Contents

RADIANT v5.0.2 - Source Export Part 5: Infrastructure & Admin UI

---

11. CDK Infrastructure Stack

File: `packages/infrastructure/lib/stacks/grimoire-stack.ts`

Purpose: AWS CDK stack defining all infrastructure for The Grimoire and Economic Governor. Creates Lambda functions, EventBridge schedules, and integrates with VPC/database resources.

Key Resources: - `GrimoireApiFunction` - REST API for Grimoire operations - `GovernorApiFunction` - REST API for Governor operations - `GrimoireCleanupFunction` - Scheduled daily cleanup Lambda - Python layer for Flyte dependencies - EventBridge rule for scheduled cleanup

```
/**
 * Grimoire & Economic Governor CDK Stack
 * RADIANT v5.0.2 - System Evolution
 *
 * This stack defines the AWS infrastructure for:
 * - The Grimoire: Self-optimizing procedural memory
 * - Economic Governor: Cost-aware model routing
 */

import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as events from 'aws-cdk-lib/aws-events';
import * as targets from 'aws-cdk-lib/aws-events-targets';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as iam from 'aws-cdk-lib/aws-iam';
import { Construct } from 'constructs';

export interface GrimoireStackProps extends cdk.StackProps {
  vpc: ec2.IVpc;
  securityGroup: ec2.ISecurityGroup;
  dbSecretArn: string;
  litellmProxyUrl: string;
```

```
  catoApiUrl: string;
  layerArn?: string;
}

export class GrimoireStack extends cdk.Stack {
  public readonly grimoireApiFunction: lambda.Function;
  public readonly governorApiFunction: lambda.Function;
  public readonly cleanupFunction: lambda.Function;

  constructor(scope: Construct, id: string, props: GrimoireStackProps) {
    super(scope, id, props);

    const { vpc, securityGroup, dbSecretArn, litellmProxyUrl, catoApiUrl } = props;

    // Common environment variables for all Lambdas
    const commonEnv = {
      DB_SECRET_ARN: dbSecretArn,
      LITELLM_PROXY_URL: litellmProxyUrl,
      CATO_API_URL: catoApiUrl,
      NODE_OPTIONS: '--enable-source-maps',
    };

    // Create shared Python layer for Flyte dependencies
    const pythonLayer = new lambda.LayerVersion(this, 'FlytePythonLayer', {
      code: lambda.Code.fromAsset('lambda-layers/flyte-deps'),
      compatibleRuntimes: [lambda.Runtime.PYTHON_3_11],
      description: 'Flyte dependencies: psycopg2, pgvector, httpx',
    });

    // ============================================================================
    // THE GRIMOIRE API
    // ============================================================================

    this.grimoireApiFunction = new lambda.Function(this, 'GrimoireApiFunction', {
      functionName: 'radiant-grimoire-api',
      runtime: lambda.Runtime.NODEJS_20_X,
      handler: 'index.handler',
      code: lambda.Code.fromAsset('lambda/grimoire-api'),
      memorySize: 512,
      timeout: cdk.Duration.seconds(30),
      vpc,
      securityGroups: [securityGroup],
      environment: {
        ...commonEnv,
        SERVICE_NAME: 'grimoire-api',
