Contents

SECTION 31: THINK TANK MODEL SELECTION & EDITABLE PRICING (v3.8.0)

NEW in v3.8.0: Users can now manually select AI models in Think Tank. All pricing is admin-editable with bulk controls and individual overrides.

31.1 Model Categories for Think Tank

Standard Models (15) - Production-Ready

ID	Display Name	Provider	Context	Input $/1M	Output $/1M	Best For
`claude-4-opus`	Claude 4 Opus	Anthropic	200K	$15.00	$75.00	Complex reasoning, analysis
`claude-4-sonnet`	Claude 4 Sonnet	Anthropic	200K	$3.00	$15.00	Quality/cost balance
`claude-3-5-haiku`	Claude 3.5 Haiku	Anthropic	200K	$0.25	$1.25	Fast responses
`gpt-4o`	GPT-4o	OpenAI	128K	$2.50	$10.00	Multimodal, reliable
`gpt-4o-mini`	GPT-4o Mini	OpenAI	128K	$0.15	$0.60	Cost-effective
`o1`	o1 Reasoning	OpenAI	200K	$15.00	$60.00	Multi-step reasoning

ID	Display Name	Provider	Context	Input $/1M	Output $/1M	Best For
gemini-2.0-pro	Gemini 2.0 Pro	Google	2M	$1.25	$5.00	Massive context
gemini-2.0-flash	Gemini 2.0 Flash	Google	1M	$0.075	$0.30	Speed, large context
grok-3	Grok 3	xAI	131K	$3.00	$15.00	Real-time info
grok-3-fast	Grok 3 Fast	xAI	131K	$1.00	$5.00	Quick responses
grok-3-mini	Grok 3 Mini	xAI	131K	$0.30	$1.50	Budget Grok
deepseek-v3	DeepSeek V3	DeepSeek	64K	$0.14	$0.28	Extremely low cost
mistral-large-2	Mistral Large 2	Mistral	128K	$2.00	$6.00	Multilingual
codestral-latest	Codestral	Mistral	32K	$0.30	$0.90	Code generation
perplexity-sonar-pro	Sonar Pro	Perplexity	128K	$3.00	$15.00	Web search

Novel Models (15) - Cutting-Edge/Experimental

ID	Display Name	Provider	Context	Input $/1M	Output $/1M	Novel Feature
o1-pro	o1 Pro	OpenAI	200K	$150.00	$600.00	Extended reasoning chains
deepseek-r1	DeepSeek R1	DeepSeek	64K	$0.55	$2.19	Open reasoning model
gemini-2.0-ultra	Gemini 2.0 Ultra	Google	2M	$5.00	$15.00	Native multimodal
gemini-2.0-pro-exp	Gemini 2.0 Pro Exp	Google	10M	$2.50	$10.00	10M token context
grok-2-vision	Grok 2 Vision	xAI	32K	$2.00	$10.00	Image understanding
grok-realtime	Grok Realtime	xAI	32K	$5.00	$20.00	Live streaming
grok-coder	Grok Coder	xAI	131K	$1.50	$7.50	Specialized code gen
grok-analyst	Grok Analyst	xAI	131K	$2.00	$10.00	Data analysis
gpt-4o-realtime	GPT-4o Realtime	OpenAI	128K	$5.00	$20.00	Voice/video streaming
claude-4-opus-agents	Claude Opus Agents	Anthropic	200K	$15.00	$75.00	Tool use, computer use
qwen-2.5-coder	Qwen 2.5 Coder	Together	128K	$0.30	$0.90	Open-source code
llama-3.3-70b	Llama 3.3 70B	Together	128K	$0.88	$0.88	Open weights
phi-4	Phi-4	Microsoft	16K	$0.07	$0.14	Small but capable

ID	Display Name	Provider	Context	Input $/1M	Output $/1M	Novel Feature
`grok-embed`	Grok Embed	xAI	8K	$0.10	-	Text embeddings
`command-r-plus`	Command R+	Cohere	128K	$2.50	$10.00	RAG optimized

31.2 Database Schema for Model Selection

```sql
-- Migration: 20241223_031_thinktank_model_selection.sql

-- Add model categorization columns
ALTER TABLE models ADD COLUMN IF NOT EXISTS is_novel BOOLEAN DEFAULT FALSE;
ALTER TABLE models ADD COLUMN IF NOT EXISTS category VARCHAR(50) DEFAULT 'general';
ALTER TABLE models ADD COLUMN IF NOT EXISTS thinktank_enabled BOOLEAN DEFAULT TRUE;
ALTER TABLE models ADD COLUMN IF NOT EXISTS thinktank_display_order INTEGER DEFAULT 100;

-- User model preferences for Think Tank
CREATE TABLE IF NOT EXISTS thinktank_user_model_preferences (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES thinktank_users(id) ON DELETE CASCADE,
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,

    -- Selection Mode: 'auto' | 'manual' | 'favorites'
    selection_mode VARCHAR(20) NOT NULL DEFAULT 'auto',

    -- Default Model (when manual mode)
    default_model_id VARCHAR(100),

    -- Favorite Models (JSON array of model IDs)
    favorite_models JSONB DEFAULT '[]'::JSONB,

    -- Category Preferences
    show_standard_models BOOLEAN DEFAULT TRUE,
    show_novel_models BOOLEAN DEFAULT TRUE,
    show_self_hosted_models BOOLEAN DEFAULT FALSE,

    -- Cost Preferences
    show_cost_per_message BOOLEAN DEFAULT TRUE,
    max_cost_per_message DECIMAL(10, 6), -- NULL = no limit
    prefer_cost_optimization BOOLEAN DEFAULT FALSE,

    -- Domain Mode Model Overrides
    -- Example: {"medical": "claude-4-opus", "code": "codestral-latest"}
    domain_mode_model_overrides JSONB DEFAULT '{}'::JSONB,

    -- Recent Models (for quick access)
    recent_models JSONB DEFAULT '[]'::JSONB,

    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(user_id)
);

CREATE INDEX idx_thinktank_model_prefs_user ON thinktank_user_model_preferences(user_id);
CREATE INDEX idx_thinktank_model_prefs_tenant ON thinktank_user_model_preferences(tenant_id);

-- Trigger for updated_at
CREATE TRIGGER update_thinktank_model_prefs_timestamp
    BEFORE UPDATE ON thinktank_user_model_preferences
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at_column();
```

