🏛️ Government Spending Tracker - Complete Codebase

This document contains the complete source code for the Government Spending Tracker project.

📋 Project Overview

- **Frontend**: React application with modern UI components
- **Backend**: FastAPI with SQLAlchemy ORM and SQLite database
- **Features**: Ministry auto-creation, role-based access, file upload, real-time budget tracking
- **Architecture**: RESTful API with JWT authentication

📁 Project Root

📄 generate_codebase_clean.py

```
import os

# Project root (change if needed)
PROJECT_ROOT = "."

# Output file
OUTPUT_FILE = "codebase.md"

# Extensions to include
INCLUDE_EXTS = (".py", ".html", ".css", ".js")

# Directories to skip
SKIP_DIRS = [".git", "__pycache__", ".venv", "node_modules",
".pytest_cache"]

with open(OUTPUT_FILE, "w", encoding="utf-8") as out:
    out.write("# 🏛️ Government Spending Tracker - Complete Codebase\n\n")
    out.write("This document contains the complete source code for the
Government Spending Tracker project.\n\n")
    out.write("## 📋 Project Overview\n\n")
    out.write("- **Frontend**: React application with modern UI
components\n")
    out.write("- **Backend**: FastAPI with SQLAlchemy ORM and SQLite
database\n")
    out.write("- **Features**: Ministry auto-creation, role-based access,
file upload, real-time budget tracking\n")
    out.write("- **Architecture**: RESTful API with JWT
authentication\n\n")
    out.write("---\n\n")

    for root, dirs, files in os.walk(PROJECT_ROOT):
        # Skip virtual envs or hidden folders
```

```
        if any(skip in root for skip in SKIP_DIRS):
            continue

        # Write folder name
        rel_path = os.path.relpath(root, PROJECT_ROOT)
        if rel_path == ".":
            rel_path = "📁 Project Root"
        else:
            rel_path = f"📁 {rel_path}"

        out.write(f"\n\n## {rel_path}\n\n")

        # Filter and sort files
        code_files = [f for f in sorted(files) if
f.endswith(INCLUDE_EXTS)]

        if not code_files:
            continue

        for file in code_files:
            filepath = os.path.join(root, file)

            # Get file extension for syntax highlighting
            ext = file.split(".")[-1] if "." in file else "text"
            syntax_map = {"py": "python", "js": "javascript", "html":
"html", "css": "css"}
            syntax = syntax_map.get(ext, ext)

            out.write(f"\n### 📄 {file}\n\n")
            out.write(f"```{syntax}\n")

            try:
                with open(filepath, "r", encoding="utf-8") as f:
                    content = f.read()
                    out.write(content)
            except Exception as e:
                out.write(f"⚠ Could not read file: {e}")

            out.write("\n```\n")

print(f"✅ Clean codebase exported to {OUTPUT_FILE}")
print("📊 File size:", os.path.getsize(OUTPUT_FILE), "bytes")
print("\n📖 To convert to PDF:")
print("   pandoc codebase.md -o codebase.pdf")
print("\n📖 To view in browser:")
print("   open codebase.md  # macOS")
print("   xdg-open codebase.md  # Linux")
```

📁 frontend

📁 frontend/public

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed
on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the
build.
      Only files inside the `public` folder can be referenced from the
HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico"
will
      work correctly both with client-side routing and a non-root public
URL.
      Learn how to configure a non-root public URL by running `npm run
build`.
    -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?
family=Inter:wght@300;400;500;600;700;800&display=swap" rel="stylesheet">
  <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
```

```
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

📁 frontend/src

📄 App.css

```
.App {
  min-height: 100vh;
  background-color: #f5f5f5;
}

* {
  box-sizing: border-box;
}

body {
  margin: 0;
  padding: 0;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #f5f5f5;
}

.action-buttons {
  display: flex;
  gap: 8px;
  align-items: center;
}

.action-buttons .btn {
  margin: 0;
}

.page-description {
  color: #666;
  margin: 8px 0 20px 0;
  font-size: 14px;
}
```