      },
      tracing: lambda.Tracing.ACTIVE,
    });
```

```
// Grant Secrets Manager access
this.grimoireApiFunction.addToRolePolicy(new iam.PolicyStatement({
  actions: ['secretsmanager:GetSecretValue'],
  resources: [dbSecretArn],
}));

// ============================================================================
// ECONOMIC GOVERNOR API
// ============================================================================

this.governorApiFunction = new lambda.Function(this, 'GovernorApiFunction', {
  functionName: 'radiant-governor-api',
  runtime: lambda.Runtime.NODEJS_20_X,
  handler: 'index.handler',
  code: lambda.Code.fromAsset('lambda/governor-api'),
  memorySize: 512,
  timeout: cdk.Duration.seconds(30),
  vpc,
  securityGroups: [securityGroup],
  environment: {
    ...commonEnv,
    SERVICE_NAME: 'governor-api',
  },
  tracing: lambda.Tracing.ACTIVE,
});

this.governorApiFunction.addToRolePolicy(new iam.PolicyStatement({
  actions: ['secretsmanager:GetSecretValue'],
  resources: [dbSecretArn],
}));

// ============================================================================
// GRIMOIRE CLEANUP (Scheduled)
// ============================================================================

this.cleanupFunction = new lambda.Function(this, 'GrimoireCleanupFunction', {
  functionName: 'radiant-grimoire-cleanup',
  runtime: lambda.Runtime.PYTHON_3_11,
  handler: 'cleanup.handler',
  code: lambda.Code.fromAsset('lambda/grimoire'),
  memorySize: 256,
  timeout: cdk.Duration.minutes(5),
  vpc,
  securityGroups: [securityGroup],
  layers: [pythonLayer],
  environment: {
    DB_SECRET_ARN: dbSecretArn,
```

```
        CLEANUP_BATCH_SIZE: '1000',
        LOW_CONFIDENCE_THRESHOLD: '0.3',
        LOW_CONFIDENCE_MAX_AGE_DAYS: '30',
      },
      tracing: lambda.Tracing.ACTIVE,
    });

    this.cleanupFunction.addToRolePolicy(new iam.PolicyStatement({
      actions: ['secretsmanager:GetSecretValue'],
      resources: [dbSecretArn],
    }));

    // Schedule cleanup to run daily at 3 AM UTC
    const cleanupRule = new events.Rule(this, 'GrimoireCleanupSchedule', {
      schedule: events.Schedule.cron({ hour: '3', minute: '0' }),
      description: 'Daily Grimoire cleanup - expired and low-confidence heuristics',
    });

    cleanupRule.addTarget(new targets.LambdaFunction(this.cleanupFunction));

    // ============================================================================
    // OUTPUTS
    // ============================================================================

    new cdk.CfnOutput(this, 'GrimoireApiArn', {
      value: this.grimoireApiFunction.functionArn,
      description: 'Grimoire API Lambda ARN',
    });

    new cdk.CfnOutput(this, 'GovernorApiArn', {
      value: this.governorApiFunction.functionArn,
      description: 'Governor API Lambda ARN',
    });

    new cdk.CfnOutput(this, 'CleanupFunctionArn', {
      value: this.cleanupFunction.functionArn,
      description: 'Grimoire Cleanup Lambda ARN',
    });
  }
}
```