31.3 Admin Editable Pricing Schema

```
-- Migration: 20241223_032_editable_pricing.sql

-- Pricing configuration table (admin-editable)
CREATE TABLE IF NOT EXISTS pricing_config (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,

    -- Global Markup Defaults
    external_default_markup DECIMAL(5, 4) DEFAULT 0.40,     -- 40%
    self_hosted_default_markup DECIMAL(5, 4) DEFAULT 0.75, -- 75%

    -- Minimum Charges
    minimum_charge_per_request DECIMAL(10, 6) DEFAULT 0.001,

    -- Grace Period for Price Increases (hours)
    price_increase_grace_period_hours INTEGER DEFAULT 24,

    -- Auto-Update Settings
    auto_update_from_providers BOOLEAN DEFAULT TRUE,
    auto_update_frequency VARCHAR(20) DEFAULT 'daily', -- 'hourly', 'daily', 'weekly'
    last_auto_update TIMESTAMPTZ,

    -- Notification Settings
    notify_on_price_change BOOLEAN DEFAULT TRUE,
    notify_threshold_percent DECIMAL(5, 2) DEFAULT 10.00, -- Notify if price changes >10%

    created_at TIMESTAMPTZ DEFAULT NOW(),
```

```sql
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(tenant_id)
);

-- Model-specific pricing overrides (admin-editable per model)
CREATE TABLE IF NOT EXISTS model_pricing_overrides (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    model_id VARCHAR(100) NOT NULL,

    -- Override Values (NULL = use defaults)
    markup_override DECIMAL(5, 4),            -- Override markup percentage
    input_price_override DECIMAL(12, 6),      -- Override input price per 1M tokens
    output_price_override DECIMAL(12, 6),     -- Override output price per 1M tokens

    -- Effective Dates (for scheduled price changes)
    effective_from TIMESTAMPTZ DEFAULT NOW(),
    effective_to TIMESTAMPTZ,

    -- Audit
    created_by UUID REFERENCES administrators(id),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(tenant_id, model_id, effective_from)
);

CREATE INDEX idx_pricing_overrides_tenant ON model_pricing_overrides(tenant_id);
CREATE INDEX idx_pricing_overrides_model ON model_pricing_overrides(model_id);
CREATE INDEX idx_pricing_overrides_effective ON model_pricing_overrides(effective_from, effect

-- Price history for auditing
CREATE TABLE IF NOT EXISTS price_history (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    model_id VARCHAR(100) NOT NULL,

    -- Previous Values
    previous_input_price DECIMAL(12, 6),
    previous_output_price DECIMAL(12, 6),
    previous_markup DECIMAL(5, 4),

    -- New Values
    new_input_price DECIMAL(12, 6),
    new_output_price DECIMAL(12, 6),
    new_markup DECIMAL(5, 4),
    -- Change Source
    change_source VARCHAR(50), -- 'admin', 'auto_sync', 'bulk_update'
    changed_by UUID REFERENCES administrators(id),

    created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_price_history_tenant_model ON price_history(tenant_id, model_id);
CREATE INDEX idx_price_history_date ON price_history(created_at);

-- View for effective pricing (combines base + overrides)
CREATE OR REPLACE VIEW effective_model_pricing AS
SELECT
    m.id AS model_id,
    m.display_name,
    m.provider_id,
    m.is_novel,
    m.category,
    m.thinktank_enabled,

    -- Base Prices
    COALESCE(m.pricing->>'input_tokens', '0')::DECIMAL AS base_input_price,
    COALESCE(m.pricing->>'output_tokens', '0')::DECIMAL AS base_output_price,

    -- Effective Markup (override > category default > global default)
    COALESCE(
        mpo.markup_override,
        CASE
            WHEN m.provider_id = 'self_hosted' THEN pc.self_hosted_default_markup
            ELSE pc.external_default_markup
        END,
        0.40
    ) AS effective_markup,

    -- Final User Prices (with markup)
    ROUND(
        COALESCE(mpo.input_price_override, COALESCE(m.pricing->>'input_tokens', '0')::DECIMAL)
        (1 + COALESCE(mpo.markup_override,
            CASE WHEN m.provider_id = 'self_hosted' THEN pc.self_hosted_default_markup ELSE pc
            0.40)),
        6
    ) AS user_input_price,

    ROUND(
        COALESCE(mpo.output_price_override, COALESCE(m.pricing->>'output_tokens', '0')::DECIMA
        (1 + COALESCE(mpo.markup_override,
            CASE WHEN m.provider_id = 'self_hosted' THEN pc.self_hosted_default_markup ELSE pc
            0.40)),
        6
    ) AS user_output_price,

    -- Override Status
    mpo.id IS NOT NULL AS has_override,
    mpo.effective_from,
    mpo.effective_to

FROM models m
LEFT JOIN pricing_config pc ON TRUE
LEFT JOIN model_pricing_overrides mpo ON m.id = mpo.model_id
    AND (mpo.effective_from <= NOW() OR mpo.effective_from IS NULL)
    AND (mpo.effective_to > NOW() OR mpo.effective_to IS NULL);
```

