Contents

SECTION 35: ADMIN DASHBOARD - MODEL & LICENSE MANAGEMENT UI (v4.1.0)

Full CRUD UI for managing AI models through the Admin Dashboard. Administrators can add/edit/delete models without any code changes.

35.1 MODEL MANAGEMENT PAGE

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

```
'use client';

import { useState } from 'react';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import { Plus, Search, RefreshCw, AlertTriangle } from 'lucide-react';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';
import { ModelTable } from '@/components/models/model-table';
import { ModelForm } from '@/components/models/model-form';
import { LicensePanel } from '@/components/models/license-panel';
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogTrigger } from '@/components/
import { toast } from '@/components/ui/use-toast';
import { api } from '@/lib/api';

export default function ModelsPage() {
  const [searchQuery, setSearchQuery] = useState('');
  const [categoryFilter, setCategoryFilter] = useState('all');
const [showAddModel, setShowAddModel] = useState(false);
const [editingModel, setEditingModel] = useState<string | null>(null);
const queryClient = useQueryClient();

const { data: models, isLoading, refetch } = useQuery({
  queryKey: ['models', categoryFilter],
  queryFn: async () => {
    const params = categoryFilter !== 'all' ? `?category=${categoryFilter}` : '';
    const response = await api.get(`/api/v2/admin/models${params}`);
    return response.data;
  },
});

const { data: licenseSummary } = useQuery({
  queryKey: ['license-summary'],
  queryFn: async () => (await api.get('/api/v2/admin/models/licenses/summary')).data,
});

const createModelMutation = useMutation({
  mutationFn: (data: any) => api.post('/api/v2/admin/models', data),
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ['models'] });
    setShowAddModel(false);
    toast({ title: 'Model Created', description: 'New model added to registry.' });
  },
});

const deleteModelMutation = useMutation({
  mutationFn: (modelId: string) => api.delete(`/api/v2/admin/models/${modelId}`),
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ['models'] });
    toast({ title: 'Model Deleted' });
  },
});

const filteredModels = models?.filter((m: any) =>
  m.displayName.toLowerCase().includes(searchQuery.toLowerCase()) ||
  m.modelId.toLowerCase().includes(searchQuery.toLowerCase())
) || [];

return (
  <div className="space-y-6">
    <div className="flex items-center justify-between">
      <div>
        <h1 className="text-3xl font-bold">AI Models</h1>
        <p className="text-muted-foreground">
          Database-driven model management - no code changes needed!
        </p>
  </div>
  <div className="flex gap-2">
    <Button variant="outline" size="sm" onClick={() => refetch()}>
      <RefreshCw className="h-4 w-4 mr-2" /> Refresh
    </Button>
    <Dialog open={showAddModel} onOpenChange={setShowAddModel}>
      <DialogTrigger asChild>
        <Button><Plus className="h-4 w-4 mr-2" /> Add Model</Button>
      </DialogTrigger>
      <DialogContent className="max-w-4xl max-h-[90vh] overflow-y-auto">
        <DialogHeader>
          <DialogTitle>Add New AI Model</DialogTitle>
        </DialogHeader>
        <ModelForm