---

12. Cleanup Lambda Handler

File: `packages/infrastructure/lambda/grimoire/cleanup.py`

Purpose: Scheduled Lambda function for daily Grimoire maintenance. Removes expired heuristics and cleans up low-confidence entries.

```python
"""
Grimoire Cleanup Lambda

RADIANT v5.0.2 - System Evolution

Scheduled maintenance job that runs daily to:
1. Delete expired heuristics (past expires_at)
2. Delete low-confidence heuristics older than 30 days
3. Log cleanup statistics for monitoring
"""

import os
import json
import boto3
import psycopg2
from datetime import datetime


# Configuration from environment
DB_SECRET_ARN = os.environ.get('DB_SECRET_ARN')
BATCH_SIZE = int(os.environ.get('CLEANUP_BATCH_SIZE', '1000'))
LOW_CONFIDENCE_THRESHOLD = float(os.environ.get('LOW_CONFIDENCE_THRESHOLD', '0.3'))
LOW_CONFIDENCE_MAX_AGE_DAYS = int(os.environ.get('LOW_CONFIDENCE_MAX_AGE_DAYS', '30'))

# System tenant ID bypasses RLS for maintenance
SYSTEM_TENANT_ID = '00000000-0000-0000-0000-000000000000'


def get_db_credentials():
    """Retrieves database credentials from Secrets Manager"""
    client = boto3.client('secretsmanager')
    response = client.get_secret_value(SecretId=DB_SECRET_ARN)
    secret = json.loads(response['SecretString'])
    return {
        'host': secret['host'],
        'database': secret['dbname'],
        'user': secret['username'],
        'password': secret['password'],
        'port': secret.get('port', 5432)
    }


def handler(event, context):
    """
    Lambda handler for Grimoire cleanup.

    Args:
        event: EventBridge scheduled event
```

```
    context: Lambda context

Returns:
    Dict with cleanup statistics
"""
print(f"Starting Grimoire cleanup at {datetime.utcnow().isoformat()}")

credentials = get_db_credentials()

conn = psycopg2.connect(**credentials)

try:
    cur = conn.cursor()

    # Set system tenant context for RLS bypass
    cur.execute("SET app.current_tenant_id = %s", (SYSTEM_TENANT_ID,))

    # 1. Delete expired heuristics
    cur.execute("""
        WITH deleted AS (
            DELETE FROM knowledge_heuristics
            WHERE expires_at < NOW()
            RETURNING id, tenant_id, domain
        )
        SELECT
            COUNT(*) as count,
            COUNT(DISTINCT tenant_id) as tenant_count
        FROM deleted
    """)
    expired_result = cur.fetchone()
    expired_count = expired_result[0] or 0
    expired_tenants = expired_result[1] or 0

    print(f"Deleted {expired_count} expired heuristics from {expired_tenants} tenants")

    # 2. Delete low-confidence stale heuristics
    cur.execute("""
        WITH deleted AS (
            DELETE FROM knowledge_heuristics
            WHERE confidence_score < %s
              AND created_at < NOW() - INTERVAL '%s days'
            RETURNING id, tenant_id, domain, confidence_score
        )
        SELECT
            COUNT(*) as count,
            AVG(confidence_score) as avg_conf
        FROM deleted
    """, (LOW_CONFIDENCE_THRESHOLD, LOW_CONFIDENCE_MAX_AGE_DAYS))
```

```
        low_conf_result = cur.fetchone()
        low_conf_count = low_conf_result[0] or 0
        avg_deleted_conf = low_conf_result[1] or 0

        print(f"Deleted {low_conf_count} low-confidence heuristics (avg conf: {avg_deleted_con

        # 3. Get remaining statistics
        cur.execute("""
            SELECT
                COUNT(*) as total,
                COUNT(DISTINCT tenant_id) as tenants,
                AVG(confidence_score) as avg_conf,
                COUNT(*) FILTER (WHERE confidence_score >= 0.8) as high_conf
            FROM knowledge_heuristics
            WHERE expires_at > NOW()
        """)
        stats = cur.fetchone()

        conn.commit()

        result = {
            'status': 'success',
            'timestamp': datetime.utcnow().isoformat(),
            'cleanup': {
                'expired_deleted': expired_count,
                'expired_tenant_count': expired_tenants,
                'low_confidence_deleted': low_conf_count,
                'total_deleted': expired_count + low_conf_count
            },
            'remaining': {
                'total_heuristics': stats[0] or 0,
                'tenant_count': stats[1] or 0,
                'avg_confidence': float(stats[2]) if stats[2] else 0,
                'high_confidence_count': stats[3] or 0
            },
            'config': {
                'low_confidence_threshold': LOW_CONFIDENCE_THRESHOLD,
                'low_confidence_max_age_days': LOW_CONFIDENCE_MAX_AGE_DAYS
            }
        }

        print(f"Cleanup complete: {json.dumps(result)}")
        return result

    except Exception as e:
        conn.rollback()
        print(f"Cleanup failed: {str(e)}")
        raise
```

```
    finally:
        conn.close()
```