📄 App.js

```
import React, { useState, useEffect } from 'react';
import CategoryManager from './CategoryManager';
import ProposalForm from './ProposalForm';
import ProposalsList from './ProposalsList';
```

```
import ContractUpload from './ContractUpload';
import Dashboard from './Dashboard';
import HistoryView from './HistoryView';
import Login from './Login';
import Sidebar from './Sidebar';
import RoleGuard from './RoleGuard';
import { UserProvider, useUser } from './UserContext';
import './App.css';
import './theme.css';

const Header = () => {
  const { user, logout } = useUser();

  return (
    <header className="app-header">
      <div className="header-content">
        <div className="header-left">
          <h1>🏛️ Government Spending Tracker</h1>
          <p className="header-subtitle">Transparent Budget Management
System</p>
        </div>
        {user && (
          <div className="header-actions">
            <button className="btn btn-secondary btn-small logout-btn"
onClick={logout}>
              <span>🚪 </span>
              Logout
            </button>
          </div>
        )}
      </div>
    </header>
  );
};

function AppContent() {
  const [proposalRefreshKey, setProposalRefreshKey] = useState(0);
  const { user, loading } = useUser();

  // Set default tab based on user role
  const getDefaultTab = (userRole) => {
    if (userRole === 'finance') {
      return 'dashboard';
    } else if (userRole === 'ministry') {
      return 'proposals';
    }
    return 'dashboard'; // fallback
  };

  const [activeTab, setActiveTab] = useState(() =>
getDefaultTab(user?.role));

  // Update activeTab when user changes
  useEffect(() => {
```

```
    if (user) {
      const defaultTab = getDefaultTab(user.role);
      setActiveTab(defaultTab);
    }
  }, [user]);

  const handleProposalCreated = () => {
    setProposalRefreshKey(prevKey => prevKey + 1);
  };

    const renderContent = () => {
    // Additional security check - redirect unauthorized users
    if (activeTab === 'dashboard' && user?.role !== 'finance') {
      setActiveTab('proposals'); // Redirect ministry users to proposals
      return <ProposalsList key={`proposals-${proposalRefreshKey}`} />;
    }
    if (activeTab === 'categories' && user?.role !== 'finance') {
      setActiveTab('proposals'); // Redirect ministry users to proposals
      return <ProposalsList key={`proposals-${proposalRefreshKey}`} />;
    }
    if ((activeTab === 'create-proposal' || activeTab === 'upload-
contract') && user?.role !== 'ministry') {
      setActiveTab('dashboard'); // Redirect finance users to dashboard
      return (
        <RoleGuard allowedRoles={['finance']}>
          <Dashboard key={`dashboard-${proposalRefreshKey}`} />
        </RoleGuard>
      );
    }

    switch (activeTab) {
      case 'dashboard':
        return (
          <RoleGuard allowedRoles={['finance']}>
            <Dashboard key={`dashboard-${proposalRefreshKey}`} />
          </RoleGuard>
        );
      case 'categories':
        return (
          <RoleGuard allowedRoles={['finance']}>
            <CategoryManager />
          </RoleGuard>
        );
      case 'proposals':
        return <ProposalsList key={`proposals-${proposalRefreshKey}`} />;
      case 'create-proposal':
        return (
          <RoleGuard allowedRoles={['ministry']}>
            <ProposalForm onCreated={handleProposalCreated} />
          </RoleGuard>
        );
      case 'upload-contract':
        return (
          <RoleGuard allowedRoles={['ministry']}>
```

```
              <ContractUpload onProposalsCreated={handleProposalCreated} />
            </RoleGuard>
          );
        case 'history':
          return <HistoryView />;
        default:
          // Default based on user role
          const defaultTab = getDefaultTab(user?.role);
          if (defaultTab === 'dashboard') {
            return (
              <RoleGuard allowedRoles={['finance']}>
                <Dashboard key={`dashboard-${proposalRefreshKey}`} />
              </RoleGuard>
            );
          } else {
            return <ProposalsList key={`proposals-${proposalRefreshKey}`}
/>;
          }
      }
    };

    if (loading) {
      return (
        <div className="loading-container">
          <div className="loading-spinner"></div>
          <span>Loading application...</span>
        </div>
      );
    }

    if (!user) {
      return <Login />;
    }

    return (
      <div className="App">
        <Header />
        <div className="app-layout">
          <Sidebar activeTab={activeTab} setActiveTab={setActiveTab} />
          <main className="main-content">
            <div className="content-container">
              {renderContent()}
            </div>
          </main>
        </div>
      </div>
    );
  }

  function App() {
    return (
      <UserProvider>
        <AppContent />
      </UserProvider>
```

```
    );
}

export default App;
```

App.test.js

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

ApprovalDialog.js

```
import React, { useState } from 'react';

const ApprovalDialog = ({ proposal, categories, onApprove, onReject,
onClose }) => {
  const [showApprove, setShowApprove] = useState(false);
  const [showReject, setShowReject] = useState(false);
  const [approvedAmount, setApprovedAmount] = useState('');
  const [decisionNotes, setDecisionNotes] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  const category = categories && Array.isArray(categories)
    ? categories.find(c => c.id === proposal.category_id)
    : null;
  const maxAmount = Math.min(proposal.requested_amount,
category?.remaining_budget || 0);

  const handleApprove = async () => {
    const amount = parseFloat(approvedAmount);
    if (isNaN(amount) || amount <= 0 || amount > maxAmount) {
      setError(`Invalid amount. Must be greater than 0 and not exceed
$${maxAmount.toLocaleString()}`);
      return;
    }

    setLoading(true);
    setError('');

    try {
```

```
      await onApprove(proposal.id, { approved_amount: amount,
decision_notes: decisionNotes || null });
    } catch (err) {
      setError('Failed to approve proposal. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  const handleReject = async () => {
    setLoading(true);
    setError('');

    try {
      await onReject(proposal.id, { decision_notes: decisionNotes || null
});
    } catch (err) {
      setError('Failed to reject proposal. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  if (showApprove) {
    return (
      <div className="modal-overlay">
        <div className="modal">
          <h3>Approve Proposal</h3>
          <p><strong>{proposal.title}</strong> - {proposal.ministry}</p>
          <p>Requested: ${proposal.requested_amount.toLocaleString()}</p>
          <p>Category remaining:
${category?.remaining_budget?.toLocaleString() || 'N/A'}</p>

          {error && (
            <div className="error-message">
              <span>⚠</span> {error}
            </div>
          )}

          <div className="form-group">
            <label>Approved Amount (max: ${maxAmount.toLocaleString()})
</label>
            <input
              type="number"
              value={approvedAmount}
              onChange={(e) => setApprovedAmount(e.target.value)}
              min="0"
              step="1000"
              placeholder={maxAmount.toString()}
              disabled={loading}
            />
          </div>

          <div className="form-group">
```

```
              <label>Decision Notes (optional)</label>
              <textarea
                value={decisionNotes}
                onChange={(e) => setDecisionNotes(e.target.value)}
                placeholder="Reason for approval..."
                rows="3"
                disabled={loading}
              />
            </div>

            <div className="form-actions">
              <button
                className="btn btn-primary"
                onClick={handleApprove}
                disabled={loading}
              >
                {loading ? 'Approving...' : 'Approve'}
              </button>
              <button
                className="btn btn-secondary"
                onClick={() => setShowApprove(false)}
                disabled={loading}
              >
                Cancel
              </button>
            </div>
          </div>
        </div>
      );
    }

    if (showReject) {
      return (
        <div className="modal-overlay">
          <div className="modal">
            <h3>Reject Proposal</h3>
            <p><strong>{proposal.title}</strong> - {proposal.ministry}</p>
            <p>Requested: ${proposal.requested_amount.toLocaleString()}</p>

            {error && (
              <div className="error-message">
                <span>⚠</span> {error}
              </div>
            )}

            <div className="form-group">
              <label>Rejection Reason (optional)</label>
              <textarea
                value={decisionNotes}
                onChange={(e) => setDecisionNotes(e.target.value)}
                placeholder="Reason for rejection..."
                rows="3"
                disabled={loading}
              />
```

```
            </div>

            <div className="form-actions">
              <button
                className="btn btn-danger"
                onClick={handleReject}
                disabled={loading}
              >
                {loading ? 'Rejecting...' : 'Reject'}
              </button>
              <button
                className="btn btn-secondary"
                onClick={() => setShowReject(false)}
                disabled={loading}
              >
                Cancel
              </button>
            </div>
          </div>
        </div>
      );
    }

    return (
      <div className="modal-overlay">
        <div className="modal">
          <h3>Decision Required</h3>
          <p><strong>{proposal.title}</strong> - {proposal.ministry}</p>
          <p>Requested: ${proposal.requested_amount.toLocaleString()}</p>

          <div className="form-actions">
            <button className="btn btn-primary" onClick={() =>
setShowApprove(true)}>Approve</button>
            <button className="btn btn-danger" onClick={() =>
setShowReject(true)}>Reject</button>
            <button className="btn btn-secondary" onClick=
{onClose}>Cancel</button>
          </div>
        </div>
      </div>
    );
};

export default ApprovalDialog;
```