          onSubmit={(data) => createModelMutation.mutate(data)}
          isLoading={createModelMutation.isPending}
          onCancel={() => setShowAddModel(false)}
        />
      </DialogContent>
    </Dialog>
  </div>
</div>

{/* Summary Cards */}
<div className="grid grid-cols-2 md:grid-cols-4 gap-4">
  <Card>
    <CardHeader className="pb-2"><CardTitle className="text-sm text-muted-foreground">To
    <CardContent><div className="text-2xl font-bold">{licenseSummary?.totalModels || 0}<
  </Card>
  <Card>
    <CardHeader className="pb-2"><CardTitle className="text-sm text-muted-foreground">Co
    <CardContent><div className="text-2xl font-bold text-green-600">{licenseSummary?.com
  </Card>
  <Card>
    <CardHeader className="pb-2"><CardTitle className="text-sm text-muted-foreground">Re
    <CardContent><div className="text-2xl font-bold text-yellow-600">{licenseSummary?.re
  </Card>
  <Card>
    <CardHeader className="pb-2"><CardTitle className="text-sm text-muted-foreground">Ex
    <CardContent><div className="text-2xl font-bold text-orange-600">{licenseSummary?.ex
  </Card>
</div>

{(licenseSummary?.nonCompliant || 0) > 0 && (
  <Card className="border-red-200 bg-red-50">
    <CardContent className="flex items-center gap-4 py-4">
      <AlertTriangle className="h-8 w-8 text-red-600" />
      <div>
```

```
          <h3 className="font-semibold text-red-800">License Compliance Alert</h3>
          <p className="text-red-700">{licenseSummary?.nonCompliant} model(s) have non-comp
        </div>
      </CardContent>
    </Card>
  )}

  <Tabs defaultValue="models" className="space-y-4">
    <TabsList>
      <TabsTrigger value="models">Models</TabsTrigger>
      <TabsTrigger value="licenses">Licenses</TabsTrigger>
      <TabsTrigger value="workflows">Workflows</TabsTrigger>
    </TabsList>

    <TabsContent value="models" className="space-y-4">
      <div className="flex gap-4">
        <div className="relative flex-1">
          <Search className="absolute left-3 top-1/2 -translate-y-1/2 h-4 w-4 text-muted-f
          <Input
            placeholder="Search models..."
            value={searchQuery}
            onChange={(e) => setSearchQuery(e.target.value)}
            className="pl-10"
          />
        </div>
        <select
          value={categoryFilter}
          onChange={(e) => setCategoryFilter(e.target.value)}
          className="px-3 py-2 border rounded-md"
        >
          <option value="all">All Categories</option>
          <option value="scientific_protein"> Protein Folding</option>
          <option value="vision_detection"> Object Detection</option>
          <option value="audio_stt"> Speech-to-Text</option>
          <option value="medical_imaging"> Medical Imaging</option>
          <option value="llm"> LLM</option>
        </select>
      </div>

      <ModelTable
        models={filteredModels}
        isLoading={isLoading}
        onEdit={setEditingModel}
        onDelete={(id) => confirm('Delete model?') && deleteModelMutation.mutate(id)}
      />
    </TabsContent>

    <TabsContent value="licenses"><LicensePanel /></TabsContent>
```

```
        <TabsContent value="workflows"><p className="text-muted-foreground">Workflow managemen
      </Tabs>
    </div>
  );
}
```