---

13. Grimoire Admin UI

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

Purpose: React/Next.js admin interface for managing The Grimoire. Provides CRUD operations and statistics visualization.

Features: - List/search heuristics by domain - Add new heuristics manually - Delete heuristics (admin only) - Reinforce (increase/decrease confidence) - View statistics dashboard

```
/**
 * Grimoire Admin Page
 * RADIANT v5.0.2 - System Evolution
 *
 * Admin interface for managing The Grimoire procedural memory.
 */

'use client';

import React, { useEffect, useState } from 'react';
import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle
} from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Badge } from '@/components/ui/badge';
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from '@/components/ui/select';
import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
```

```
} from '@/components/ui/table';
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from '@/components/ui/dialog';
import { Textarea } from '@/components/ui/textarea';
import {
  BookOpen,
  Plus,
  Trash2,
  ThumbsUp,
  ThumbsDown,
  Search,
  RefreshCw
} from 'lucide-react';
import { useToast } from '@/hooks/use-toast';

interface Heuristic {
  id: string;
  domain: string;
  heuristic_text: string;
  confidence_score: number;
  created_at: string;
  expires_at: string;
}

interface GrimoireStats {
  total_heuristics: number;
  total_high_confidence: number;
  total_expiring_soon: number;
  domain_count: number;
  by_domain: Record<string, {
    total: number;
    avg_confidence: number;
    high_confidence: number;
    expiring_soon: number;
  }>;
}

const DOMAINS = ['general', 'medical', 'financial', 'legal', 'technical', 'creative'];

export default function GrimoirePage() {
  const [heuristics, setHeuristics] = useState<Heuristic[]>([]);
```

```
const [stats, setStats] = useState<GrimoireStats | null>(null);
const [loading, setLoading] = useState(true);
const [selectedDomain, setSelectedDomain] = useState<string>('all');
const [searchQuery, setSearchQuery] = useState('');
const [newHeuristic, setNewHeuristic] = useState({ domain: 'general', text: '' });
const [dialogOpen, setDialogOpen] = useState(false);
const { toast } = useToast();

const fetchData = async () => {
  setLoading(true);
  try {
    const params = new URLSearchParams();
    if (selectedDomain !== 'all') params.set('domain', selectedDomain);
    if (searchQuery) params.set('search', searchQuery);

    const [heuristicsRes, statsRes] = await Promise.all([
      fetch(`/api/grimoire/heuristics?${params}`),
      fetch('/api/grimoire/stats')
    ]);

    if (heuristicsRes.ok) {
      const data = await heuristicsRes.json();
      setHeuristics(data.heuristics);
    }

    if (statsRes.ok) {
      const data = await statsRes.json();
      setStats(data);
    }
  } catch (error) {
    toast({ title: 'Error', description: 'Failed to fetch data', variant: 'destructive' });
  } finally {
    setLoading(false);
  }
};

useEffect(() => {
  fetchData();
}, [selectedDomain]);

const handleSearch = () => {
  fetchData();
};

const handleAddHeuristic = async () => {
  if (!newHeuristic.text.trim()) {
    toast({ title: 'Error', description: 'Heuristic text is required', variant: 'destructive
    return;
```

```
  }

  try {
    const res = await fetch('/api/grimoire/heuristics', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        domain: newHeuristic.domain,
        heuristic_text: newHeuristic.text
      })
    });

    if (res.ok) {
      toast({ title: 'Success', description: 'Heuristic added' });
      setNewHeuristic({ domain: 'general', text: '' });
      setDialogOpen(false);
      fetchData();
    } else {
      throw new Error('Failed to add');
    }
  } catch (error) {
    toast({ title: 'Error', description: 'Failed to add heuristic', variant: 'destructive' })
  }
};

const handleDelete = async (id: string) => {
  if (!confirm('Delete this heuristic?')) return;

  try {
    const res = await fetch(`/api/grimoire/heuristics/${id}`, { method: 'DELETE' });
    if (res.ok) {
      toast({ title: 'Success', description: 'Heuristic deleted' });
      fetchData();
    }
  } catch (error) {
    toast({ title: 'Error', description: 'Failed to delete', variant: 'destructive' });
  }
};

const handleReinforce = async (id: string, positive: boolean) => {
  try {
    const res = await fetch(`/api/grimoire/heuristics/${id}/reinforce`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ positive })
    });

    if (res.ok) {
```

```
      toast({
        title: 'Success',
        description: positive ? 'Confidence increased' : 'Confidence decreased'
      });
      fetchData();
    }
  } catch (error) {
    toast({ title: 'Error', description: 'Failed to reinforce', variant: 'destructive' });
  }
};

const getConfidenceBadge = (score: number) => {
  if (score >= 0.8) return <Badge className="bg-green-500">High ({(score * 100).toFixed(0)}%)
  if (score >= 0.5) return <Badge className="bg-yellow-500">Medium ({(score * 100).toFixed(0)
  return <Badge className="bg-red-500">Low ({(score * 100).toFixed(0)}%)</Badge>;
};

return (
  <div className="space-y-6 p-6">
    <div className="flex items-center justify-between">
      <div className="flex items-center gap-2">
        <BookOpen className="h-8 w-8" />
        <div>
          <h1 className="text-2xl font-bold">The Grimoire</h1>
          <p className="text-muted-foreground">Self-optimizing procedural memory</p>
        </div>
      </div>