CategoryManager.css

```
.category-manager {
  max-width: 1200px;
  margin: 0 auto;
```

```
    padding: 20px;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.header {
    text-align: center;
    margin-bottom: 30px;
}

.header h1 {
    color: #2c3e50;
    margin-bottom: 10px;
    font-size: 2.5rem;
}

.header h2 {
    color: #7f8c8d;
    margin-bottom: 20px;
    font-size: 1.5rem;
}

.error {
    background-color: #e74c3c;
    color: white;
    padding: 10px;
    border-radius: 5px;
    margin-bottom: 20px;
}

.loading {
    text-align: center;
    font-size: 1.2rem;
    color: #7f8c8d;
    padding: 40px;
}

.actions {
    display: flex;
    gap: 10px;
    margin-bottom: 30px;
}

.btn {
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 1rem;
    transition: background-color 0.3s;
}

.btn-primary {
    background-color: #3498db;
    color: white;
```

```
}

.btn-primary:hover {
  background-color: #2980b9;
}

.btn-secondary {
  background-color: #95a5a6;
  color: white;
}

.btn-secondary:hover {
  background-color: #7f8c8d;
}

.btn-danger {
  background-color: #e74c3c;
  color: white;
}

.btn-danger:hover {
  background-color: #c0392b;
}

.btn-small {
  padding: 5px 10px;
  font-size: 0.9rem;
  margin-right: 5px;
}

.form-container {
  background-color: #f8f9fa;
  padding: 20px;
  border-radius: 8px;
  margin-bottom: 30px;
  border: 1px solid #dee2e6;
}

.form-container h3 {
  margin-top: 0;
  color: #2c3e50;
}

.form-group {
  margin-bottom: 15px;
}

.form-group label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
  color: #2c3e50;
}
```

```css
.form-group input {
  width: 100%;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 1rem;
  box-sizing: border-box;
}

.form-group input:focus {
  outline: none;
  border-color: #3498db;
  box-shadow: 0 0 5px rgba(52, 152, 219, 0.3);
}

.form-actions {
  display: flex;
  gap: 10px;
}

.categories-table {
  background-color: white;
  border-radius: 8px;
  overflow: hidden;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.categories-table h3 {
  background-color: #2c3e50;
  color: white;
  margin: 0;
  padding: 15px 20px;
}

table {
  width: 100%;
  border-collapse: collapse;
}

th, td {
  padding: 12px 20px;
  text-align: left;
  border-bottom: 1px solid #dee2e6;
}

th {
  background-color: #f8f9fa;
  font-weight: bold;
  color: #2c3e50;
}

tr:hover {
  background-color: #f8f9fa;
}
```

```
.no-data {
  text-align: center;
  padding: 40px;
  color: #7f8c8d;
  font-style: italic;
}

/* Responsive design */
@media (max-width: 768px) {
  .category-manager {
    padding: 10px;
  }

  .header h1 {
    font-size: 2rem;
  }

  .actions {
    flex-direction: column;
  }

  table {
    font-size: 0.9rem;
  }

  th, td {
    padding: 8px 10px;
  }
}

/* Modal styles for approval dialogs */
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 1000;
}

.modal {
  background: white;
  padding: 20px;
  border-radius: 8px;
  max-width: 500px;
  width: 90%;
  max-height: 80vh;
  overflow-y: auto;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
```

```
}

.modal h3 {
  margin-top: 0;
  color: #2c3e50;
}

.modal p {
  margin: 10px 0;
  color: #7f8c8d;
}

.modal textarea {
  width: 100%;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 1rem;
  box-sizing: border-box;
  resize: vertical;
}

.modal textarea:focus {
  outline: none;
  border-color: #3498db;
  box-shadow: 0 0 5px rgba(52, 152, 219, 0.3);
}
```

