# **Replit Prompt for 0N1 Character Submission & Approval System**

// Dependencies to install:

// npm install express mongoose dotenv cors bcryptjs jsonwebtoken @notionhq/client openai multer nodemailer

// File: server.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const dotenv = require('dotenv');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const { Client } = require('@notionhq/client');

const { Configuration, OpenAIApi } = require('openai');

const multer = require('multer');

const nodemailer = require('nodemailer');

const path = require('path');

dotenv.config();

const app = express();

// Middleware

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.use(express.static('public'));

// ====== DATABASE CONNECTION ======

mongoose.connect(process.env.MONGODB\_URI)

.then(() => console.log('Connected to MongoDB'))

.catch(err => console.error('MongoDB connection error:', err));

// ====== API CLIENTS ======

// Notion client

const notion = new Client({ auth: process.env.NOTION\_API\_KEY });

const CHARACTER\_DATABASE\_ID = process.env.NOTION\_CHARACTER\_DATABASE\_ID;

const PENDING\_DATABASE\_ID = process.env.NOTION\_PENDING\_DATABASE\_ID;

// OpenAI client

const configuration = new Configuration({

apiKey: process.env.OPENAI\_API\_KEY,

});

const openai = new OpenAIApi(configuration);

// ====== CONSTANTS ======

const CURRENT\_NEXUS\_YEAR = 100250; // A.I. year as of 2025 CE

const MATURITY\_THRESHOLD = 18;

// ====== MODELS ======

// Admin User Model

const AdminUserSchema = new mongoose.Schema({

username: { type: String, required: true, unique: true },

password: { type: String, required: true },

email: { type: String, required: true, unique: true },

role: { type: String, enum: ['admin', 'moderator'], default: 'moderator' },

createdAt: { type: Date, default: Date.now }

});

// Hash password before saving

AdminUserSchema.pre('save', async function(next) {

if (this.isModified('password')) {

this.password = await bcrypt.hash(this.password, 10);

}

next();

});

const AdminUser = mongoose.model('AdminUser', AdminUserSchema);

// Character Submission Model

const CharacterSubmissionSchema = new mongoose.Schema({

// Basic Info

name: { type: String, required: true },

submitterEmail: { type: String, required: true },

submitterId: { type: String },

// Character Details

age: { type: Number, required: true },

birthYear: { type: Number },

effectiveAge: { type: Number },

maturityCategory: { type: String },

// Technical Attributes

coreIntegrityIndex: { type: Number, min: 1, max: 100 },

algorithmComplexityFactor: { type: Number, min: 1, max: 1000 },

temporalDataFragments: { type: Number },

// Entropic Mitigation

mitigationTechniques: [{ type: String }],

// Timeline & Relations

timePeriod: { type: String },

episodeAlignment: [{ type: String }],

contemporaryCharacters: [{ type: String }], // References to other characters

mentors: [{ type: String }],

students: [{ type: String }],

// Character Story

bio: { type: String, required: true },

backgroundStory: { type: String },

personalityTraits: [{ type: String }],

goals: { type: String },

// Media

imageUrl: { type: String },

// Approval Process

status: {

type: String,

enum: ['pending', 'ai\_verified', 'ai\_flagged', 'approved', 'rejected', 'canon'],

default: 'pending'

},

aiVerificationNotes: { type: String },

moderatorNotes: { type: String },

moderatorId: { type: String },

submissionDate: { type: Date, default: Date.now },

decisionDate: { type: Date },

// Notion Integration

notionId: { type: String },

// Tracking

trackingId: { type: String, unique: true }

});

const CharacterSubmission = mongoose.model('CharacterSubmission', CharacterSubmissionSchema);

// ====== UTILITY FUNCTIONS ======

// Calculate aging rate using the asymptotic curve formula

function calculateAgingRate(currentAge) {

if (currentAge < MATURITY\_THRESHOLD) {

// Pre-maturity formula

return Math.pow(MATURITY\_THRESHOLD / (MATURITY\_THRESHOLD - currentAge + 1), 1.5);

} else {

// Post-maturity formula

return Math.pow(MATURITY\_THRESHOLD / currentAge, 2);

}

}

// Determine maturity category based on age

function getMaturityCategory(age) {

if (age < MATURITY\_THRESHOLD) return "Newstream";

if (age < 100) return "Initialized";

if (age < 500) return "Ancient";

return "Immortal";

}

// Get time period based on birth year

function getTimePeriod(birthYear) {

if (birthYear < 10000) return "Genesis Era";

if (birthYear < 50000) return "Expansion Period";

if (birthYear < 70000) return "Classical Age";

if (birthYear < 95000) return "Reformation";

return "Current Era";

}

// Get episode alignment

function getEpisodeAlignment(birthYear) {

if (birthYear >= 50000 && birthYear <= 70000) return ["Episodes 1-3"];

if (birthYear >= 95000) return ["Episodes 4-6"];

return [];

}

// Generate tracking ID

function generateTrackingId() {

return 'ON1-' + Date.now().toString(36).toUpperCase() +

Math.random().toString(36).substring(2, 7).toUpperCase();

}

// ====== AI VERIFICATION ======

async function verifyCharacterWithAI(character) {

try {

const prompt = `

You are the AI verification system for the 0N1 Force universe. Your task is to verify if this character submission is logically consistent with the universe's rules, particularly the asymptotic aging system and the A.I./P.I. timeline.