31.4 Admin Pricing Dashboard

```
// apps/admin-dashboard/src/app/(dashboard)/models/pricing/page.tsx

'use client';

import { useState } from 'react';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import { Card, CardContent, CardHeader, CardTitle, CardDescription } from '@/components/ui/card
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Label } from '@/components/ui/label';
import { Switch } from '@/components/ui/switch';
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';
import { Badge } from '@/components/ui/badge';
import { Slider } from '@/components/ui/slider';
import {
  Table, TableBody, TableCell, TableHead, TableHeader, TableRow
} from '@/components/ui/table';
import {
  Dialog, DialogContent, DialogHeader, DialogTitle, DialogTrigger, DialogFooter
} from '@/components/ui/dialog';
import { toast } from 'sonner';
import { Save, RefreshCw, DollarSign, Percent, History, Edit2 } from 'lucide-react';

interface PricingConfig {
  externalDefaultMarkup: number;
  selfHostedDefaultMarkup: number;
  minimumChargePerRequest: number;
  priceIncreaseGracePeriodHours: number;
  autoUpdateFromProviders: boolean;
  autoUpdateFrequency: 'hourly' | 'daily' | 'weekly';
  notifyOnPriceChange: boolean;
  notifyThresholdPercent: number;
}

interface ModelPricing {
  modelId: string;
  displayName: string;
  providerId: string;
  isNovel: boolean;
  category: string;
  baseInputPrice: number;
  baseOutputPrice: number;
  effectiveMarkup: number;
  userInputPrice: number;
  userOutputPrice: number;
  hasOverride: boolean;
}

export default function ModelPricingPage() {
  const queryClient = useQueryClient();
  const [selectedModel, setSelectedModel] = useState<ModelPricing | null>(null);
  const [bulkMarkup, setBulkMarkup] = useState({ external: 40, selfHosted: 75 });

  // Fetch pricing config
  const { data: config, isLoading: configLoading } = useQuery<PricingConfig>({
    queryKey: ['pricing-config'],
    queryFn: () => fetch('/api/admin/pricing/config').then(r => r.json()),
  });

  // Fetch all model pricing
  const { data: models, isLoading: modelsLoading } = useQuery<ModelPricing[]>({
    queryKey: ['model-pricing'],
    queryFn: () => fetch('/api/admin/pricing/models').then(r => r.json()),
  });

  // Update config mutation
  const updateConfigMutation = useMutation({
    mutationFn: (data: Partial<PricingConfig>) =>
      fetch('/api/admin/pricing/config', {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data),
      }).then(r => r.json()),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['pricing-config'] });
      toast.success('Pricing configuration updated');
    },
  });
```

```
// Bulk update markup mutation
const bulkUpdateMutation = useMutation({
  mutationFn: (data: { type: 'external' | 'self_hosted'; markup: number }) =>
    fetch('/api/admin/pricing/bulk-update', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data),
    }).then(r => r.json()),
  onSuccess: (_, variables) => {
    queryClient.invalidateQueries({ queryKey: ['model-pricing'] });
    toast.success(`All ${variables.type === 'external' ? 'external' : 'self-hosted'} models u
  },
});

// Individual model override mutation
const overrideMutation = useMutation({
  mutationFn: (data: { modelId: string; markup?: number; inputPrice?: number; outputPrice?: n
    fetch(`/api/admin/pricing/models/${data.modelId}/override`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data),
    }).then(r => r.json()),
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ['model-pricing'] });
    setSelectedModel(null);
    toast.success('Model pricing override saved');
  },
});

// Clear override mutation
const clearOverrideMutation = useMutation({
  mutationFn: (modelId: string) =>
    fetch(`/api/admin/pricing/models/${modelId}/override`, {
      method: 'DELETE',
    }).then(r => r.json()),
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ['model-pricing'] });
    toast.success('Pricing override removed');
  },
});

const [localConfig, setLocalConfig] = useState<Partial<PricingConfig>>(config || {});

return (
  <div className="space-y-6">
    <div className="flex justify-between items-center">
      <div>
```

```
        <h1 className="text-2xl font-bold">Model Pricing</h1>
        <p className="text-muted-foreground">
          Configure pricing markups and overrides for all AI models
        </p>
      </div>
      <div className="flex gap-2">
        <Button variant="outline" onClick={() => queryClient.invalidateQueries()}>
          <RefreshCw className="h-4 w-4 mr-2" />
          Refresh
        </Button>
        <Button onClick={() => updateConfigMutation.mutate(localConfig)}>
          <Save className="h-4 w-4 mr-2" />
          Save Config
        </Button>
      </div>
    </div>

    <Tabs defaultValue="config">
      <TabsList>
        <TabsTrigger value="config">Global Configuration</TabsTrigger>
        <TabsTrigger value="bulk">Bulk Updates</TabsTrigger>
        <TabsTrigger value="models">Individual Models</TabsTrigger>
        <TabsTrigger value="history">Price History</TabsTrigger>
      </TabsList>

      {/* Global Configuration Tab */}
      <TabsContent value="config" className="space-y-4">
        <div className="grid gap-4 md:grid-cols-2">
          <Card>
            <CardHeader>
              <CardTitle className="flex items-center gap-2">
                <Percent className="h-5 w-5" />
                Default Markups
              </CardTitle>
              <CardDescription>
                Global markup percentages applied to all models
              </CardDescription>
            </CardHeader>
            <CardContent className="space-y-6">
              <div className="space-y-3">
                <div className="flex justify-between">
                  <Label>External Provider Markup</Label>
                  <span className="font-mono text-sm">
                    {((localConfig.externalDefaultMarkup || config?.externalDefaultMarkup ||
                  </span>
                </div>
                <Slider
                  value={[(localConfig.externalDefaultMarkup || config?.externalDefaultMarku
                  min={0}
                  max={200}
                  step={5}
                  onValueChange={([value]) =>
                    setLocalConfig({ ...localConfig, externalDefaultMarkup: value / 100 })
                  }
                />
                <p className="text-xs text-muted-foreground">
                  Applied to OpenAI, Anthropic, Google, xAI, Mistral, etc.
                </p>
              </div>

              <div className="space-y-3">
                <div className="flex justify-between">
                  <Label>Self-Hosted Model Markup</Label>
                  <span className="font-mono text-sm">
                    {((localConfig.selfHostedDefaultMarkup || config?.selfHostedDefaultMarkup
                  </span>
                </div>
                <Slider
                  value={[(localConfig.selfHostedDefaultMarkup || config?.selfHostedDefaultMa
                  min={0}
                  max={300}
                  step={5}
                  onValueChange={([value]) =>
                    setLocalConfig({ ...localConfig, selfHostedDefaultMarkup: value / 100 })
                  }
                />
                <p className="text-xs text-muted-foreground">
                  Applied to SageMaker-hosted models (covers compute costs)
                </p>
              </div>
            </CardContent>
          </Card>

          <Card>
            <CardHeader>
              <CardTitle className="flex items-center gap-2">
                <DollarSign className="h-5 w-5" />
                Pricing Rules
              </CardTitle>
              <CardDescription>
                Additional pricing configuration
              </CardDescription>
            </CardHeader>
            <CardContent className="space-y-4">
              <div className="space-y-2">
                <Label>Minimum Charge Per Request ($)</Label>
```

```
  <Input
    type="number"
    step="0.0001"
    value={localConfig.minimumChargePerRequest || config?.minimumChargePerReque
    onChange={(e) =>
      setLocalConfig({ ...localConfig, minimumChargePerRequest: parseFloat(e.ta
    }
  />
</div>

<div className="space-y-2">
  <Label>Price Increase Grace Period (hours)</Label>
  <Input
    type="number"
    min={0}
    max={168}
    value={localConfig.priceIncreaseGracePeriodHours || config?.priceIncreaseGr
    onChange={(e) =>
      setLocalConfig({ ...localConfig, priceIncreaseGracePeriodHours: parseInt
    }
  />
  <p className="text-xs text-muted-foreground">
    Delay before price increases take effect
  </p>
</div>

<div className="flex items-center justify-between">
  <div>
    <Label>Auto-Update from Providers</Label>
    <p className="text-xs text-muted-foreground">
      Automatically sync base prices from provider APIs
    </p>
  </div>
  <Switch
    checked={localConfig.autoUpdateFromProviders ?? config?.autoUpdateFromProvi
    onCheckedChange={(checked) =>
      setLocalConfig({ ...localConfig, autoUpdateFromProviders: checked })
    }
  />
</div>

<div className="flex items-center justify-between">
  <div>
    <Label>Notify on Price Changes</Label>
    <p className="text-xs text-muted-foreground">
      Alert when provider prices change significantly
    </p>
  </div>
```

```
            <Switch
              checked={localConfig.notifyOnPriceChange ?? config?.notifyOnPriceChange ??
              onCheckedChange={(checked) =>
                setLocalConfig({ ...localConfig, notifyOnPriceChange: checked })
              }
            />
          </div>
        </CardContent>
      </Card>
    </div>
  </TabsContent>

  {/* Bulk Updates Tab */}
  <TabsContent value="bulk" className="space-y-4">
    <div className="grid gap-4 md:grid-cols-2">
      <Card>
        <CardHeader>
          <CardTitle>External Providers</CardTitle>
          <CardDescription>
            Update markup for all external AI providers at once
          </CardDescription>
        </CardHeader>
        <CardContent className="space-y-4">
          <div className="space-y-3">
            <div className="flex justify-between">
              <Label>Markup Percentage</Label>