35.2 API ENDPOINTS

packages/lambda/admin/orchestration.ts

```
/**
 * RADIANT v4.1.0 - Admin Orchestration API
 */

import { APIGatewayProxyHandler } from 'aws-lambda';
import { Pool } from 'pg';
import { getOrchestrationEngine } from '@radiant/services/orchestration';
import { requirePermission } from '../auth/permissions';
import { createResponse, createErrorResponse } from '../utils/response';

const pool = new Pool({ connectionString: process.env.DATABASE_URL });

export const listModels: APIGatewayProxyHandler = async (event) => {
  try {
    await requirePermission(event, 'models:read');
    const { category, status, minTier, providerType } = event.queryStringParameters || {};
    const engine = getOrchestrationEngine(pool);
    const models = await engine.listModels({ category, status, minTier: minTier ? parseInt(min
    return createResponse(200, models);
  } catch (error: any) {
    return createErrorResponse(error);
  }
};

export const getModel: APIGatewayProxyHandler = async (event) => {
  try {
    await requirePermission(event, 'models:read');
    const modelId = event.pathParameters?.modelId;
    if (!modelId) return createResponse(400, { error: 'Model ID required' });

    const engine = getOrchestrationEngine(pool);
    const model = await engine.getModelConfig(modelId);
    if (!model) return createResponse(404, { error: 'Model not found' });
    return createResponse(200, model);
  } catch (error: any) {
    return createErrorResponse(error);
  }
```

```
};

export const createModel: APIGatewayProxyHandler = async (event) => {
  try {
    const admin = await requirePermission(event, 'models:write');
    const data = JSON.parse(event.body || '{}');
    const engine = getOrchestrationEngine(pool);
    const modelUuid = await engine.createModel(data, admin.id);
    return createResponse(201, { id: modelUuid, modelId: data.modelId, message: 'Model created
  } catch (error: any) {
    return createErrorResponse(error);
  }
};

export const updateModel: APIGatewayProxyHandler = async (event) => {
  try {
    const admin = await requirePermission(event, 'models:write');
    const modelId = event.pathParameters?.modelId;
    if (!modelId) return createResponse(400, { error: 'Model ID required' });

    const updates = JSON.parse(event.body || '{}');
    const engine = getOrchestrationEngine(pool);
    await engine.updateModel(modelId, updates, admin.id);
    return createResponse(200, { success: true });
  } catch (error: any) {
    return createErrorResponse(error);
  }
};

export const deleteModel: APIGatewayProxyHandler = async (event) => {
  try {
    const admin = await requirePermission(event, 'models:write');
    const modelId = event.pathParameters?.modelId;
    if (!modelId) return createResponse(400, { error: 'Model ID required' });

    const engine = getOrchestrationEngine(pool);
    await engine.deleteModel(modelId, admin.id);
    return createResponse(200, { success: true });
  } catch (error: any) {
    return createErrorResponse(error);
  }
};

export const getLicenseSummary: APIGatewayProxyHandler = async (event) => {
  try {
    await requirePermission(event, 'models:read');
    const engine = getOrchestrationEngine(pool);
    const summary = await engine.getLicenseSummary();
    return createResponse(200, summary);
  } catch (error: any) {
    return createErrorResponse(error);
  }
};

export const listWorkflows: APIGatewayProxyHandler = async (event) => {
  try {
    await requirePermission(event, 'workflows:read');
    const result = await pool.query(`
      SELECT workflow_id, name, description, category, version, min_tier
      FROM workflow_definitions WHERE enabled = true ORDER BY category, name
    `);
    return createResponse(200, result.rows);
  } catch (error: any) {
    return createErrorResponse(error);
  }
};

export const executeWorkflow: APIGatewayProxyHandler = async (event) => {
  try {
    const admin = await requirePermission(event, 'workflows:execute');
    const workflowId = event.pathParameters?.workflowId;
    if (!workflowId) return createResponse(400, { error: 'Workflow ID required' });

    const { tenantId, userId, parameters } = JSON.parse(event.body || '{}');
    const engine = getOrchestrationEngine(pool);
    const executionId = await engine.executeWorkflow(workflowId, tenantId, userId, parameters)
    return createResponse(202, { executionId, status: 'started' });
  } catch (error: any) {
    return createErrorResponse(error);
  }
};
```

35.3 VERIFICATION COMMANDS

```
# Apply orchestration migration
psql $DATABASE_URL -f packages/database/migrations/034_orchestration_engine.sql

# Apply AlphaFold 2 seed data
psql $DATABASE_URL -f packages/database/migrations/034a_seed_alphafold2.sql

# Verify AlphaFold 2 is in registry
psql $DATABASE_URL -c "SELECT model_id, display_name, status FROM ai_models WHERE model_id = '

# Verify licenses
```

```
psql $DATABASE_URL -c "SELECT m.display_name, l.license_spdx, l.commercial_use FROM model_licen

# Test get_model_config function
psql $DATABASE_URL -c "SELECT get_model_config('alphafold2')" | head -50

# Test get_workflow_config function
psql $DATABASE_URL -c "SELECT get_workflow_config('protein_folding_alphafold2')" | head -50

# Verify audit log is recording
psql $DATABASE_URL -c "SELECT entity_type, action, change_summary, created_at FROM orchestratio
```