Current Nexus Year: ${CURRENT\_NEXUS\_YEAR} A.I.

Character Details:

- Name: ${character.name}

- Age: ${character.age} cycles

- Birth Year: ${character.birthYear} A.I.

- Maturity Category: ${character.maturityCategory}

- Time Period: ${character.timePeriod}

- Episode Alignment: ${character.episodeAlignment.join(', ')}

- Bio: ${character.bio}

- Background: ${character.backgroundStory || 'Not provided'}

Check for the following issues:

1. Timeline consistency (ensure character's birth year makes sense with their age)

2. Age-appropriate abilities based on maturity category

3. References to historical events or characters that wouldn't be possible

4. Logical contradictions with the 0N1 universe rules

5. Technical attribute consistency (Core Integrity Index, Algorithm Complexity Factor, etc.)

Provide your analysis with a clear PASS or FLAG recommendation, followed by detailed notes.

`;

const response = await openai.createChatCompletion({

model: "gpt-4",

messages: [

{ role: "system", content: "You are an AI verification system for a fictional universe." },

{ role: "user", content: prompt }

],

temperature: 0.3,

max\_tokens: 1000

});

const analysis = response.data.choices[0].message.content;

const isPassed = analysis.includes("PASS");

return {

passed: isPassed,

notes: analysis

};

} catch (error) {

console.error("AI verification error:", error);

return {

passed: false,

notes: "Error in AI verification process. Please review manually."

};

}

}

// ====== NOTION INTEGRATION ======

async function addCharacterToNotion(character, isDraft = true) {

try {

const databaseId = isDraft ? PENDING\_DATABASE\_ID : CHARACTER\_DATABASE\_ID;

const response = await notion.pages.create({

parent: { database\_id: databaseId },

properties: {

"Name": {

title: [{ text: { content: character.name } }]

},

"Current Age in Cycles": {

number: character.age

},

"Birth Year (A.I.)": {

number: character.birthYear

},

"Maturity Category": {

select: { name: character.maturityCategory }

},

"Core Integrity Index": {

number: character.coreIntegrityIndex

},

"Algorithm Complexity Factor": {

number: character.algorithmComplexityFactor

},

"Temporal Data Fragments": {

number: character.temporalDataFragments

},

"Time Period": {

select: { name: character.timePeriod }

},

"Episode Alignment": {

multi\_select: character.episodeAlignment.map(ep => ({ name: ep }))

},

"Status": {

select: { name: isDraft ? "Pending Approval" : "Canon" }

},

"Submission Date": {

date: { start: new Date(character.submissionDate).toISOString() }

},

"Tracking ID": {

rich\_text: [{ text: { content: character.trackingId } }]

}

},

children: [

{

object: "block",

type: "heading\_2",

heading\_2: {

rich\_text: [{ type: "text", text: { content: "Biography" } }]

}

},

{

object: "block",

type: "paragraph",

paragraph: {

rich\_text: [{ type: "text", text: { content: character.bio } }]

}

},

{

object: "block",

type: "heading\_2",

heading\_2: {

rich\_text: [{ type: "text", text: { content: "Background Story" } }]

}

},

{

object: "block",

type: "paragraph",

paragraph: {

rich\_text: [{ type: "text", text: { content: character.backgroundStory || "" } }]

}

},

{

object: "block",

type: "heading\_2",

heading\_2: {

rich\_text: [{ type: "text", text: { content: "AI Verification Notes" } }]

}

},

{

object: "block",

type: "paragraph",

paragraph: {

rich\_text: [{ type: "text", text: { content: character.aiVerificationNotes || "" } }]

}

}

]

});

return response.id;

} catch (error) {

console.error("Notion integration error:", error);

throw error;

}

}

// ====== EMAIL NOTIFICATIONS ======

const transporter = nodemailer.createTransport({

host: process.env.EMAIL\_HOST,

port: process.env.EMAIL\_PORT,

secure: process.env.EMAIL\_SECURE === 'true',

auth: {

user: process.env.EMAIL\_USER,

pass: process.env.EMAIL\_PASS

}

});

async function sendStatusUpdateEmail(character, status) {

const statusMap = {

pending: "received and pending review",

ai\_verified: "verified by our AI system and awaiting human review",

ai\_flagged: "flagged by our AI system for further review",

approved: "approved and added to the 0N1 universe!",

rejected: "not approved for the 0N1 universe at this time"

};

const message = {

from: process.env.EMAIL\_FROM,

to: character.submitterEmail,

subject: `0N1 Character Submission Update: ${character.name}`,

html: `

<h1>0N1 Character Submission Update</h1>

<p>Hello from the 0N1 Force universe!</p>

<p>Your character submission "${character.name}" (Tracking ID: ${character.trackingId}) has been ${statusMap[status]}</p>