CategoryManager.js

```
import React, { useState, useEffect } from 'react';
import { categoryAPI } from './api';
import './CategoryManager.css';

const CategoryManager = () => {
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [showForm, setShowForm] = useState(false);
  const [editingCategory, setEditingCategory] = useState(null);
  const [formData, setFormData] = useState({
    name: '',
    allocated_budget: ''
  });

  // Load categories on component mount
  useEffect(() => {
    loadCategories();
  }, []);
```

```
  const loadCategories = async () => {
    try {
      setLoading(true);
      const data = await categoryAPI.getAll();
      setCategories(data);
      // notify other parts of the app (e.g., ProposalForm)
      if (typeof window !== "undefined") {
        window.dispatchEvent(new Event("categories-updated"));
      }
      setError(null);
    } catch (err) {
      // Handle different error formats
      let errorMessage = 'Failed to load categories';

      if (err?.response?.data) {
        const errorData = err.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    } finally {
      setLoading(false);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(null);

    try {
      // Validate name
      if (!formData.name.trim()) {
        setError('Category name is required.');
        return;
      }
```

```
      // Validate budget
      const budgetString = (formData.allocated_budget ||
'').toString().trim();
      if (!budgetString) {
        setError('Allocated budget is required.');
        return;
      }

      const budget = parseFloat(budgetString);
      if (isNaN(budget) || budget <= 0) {
        setError('Allocated budget must be a number greater than 0.');
        return;
      }

      if (editingCategory) {
        await categoryAPI.update(editingCategory.id, {
          name: formData.name,
          allocated_budget: budget
        });
      } else {
        await categoryAPI.create({
          name: formData.name,
          allocated_budget: budget
        });
      }

      setFormData({ name: '', allocated_budget: '' });
      setShowForm(false);
      setEditingCategory(null);
      await loadCategories();
    } catch (err) {
      // Handle different error formats
      let errorMessage = editingCategory ? 'Failed to update category' :
'Failed to create category';

      if (err?.response?.data) {
        const errorData = err.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
```

```
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    }
  };

  const handleEdit = (category) => {
    setEditingCategory(category);
    setFormData({
      name: category.name || '',
      allocated_budget: category.allocated_budget ?
category.allocated_budget.toString() : ''
    });
    setShowForm(true);
  };

  const handleDelete = async (id) => {
    if (!window.confirm('Are you sure you want to delete this category?'))
{
      return;
    }

    try {
      await categoryAPI.delete(id);
      await loadCategories();
    } catch (err) {
      // Handle different error formats
      let errorMessage = 'Failed to delete category';

      if (err?.response?.data) {
        const errorData = err.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
```

```
        }
      }

      setError(errorMessage);
    }
  };

  const handleCancel = () => {
    setFormData({ name: '', allocated_budget: '' });
    setShowForm(false);
    setEditingCategory(null);
    setError(null);
  };

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

  if (loading) {
    return (
      <div className="card">
        <h2>Category Management</h2>
        <div className="loading">Loading categories...</div>
      </div>
    );
  }

  return (
    <div className="card">
      <h2>Category Management</h2>
      {error && <div className="error-message">{error}</div>}

      {!showForm ? (
        <div>
          <div className="form-actions">
            <button
              className="btn btn-primary"
              onClick={() => setShowForm(true)}
            >
              Add New Category
            </button>
          </div>

          <table className="categories-table">
            <thead>
              <tr>
                <th>Name</th>
                <th>Allocated Budget</th>
                <th>Remaining Budget</th>
                <th>Created</th>
                <th>Actions</th>
              </tr>
            </thead>
```

```
              <tbody>
                {categories.map(category => (
                  <tr key={category.id}>
                    <td>{category.name}</td>
                    <td>${category.allocated_budget.toLocaleString()}</td>
                    <td>${category.remaining_budget.toLocaleString()}</td>
                    <td>{new Date(category.created_at).toLocaleDateString()}
</td>
                    <td>
                      <button
                        className="btn btn-small btn-secondary"
                        onClick={() => handleEdit(category)}
                      >
                        Edit
                      </button>
                      <button
                        className="btn btn-small btn-secondary"
                        onClick={() => handleDelete(category.id)}
                        style={{ marginLeft: '8px', backgroundColor:
'#ef4444', color: 'white' }}
                      >
                        Delete
                      </button>
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        ) : (
          <form onSubmit={handleSubmit}>
            <div className="form-group">
              <label htmlFor="name">Category Name</label>
              <input
                type="text"
                id="name"
                name="name"
                value={formData.name}
                onChange={handleInputChange}
                placeholder="e.g., Education, Health, Defense"
                required
              />
            </div>

            <div className="form-group">
              <label htmlFor="allocated_budget">Allocated Budget</label>
              <input
                type="text"
                id="allocated_budget"
                name="allocated_budget"
                value={formData.allocated_budget}
                onChange={handleInputChange}
                placeholder="e.g., 1000000"
                pattern="[0-9]*"
```

```
                inputMode="numeric"
                required
              />
            </div>

            <div className="form-actions">
              <button type="submit" className="btn btn-primary">
                {editingCategory ? 'Update Category' : 'Create Category'}
              </button>
              <button
                type="button"
                className="btn btn-secondary"
                onClick={handleCancel}
              >
                Cancel
              </button>
            </div>
          </form>
        )}
      </div>
    );
};

export default CategoryManager;
```