              <span className="font-mono text-sm">{bulkMarkup.external}%</span>
            </div>
            <Slider
              value={[bulkMarkup.external]}
              min={0}
              max={200}
              step={5}
              onValueChange={([value]) => setBulkMarkup({ ...bulkMarkup, external: value
            />
          </div>
          <Button
            className="w-full"
            onClick={() => bulkUpdateMutation.mutate({ type: 'external', markup: bulkMar
            disabled={bulkUpdateMutation.isPending}
          >
            Apply to All External Models
          </Button>
          <p className="text-xs text-muted-foreground">
            Affects: OpenAI, Anthropic, Google, xAI, Mistral, Perplexity, DeepSeek, Cohe
          </p>
        </CardContent>
      </Card>
```

```
        <Card>
          <CardHeader>
            <CardTitle>Self-Hosted Models</CardTitle>
            <CardDescription>
              Update markup for all SageMaker-hosted models at once
            </CardDescription>
          </CardHeader>
          <CardContent className="space-y-4">
            <div className="space-y-3">
              <div className="flex justify-between">
                <Label>Markup Percentage</Label>
                <span className="font-mono text-sm">{bulkMarkup.selfHosted}%</span>
              </div>
              <Slider
                value={[bulkMarkup.selfHosted]}
                min={0}
                max={300}
                step={5}
                onValueChange={([value]) => setBulkMarkup({ ...bulkMarkup, selfHosted: valu
              />
            </div>
            <Button
              className="w-full"
              onClick={() => bulkUpdateMutation.mutate({ type: 'self_hosted', markup: bulkM
              disabled={bulkUpdateMutation.isPending}
            >
              Apply to All Self-Hosted Models
            </Button>
            <p className="text-xs text-muted-foreground">
              Affects: Stable Diffusion, Whisper, SAM 2, YOLO, MusicGen, etc.
            </p>
          </CardContent>
        </Card>
      </div>
    </TabsContent>

    {/* Individual Models Tab */}
    <TabsContent value="models">
      <Card>
        <CardHeader>
          <CardTitle>Model Pricing Details</CardTitle>
          <CardDescription>
            View and override pricing for individual models
          </CardDescription>
        </CardHeader>
        <CardContent>
          <Table>
```

```
<TableHeader>
  <TableRow>
    <TableHead>Model</TableHead>
    <TableHead>Provider</TableHead>
    <TableHead>Category</TableHead>
    <TableHead className="text-right">Base Input</TableHead>
    <TableHead className="text-right">Base Output</TableHead>
    <TableHead className="text-right">Markup</TableHead>
    <TableHead className="text-right">User Input</TableHead>
    <TableHead className="text-right">User Output</TableHead>
    <TableHead>Override</TableHead>
    <TableHead></TableHead>
  </TableRow>
</TableHeader>
<TableBody>
  {(models || []).map((model) => (
    <TableRow key={model.modelId}>
      <TableCell>
        <div className="font-medium">{model.displayName}</div>
        <div className="text-xs text-muted-foreground font-mono">{model.modelI
      </TableCell>
      <TableCell>{model.providerId}</TableCell>
      <TableCell>
        <Badge variant={model.isNovel ? 'secondary' : 'outline'}>
          {model.isNovel ? 'Novel' : 'Standard'}
        </Badge>
      </TableCell>
      <TableCell className="text-right font-mono">
        ${model.baseInputPrice.toFixed(2)}
      </TableCell>
      <TableCell className="text-right font-mono">
        ${model.baseOutputPrice.toFixed(2)}
      </TableCell>
      <TableCell className="text-right font-mono">
        {(model.effectiveMarkup * 100).toFixed(0)}%
      </TableCell>
      <TableCell className="text-right font-mono text-green-600">
        ${model.userInputPrice.toFixed(2)}
      </TableCell>
      <TableCell className="text-right font-mono text-green-600">
        ${model.userOutputPrice.toFixed(2)}
      </TableCell>
      <TableCell>
        {model.hasOverride ? (
          <Badge variant="default">Custom</Badge>
        ) : (
          <Badge variant="outline">Default</Badge>
        )}
```

```
                      </TableCell>
                      <TableCell>
                        <Dialog>
                          <DialogTrigger asChild>
                            <Button variant="ghost" size="sm" onClick={() => setSelectedModel(r
                              <Edit2 className="h-4 w-4" />
                            </Button>
                          </DialogTrigger>
                          <DialogContent>
                            <DialogHeader>
                              <DialogTitle>Edit Pricing: {model.displayName}</DialogTitle>
                            </DialogHeader>
                            <ModelPricingEditor
                              model={model}
                              onSave={(data) => overrideMutation.mutate({ modelId: model.modelI
                              onClear={() => clearOverrideMutation.mutate(model.modelId)}
                            />
                          </DialogContent>
                        </Dialog>
                      </TableCell>
                    </TableRow>
                  ))}
                </TableBody>
              </Table>
            </CardContent>
          </Card>
        </TabsContent>

        {/* Price History Tab */}
        <TabsContent value="history">
          <PriceHistoryTable />
        </TabsContent>
      </Tabs>
    </div>
  );
}

// Model Pricing Editor Component
function ModelPricingEditor({
  model,
  onSave,
  onClear
}: {
  model: ModelPricing;
  onSave: (data: any) => void;
  onClear: () => void;
}) {
  const [markup, setMarkup] = useState(model.effectiveMarkup * 100);
  const [inputPrice, setInputPrice] = useState<number | null>(null);
  const [outputPrice, setOutputPrice] = useState<number | null>(null);

  return (
    <div className="space-y-4">
      <div className="space-y-2">
        <Label>Custom Markup (%)</Label>
        <div className="flex items-center gap-4">
          <Slider
            value={[markup]}
            min={0}
            max={200}
            step={5}
            onValueChange={([value]) => setMarkup(value)}
            className="flex-1"
          />
          <span className="font-mono w-16 text-right">{markup}%</span>
        </div>
      </div>

      <div className="grid grid-cols-2 gap-4">
        <div className="space-y-2">
          <Label>Override Input Price ($/1M tokens)</Label>
          <Input
            type="number"
            step="0.01"
            placeholder={`Default: $${model.baseInputPrice.toFixed(2)}`}
            value={inputPrice ?? ''}
            onChange={(e) => setInputPrice(e.target.value ? parseFloat(e.target.value) : null)
          />
        </div>
        <div className="space-y-2">
          <Label>Override Output Price ($/1M tokens)</Label>
          <Input
            type="number"
            step="0.01"
            placeholder={`Default: $${model.baseOutputPrice.toFixed(2)}`}
            value={outputPrice ?? ''}
            onChange={(e) => setOutputPrice(e.target.value ? parseFloat(e.target.value) : null
          />
        </div>
      </div>

      <div className="bg-muted p-3 rounded-lg">
        <div className="text-sm font-medium mb-2">Preview User Prices</div>
        <div className="grid grid-cols-2 gap-4 text-sm">
          <div>
            Input: <span className="font-mono text-green-600">
```

```
              ${((inputPrice ?? model.baseInputPrice) * (1 + markup / 100)).toFixed(2)}/1M
            </span>
          </div>
          <div>
            Output: <span className="font-mono text-green-600">
              ${((outputPrice ?? model.baseOutputPrice) * (1 + markup / 100)).toFixed(2)}/1M
            </span>
          </div>
        </div>
      </div>

      <DialogFooter className="flex justify-between">
        {model.hasOverride && (
          <Button variant="outline" onClick={onClear}>
            Clear Override
          </Button>
        )}
        <Button onClick={() => onSave({ markup: markup / 100, inputPrice, outputPrice })}>
          Save Override
        </Button>
      </DialogFooter>
    </div>
  );
}

// Price History Table Component
function PriceHistoryTable() {
  const { data: history } = useQuery({
    queryKey: ['price-history'],
    queryFn: () => fetch('/api/admin/pricing/history?limit=100').then(r => r.json()),
  });

  return (
    <Card>
      <CardHeader>
        <CardTitle className="flex items-center gap-2">
          <History className="h-5 w-5" />
          Price Change History
        </CardTitle>
      </CardHeader>
      <CardContent>
        <Table>
          <TableHeader>
            <TableRow>
              <TableHead>Date</TableHead>
              <TableHead>Model</TableHead>
              <TableHead>Change Type</TableHead>
              <TableHead className="text-right">Previous</TableHead>
              <TableHead className="text-right">New</TableHead>
              <TableHead>Changed By</TableHead>
            </TableRow>
          </TableHeader>
          <TableBody>
            {(history || []).map((entry: any) => (
              <TableRow key={entry.id}>
                <TableCell className="text-sm">
                  {new Date(entry.createdAt).toLocaleString()}
                </TableCell>
                <TableCell className="font-mono text-sm">{entry.modelId}</TableCell>
                <TableCell>
                  <Badge variant="outline">{entry.changeSource}</Badge>
                </TableCell>
                <TableCell className="text-right font-mono text-sm">
                  {entry.previousMarkup ? `${(entry.previousMarkup * 100).toFixed(0)}%` : '-'}
                </TableCell>
                <TableCell className="text-right font-mono text-sm">
                  {entry.newMarkup ? `${(entry.newMarkup * 100).toFixed(0)}%` : '-'}
                </TableCell>
                <TableCell className="text-sm">{entry.changedByEmail || 'System'}</TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </CardContent>
    </Card>
  );
}
```