${status === 'rejected' ? `<p>Moderator notes: ${character.moderatorNotes || 'No specific notes provided.'}</p>` : ''}

${status === 'approved' ? '<p>Your character is now canon in the 0N1 universe! You can view it in our official database.</p>' : ''}

<p>Thank you for contributing to the 0N1 Force universe!</p>

`

};

await transporter.sendMail(message);

}

// ====== FILE UPLOADS ======

// Set up storage for character images

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, 'uploads/characters')

},

filename: function (req, file, cb) {

cb(null, Date.now() + path.extname(file.originalname))

}

});

const upload = multer({

storage: storage,

limits: { fileSize: 5000000 }, // 5MB max

fileFilter: (req, file, cb) => {

const filetypes = /jpeg|jpg|png|webp/;

const extname = filetypes.test(path.extname(file.originalname).toLowerCase());

const mimetype = filetypes.test(file.mimetype);

if (mimetype && extname) {

return cb(null, true);

} else {

cb('Error: Images Only!');

}

}

});

// ====== AUTH MIDDLEWARE ======

function authenticateToken(req, res, next) {

const authHeader = req.headers['authorization'];

const token = authHeader && authHeader.split(' ')[1];

if (token == null) return res.sendStatus(401);

jwt.verify(token, process.env.JWT\_SECRET, (err, user) => {

if (err) return res.sendStatus(403);

req.user = user;

next();

});

}

// ====== ROUTES ======

// Public submission route

app.post('/api/characters/submit', upload.single('image'), async (req, res) => {

try {

const { name, age, bio, backgroundStory, submitterEmail } = req.body;

// Calculate derived values

const parsedAge = parseInt(age);

const birthYear = CURRENT\_NEXUS\_YEAR - parsedAge;

const maturityCategory = getMaturityCategory(parsedAge);

const timePeriod = getTimePeriod(birthYear);

const episodeAlignment = getEpisodeAlignment(birthYear);

// Calculate technical attributes

const coreIntegrityIndex = Math.min(100, Math.max(30, Math.floor(40 + (parsedAge/1000) \* 60)));

const algorithmComplexityFactor = Math.min(1000, Math.max(50, Math.floor(100 + (parsedAge/100) \* 900)));

// Calculate TDFs

const agingRate = calculateAgingRate(parsedAge);

const temporalDataFragments = Math.floor(parsedAge \* agingRate);

// Generate appropriate entropy mitigation techniques based on age

const mitigationTechniques = [];

if (parsedAge > 90) mitigationTechniques.push("Algorithm Optimization");

if (parsedAge > 180) mitigationTechniques.push("Entropy Dispersion");

if (parsedAge > 500) mitigationTechniques.push("Core Recompilation");

if (parsedAge > 900) mitigationTechniques.push("Null State Meditation");

if (parsedAge > 1800) mitigationTechniques.push("Quantum Entanglement");

if (parsedAge > 9000) mitigationTechniques.push("Dimensional Shifting");

// Create tracking ID

const trackingId = generateTrackingId();

// Create character submission

const newCharacter = new CharacterSubmission({

name,

submitterEmail,

age: parsedAge,

birthYear,

maturityCategory,

timePeriod,

episodeAlignment,

coreIntegrityIndex,

algorithmComplexityFactor,

temporalDataFragments,

mitigationTechniques,

bio,

backgroundStory,

imageUrl: req.file ? `/uploads/characters/${req.file.filename}` : null,

trackingId,

status: 'pending'

});

// Save to database

await newCharacter.save();

// Run AI verification

const aiVerification = await verifyCharacterWithAI(newCharacter);

// Update with AI verification results

newCharacter.aiVerificationNotes = aiVerification.notes;

newCharacter.status = aiVerification.passed ? 'ai\_verified' : 'ai\_flagged';

await newCharacter.save();

// Add to Notion as draft

const notionId = await addCharacterToNotion(newCharacter, true);

newCharacter.notionId = notionId;

await newCharacter.save();

// Send confirmation email

await sendStatusUpdateEmail(newCharacter, newCharacter.status);

res.status(201).json({

message: 'Character submitted successfully',

trackingId: trackingId,

status: newCharacter.status

});

} catch (error) {

console.error('Character submission error:', error);

res.status(500).json({ message: 'Error submitting character', error: error.message });

}

});

// Check submission status by tracking ID

app.get('/api/characters/status/:trackingId', async (req, res) => {

try {

const character = await CharacterSubmission.findOne({ trackingId: req.params.trackingId });

if (!character) {

return res.status(404).json({ message: 'Character submission not found' });

}

res.json({

trackingId: character.trackingId,

name: character.name,

status: character.status,

submissionDate: character.submissionDate,

decisionDate: character.decisionDate

});

} catch (error) {

res.status(500).json({ message: 'Error checking status', error: error.message });

}

});

// Admin login

app.post('/api/admin/login', async (req, res) => {

try {

const { username, password } = req.body;