ContractUpload.js

```
import React, { useState } from 'react';
import { uploadAPI, proposalAPI } from './api';

const ContractUpload = ({ onCreated }) => {
  const [file, setFile] = useState(null);
  const [drafts, setDrafts] = useState([]);
  const [error, setError] = useState(null);
  const [parsing, setParsing] = useState(false);

  const onSelect = (e) => {
    setFile(e.target.files[0] || null);
    setDrafts([]);
    setError(null);
  };

    const onParse = async () => {
    if (!file) { setError('Please choose a JSON or CSV file.'); return; }
    try {
      setParsing(true);
      const res = await uploadAPI.parse(file);

      // Get existing proposals to check for duplicates
      const existingProposals = await proposalAPI.getAll();
```

```
      // Initialize row-level flags and check for existing proposals
      const initializedDrafts = (res.drafts || []).map(d => {
        const isExisting = existingProposals.some(existing =>
          existing.ministry_name === d.ministry_name &&
          existing.title === d.title &&
          existing.requested_amount === d.requested_amount
        );

        return {
          ...d,
          isCreating: false,
          isCreated: isExisting  // Mark as created if it already exists
        };
      });

      setDrafts(initializedDrafts);
      setError(null);
    } catch (e) {
      // Handle different error formats
      let errorMessage = 'Failed to parse file';

      if (e?.response?.data) {
        const errorData = e.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    } finally {
      setParsing(false);
    }
  };

  const createProposal = async (d, idx) => {
    try {
```

```
      if (!d.valid) { return; }
      // Guard: skip if already created or being created
      if (d.isCreating || d.isCreated) { return; }

      // Mark as creating
      setDrafts(prev => prev.map((x, i) => i === idx ? { ...x, isCreating:
true } : x));

      await proposalAPI.create({
        ministry_name: d.ministry_name,
        category_id: d.category_id,
        title: d.title,
        description: d.description || null,
        requested_amount: d.requested_amount
      });

      // Mark as created
      setDrafts(prev => prev.map((x, i) => i === idx ? { ...x, isCreating:
false, isCreated: true } : x));
      onCreated && onCreated();
    } catch (e) {
      // Reset creating flag on error
      setDrafts(prev => prev.map((x, i) => i === idx ? { ...x, isCreating:
false } : x));

      // Handle different error formats
      let errorMessage = 'Failed to create proposal from draft';

      if (e?.response?.data) {
        const errorData = e.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    }
```

```
  };

  return (
    <div className="form-container" style={{ marginTop: 20 }}>
      <h3>Contract Upload</h3>
      {error && <div className="error">{error}</div>}
      <div className="form-group">
        <input type="file" accept=".json,.csv" onChange={onSelect} />
      </div>
      <div className="form-actions">
        <button className="btn btn-primary" onClick={onParse} disabled=
{parsing}>
          {parsing ? 'Parsing...' : 'Parse'}
        </button>
      </div>

      {drafts.length > 0 && (
        <div className="categories-table" style={{ marginTop: 20 }}>
          <h3>Parsed Drafts</h3>
          <table>
            <thead>
              <tr>
                <th>Valid</th>
                <th>Ministry</th>
                <th>Category</th>
                <th>Title</th>
                <th>Requested</th>
                <th>Action</th>
              </tr>
            </thead>
            <tbody>
              {drafts.map((d, i) => (
                <tr key={i}>
                  <td>{d.valid ? 'Yes' : 'No'}</td>
                  <td>{d.ministry_name || '-'}</td>
                  <td>{d.category_name || '-'}</td>
                  <td>{d.title || '-'}</td>
                  <td>{d.requested_amount != null ? new
Intl.NumberFormat('en-US', { style: 'currency', currency: 'USD',
minimumFractionDigits: 0 }).format(d.requested_amount) : '-'}</td>
                  <td>
                    <button
                      className="btn btn-small btn-primary"
                      disabled={!d.valid || d.isCreating || d.isCreated}
                      onClick={() => createProposal(d, i)}
                    >
                      {d.isCreated ? 'Created' : (d.isCreating ?
'Creating…' : 'Create Proposal')}
                    </button>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
```

```
          </div>
        )}
        <div className='form-actions' style={{ marginTop: 10 }}>
          <button className="btn btn-secondary" onClick={() => {
setDrafts([]); setFile(null); setError(null); }}>
            Clear All
          </button>
        </div>
      </div>
    );
};

export default ContractUpload;
```