31.5 Think Tank Model Selection UI

```
// apps/thinktank/src/components/chat/model-selector.tsx

'use client';

import { useState, useMemo } from 'react';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import {
  Popover, PopoverContent, PopoverTrigger
} from '@/components/ui/popover';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Badge } from '@/components/ui/badge';
import { Switch } from '@/components/ui/switch';
import { Label } from '@/components/ui/label';
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';
import { ScrollArea } from '@/components/ui/scroll-area';
import { cn } from '@/lib/utils';
import {
  ChevronDown, Search, Star, StarOff, Sparkles, Zap,
  DollarSign, Brain, Check, Settings2
} from 'lucide-react';

interface Model {
  id: string;
  displayName: string;
  providerId: string;
  providerName: string;
  isNovel: boolean;
  category: string;
  contextWindow: number;
  capabilities: string[];
  userInputPrice: number;
  userOutputPrice: number;
  isFavorite?: boolean;
}

interface ModelPreferences {
  selectionMode: 'auto' | 'manual' | 'favorites';
  defaultModelId?: string;
  favoriteModels: string[];
  showStandardModels: boolean;
  showNovelModels: boolean;
  showCostPerMessage: boolean;
  maxCostPerMessage?: number;
}

interface ModelSelectorProps {
  selectedModel: string | null; // null = Auto
  onModelChange: (modelId: string | null) => void;
  disabled?: boolean;
}

export function ModelSelector({ selectedModel, onModelChange, disabled }: ModelSelectorProps)
  const [open, setOpen] = useState(false);
  const [search, setSearch] = useState('');
  const queryClient = useQueryClient();

  // Fetch available models
  const { data: models = [] } = useQuery<Model[]>({
    queryKey: ['thinktank-models'],
    queryFn: () => fetch('/api/thinktank/models').then(r => r.json()),
  });
```

```
// Fetch user preferences
const { data: preferences } = useQuery<ModelPreferences>({
  queryKey: ['thinktank-model-preferences'],
  queryFn: () => fetch('/api/thinktank/preferences/models').then(r => r.json()),
});

// Toggle favorite mutation
const toggleFavoriteMutation = useMutation({
  mutationFn: (modelId: string) =>
    fetch('/api/thinktank/preferences/models/favorite', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ modelId }),
    }).then(r => r.json()),
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ['thinktank-model-preferences'] });
    queryClient.invalidateQueries({ queryKey: ['thinktank-models'] });
  },
});

// Filter and group models
const { standardModels, novelModels, favoriteModels } = useMemo(() => {
  const filtered = models.filter(m =>
    m.displayName.toLowerCase().includes(search.toLowerCase()) ||
    m.providerId.toLowerCase().includes(search.toLowerCase())
  );

  return {
    standardModels: filtered.filter(m => !m.isNovel && preferences?.showStandardModels !== fa
    novelModels: filtered.filter(m => m.isNovel && preferences?.showNovelModels !== false),
    favoriteModels: filtered.filter(m => preferences?.favoriteModels?.includes(m.id)),
  };
}, [models, search, preferences]);

// Get selected model details
const selectedModelDetails = selectedModel
  ? models.find(m => m.id === selectedModel)
  : null;

// Format price for display
const formatPrice = (inputPrice: number, outputPrice: number) => {
  const avgPrice = (inputPrice + outputPrice) / 2;
  if (avgPrice < 1) return `$${avgPrice.toFixed(3)}/1K`;
  return `$${avgPrice.toFixed(2)}/1M`;
};

return (
  <Popover open={open} onOpenChange={setOpen}>
    <PopoverTrigger asChild>
      <Button
        variant="outline"
        role="combobox"
        aria-expanded={open}
        disabled={disabled}
        className="justify-between min-w-[200px]"
      >
        <div className="flex items-center gap-2">
          {selectedModel === null ? (
            <>
              <Brain className="h-4 w-4 text-purple-500" />
              <span>Auto</span>
              <Badge variant="secondary" className="text-xs">RADIANT Brain</Badge>
            </>
          ) : (
            <>
              {selectedModelDetails?.isNovel && <Sparkles className="h-4 w-4 text-amber-500"
              <span>{selectedModelDetails?.displayName || selectedModel}</span>
              {preferences?.showCostPerMessage && selectedModelDetails && (
                <span className="text-xs text-muted-foreground">
                  {formatPrice(selectedModelDetails.userInputPrice, selectedModelDetails.use
                </span>
              )}
            </>
          )}
        </div>
        <ChevronDown className="ml-2 h-4 w-4 shrink-0 opacity-50" />
      </Button>
    </PopoverTrigger>

    <PopoverContent className="w-[400px] p-0" align="start">
      <div className="p-3 border-b">
        <div className="relative">
          <Search className="absolute left-3 top-1/2 -translate-y-1/2 h-4 w-4 text-muted-fore
          <Input
            placeholder="Search models..."
            value={search}
            onChange={(e) => setSearch(e.target.value)}
            className="pl-9"
          />
        </div>
      </div>

      <Tabs defaultValue="all" className="w-full">
        <TabsList className="w-full justify-start rounded-none border-b bg-transparent p-0">
          <TabsTrigger value="all" className="rounded-none data-[state=active]:border-b-2">
            All
          </TabsTrigger>
          <TabsTrigger value="favorites" className="rounded-none data-[state=active]:border-l
            <Star className="h-3 w-3 mr-1" />
            Favorites
          </TabsTrigger>
          <TabsTrigger value="standard" className="rounded-none data-[state=active]:border-b-
            Standard
          </TabsTrigger>
          <TabsTrigger value="novel" className="rounded-none data-[state=active]:border-b-2">
            <Sparkles className="h-3 w-3 mr-1" />
            Novel
          </TabsTrigger>
        </TabsList>

        <ScrollArea className="h-[300px]">
          {/* Auto Option */}
          <div className="p-1">
            <ModelOption
              model={null}
              isSelected={selectedModel === null}
              onSelect={() => {
                onModelChange(null);
                setOpen(false);
              }}
              showCost={false}
            />
          </div>

          <TabsContent value="all" className="m-0 p-1">
            {favoriteModels.length > 0 && (
              <div className="mb-2">
                <div className="px-2 py-1 text-xs font-medium text-muted-foreground">Favorite
                {favoriteModels.map(model => (
                  <ModelOption
                    key={model.id}
                    model={model}
                    isSelected={selectedModel === model.id}
                    isFavorite={true}
                    onSelect={() => {
                      onModelChange(model.id);
                      setOpen(false);
                    }}
                    onToggleFavorite={() => toggleFavoriteMutation.mutate(model.id)}
                    showCost={preferences?.showCostPerMessage}
                  />
                ))}
              </div>
```

```
  )}

  {standardModels.length > 0 && (
    <div className="mb-2">
      <div className="px-2 py-1 text-xs font-medium text-muted-foreground">Standard
      {standardModels.map(model => (
        <ModelOption
          key={model.id}
          model={model}
          isSelected={selectedModel === model.id}
          isFavorite={preferences?.favoriteModels?.includes(model.id)}
          onSelect={() => {
            onModelChange(model.id);
            setOpen(false);
          }}
          onToggleFavorite={() => toggleFavoriteMutation.mutate(model.id)}
          showCost={preferences?.showCostPerMessage}
        />
      ))}
    </div>
  )}

  {novelModels.length > 0 && (
    <div>
      <div className="px-2 py-1 text-xs font-medium text-muted-foreground flex item
        <Sparkles className="h-3 w-3" />
        Novel / Experimental
      </div>
      {novelModels.map(model => (
        <ModelOption
          key={model.id}
          model={model}
          isSelected={selectedModel === model.id}
          isFavorite={preferences?.favoriteModels?.includes(model.id)}
          onSelect={() => {
            onModelChange(model.id);
            setOpen(false);
          }}
          onToggleFavorite={() => toggleFavoriteMutation.mutate(model.id)}
          showCost={preferences?.showCostPerMessage}
        />
      ))}
    </div>
  )}
</TabsContent>

<TabsContent value="favorites" className="m-0 p-1">
  {favoriteModels.length === 0 ? (
    <div className="p-4 text-center text-sm text-muted-foreground">
      No favorite models yet. Star models to add them here.
    </div>
  ) : (
    favoriteModels.map(model => (
      <ModelOption
        key={model.id}
        model={model}
        isSelected={selectedModel === model.id}
        isFavorite={true}
        onSelect={() => {
          onModelChange(model.id);
          setOpen(false);
        }}
        onToggleFavorite={() => toggleFavoriteMutation.mutate(model.id)}
        showCost={preferences?.showCostPerMessage}
      />
    ))
  )}
</TabsContent>

<TabsContent value="standard" className="m-0 p-1">
  {standardModels.map(model => (
    <ModelOption
      key={model.id}
      model={model}
      isSelected={selectedModel === model.id}
      isFavorite={preferences?.favoriteModels?.includes(model.id)}
      onSelect={() => {
        onModelChange(model.id);
        setOpen(false);
      }}
      onToggleFavorite={() => toggleFavoriteMutation.mutate(model.id)}
      showCost={preferences?.showCostPerMessage}
    />
  ))}
</TabsContent>

<TabsContent value="novel" className="m-0 p-1">
  {novelModels.map(model => (
    <ModelOption
      key={model.id}
      model={model}
      isSelected={selectedModel === model.id}
      isFavorite={preferences?.favoriteModels?.includes(model.id)}
      onSelect={() => {
        onModelChange(model.id);
        setOpen(false);
```

```
                  }}
                  onToggleFavorite={() => toggleFavoriteMutation.mutate(model.id)}
                  showCost={preferences?.showCostPerMessage}
                />
              ))}
            </TabsContent>
          </ScrollArea>
        </Tabs>

        {/* Settings Footer */}
        <div className="border-t p-2 flex justify-between items-center">
          <div className="flex items-center gap-2 text-xs text-muted-foreground">
            <DollarSign className="h-3 w-3" />
            <span>Prices per 1M tokens</span>
          </div>
          <Button variant="ghost" size="sm" className="text-xs">
            <Settings2 className="h-3 w-3 mr-1" />
            Preferences
          </Button>
        </div>
      </PopoverContent>
    </Popover>
  );
}

// Individual Model Option Component
function ModelOption({
  model,
  isSelected,
  isFavorite,
  onSelect,
  onToggleFavorite,
  showCost,
}: {
  model: Model | null;
  isSelected: boolean;
  isFavorite?: boolean;
  onSelect: () => void;
  onToggleFavorite?: () => void;
  showCost?: boolean;
}) {
  // Auto option
  if (model === null) {
    return (
      <div
        className={cn(
          "flex items-center gap-3 p-2 rounded-md cursor-pointer hover:bg-accent",
          isSelected && "bg-accent"
        )}
        onClick={onSelect}
      >
        <Brain className="h-5 w-5 text-purple-500" />
        <div className="flex-1">
          <div className="flex items-center gap-2">
            <span className="font-medium">Auto</span>
            <Badge variant="secondary" className="text-xs">RADIANT Brain</Badge>
          </div>
          <div className="text-xs text-muted-foreground">
            Intelligently selects the best model for your task
          </div>
        </div>
        {isSelected && <Check className="h-4 w-4 text-primary" />}
      </div>
    );
  }

  return (
    <div
      className={cn(
        "flex items-center gap-3 p-2 rounded-md cursor-pointer hover:bg-accent group",
        isSelected && "bg-accent"
      )}
      onClick={onSelect}
    >
      <div className="flex-shrink-0">
        {model.isNovel ? (
          <Sparkles className="h-5 w-5 text-amber-500" />
        ) : (
          <Zap className="h-5 w-5 text-blue-500" />
        )}
      </div>

      <div className="flex-1 min-w-0">
        <div className="flex items-center gap-2">
          <span className="font-medium truncate">{model.displayName}</span>
          {model.isNovel && (
            <Badge variant="outline" className="text-xs text-amber-600 border-amber-300">
              Novel
            </Badge>
          )}
        </div>
        <div className="flex items-center gap-2 text-xs text-muted-foreground">
          <span>{model.providerName}</span>
          <span>•</span>
          <span>{(model.contextWindow / 1000).toFixed(0)}K context</span>
          {showCost && (
            <>
              <span>•</span>
              <span className="text-green-600">
                ${((model.userInputPrice + model.userOutputPrice) / 2).toFixed(2)}/1M
              </span>
            </>
          )}
        </div>
      </div>

      <div className="flex items-center gap-1">
        {onToggleFavorite && (
          <button
            className="p-1 opacity-0 group-hover:opacity-100 transition-opacity"
            onClick={(e) => {
              e.stopPropagation();
              onToggleFavorite();
            }}
          >
            {isFavorite ? (
              <Star className="h-4 w-4 text-yellow-500 fill-yellow-500" />
            ) : (
              <StarOff className="h-4 w-4 text-muted-foreground" />
            )}
          </button>
        )}
        {isSelected && <Check className="h-4 w-4 text-primary" />}
      </div>
    </div>
  );
}
```