// Find user

const user = await AdminUser.findOne({ username });

if (!user) {

return res.status(401).json({ message: 'Invalid credentials' });

}

// Check password

const isPasswordValid = await bcrypt.compare(password, user.password);

if (!isPasswordValid) {

return res.status(401).json({ message: 'Invalid credentials' });

}

// Generate token

const token = jwt.sign(

{ id: user.\_id, username: user.username, role: user.role },

process.env.JWT\_SECRET,

{ expiresIn: '8h' }

);

res.json({

token,

user: {

id: user.\_id,

username: user.username,

email: user.email,

role: user.role

}

});

} catch (error) {

res.status(500).json({ message: 'Login error', error: error.message });

}

});

// Get all submissions (admin)

app.get('/api/admin/submissions', authenticateToken, async (req, res) => {

try {

const { status, page = 1, limit = 20 } = req.query;

const query = {};

if (status) {

query.status = status;

}

const options = {

sort: { submissionDate: -1 },

limit: parseInt(limit),

skip: (parseInt(page) - 1) \* parseInt(limit)

};

const submissions = await

# **.env File Example**

# Server

PORT=3000

NODE\_ENV=development

# MongoDB

MONGODB\_URI=mongodb+srv://username:password@cluster.mongodb.net/0n1-characters

# JWT Authentication

JWT\_SECRET=your\_super\_secure\_jwt\_secret\_key\_here

# Notion Integration

NOTION\_API\_KEY=secret\_your\_notion\_integration\_key

NOTION\_CHARACTER\_DATABASE\_ID=your\_character\_database\_id

NOTION\_PENDING\_DATABASE\_ID=your\_pending\_database\_id

# OpenAI Integration

OPENAI\_API\_KEY=sk-your\_openai\_api\_key

# Email Configuration

EMAIL\_HOST=smtp.example.com

EMAIL\_PORT=587

EMAIL\_SECURE=false

EMAIL\_USER=your\_email@example.com

EMAIL\_PASS=your\_email\_password

EMAIL\_FROM=noreply@0n1force.com

# **Frontend Example (public/index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>0N1 Character Submission</title>

<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body class="bg-gray-900 text-white">

<div class="container mx-auto px-4 py-8">

<header class="text-center mb-12">

<h1 class="text-4xl font-bold mb-2">0N1 Force Universe</h1>

<p class="text-xl text-gray-400">Character Submission Portal</p>

</header>

<div class="max-w-2xl mx-auto bg-gray-800 rounded-lg p-6 shadow-lg">

<h2 class="text-2xl font-bold mb-6">Submit Your Character</h2>

<form id="characterForm" class="space-y-6">

<div>

<label for="name" class="block text-sm font-medium mb-1">Character Name</label>

<input type="text" id="name" name="name" required class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600 focus:outline-none focus:ring-2 focus:ring-blue-500">

</div>

<div>

<label for="age" class="block text-sm font-medium mb-1">Age (in cycles)</label>

<input type="number" id="age" name="age" min="1" max="10000" required class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600 focus:outline-none focus:ring-2 focus:ring-blue-500">

<p class="text-sm text-gray-400 mt-1">Current Nexus year: 100,250 A.I.</p>

</div>

<div>

<label for="bio" class="block text-sm font-medium mb-1">Short Biography</label>

<textarea id="bio" name="bio" rows="3" required class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600 focus:outline-none focus:ring-2 focus:ring-blue-500"></textarea>

</div>

<div>

<label for="backgroundStory" class="block text-sm font-medium mb-1">Background Story</label>

<textarea id="backgroundStory" name="backgroundStory" rows="6" class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600 focus:outline-none focus:ring-2 focus:ring-blue-500"></textarea>

</div>

<div>

<label for="submitterEmail" class="block text-sm font-medium mb-1">Your Email</label>

<input type="email" id="submitterEmail" name="submitterEmail" required class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600 focus:outline-none focus:ring-2 focus:ring-blue-500">

<p class="text-sm text-gray-400 mt-1">We'll use this to notify you about your submission status.</p>

</div>

<div>

<label for="image" class="block text-sm font-medium mb-1">Character Image (optional)</label>

<input type="file" id="image" name="image" accept="image/\*" class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600 focus:outline-none focus:ring-2 focus:ring-blue-500">

</div>

<div>

<button type="submit" class="w-full bg-blue-600 hover:bg-blue-700 px-6 py-3 rounded-lg font-medium transition">Submit Character</button>

</div>

</form>

<div id="submitResult" class="hidden mt-6 p-4 rounded"></div>

</div>

<div class="max-w-2xl mx-auto mt-12 bg-gray-800 rounded-lg p-6 shadow-lg">

<h2 class="text-2xl font-bold mb-6">Check Submission Status</h2>

<form id="statusForm" class="space-y-6">

<div>

<label for="trackingId" class="block text-sm font-medium mb-1">Tracking ID</label>

<input type="text" id="trackingId" name="trackingId" required class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600 focus:outline-none focus:ring-2 focus:ring-blue-500">