Dashboard.js

```
import React, { useEffect, useState } from 'react';
import { Bar } from 'react-chartjs-2';
import { Chart as ChartJS, ArcElement, Tooltip, Legend, BarElement,
CategoryScale, LinearScale } from 'chart.js';
import { dashboardAPI } from './api';

ChartJS.register(ArcElement, Tooltip, Legend, BarElement, CategoryScale,
LinearScale);

const Dashboard = ({ refreshKey }) => {
  const chartOptions = {
    responsive: true,
    maintainAspectRatio: true,
    aspectRatio: 3,
    layout: { padding: 0 },
    plugins: {
      legend: {
        position: 'bottom',
        labels: {
          boxWidth: 12,
          font: { size: 11, family: 'Inter' },
          padding: 15
        }
      },
      tooltip: {
        titleFont: { size: 12, family: 'Inter' },
        bodyFont: { size: 11, family: 'Inter' },
        backgroundColor: 'rgba(0, 0, 0, 0.8)',
        titleColor: 'white',
        bodyColor: 'white',
        borderColor: 'rgba(255, 255, 255, 0.1)',
        borderWidth: 1,
        cornerRadius: 8
      }
```

```
      },
      scales: {
        x: {
          ticks: {
            font: { size: 10, family: 'Inter' },
            color: 'var(--color-text-secondary)'
          },
          grid: {
            color: 'rgba(0, 0, 0, 0.05)'
          }
        },
        y: {
          ticks: {
            font: { size: 10, family: 'Inter' },
            color: 'var(--color-text-secondary)',
            callback: function(value) {
              return '$' + value.toLocaleString();
            }
          },
          grid: {
            color: 'rgba(0, 0, 0, 0.05)'
          }
        }
      }
    };

    const [summary, setSummary] = useState(null);
    const [error, setError] = useState(null);
    const [loading, setLoading] = useState(true);

    const load = async () => {
      try {
        setLoading(true);
        setError(null);
        const data = await dashboardAPI.getSummary();
        setSummary(data);
      } catch (err) {
        setError(err?.response?.data?.detail || 'Failed to load dashboard
data');
        console.error('Dashboard error:', err);
      } finally {
        setLoading(false);
      }
    };

    useEffect(() => {
      load();
    }, [refreshKey]);

    if (loading) {
      return (
        <div className="card dashboard-card">
          <div className="card-header">
            <h2>📊 Dashboard</h2>
```

```
            <p>Financial overview and analytics</p>
          </div>
          <div className="loading">
            <div className="loading-spinner"></div>
            <span>Loading dashboard data...</span>
          </div>
        </div>
      );
    }

    if (error) {
      return (
        <div className="card dashboard-card">
          <div className="card-header">
            <h2>📊 Dashboard</h2>
            <p>Financial overview and analytics</p>
          </div>
          <div className="error-message">
            <span>⚠</span> {error}
          </div>
          <button className="btn btn-primary" onClick={load}>
            <span>🔄 </span>
            Retry
          </button>
        </div>
      );
    }

    if (!summary) {
      return (
        <div className="card dashboard-card">
          <div className="card-header">
            <h2>📊 Dashboard</h2>
            <p>Financial overview and analytics</p>
          </div>
          <div className="error-message">
            <span>📭 </span> No data available
          </div>
        </div>
      );
    }

    // Category budget chart
    const categoryChartData = {
      labels: summary.categories.map(c => c.name),
      datasets: [
        {
          label: 'Allocated Budget',
          data: summary.categories.map(c => c.allocated_budget),
          backgroundColor: 'rgba(37, 99, 235, 0.8)',
          borderColor: 'rgba(37, 99, 235, 1)',
          borderWidth: 2,
          borderRadius: 4,
          borderSkipped: false,
```

```
      },
      {
        label: 'Remaining Budget',
        data: summary.categories.map(c => c.remaining_budget),
        backgroundColor: 'rgba(16, 185, 129, 0.8)',
        borderColor: 'rgba(16, 185, 129, 1)',
        borderWidth: 2,
        borderRadius: 4,
        borderSkipped: false,
      }
    ]
  };

  // Ministry spending chart
  const ministryChartData = {
    labels: summary.ministries.map(m => m.ministry),
    datasets: [
      {
        label: 'Requested Amount',
        data: summary.ministries.map(m => m.requested_total),
        backgroundColor: 'rgba(239, 68, 68, 0.8)',
        borderColor: 'rgba(239, 68, 68, 1)',
        borderWidth: 2,
        borderRadius: 4,
        borderSkipped: false,
      },
      {
        label: 'Approved Amount',
        data: summary.ministries.map(m => m.approved_total),
        backgroundColor: 'rgba(124, 58, 237, 0.8)',
        borderColor: 'rgba(124, 58, 237, 1)',
        borderWidth: 2,
        borderRadius: 4,
        borderSkipped: false,
      }
    ]
  };

  return (
    <div className="card dashboard-card">
      <div className="card-header">
        <h2>📊 Dashboard</h2>
        <p>Financial overview and analytics</p>
      </div>

      {/* KPI Summary */}
      <div className="kpi-summary">
        <div className="kpi-item">
          <div className="kpi-icon">💰</div>
          <h3>Total Allocated</h3>
          <p>${summary.kpis.total_allocated.toLocaleString()}</p>
        </div>
        <div className="kpi-item">
          <div className="kpi-icon">💳</div>
```

```
            <h3>Total Remaining</h3>
            <p>${summary.kpis.total_remaining.toLocaleString()}</p>
          </div>
          <div className="kpi-item">
            <div className="kpi-icon">✅</div>
            <h3>Total Approved</h3>
            <p>${summary.kpis.total_approved.toLocaleString()}</p>
          </div>
          <div className="kpi-item">
            <div className="kpi-icon">📈</div>
            <h3>Utilization Rate</h3>
            <p>{((summary.kpis.total_allocated -
summary.kpis.total_remaining) / summary.kpis.total_allocated *
100).toFixed(1)}%</p>
          </div>
        </div>

        {/* Charts */}
        <div className="charts-grid">
          <div className="chart-section">
            <div className="chart-header">
              <h3>📊 Budget by Category</h3>
              <p>Allocated vs Remaining Budget</p>
            </div>
            <div className="chart-container">
              <Bar data={categoryChartData} options={chartOptions} />
            </div>
          </div>

          <div className="chart-section">
            <div className="chart-header">
              <h3>🏛️ Spending by Ministry</h3>
              <p>Requested vs Approved Amounts</p>
            </div>
            <div className="chart-container">
              <Bar data={ministryChartData} options={chartOptions} />
            </div>
          </div>
        </div>
      </div>
    );
};

export default Dashboard;
```

📄 EditProposalDialog.js

```
import React, { useState } from 'react';

const EditProposalDialog = ({ proposal, categories, onSubmit, onClose })
```

```
=> {
  const [formData, setFormData] = useState({
    title: proposal.title || '',
    description: proposal.description || '',
    requested_amount: proposal.requested_amount?.toString() || '',
    category_id: proposal.category_id?.toString() || ''
  });
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(false);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const validateForm = () => {
    const errors = [];

    if (!formData.title.trim()) {
      errors.push('Title is required');
    }

    if (!formData.requested_amount ||
parseFloat(formData.requested_amount) <= 0) {
      errors.push('Requested amount must be greater than 0');
    }

    if (!formData.category_id) {
      errors.push('Category is required');
    }

    return errors;
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(null);

    const validationErrors = validateForm();
    if (validationErrors.length > 0) {
      setError(validationErrors.join('; '));
      return;
    }

    setLoading(true);

    try {
      const submitData = {
        title: formData.title.trim(),
        description: formData.description.trim() || null,
        requested_amount: parseFloat(formData.requested_amount),
```

```
        category_id: parseInt(formData.category_id)
      };

      await onSubmit(submitData);
    } catch (err) {
      setError(err.message || 'Failed to update proposal');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="modal-overlay">
      <div className="modal-content">
        <div className="modal-header">
          <h2>Edit Proposal #{proposal.id}</h2>
          <button className="modal-close" onClick={onClose}>×</button>
        </div>

        <form onSubmit={handleSubmit}>
          {error && <div className="error-message">{error}</div>}

          <div className="form-group">
            <label htmlFor="title">Title *</label>
            <input
              type="text"
              id="title"
              name="title"
              value={formData.title}
              onChange={handleInputChange}
              required
            />
          </div>

          <div className="form-group">
            <label htmlFor="description">Description</label>
            <textarea
              id="description"
              name="description"
              value={formData.description}
              onChange={handleInputChange}
              rows="3"
            />
          </div>

          <div className="form-group">
            <label htmlFor="requested_amount">Requested Amount *</label>
            <input
              type="text"
              id="requested_amount"
              name="requested_amount"
              value={formData.requested_amount}
              onChange={handleInputChange}
              placeholder="e.g., 1000000"
```

```
                pattern="[0-9]*"
                inputMode="numeric"
                required
              />
            </div>

            <div className="form-group">
              <label htmlFor="category_id">Category *</label>
              <select
                id="category_id"
                name="category_id"
                value={formData.category_id}
                onChange={handleInputChange}
                required
              >
                <option value="">Select Category</option>
                {categories.map(category => (
                  <option key={category.id} value={category.id}>
                    {category.name}
                  </option>
                ))}
              </select>
            </div>

            <div className="form-actions">
              <button
                type="submit"
                className="btn btn-primary"
                disabled={loading}
              >
                {loading ? 'Updating...' : 'Update Proposal'}
              </button>
              <button
                type="button"
                className="btn btn-secondary"
                onClick={onClose}
                disabled={loading}
              >
                Cancel
              </button>
            </div>
          </form>
        </div>
      </div>
    );
};

export default EditProposalDialog;
```

