	`/api/thinktank/ui-improvement/complete`	Complete session

Implementation

- **Lambda**: `lambda/thinktank/ui-feedback.ts`, `ui-improvement.ts`

---

28. Multipage Apps

User-generated multipage applications from the artifact engine.

API Endpoints

Method	Endpoint	Purpose
GET	`/api/thinktank/multipage-apps`	List user's apps
GET	`/api/thinktank/multipage-apps/:id`	Get app details
POST	`/api/thinktank/multipage-apps`	Create new app
PUT	`/api/thinktank/multipage-apps/:id`	Update app
DELETE	`/api/thinktank/multipage-apps/:id`	Delete app

App Structure

```typescript
interface MultipageApp {
  id: string;
  name: string;
  description: string;
  icon: string;
  theme: { primaryColor: string; mode: 'light' | 'dark' };
  pages: Array<{ id: string; name: string; icon: string; content: object }>;
  navigation: { type: 'tabs' | 'sidebar' | 'drawer'; position: string };
  sharedState: object;
  isPublished: boolean;
  version: number;
}
```

Implementation

- **Lambda**: `lambda/thinktank/multipage-apps.ts`

- **Migration**: `174_thinktank_missing_features.sql`

---

29. Consumer App Components (v5.24.0)

New components added to the Think Tank consumer app:

Voice Input

- Whisper-based speech recognition
- Audio level visualization
- Cross-browser support

File Attachments

- Drag-and-drop upload
- Image preview
- File type validation

Brain Plan Viewer

- AGI plan visualization
- Step progress tracking
- Mode and model display

Cato Mood Selector

- 5 moods: Balanced, Scout, Sage, Spark, Guide
- Dropdown, inline, and compact variants

Time Machine

- Video-editor style timeline
- Snapshot bookmarking
- Branch creation

Component Files

```
apps/thinktank/components/chat/
  voice-input.tsx
  file-attachments.tsx
  brain-plan-viewer.tsx
  cato-mood-selector.tsx
  time-machine.tsx
  index.ts
```

Appendix A: Application Architecture

Think Tank Admin Tech Stack

Technology	Version	Purpose
Next.js	14.x	React framework
TypeScript	5.x	Type safety
Tailwind CSS	3.x	Styling
shadcn/ui	latest	Component library
React Query	5.x	Data fetching
Lucide	latest	Icons

API Authentication

Think Tank Admin uses Cognito authentication via the `ThinkTankAuthStack`:

```
// API client configuration
const api = createApiClient({
  baseUrl: process.env.NEXT_PUBLIC_API_URL,
  getToken: () => auth.getAccessToken(),
});
```

CDK Stack

- **Stack**: `ThinkTankAdminApiStack`
- **File**: `lib/stacks/thinktank-admin-api-stack.ts`
- **Handler**: `lambda/thinktank-admin/handler.ts` (consolidated router)

Appendix B: Adding New Features

When adding features to Think Tank Admin:

1. Create page in `apps/thinktank-admin/app/(dashboard)/`
2. Add Lambda handler in `lambda/thinktank/`
3. Update consolidated handler in `lambda/thinktank/handler.ts`
4. **Update this guide** with full documentation
5. Add to CHANGELOG.md

See `/.windsurf/workflows/documentation-consolidation.md` for documentation policy.