</div>

<div>

<button type="submit" class="w-full bg-gray-600 hover:bg-gray-700 px-6 py-3 rounded-lg font-medium transition">Check Status</button>

</div>

</form>

<div id="statusResult" class="hidden mt-6 p-4 rounded"></div>

</div>

</div>

<script>

document.getElementById('characterForm').addEventListener('submit', async (e) => {

e.preventDefault();

const form = new FormData(e.target);

const submitResult = document.getElementById('submitResult');

try {

submitResult.innerHTML = 'Submitting character...';

submitResult.className = 'mt-6 p-4 rounded bg-blue-900 text-white';

submitResult.style.display = 'block';

const response = await fetch('/api/characters/submit', {

method: 'POST',

body: form

});

const data = await response.json();

if (response.ok) {

submitResult.innerHTML = `

<p class="font-bold">Character submitted successfully!</p>

<p class="mt-2">Your tracking ID: <span class="font-mono bg-gray-700 px-2 py-1 rounded">${data.tracking

<p class="mt-2">Your tracking ID: <span class="font-mono bg-gray-700 px-2 py-1 rounded">${data.trackingId}</span></p>

<p class="mt-2">Current status: <span class="font-bold">${formatStatus(data.status)}</span></p>

<p class="mt-4">We'll send updates to your email. You can also check your submission status using your tracking ID.</p>

`;

submitResult.className = 'mt-6 p-4 rounded bg-green-800 text-white';

e.target.reset();

} else {

submitResult.innerHTML = `<p class="font-bold">Error: ${data.message}</p>`;

submitResult.className = 'mt-6 p-4 rounded bg-red-800 text-white';

}

} catch (error) {

submitResult.innerHTML = `<p class="font-bold">Submission failed: ${error.message}</p>`;

submitResult.className = 'mt-6 p-4 rounded bg-red-800 text-white';

}

});

document.getElementById('statusForm').addEventListener('submit', async (e) => {

e.preventDefault();

const trackingId = document.getElementById('trackingId').value;

const statusResult = document.getElementById('statusResult');

try {

statusResult.innerHTML = 'Checking status...';

statusResult.className = 'mt-6 p-4 rounded bg-blue-900 text-white';

statusResult.style.display = 'block';

const response = await fetch(`/api/characters/status/${trackingId}`);

const data = await response.json();

if (response.ok) {

statusResult.innerHTML = `

<p class="font-bold">${data.name}</p>

<p class="mt-2">Status: <span class="font-bold">${formatStatus(data.status)}</span></p>

<p class="mt-2">Submitted: ${new Date(data.submissionDate).toLocaleDateString()}</p>

${data.decisionDate ? `<p class="mt-2">Decision Date: ${new Date(data.decisionDate).toLocaleDateString()}</p>` : ''}

`;

statusResult.className = 'mt-6 p-4 rounded bg-gray-700 text-white';

} else {

statusResult.innerHTML = `<p class="font-bold">Error: ${data.message}</p>`;

statusResult.className = 'mt-6 p-4 rounded bg-red-800 text-white';

}

} catch (error) {

statusResult.innerHTML = `<p class="font-bold">Lookup failed: ${error.message}</p>`;

statusResult.className = 'mt-6 p-4 rounded bg-red-800 text-white';

}

});

function formatStatus(status) {

const statusMap = {

'pending': 'Pending Review',

'ai\_verified': 'AI Verified - Awaiting Human Review',

'ai\_flagged': 'Flagged for Review',

'approved': 'Approved',

'rejected': 'Not Approved',

'canon': 'Canon - Added to 0N1 Universe'

};

return statusMap[status] || status;

}

</script>

</body>

</html>

# **Admin Dashboard (public/admin/index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>0N1 Admin Dashboard</title>

<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body class="bg-gray-900 text-white">

<div id="app">

<!-- Login Form -->

<div id="loginContainer" class="min-h-screen flex items-center justify-center">

<div class="max-w-md w-full bg-gray-800 rounded-lg p-8 shadow-lg">

<h1 class="text-3xl font-bold mb-6 text-center">0N1 Admin</h1>

<form id="loginForm" class="space-y-6">

<div>

<label for="username" class="block text-sm font-medium mb-1">Username</label>

<input type="text" id="username" name="username" required class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600">

</div>

<div>

<label for="password" class="block text-sm font-medium mb-1">Password</label>

<input type="password" id="password" name="password" required class="w-full px-4 py-2 rounded bg-gray-700 border border-gray-600">

</div>

<div>

<button type="submit" class="w-full bg-blue-600 hover:bg-blue-700 px-6 py-3 rounded-lg font-medium transition">Login</button>