HistoryView.js

```
import React, { useEffect, useState, useCallback } from 'react';
import { historyAPI, categoryAPI } from './api';

const HistoryView = () => {
  const [proposals, setProposals] = useState([]);
  const [categories, setCategories] = useState([]);
  const [filters, setFilters] = useState({
    category_id: '',
    status: '',
    date_from: '',
    date_to: ''
  });
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(false);
  const [stats, setStats] = useState({
    total: 0,
    pending: 0,
    approved: 0,
    rejected: 0,
    totalRequested: 0,
    totalApproved: 0
  });

  const cleanFilters = (f) =>
Object.fromEntries(Object.entries(f).filter(([_, v]) => v !== '' && v !=
null));

  const loadHistory = useCallback(async () => {
    try {
      setLoading(true);
      setError(null);
      const data = await historyAPI.list(cleanFilters(filters));
      setProposals(data);

      // Calculate stats
      const stats = {
        total: data.length,
        pending: data.filter(p => p.status === 'Pending').length,
        approved: data.filter(p => p.status === 'Approved').length,
        rejected: data.filter(p => p.status === 'Rejected').length,
        totalRequested: data.reduce((sum, p) => sum + (p.requested_amount
|| 0), 0),
        totalApproved: data.reduce((sum, p) => sum + (p.approved_amount ||
0), 0)
      };
      setStats(stats);
    } catch (err) {
      setError(err?.response?.data?.detail || 'Failed to load proposal
history');
      console.error('History loading error:', err);
    } finally {
      setLoading(false);
    }
```

```
  }, [filters]);

  const loadCategories = useCallback(async () => {
    try {
      const data = await categoryAPI.getAll();
      setCategories(data);
    } catch (err) {
      console.error('Failed to load categories:', err);
    }
  }, []);

  useEffect(() => {
    loadHistory();
    loadCategories();
  }, [loadHistory, loadCategories]);

  const handleFilterChange = (e) => {
    const { name, value } = e.target;
    setFilters(prev => ({ ...prev, [name]: value }));
  };

  const applyFilters = () => {
    loadHistory();
  };

  const clearFilters = () => {
    setFilters({ ministry: '', category_id: '', status: '', date_from: '',
date_to: '' });
  };

  const getStatusBadge = (status) => {
    const statusConfig = {
      'Pending': { icon: '⏳', class: 'status-pending', color: '#f59e0b'
},
      'Approved': { icon: '✅', class: 'status-approved', color: '#10b981'
},
      'Rejected': { icon: '❌', class: 'status-rejected', color: '#ef4444'
}
    };

    const config = statusConfig[status] || { icon: '?', class: 'status-
unknown', color: '#6b7280' };

    return (
      <span className={`status-badge ${config.class}`} style={{
backgroundColor: `${config.color}20`, color: config.color }}>
        <span className="status-icon">{config.icon}</span>
        {status}
      </span>
    );
  };

  const formatCurrency = (amount) => {
    return new Intl.NumberFormat('en-US', {
```

```
      style: 'currency',
      currency: 'USD',
      minimumFractionDigits: 0
    }).format(amount || 0);
  };

  const formatDate = (dateString) => {
    if (!dateString) return '-';
    return new Date(dateString).toLocaleDateString('en-US', {
      year: 'numeric',
      month: 'short',
      day: 'numeric',
      hour: '2-digit',
      minute: '2-digit'
    });
  };

  return (
    <div className="card history-card">
      <div className="card-header">
        <h2>📜 Proposal History</h2>
        <p>Complete transaction history and approval records</p>
      </div>

      {/* Statistics Cards */}
      <div className="history-stats">
        <div className="stat-card">
          <div className="stat-icon">📊</div>
          <div className="stat-content">
            <h3>Total Proposals</h3>
            <p>{stats.total}</p>
          </div>
        </div>
        <div className="stat-card">
          <div className="stat-icon">⏳</div>
          <div className="stat-content">
            <h3>Pending</h3>
            <p>{stats.pending}</p>
          </div>
        </div>
        <div className="stat-card">
          <div className="stat-icon">✅</div>
          <div className="stat-content">
            <h3>Approved</h3>
            <p>{stats.approved}</p>
          </div>
        </div>
        <div className="stat-card">
          <div className="stat-icon">❌</div>
          <div className="stat-content">
            <h3>Rejected</h3>
            <p>{stats.rejected}</p>
          </div>
        </div>
```

```
          <div className="stat-card">
            <div className="stat-icon">💰 </div>
            <div className="stat-content">
              <h3>Total Requested</h3>
              <p>{formatCurrency(stats.totalRequested)}</p>
            </div>
          </div>
          <div className="stat-card">
            <div className="stat-icon">✅ </div>
            <div className="stat-content">
              <h3>Total Approved</h3>
              <p>{formatCurrency(stats.totalApproved)}</p>
            </div>
          </div>
        </div>

        {/* Filters */}
        <div className="filters-section">
          <h3>🔍  Filters</h3>
          <div className="filters-grid">
            <div className="form-group">
              <label htmlFor="ministry">Ministry</label>
              <input
                type="text"
                id="ministry"
                name="ministry"
                value={filters.ministry}
                onChange={handleFilterChange}
                placeholder="Filter by ministry"
              />
            </div>

            <div className="form-group">
              <label htmlFor="status">Status</label>
              <select name="status" value={filters.status} onChange=
{handleFilterChange}>
                <option value="">All Statuses</option>
                <option value="Pending">Pending</option>
                <option value="Approved">Approved</option>
                <option value="Rejected">Rejected</option>
              </select>
            </div>

            <div className="form-group">
              <label htmlFor="category_id">Category</label>
              <select name="category_id" value={filters.category_id}
onChange={handleFilterChange}>
                <option value="">All Categories</option>
                {categories.map(c => (
                  <option key={c.id} value={c.id}>{c.name}</option>
                ))}
              </select>
            </div>
```

```
            <div className="form-group">
              <label htmlFor="date_from">Date From</label>
              <input
                type="date"
                id="date_from"
                name="date_from"
                value={filters.date_from}
                onChange={handleFilterChange}
              />
            </div>

            <div className="form-group">
              <label htmlFor="date_to">Date To</label>
              <input
                type="date"
                id="date_to"
                name="date_to"
                value={filters.date_to}
                onChange={handleFilterChange}
              />
            </div>
          </div>

          <div className="filter-actions">
            <button
              className="btn btn-primary"
              onClick={applyFilters}
              disabled={loading}
            >
              {loading ? (
                <>
                  <span className="spinner"></span>
                  Loading...
                </>
              ) : (
                <>
                  <span>🔍 </span>
                  Apply Filters
                </>
              )}
            </button>
            <button className="btn btn-secondary" onClick={clearFilters}>
              <span>🔄 </span>
              Clear Filters
            </button>
          </div>
        </div>

        {/* Error Message */}
        {error && (
          <div className="error-message">
            <span>⚠</span> {error}
          </div>
        )}
```

```
      {/* Proposals Table */}
      <div className="table-section">
        <div className="table-header">
          <h3>📋 Proposal Records</h3>
          <div className="table-actions">
            <button className="btn btn-secondary btn-small">
              <span>📊</span>
              Export CSV
            </button>
          </div>
        </div>

        {loading ? (
          <div className="loading">
            <div className="loading-spinner"></div>
            <span>Loading proposal history...</span>
          </div>
        ) : proposals.length === 0 ? (
          <div className="no-data">
            <div className="no-data-icon">📪</div>
            <h3>No Proposals Found</h3>
            <p>No proposals match your current filters. Try adjusting your
search criteria.</p>
          </div>
        ) : (
          <div className="table-container">
            <table className="proposals-table">
              <thead>
                <tr>
                  <th>ID</th>
                  <th>Ministry</th>
                  <th>Category</th>
                  <th>Title</th>
                  <th>Requested</th>
                  <th>Status</th>
                  <th>Approved</th>
                  <th>Decided At</th>
                  <th>Created</th>
                </tr>
              </thead>
              <tbody>
                {proposals.map(proposal => (
                  <tr key={proposal.id}>
                    <td>
                      <span className="proposal-id">#{proposal.id}</span>
                    </td>
                    <td>
                      <span className="ministry-name">{proposal.ministry}
</span>
                    </td>
                    <td>
                      <span className="category-name">
                        {categories.find(c => c.id ===
```

```
proposal.category_id)?.name || 'Unknown'}
                      </span>
                    </td>
                    <td>
                      <span className="proposal-title" title=
{proposal.title}>
                        {proposal.title}
                      </span>
                    </td>
                    <td>
                      <span className="amount requested">
{formatCurrency(proposal.requested_amount)}</span>
                    </td>
                    <td>{getStatusBadge(proposal.status)}</td>
                    <td>
                      <span className="amount approved">
                        {proposal.approved_amount ?
formatCurrency(proposal.approved_amount) : '-'}
                      </span>
                    </td>
                    <td>
                      <span className="date">
{formatDate(proposal.decided_at)}</span>
                    </td>
                    <td>
                      <span className="date">
{formatDate(proposal.created_at)}</span>
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        )}
      </div>
    </div>
  );
};

export default HistoryView;
```