31.6 Think Tank Model API Endpoints

```
// packages/lambdas/src/handlers/thinktank/models.ts

import { APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
import { Pool } from 'pg';
import { createLogger, corsHeaders } from '@radiant/shared';
import { verifyJWT, getUserFromToken } from '../auth/jwt';

const pool = new Pool({ connectionString: process.env.DATABASE_URL });
const logger = createLogger('thinktank-models');

// GET /api/thinktank/models - List available models for Think Tank users
export async function listModels(event: APIGatewayProxyEvent): Promise<APIGatewayProxyResult>
```

```
try {
  const user = await getUserFromToken(event);
  const client = await pool.connect();

  try {
    // Get models with effective pricing and user favorites
    const result = await client.query(`
      SELECT
        m.id,
        m.display_name,
        m.provider_id,
        p.display_name as provider_name,
        m.is_novel,
        m.category,
        m.context_window,
        m.capabilities,
        m.thinktank_enabled,

        -- Effective pricing (from view)
        emp.user_input_price,
        emp.user_output_price,
        emp.effective_markup,

        -- User favorites
        $1 = ANY(COALESCE(ump.favorite_models, '[]')::text[]) as is_favorite

      FROM models m
      JOIN providers p ON m.provider_id = p.id
      LEFT JOIN effective_model_pricing emp ON m.id = emp.model_id
      LEFT JOIN thinktank_user_model_preferences ump ON ump.user_id = $2

      WHERE m.thinktank_enabled = true
        AND m.is_enabled = true
        AND m.status = 'active'

      ORDER BY
        m.is_novel ASC,
        m.thinktank_display_order ASC,
        m.display_name ASC
    `, [user.id, user.id]);

    return {
      statusCode: 200,
      headers: corsHeaders,
      body: JSON.stringify(result.rows.map(row => ({
        id: row.id,
        displayName: row.display_name,
        providerId: row.provider_id,
          providerName: row.provider_name,
          isNovel: row.is_novel,
          category: row.category,
          contextWindow: row.context_window,
          capabilities: row.capabilities || [],
          userInputPrice: parseFloat(row.user_input_price) || 0,
          userOutputPrice: parseFloat(row.user_output_price) || 0,
          isFavorite: row.is_favorite,
        })),
      };
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to list models', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to load models' }),
    };
  }
}

// GET /api/thinktank/preferences/models - Get user's model preferences
export async function getModelPreferences(event: APIGatewayProxyEvent): Promise<APIGatewayProxy
  try {
    const user = await getUserFromToken(event);
    const client = await pool.connect();

    try {
      const result = await client.query(`
        SELECT
          selection_mode,
          default_model_id,
          favorite_models,
          show_standard_models,
          show_novel_models,
          show_self_hosted_models,
          show_cost_per_message,
          max_cost_per_message,
          prefer_cost_optimization,
          domain_mode_model_overrides,
          recent_models
        FROM thinktank_user_model_preferences
        WHERE user_id = $1
      `, [user.id]);

      if (result.rows.length === 0) {
        // Return defaults
        return {
          statusCode: 200,
          headers: corsHeaders,
          body: JSON.stringify({
            selectionMode: 'auto',
            defaultModelId: null,
            favoriteModels: [],
            showStandardModels: true,
            showNovelModels: true,
            showSelfHostedModels: false,
            showCostPerMessage: true,
            maxCostPerMessage: null,
            preferCostOptimization: false,
            domainModeModelOverrides: {},
            recentModels: [],
          }),
        };
      }

      const row = result.rows[0];
      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({
          selectionMode: row.selection_mode,
          defaultModelId: row.default_model_id,
          favoriteModels: row.favorite_models || [],
          showStandardModels: row.show_standard_models,
          showNovelModels: row.show_novel_models,
          showSelfHostedModels: row.show_self_hosted_models,
          showCostPerMessage: row.show_cost_per_message,
          maxCostPerMessage: row.max_cost_per_message ? parseFloat(row.max_cost_per_message) :
          preferCostOptimization: row.prefer_cost_optimization,
          domainModeModelOverrides: row.domain_mode_model_overrides || {},
          recentModels: row.recent_models || [],
        }),
      };
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to get model preferences', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to load preferences' }),
    };
  }
}

// PUT /api/thinktank/preferences/models - Update user's model preferences
export async function updateModelPreferences(event: APIGatewayProxyEvent): Promise<APIGatewayP
  try {
    const user = await getUserFromToken(event);
    const body = JSON.parse(event.body || '{}');
    const client = await pool.connect();

    try {
      await client.query(`
        INSERT INTO thinktank_user_model_preferences (
          user_id, tenant_id, selection_mode, default_model_id, favorite_models,
          show_standard_models, show_novel_models, show_self_hosted_models,
          show_cost_per_message, max_cost_per_message, prefer_cost_optimization,
          domain_mode_model_overrides
        ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12)
        ON CONFLICT (user_id) DO UPDATE SET
          selection_mode = COALESCE($3, thinktank_user_model_preferences.selection_mode),
          default_model_id = COALESCE($4, thinktank_user_model_preferences.default_model_id),
          favorite_models = COALESCE($5, thinktank_user_model_preferences.favorite_models),
          show_standard_models = COALESCE($6, thinktank_user_model_preferences.show_standard_mo
          show_novel_models = COALESCE($7, thinktank_user_model_preferences.show_novel_models)
          show_self_hosted_models = COALESCE($8, thinktank_user_model_preferences.show_self_hos
          show_cost_per_message = COALESCE($9, thinktank_user_model_preferences.show_cost_per_m
          max_cost_per_message = $10,
          prefer_cost_optimization = COALESCE($11, thinktank_user_model_preferences.prefer_cost
          domain_mode_model_overrides = COALESCE($12, thinktank_user_model_preferences.domain_m
          updated_at = NOW()
      `, [
        user.id,
        user.tenantId,
        body.selectionMode,
        body.defaultModelId,
        JSON.stringify(body.favoriteModels || []),
        body.showStandardModels,
        body.showNovelModels,
        body.showSelfHostedModels,
        body.showCostPerMessage,
        body.maxCostPerMessage,
        body.preferCostOptimization,
        JSON.stringify(body.domainModeModelOverrides || {}),
      ]);

      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({ success: true }),
      };
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to update model preferences', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to save preferences' }),
    };
  }
}

// POST /api/thinktank/preferences/models/favorite - Toggle favorite model
export async function toggleFavoriteModel(event: APIGatewayProxyEvent): Promise<APIGatewayProxy
  try {
    const user = await getUserFromToken(event);
    const { modelId } = JSON.parse(event.body || '{}');

    if (!modelId) {
      return {
        statusCode: 400,
        headers: corsHeaders,
        body: JSON.stringify({ error: 'modelId is required' }),
      };
    }

    const client = await pool.connect();

    try {
      // Check if model is currently a favorite
      const currentResult = await client.query(`
        SELECT favorite_models FROM thinktank_user_model_preferences WHERE user_id = $1
      `, [user.id]);

      let favorites: string[] = currentResult.rows[0]?.favorite_models || [];

      if (favorites.includes(modelId)) {
        // Remove from favorites
        favorites = favorites.filter(id => id !== modelId);
      } else {
        // Add to favorites
        favorites.push(modelId);
      }

      // Update or insert
      await client.query(`
        INSERT INTO thinktank_user_model_preferences (user_id, tenant_id, favorite_models)
        VALUES ($1, $2, $3)
        ON CONFLICT (user_id) DO UPDATE SET
          favorite_models = $3,
          updated_at = NOW()
      `, [user.id, user.tenantId, JSON.stringify(favorites)]);

      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({
          success: true,
          isFavorite: favorites.includes(modelId),
          favoriteModels: favorites,
        }),
      };
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to toggle favorite', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to update favorites' }),
    };
  }
}
```