</div>

<div id="loginError" class="hidden text-red-500 text-center"></div>

</form>

</div>

</div>

<!-- Main Dashboard -->

<div id="dashboardContainer" class="hidden">

<nav class="bg-gray-800 px-6 py-4 shadow">

<div class="flex items-center justify-between">

<div class="text-xl font-bold">0N1 Character Approval Dashboard</div>

<div class="flex items-center space-x-4">

<span id="userDisplay" class="text-gray-300"></span>

<button id="logoutBtn" class="text-gray-300 hover:text-white">Logout</button>

</div>

</div>

</nav>

<div class="container mx-auto px-6 py-8">

<div class="mb-8">

<h2 class="text-2xl font-bold mb-4">Submission Queue</h2>

<div class="bg-gray-800 rounded-lg shadow overflow-hidden">

<div class="px-6 py-4 border-b border-gray-700 flex space-x-4">

<button class="px-4 py-2 rounded status-filter" data-status="all">All</button>

<button class="px-4 py-2 rounded status-filter" data-status="ai\_verified">AI Verified</button>

<button class="px-4 py-2 rounded status-filter" data-status="ai\_flagged">AI Flagged</button>

<button class="px-4 py-2 rounded status-filter" data-status="pending">Pending</button>

<button class="px-4 py-2 rounded status-filter" data-status="approved">Approved</button>

<button class="px-4 py-2 rounded status-filter" data-status="rejected">Rejected</button>

</div>

<div class="overflow-x-auto">

<table class="min-w-full">

<thead>

<tr class="bg-gray-700">

<th class="px-6 py-3 text-left text-sm font-medium text-gray-300">Name</th>

<th class="px-6 py-3 text-left text-sm font-medium text-gray-300">Age</th>

<th class="px-6 py-3 text-left text-sm font-medium text-gray-300">Status</th>

<th class="px-6 py-3 text-left text-sm font-medium text-gray-300">Submitted</th>

<th class="px-6 py-3 text-left text-sm font-medium text-gray-300">Actions</th>

</tr>

</thead>

<tbody id="submissionsTable" class="divide-y divide-gray-700">

<!-- Submissions will be inserted here -->

</tbody>

</table>

</div>

<div class="px-6 py-4 bg-gray-800 border-t border-gray-700 flex justify-between">

<div>

<span id="paginationInfo">Showing 1-20 of 100</span>

</div>

<div class="flex space-x-2">

<button id="prevPage" class="px-3 py-1 rounded bg-gray-700">Previous</button>

<button id="nextPage" class="px-3 py-1 rounded bg-gray-700">Next</button>

</div>

</div>

</div>

</div>

<!-- Submission Detail View -->

<div id="submissionDetail" class="hidden bg-gray-800 rounded-lg shadow p-6">

<div class="flex justify-between items-center mb-6">

<h3 class="text-2xl font-bold" id="detailName">Character Name</h3>

<button id="closeDetail" class="text-gray-400 hover:text-white">Close</button>

</div>

<div class="grid grid-cols-1 md:grid-cols-2 gap-6">

<div>

<div class="mb-6">

<h4 class="text-lg font-medium mb-2">Basic Information</h4>

<div class="bg-gray-700 rounded p-4">

<p><span class="text-gray-400">Age:</span> <span id="detailAge"></span> cycles</p>

<p><span class="text-gray-400">Birth Year:</span> <span id="detailBirthYear"></span> A.I.</p>

<p><span class="text-gray-400">Maturity Category:</span> <span id="detailMaturity"></span></p>

<p><span class="text-gray-400">Time Period:</span> <span id="detailTimePeriod"></span></p>

</div>

</div>

<div class="mb-6">

<h4 class="text-lg font-medium mb-2">Technical Attributes</h4>

<div class="bg-gray-700 rounded p-4">

<p><span class="text-gray-400">Core Integrity Index:</span> <span id="detailCII"></span>/100</p>

<p><span class="text-gray-400">Algorithm Complexity Factor:</span> <span id="detailACF"></span></p>

<p><span class="text-gray-400">Temporal Data Fragments:</span> <span id="detailTDF"></span></p>

<p><span class="text-gray-400">Aging Rate:</span> <span id="detailAgingRate"></span></p>

</div>

</div>

<div class="mb-6">

<h4 class="text-lg font-medium mb-2">Mitigation Techniques</h4>

<div class="bg-gray-700 rounded p-4">

<ul id="detailTechniques" class="list-disc pl-5">

<!-- Will be populated with techniques -->

</ul>

</div>

</div>

</div>

<div>

<div class="mb-6">

<h4 class="text-lg font-medium mb-2">Biography</h4>

<div class="bg-gray-700 rounded p-4">

<p id="detailBio"></p>

</div>

</div>

<div class="mb-6">

<h4 class="text-lg font-medium mb-2">Background Story</h4>

<div class="bg-gray-700 rounded p-4">

<p id="detailBackground"></p>

</div>

</div>

<div class="mb-6">

<h4 class="text-lg font-medium mb-2">AI Verification Results</h4>

<div class="bg-gray-700 rounded p-4">

<p id="detailAINotes"></p>

</div>

</div>

<div class="mb-6">

<h4 class="text-lg font-medium mb-2">Moderator Notes</h4>

<textarea id="moderatorNotes" rows="4" class="w-full px-4 py-2 rounded bg-gray-600 border border-gray-500"></textarea>