      <Dialog open={dialogOpen} onOpenChange={setDialogOpen}>
        <DialogTrigger asChild>
          <Button><Plus className="mr-2 h-4 w-4" /> Add Heuristic</Button>
        </DialogTrigger>
        <DialogContent>
          <DialogHeader>
            <DialogTitle>Add New Heuristic</DialogTitle>
            <DialogDescription>
              Manually add a procedural memory heuristic
            </DialogDescription>
          </DialogHeader>
          <div className="space-y-4 py-4">
            <Select
              value={newHeuristic.domain}
              onValueChange={(v) => setNewHeuristic(prev => ({ ...prev, domain: v }))}
            >
              <SelectTrigger>
                <SelectValue placeholder="Select domain" />
              </SelectTrigger>
              <SelectContent>
```

```
            {DOMAINS.map(d => (
              <SelectItem key={d} value={d}>{d.charAt(0).toUpperCase() + d.slice(1)}</Sel
            ))}
          </SelectContent>
        </Select>
        <Textarea
          placeholder="When [condition], always [action]..."
          value={newHeuristic.text}
          onChange={(e) => setNewHeuristic(prev => ({ ...prev, text: e.target.value }))}
          rows={4}
        />
      </div>
      <DialogFooter>
        <Button variant="outline" onClick={() => setDialogOpen(false)}>Cancel</Button>
        <Button onClick={handleAddHeuristic}>Add Heuristic</Button>
      </DialogFooter>
    </DialogContent>
  </Dialog>
</div>

{/* Statistics Cards */}
{stats && (
  <div className="grid grid-cols-4 gap-4">
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>Total Heuristics</CardDescription>
        <CardTitle className="text-3xl">{stats.total_heuristics}</CardTitle>
      </CardHeader>
    </Card>
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>High Confidence</CardDescription>
        <CardTitle className="text-3xl text-green-500">{stats.total_high_confidence}</Ca
      </CardHeader>
    </Card>
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>Expiring Soon</CardDescription>
        <CardTitle className="text-3xl text-yellow-500">{stats.total_expiring_soon}</Car
      </CardHeader>
    </Card>
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>Domains</CardDescription>
        <CardTitle className="text-3xl">{stats.domain_count}</CardTitle>
      </CardHeader>
    </Card>
  </div>
```

```
)}

{/* Filters */}
<Card>
  <CardContent className="pt-6">
    <div className="flex gap-4">
      <Select value={selectedDomain} onValueChange={setSelectedDomain}>
        <SelectTrigger className="w-48">
          <SelectValue placeholder="Filter by domain" />
        </SelectTrigger>
        <SelectContent>
          <SelectItem value="all">All Domains</SelectItem>
          {DOMAINS.map(d => (
            <SelectItem key={d} value={d}>{d.charAt(0).toUpperCase() + d.slice(1)}</Selec
          ))}
        </SelectContent>
      </Select>
      <div className="flex-1 flex gap-2">
        <Input
          placeholder="Search heuristics..."
          value={searchQuery}
          onChange={(e) => setSearchQuery(e.target.value)}
          onKeyDown={(e) => e.key === 'Enter' && handleSearch()}
        />
        <Button variant="outline" onClick={handleSearch}>
          <Search className="h-4 w-4" />
        </Button>
      </div>
      <Button variant="outline" onClick={fetchData}>
        <RefreshCw className={`h-4 w-4 ${loading ? 'animate-spin' : ''}`} />
      </Button>
    </div>
  </CardContent>
</Card>

{/* Heuristics Table */}
<Card>
  <CardContent className="pt-6">
    <Table>
      <TableHeader>
        <TableRow>
          <TableHead className="w-24">Domain</TableHead>
          <TableHead>Heuristic</TableHead>
          <TableHead className="w-32">Confidence</TableHead>
          <TableHead className="w-32">Created</TableHead>
          <TableHead className="w-40">Actions</TableHead>
        </TableRow>
      </TableHeader>
```

```
<TableBody>
  {heuristics.map((h) => (
    <TableRow key={h.id}>
      <TableCell>
        <Badge variant="outline">{h.domain}</Badge>
      </TableCell>
      <TableCell className="max-w-md truncate">{h.heuristic_text}</TableCell>
      <TableCell>{getConfidenceBadge(h.confidence_score)}</TableCell>
      <TableCell className="text-sm text-muted-foreground">
        {new Date(h.created_at).toLocaleDateString()}
      </TableCell>
      <TableCell>
        <div className="flex gap-1">
          <Button
            size="sm"
            variant="ghost"
            onClick={() => handleReinforce(h.id, true)}
            title="Increase confidence"
          >
            <ThumbsUp className="h-4 w-4 text-green-500" />
          </Button>
          <Button
            size="sm"
            variant="ghost"
            onClick={() => handleReinforce(h.id, false)}
            title="Decrease confidence"
          >
            <ThumbsDown className="h-4 w-4 text-red-500" />
          </Button>
          <Button
            size="sm"
            variant="ghost"
            onClick={() => handleDelete(h.id)}
            title="Delete"
          >
            <Trash2 className="h-4 w-4 text-destructive" />
          </Button>
        </div>
      </TableCell>
    </TableRow>
  ))}
  {heuristics.length === 0 && (
    <TableRow>
      <TableCell colSpan={5} className="text-center text-muted-foreground py-8">
        No heuristics found
      </TableCell>
    </TableRow>
  )}
```

```
            </TableBody>
          </Table>
        </CardContent>
      </Card>
    </div>
  );
}
```