31.7 Admin Pricing API Endpoints

```
// packages/lambdas/src/handlers/admin/pricing.ts

import { APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
import { Pool } from 'pg';
import { createLogger, corsHeaders } from '@radiant/shared';
import { requireAdmin } from '../auth/admin';

const pool = new Pool({ connectionString: process.env.DATABASE_URL });
const logger = createLogger('admin-pricing');

// GET /api/admin/pricing/config
export async function getPricingConfig(event: APIGatewayProxyEvent): Promise<APIGatewayProxyRes
  try {
    await requireAdmin(event);
    const client = await pool.connect();

    try {
      const result = await client.query(`
        SELECT * FROM pricing_config LIMIT 1
      `);

      if (result.rows.length === 0) {
        // Return defaults
        return {
          statusCode: 200,
          headers: corsHeaders,
          body: JSON.stringify({
            externalDefaultMarkup: 0.40,
            selfHostedDefaultMarkup: 0.75,
            minimumChargePerRequest: 0.001,
            priceIncreaseGracePeriodHours: 24,
            autoUpdateFromProviders: true,
            autoUpdateFrequency: 'daily',
            notifyOnPriceChange: true,
            notifyThresholdPercent: 10,
          }),
        };
      }

      const row = result.rows[0];
      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({
          externalDefaultMarkup: parseFloat(row.external_default_markup),
          selfHostedDefaultMarkup: parseFloat(row.self_hosted_default_markup),
          minimumChargePerRequest: parseFloat(row.minimum_charge_per_request),
          priceIncreaseGracePeriodHours: row.price_increase_grace_period_hours,
          autoUpdateFromProviders: row.auto_update_from_providers,
          autoUpdateFrequency: row.auto_update_frequency,
          lastAutoUpdate: row.last_auto_update,
          notifyOnPriceChange: row.notify_on_price_change,
          notifyThresholdPercent: parseFloat(row.notify_threshold_percent),
        }),
      };
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to get pricing config', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to load pricing config' }),
    };
  }
}

// PUT /api/admin/pricing/config
export async function updatePricingConfig(event: APIGatewayProxyEvent): Promise<APIGatewayProx
  try {
    const admin = await requireAdmin(event);
    const body = JSON.parse(event.body || '{}');
    const client = await pool.connect();

    try {
      await client.query(`
        INSERT INTO pricing_config (
          tenant_id, external_default_markup, self_hosted_default_markup,
          minimum_charge_per_request, price_increase_grace_period_hours,
          auto_update_from_providers, auto_update_frequency,
          notify_on_price_change, notify_threshold_percent
        ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)
        ON CONFLICT (tenant_id) DO UPDATE SET
          external_default_markup = COALESCE($2, pricing_config.external_default_markup),
          self_hosted_default_markup = COALESCE($3, pricing_config.self_hosted_default_markup)
          minimum_charge_per_request = COALESCE($4, pricing_config.minimum_charge_per_request)
          price_increase_grace_period_hours = COALESCE($5, pricing_config.price_increase_grace
          auto_update_from_providers = COALESCE($6, pricing_config.auto_update_from_providers)
          auto_update_frequency = COALESCE($7, pricing_config.auto_update_frequency),
          notify_on_price_change = COALESCE($8, pricing_config.notify_on_price_change),
          notify_threshold_percent = COALESCE($9, pricing_config.notify_threshold_percent),
          updated_at = NOW()
      `, [
        admin.tenantId,
        body.externalDefaultMarkup,
        body.selfHostedDefaultMarkup,
        body.minimumChargePerRequest,
        body.priceIncreaseGracePeriodHours,
        body.autoUpdateFromProviders,
        body.autoUpdateFrequency,
        body.notifyOnPriceChange,
        body.notifyThresholdPercent,
      ]);

      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({ success: true }),
      };
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to update pricing config', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to save config' }),
    };
  }
}

// POST /api/admin/pricing/bulk-update - Bulk update markups
export async function bulkUpdatePricing(event: APIGatewayProxyEvent): Promise<APIGatewayProxyRe
  try {
    const admin = await requireAdmin(event);
    const { type, markup } = JSON.parse(event.body || '{}');

    if (!type || markup === undefined) {
      return {
        statusCode: 400,
        headers: corsHeaders,
        body: JSON.stringify({ error: 'type and markup are required' }),
      };
    }

    const client = await pool.connect();

    try {
      await client.query('BEGIN');

      // Get affected models
      const modelsResult = await client.query(`
        SELECT id, pricing->>'billed_markup' as current_markup
        FROM models
        WHERE provider_id ${type === 'self_hosted' ? "= 'self_hosted'" : "!= 'self_hosted'"}
      `);

      // Record price history for each model
      for (const model of modelsResult.rows) {
        await client.query(`
          INSERT INTO price_history (tenant_id, model_id, previous_markup, new_markup, change_
          VALUES ($1, $2, $3, $4, 'bulk_update', $5)
        `, [admin.tenantId, model.id, parseFloat(model.current_markup) || 0, markup / 100, adm
      }

      // Update all models of the specified type
      await client.query(`
        UPDATE models
        SET pricing = jsonb_set(
          COALESCE(pricing, '{}'::jsonb),
          '{billed_markup}',
          to_jsonb($1::numeric)
        ),
        updated_at = NOW()
        WHERE provider_id ${type === 'self_hosted' ? "= 'self_hosted'" : "!= 'self_hosted'"}
      `, [markup / 100]);

      // Also update the default in pricing_config
      if (type === 'self_hosted') {
        await client.query(`
          UPDATE pricing_config SET self_hosted_default_markup = $1, updated_at = NOW()
          WHERE tenant_id = $2
        `, [markup / 100, admin.tenantId]);
      } else {
        await client.query(`
          UPDATE pricing_config SET external_default_markup = $1, updated_at = NOW()
          WHERE tenant_id = $2
        `, [markup / 100, admin.tenantId]);
      }

      await client.query('COMMIT');

      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({
          success: true,
          modelsUpdated: modelsResult.rows.length,
        }),
      };
    } catch (error) {
      await client.query('ROLLBACK');
      throw error;
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to bulk update pricing', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to update pricing' }),
    };
  }
```

```
}

// PUT /api/admin/pricing/models/:modelId/override - Set individual model override
export async function setModelPricingOverride(event: APIGatewayProxyEvent): Promise<APIGateway
  try {
    const admin = await requireAdmin(event);
    const modelId = event.pathParameters?.modelId;
    const { markup, inputPrice, outputPrice } = JSON.parse(event.body || '{}');

    if (!modelId) {
      return {
        statusCode: 400,
        headers: corsHeaders,
        body: JSON.stringify({ error: 'modelId is required' }),
      };
    }

    const client = await pool.connect();

    try {
      await client.query('BEGIN');

      // Get current values for history
      const currentResult = await client.query(`
        SELECT markup_override, input_price_override, output_price_override
        FROM model_pricing_overrides
        WHERE tenant_id = $1 AND model_id = $2
        ORDER BY effective_from DESC
        LIMIT 1
      `, [admin.tenantId, modelId]);

      const current = currentResult.rows[0];

      // Record history
      await client.query(`
        INSERT INTO price_history (
          tenant_id, model_id,
          previous_markup, new_markup,
          previous_input_price, new_input_price,
          previous_output_price, new_output_price,
          change_source, changed_by
        ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, 'admin', $9)
      `, [
        admin.tenantId, modelId,
        current?.markup_override, markup,
        current?.input_price_override, inputPrice,
        current?.output_price_override, outputPrice,
        admin.id,
      ]);

      // Insert or update override
      await client.query(`
        INSERT INTO model_pricing_overrides (
          tenant_id, model_id, markup_override, input_price_override, output_price_override, c
        ) VALUES ($1, $2, $3, $4, $5, $6)
        ON CONFLICT (tenant_id, model_id, effective_from) DO UPDATE SET
          markup_override = $3,
          input_price_override = $4,
          output_price_override = $5,
          updated_at = NOW()
      `, [admin.tenantId, modelId, markup, inputPrice, outputPrice, admin.id]);

      await client.query('COMMIT');

      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({ success: true }),
      };
    } catch (error) {
      await client.query('ROLLBACK');
      throw error;
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to set pricing override', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to save override' }),
    };
  }
}

// DELETE /api/admin/pricing/models/:modelId/override - Remove override
export async function deleteModelPricingOverride(event: APIGatewayProxyEvent): Promise<APIGate
  try {
    const admin = await requireAdmin(event);
    const modelId = event.pathParameters?.modelId;

    if (!modelId) {
      return {
        statusCode: 400,
        headers: corsHeaders,
        body: JSON.stringify({ error: 'modelId is required' }),
      };
    }

    const client = await pool.connect();

    try {
      await client.query(`
        DELETE FROM model_pricing_overrides
        WHERE tenant_id = $1 AND model_id = $2
      `, [admin.tenantId, modelId]);

      return {
        statusCode: 200,
        headers: corsHeaders,
        body: JSON.stringify({ success: true }),
      };
    } finally {
      client.release();
    }
  } catch (error) {
    logger.error('Failed to delete pricing override', { error });
    return {
      statusCode: 500,
      headers: corsHeaders,
      body: JSON.stringify({ error: 'Failed to remove override' }),
    };
  }
}
```