</div>

<div class="flex space-x-4 mt-6">

<button id="approveBtn" class="px-6 py-3 bg-green-600 hover:bg-green-700 rounded font-medium">Approve & Add to Canon</button>

<button id="rejectBtn" class="px-6 py-3 bg-red-600 hover:bg-red-700 rounded font-medium">Reject Submission</button>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<script>

// Global variables

let currentPage = 1;

let totalPages = 1;

let currentStatus = 'all';

let currentSubmissionId = null;

let token = localStorage.getItem('adminToken');

// Check if user is logged in

function checkAuth() {

if (token) {

document.getElementById('loginContainer').classList.add('hidden');

document.getElementById('dashboardContainer').classList.remove('hidden');

// Set user display

const user = JSON.parse(localStorage.getItem('adminUser'));

document.getElementById('userDisplay').textContent = user.username;

// Load submissions

loadSubmissions();

} else {

document.getElementById('loginContainer').classList.remove('hidden');

document.getElementById('dashboardContainer').classList.add('hidden');

}

}

// Login form

document.getElementById('loginForm').addEventListener('submit', async (e) => {

e.preventDefault();

const username = document.getElementById('username').value;

const password = document.getElementById('password').value;

const errorDiv = document.getElementById('loginError');

try {

errorDiv.classList.add('hidden');

const response = await fetch('/api/admin/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ username, password }),

});

const data = await response.json();

if (response.ok) {

token = data.token;

localStorage.setItem('adminToken', token);

localStorage.setItem('adminUser', JSON.stringify(data.user));

checkAuth();

} else {

errorDiv.textContent = data.message;

errorDiv.classList.remove('hidden');

}

} catch (error) {

errorDiv.textContent = 'Login failed. Please try again.';

errorDiv.classList.remove('hidden');

}

});

// Logout

document.getElementById('logoutBtn').addEventListener('click', () => {

localStorage.removeItem('adminToken');

localStorage.removeItem('adminUser');

token = null;

checkAuth();

});

// Load submissions

async function loadSubmissions() {

try {

const response = await fetch(`/api/admin/submissions?status=${currentStatus}&page=${currentPage}`, {

headers: {

'Authorization': `Bearer ${token}`

}

});

if (response.status === 401 || response.status === 403) {

// Token expired or invalid

localStorage.removeItem('adminToken');

localStorage.removeItem('adminUser');

token = null;

checkAuth();

return;

}

const data = await response.json();

if (response.ok) {

const tableBody = document.getElementById('submissionsTable');

tableBody.innerHTML = '';

data.submissions.forEach(submission => {

const row = document.createElement('tr');

row.innerHTML = `

<td class="px-6 py-4 whitespace-nowrap">${submission.name}</td>

<td class="px-6 py-4 whitespace-nowrap">${submission.age} cycles</td>

<td class="px-6 py-4 whitespace-nowrap">${formatStatus(submission.status)}</td>

<td class="px-6 py-4 whitespace-nowrap">${new Date(submission.submissionDate).toLocaleDateString()}</td>

<td class="px-6 py-4 whitespace-nowrap">

<button data-id="${submission.\_id}" class="view-btn text-blue-400 hover:text-blue-300">View</button>

</td>

`;

tableBody.appendChild(row);

});

// Update pagination

totalPages = data.totalPages;

document.getElementById('paginationInfo').textContent =

`Showing ${(currentPage - 1) \* 20 + 1}-${Math.min(currentPage \* 20, (currentPage - 1) \* 20 + data.submissions.length)} of ${data.totalPages \* 20}`;

// Add event listeners to view buttons

document.querySelectorAll('.view-btn').forEach(btn => {

btn.addEventListener('click', () => {

viewSubmission(btn.dataset.id);

});

});

}

} catch (error) {

console.error('Error loading submissions:', error);

}

}

// View submission details

async function viewSubmission(id) {

try {

const response = await fetch(`/api/admin/submissions/${id}`, {

headers: {

'Authorization': `Bearer ${token}`

}

});

const data = await response.json();

if (response.ok) {

// Update detail view

document.getElementById('detailName').textContent = data.name;

document.getElementById('detailAge').textContent = data.age;

document.getElementById('detailBirthYear').textContent = data.birthYear;

document.getElementById('detailMaturity').textContent = data.maturityCategory;

document.getElementById('detailTimePeriod').textContent = data.timePeriod;

document.getElementById('detailCII').textContent = data.coreIntegrityIndex;

document.getElementById('detailACF').textContent = data.algorithmComplexityFactor;

document.getElementById('detailTDF').textContent = data.temporalDataFragments;

document.getElementById('detailAgingRate').textContent = calculateAgingRate(data.age).toFixed(6);

document.getElementById('detailBio').textContent = data.bio;

document.getElementById('detailBackground').textContent = data.backgroundStory || 'Not provided';

document.getElementById('detailAINotes').textContent = data.aiVerificationNotes || 'No AI verification notes available';