---

14. Governor Admin UI

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

Purpose: React/Next.js admin interface for managing the Economic Governor. Provides configuration and statistics visualization.

Features: - Configure Governor mode per domain - View savings statistics and trends - View recent routing decisions - Cost optimization analytics

```
/**
 * Economic Governor Admin Page
 * RADIANT v5.0.2 - System Evolution
 *
 * Admin interface for managing the Economic Governor.
 */

'use client';

import React, { useEffect, useState } from 'react';
import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle
} from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { Badge } from '@/components/ui/badge';
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from '@/components/ui/select';
import {
  Table,
  TableBody,
  TableCell,
```

```
  TableHead,
  TableHeader,
  TableRow,
} from '@/components/ui/table';
import {
  Gauge,
  DollarSign,
  TrendingDown,
  RefreshCw,
  ArrowRight
} from 'lucide-react';
import { useToast } from '@/hooks/use-toast';

type GovernorMode = 'performance' | 'balanced' | 'cost_saver' | 'off';

interface DomainConfig {
  domain: string;
  mode: GovernorMode;
  updated_at: string;
}

interface GovernorStats {
  period: { days: number };
  summary: {
    totalDecisions: number;
    avgComplexity: number;
    totalSavings: number;
    modelSwaps: number;
    taskDistribution: {
      simple: number;
      medium: number;
      complex: number;
    };
  };
  daily: Array<{
    day: string;
    decisions: number;
    savings: number;
    avg_complexity: number;
  }>;
  byMode: Array<{
    mode: string;
    count: number;
    savings: number;
  }>;
}

interface RecentDecision {
```

```
  id: string;
  executionId: string;
  originalModel: string;
  selectedModel: string;
  complexityScore: number;
  savingsAmount: number;
  mode: string;
  reason: string;
  createdAt: string;
}

const MODES: { value: GovernorMode; label: string; description: string }[] = [
  { value: 'off', label: 'Off', description: 'No optimization - use requested models' },
  { value: 'performance', label: 'Performance', description: 'Prioritize quality over cost' },
  { value: 'balanced', label: 'Balanced', description: 'Balance cost and quality' },
  { value: 'cost_saver', label: 'Cost Saver', description: 'Maximize cost savings' },
];

const DOMAINS = ['general', 'medical', 'financial', 'legal', 'technical', 'creative'];

export default function GovernorPage() {
  const [configs, setConfigs] = useState<DomainConfig[]>([]);
  const [stats, setStats] = useState<GovernorStats | null>(null);
  const [recentDecisions, setRecentDecisions] = useState<RecentDecision[]>([]);
  const [loading, setLoading] = useState(true);
  const { toast } = useToast();

  const fetchData = async () => {
    setLoading(true);
    try {
      const [configRes, statsRes, recentRes] = await Promise.all([
        fetch('/api/governor/config'),
        fetch('/api/governor/statistics?days=30'),
        fetch('/api/governor/recent?limit=10')
      ]);

      if (configRes.ok) {
        const data = await configRes.json();
        setConfigs(data.domains);
      }

      if (statsRes.ok) {
        const data = await statsRes.json();
        setStats(data);
      }

      if (recentRes.ok) {
        const data = await recentRes.json();
```

```
      setRecentDecisions(data.decisions);
    }
  } catch (error) {
    toast({ title: 'Error', description: 'Failed to fetch data', variant: 'destructive' });
  } finally {
    setLoading(false);
  }
};

useEffect(() => {
  fetchData();
}, []);

const handleModeChange = async (domain: string, mode: GovernorMode) => {
  try {
    const res = await fetch('/api/governor/config', {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ domain, mode })
    });

    if (res.ok) {
      toast({ title: 'Success', description: `${domain} mode updated to ${mode}` });
      fetchData();
    }
  } catch (error) {
    toast({ title: 'Error', description: 'Failed to update mode', variant: 'destructive' });
  }
};

const getModeBadge = (mode: string) => {
  const colors: Record<string, string> = {
    off: 'bg-gray-500',
    performance: 'bg-blue-500',
    balanced: 'bg-green-500',
    cost_saver: 'bg-yellow-500'
  };
  return <Badge className={colors[mode] || 'bg-gray-500'}>{mode.replace('_', ' ')}</Badge>;
};

const getComplexityBadge = (score: number) => {
  if (score <= 4) return <Badge className="bg-green-500">Simple ({score})</Badge>;
  if (score <= 8) return <Badge className="bg-yellow-500">Medium ({score})</Badge>;
  return <Badge className="bg-red-500">Complex ({score})</Badge>;
};

return (
  <div className="space-y-6 p-6">
```

```
<div className="flex items-center justify-between">
  <div className="flex items-center gap-2">
    <Gauge className="h-8 w-8" />
    <div>
      <h1 className="text-2xl font-bold">Economic Governor</h1>
      <p className="text-muted-foreground">Cost-aware model routing</p>
    </div>
  </div>
  <Button variant="outline" onClick={fetchData}>
    <RefreshCw className={`h-4 w-4 mr-2 ${loading ? 'animate-spin' : ''}`} />
    Refresh
  </Button>
</div>

{/* Summary Statistics */}
{stats && (
  <div className="grid grid-cols-4 gap-4">
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>Total Decisions (30d)</CardDescription>
        <CardTitle className="text-3xl">{stats.summary.totalDecisions}</CardTitle>
      </CardHeader>
    </Card>
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>Total Savings</CardDescription>
        <CardTitle className="text-3xl text-green-500">
          <DollarSign className="inline h-6 w-6" />
          {stats.summary.totalSavings.toFixed(2)}
        </CardTitle>