31.8 Integration with Chat Handler

```
// apps/thinktank/src/components/chat/chat-input.tsx (updated)

import { ModelSelector } from './model-selector';

export function ChatInput({ onSend, disabled }: ChatInputProps) {
  const [message, setMessage] = useState('');
  const [selectedModel, setSelectedModel] = useState<string | null>(null); // null = Auto

  const handleSend = () => {
    if (!message.trim()) return;

    onSend({
      content: message,
      modelId: selectedModel, // Will be resolved by RADIANT Brain if null
    });
    setMessage('');
  };

  return (
    <div className="border-t p-4">
      <div className="flex items-end gap-3">
        <div className="flex-1">
          <Textarea
            value={message}
            onChange={(e) => setMessage(e.target.value)}
            placeholder="Type your message..."
            className="min-h-[60px] resize-none"
            disabled={disabled}
            onKeyDown={(e) => {
              if (e.key === 'Enter' && !e.shiftKey) {
                e.preventDefault();
                handleSend();
              }
            }}
          />
        </div>
        <div className="flex flex-col gap-2">
          <ModelSelector
            selectedModel={selectedModel}
            onModelChange={setSelectedModel}
            disabled={disabled}
          />
          <Button onClick={handleSend} disabled={disabled || !message.trim()}>
            <Send className="h-4 w-4" />
          </Button>
        </div>
      </div>
    </div>
  );
}
```