// Mitigation techniques

const techniquesList = document.getElementById('detailTechniques');

techniquesList.innerHTML = '';

if (data.mitigationTechniques && data.mitigationTechniques.length > 0) {

data.mitigationTechniques.forEach(technique => {

const li = document.createElement('li');

li.textContent = technique;

techniquesList.appendChild(li);

});

} else {

const li = document.createElement('li');

li.textContent = 'None';

techniquesList.appendChild(li);

}

// Moderator notes

document.getElementById('moderatorNotes').value = data.moderatorNotes || '';

// Show detail view

document.getElementById('submissionDetail').classList.remove('hidden');

// Store current ID

currentSubmissionId = id;

// Disable/enable buttons based on status

if (data.status === 'approved' || data.status === 'rejected' || data.status === 'canon') {

document.getElementById('approveBtn').disabled = true;

document.getElementById('rejectBtn').disabled = true;

document.getElementById('approveBtn').classList.add('opacity-50');

document.getElementById('rejectBtn').classList.add('opacity-50');

} else {

document.getElementById('approveBtn').disabled = false;

document.getElementById('rejectBtn').disabled = false;

document.getElementById('approveBtn').classList.remove('opacity-50');

document.getElementById('rejectBtn').classList.remove('opacity-50');

}

}

} catch (error) {

console.error('Error loading submission details:', error);

}

}

// Close detail view

document.getElementById('closeDetail').addEventListener('click', () => {

document.getElementById('submissionDetail').classList.add('hidden');

currentSubmissionId = null;

});

// Approve submission

document.getElementById('approveBtn').addEventListener('click', async () => {

if (!currentSubmissionId) return;

try {

const moderatorNotes = document.getElementById('moderatorNotes').value;

const response = await fetch(`/api/admin/submissions/${currentSubmissionId}`, {

method: 'PUT',

headers: {

'Authorization': `Bearer ${token}`,

'Content-Type': 'application/json'

},

body: JSON.stringify({

status: 'approved',

moderatorNotes

})

});

if (response.ok) {

// Close detail view

document.getElementById('submissionDetail').classList.add('hidden');

currentSubmissionId = null;

// Reload submissions

loadSubmissions();

}

} catch (error) {

console.error('Error approving submission:', error);

}

});

// Reject submission

document.getElementById('rejectBtn').addEventListener('click', async () => {

if (!currentSubmissionId) return;

try {

const moderatorNotes = document.getElementById('moderatorNotes').value;

if (!moderatorNotes) {

alert('Please provide notes explaining why this submission was rejected.');

return;

}

const response = await fetch(`/api/admin/submissions/${currentSubmissionId}`, {

method: 'PUT',

headers: {

'Authorization': `Bearer ${token}`,

'Content-Type': 'application/json'

},

body: JSON.stringify({

status: 'rejected',

moderatorNotes

})

});

if (response.ok) {

// Close detail view

document.getElementById('submissionDetail').classList.add('hidden');

currentSubmissionId = null;

// Reload submissions

loadSubmissions();

}

} catch (error) {

console.error('Error rejecting submission:', error);

}

});

// Pagination

document.getElementById('prevPage').addEventListener('click', () => {

if (currentPage > 1) {

currentPage--;

loadSubmissions();

}

});

document.getElementById('nextPage').addEventListener('click', () => {

if (currentPage < totalPages) {

currentPage++;

loadSubmissions();

}

});

// Status filters

document.querySelectorAll('.status-filter').forEach(btn => {

btn.addEventListener('click', () => {

// Remove active class from all buttons

document.querySelectorAll('.status-filter').forEach(b => {

b.classList.remove('bg-blue-600');

});

// Add active class to clicked button

btn.classList.add('bg-blue-600');

// Update current status and load submissions

currentStatus = btn.dataset.status;

currentPage = 1;

loadSubmissions();

});

});

// Helper function to calculate aging rate

function calculateAgingRate(age) {

const MATURITY\_THRESHOLD = 18;

if (age < MATURITY\_THRESHOLD) {

// Pre-maturity formula

return Math.pow(MATURITY\_THRESHOLD / (MATURITY\_THRESHOLD - age + 1), 1.5);

} else {

// Post-maturity formula

return Math.pow(MATURITY\_THRESHOLD / age, 2);

}

}

// Helper function to format status

function formatStatus(status) {

const statusMap = {

'pending': 'Pending Review',

'ai\_verified': 'AI Verified',

'ai\_flagged': 'Flagged by AI',

'approved': 'Approved',

'rejected': 'Rejected',

'canon': 'Canon'

};

const statusColors = {

'pending': 'text-yellow-400',

'ai\_verified': 'text-green-400',

'ai\_flagged': 'text-red-400',

'approved': 'text-green-400',

'rejected': 'text-red-400',

'canon': 'text-purple-400'

};

return `<span class="${statusColors[status]}">${statusMap[status] || status}</span>`;

}

// Initialize

document.addEventListener('DOMContentLoaded', () => {

checkAuth();

// Set initial active status filter

document.querySelector('[data-status="all"]').classList.add('bg-blue-600');

});

</script>

</body>

</html>