      </CardHeader>
    </Card>
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>Model Swaps</CardDescription>
        <CardTitle className="text-3xl flex items-center gap-2">
          <TrendingDown className="h-6 w-6 text-blue-500" />
          {stats.summary.modelSwaps}
        </CardTitle>
      </CardHeader>
    </Card>
    <Card>
      <CardHeader className="pb-2">
        <CardDescription>Avg Complexity</CardDescription>
        <CardTitle className="text-3xl">{stats.summary.avgComplexity.toFixed(1)}/10</Card
      </CardHeader>
    </Card>
  </div>
```

```
)}

{/* Task Distribution */}
{stats && (
  <Card>
    <CardHeader>
      <CardTitle>Task Distribution</CardTitle>
      <CardDescription>Last 30 days by complexity</CardDescription>
    </CardHeader>
    <CardContent>
      <div className="flex gap-4">
        <div className="flex-1 text-center p-4 bg-green-500/10 rounded-lg">
          <div className="text-3xl font-bold text-green-500">
            {stats.summary.taskDistribution.simple}
          </div>
          <div className="text-sm text-muted-foreground">Simple (1-4)</div>
        </div>
        <div className="flex-1 text-center p-4 bg-yellow-500/10 rounded-lg">
          <div className="text-3xl font-bold text-yellow-500">
            {stats.summary.taskDistribution.medium}
          </div>
          <div className="text-sm text-muted-foreground">Medium (5-8)</div>
        </div>
        <div className="flex-1 text-center p-4 bg-red-500/10 rounded-lg">
          <div className="text-3xl font-bold text-red-500">
            {stats.summary.taskDistribution.complex}
          </div>
          <div className="text-sm text-muted-foreground">Complex (9-10)</div>
        </div>
      </div>
    </CardContent>
  </Card>
)}

{/* Domain Configuration */}
<Card>
  <CardHeader>
    <CardTitle>Domain Configuration</CardTitle>
    <CardDescription>Set Governor mode per domain</CardDescription>
  </CardHeader>
  <CardContent>
    <Table>
      <TableHeader>
        <TableRow>
          <TableHead>Domain</TableHead>
          <TableHead>Current Mode</TableHead>
          <TableHead>Change Mode</TableHead>
        </TableRow>
```

```
      </TableHeader>
      <TableBody>
        {DOMAINS.map(domain => {
          const config = configs.find(c => c.domain === domain);
          const currentMode = config?.mode || 'balanced';
          return (
            <TableRow key={domain}>
              <TableCell className="font-medium">
                {domain.charAt(0).toUpperCase() + domain.slice(1)}
              </TableCell>
              <TableCell>{getModeBadge(currentMode)}</TableCell>
              <TableCell>
                <Select
                  value={currentMode}
                  onValueChange={(v) => handleModeChange(domain, v as GovernorMode)}
                >
                  <SelectTrigger className="w-48">
                    <SelectValue />
                  </SelectTrigger>
                  <SelectContent>
                    {MODES.map(m => (
                      <SelectItem key={m.value} value={m.value}>
                        <div className="flex flex-col">
                          <span>{m.label}</span>
                          <span className="text-xs text-muted-foreground">{m.description
                        </div>
                      </SelectItem>
                    ))}
                  </SelectContent>
                </Select>
              </TableCell>
            </TableRow>
          );
        })}
      </TableBody>
    </Table>
  </CardContent>
</Card>

{/* Recent Decisions */}
<Card>
  <CardHeader>
    <CardTitle>Recent Routing Decisions</CardTitle>
    <CardDescription>Last 10 model routing decisions</CardDescription>
  </CardHeader>
  <CardContent>
    <Table>
      <TableHeader>
```

```
            <TableRow>
              <TableHead>Time</TableHead>
              <TableHead>Complexity</TableHead>
              <TableHead>Routing</TableHead>
              <TableHead>Savings</TableHead>
              <TableHead>Reason</TableHead>
            </TableRow>
          </TableHeader>
          <TableBody>
            {recentDecisions.map((d) => (
              <TableRow key={d.id}>
                <TableCell className="text-sm text-muted-foreground">
                  {new Date(d.createdAt).toLocaleTimeString()}
                </TableCell>
                <TableCell>{getComplexityBadge(d.complexityScore)}</TableCell>
                <TableCell>
                  <div className="flex items-center gap-2 text-sm">
                    <code className="bg-muted px-1 rounded">{d.originalModel}</code>
                    {d.originalModel !== d.selectedModel && (
                      <>
                        <ArrowRight className="h-4 w-4" />
                        <code className="bg-muted px-1 rounded">{d.selectedModel}</code>
                      </>
                    )}
                  </div>
                </TableCell>
                <TableCell>
                  {d.savingsAmount > 0 && (
                    <span className="text-green-500">${d.savingsAmount.toFixed(4)}</span>
                  )}
                </TableCell>
                <TableCell className="max-w-xs truncate text-sm text-muted-foreground">
                  {d.reason}
                </TableCell>
              </TableRow>
            ))}
            {recentDecisions.length === 0 && (
              <TableRow>
                <TableCell colSpan={5} className="text-center text-muted-foreground py-8">
                  No recent decisions
                </TableCell>
              </TableRow>
            )}
          </TableBody>
        </Table>
      </CardContent>
    </Card>
  </div>
```

```
  );
}
```

---

15. Think Tank Workflow Integration

File: `packages/flyte/workflows/think_tank_workflow.py` (excerpt)

Purpose: Shows how The Grimoire and Economic Governor integrate into the main Think Tank workflow.

```python
"""
Think Tank Workflow - Grimoire & Governor Integration

RADIANT v5.0.2 - System Evolution

This excerpt shows how The Grimoire and Economic Governor
integrate into the Human-in-the-Loop Think Tank workflow.
"""

from flytekit import workflow, dynamic
from dataclasses import dataclass
from typing import Optional, List

from radiant.flyte.workflows.grimoire_tasks import consult_grimoire, librarian_review
from radiant.flyte.utils import EconomicGovernor, GovernorDecision


@dataclass
class ThinkTankRequest:
    tenant_id: str
    user_id: str
    prompt: str
    domain: str
    session_id: str
    model: Optional[str] = None


@dataclass
class ThinkTankResult:
    response: str
    execution_id: str
    model_used: str
    grimoire_heuristics_used: int
    governor_savings: float
    was_successful: bool
```

```python
@workflow
def think_tank_hitl_workflow(request: ThinkTankRequest) -> ThinkTankResult:
    """
    Human-in-the-Loop reasoning workflow with procedural memory and cost optimization.

    Flow:
    1. Consult Grimoire for relevant heuristics
    2. Economic Governor evaluates task complexity
    3. Execute with optimal model
    4. On success, Librarian reviews for new heuristics
    """

    # Step 1: Consult The Grimoire
    grimoire_result = consult_grimoire(
        tenant_id=request.tenant_id,
        prompt=request.prompt,
        domain=request.domain
    )

    # Step 2: Economic Governor evaluation
    governor = EconomicGovernor()
    decision = governor.evaluate_task(
        task={'type': 'think_tank', 'prompt': request.prompt, 'context': {}},
        agent={'agent_id': 'thinktank', 'role': 'reasoner', 'model': request.model or 'gpt-4o'
        domain=request.domain
    )

    # Step 3: Build augmented prompt with heuristics
    augmented_prompt = grimoire_result.context_injection + "\n\n" + request.prompt

    # Step 4: Execute with selected model
    # ... (AI execution logic)
    response = "..."  # Placeholder
    execution_id = "..."
    was_successful = True

    # Step 5: Librarian review on success
    if was_successful:
        librarian_review(
            tenant_id=request.tenant_id,
            execution_id=execution_id,
            prompt=request.prompt,
            response=response,
            domain=request.domain,
            was_successful=True,
            user_rating=None  # Will be updated async
        )
```

```
    return ThinkTankResult(
        response=response,
        execution_id=execution_id,
        model_used=decision.selected_model,
        grimoire_heuristics_used=len(grimoire_result.heuristics),
        governor_savings=decision.estimated_savings,
        was_successful=was_successful
    )
```

Export Complete

This concludes the comprehensive source export for **The Grimoire** and **Economic Governor** systems in RADIANT v5.0.2.

Files Included: 1. Database Schema Migration (SQL) 2. Economic Governor Service (TypeScript) 3. Governor Module Exports (TypeScript) 4. Grimoire API Handler (TypeScript) 5. Governor API Handler (TypeScript) 6. Grimoire Flyte Tasks (Python) 7. Database Utilities (Python) 8. Embedding Utilities (Python) 9. Cato Safety Client (Python) 10. Python Utils Module Exports (Python) 11. CDK Infrastructure Stack (TypeScript) 12. Cleanup Lambda Handler (Python) 13. Grimoire Admin UI (React/TSX) 14. Governor Admin UI (React/TSX) 15. Think Tank Workflow Integration (Python)

Total: 15 source files across 5 documentation parts.