Login.js

```
import React, { useState } from 'react';
import { authAPI } from './api';
import { useUser } from './UserContext';

const Login = () => {
  const [credentials, setCredentials] = useState({ username: '', password:
'' });
```

```
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const { login } = useUser();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError('');

    try {
      const response = await authAPI.login(credentials.username,
credentials.password);
      localStorage.setItem('authToken', response.access_token);
      localStorage.setItem('user', JSON.stringify(response.user));
      login(response.user); // Use the context login function
    } catch (err) {
      // Handle different error formats
      let errorMessage = 'Login failed';

      if (err?.response?.data) {
        const errorData = err.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    } finally {
      setLoading(false);
    }
  };

  const handleChange = (e) => {
    setCredentials({
      ...credentials,
      [e.target.name]: e.target.value
    });
```

```
  };

  const handleQuickLogin = (username, password) => {
    setCredentials({ username, password });
  };

  return (
    <div className="login-container">
      <div className="login-card">
        <div className="login-header">
          <h2>🏛️ Government Spending Tracker</h2>
          <h3>Secure Access Portal</h3>
          <p>Transparent budget management and proposal tracking</p>
        </div>

        <form onSubmit={handleSubmit} className="login-form">
          <div className="form-group">
            <label htmlFor="username">Username</label>
            <input
              type="text"
              id="username"
              name="username"
              value={credentials.username}
              onChange={handleChange}
              placeholder="Enter your username"
              required
            />
          </div>

          <div className="form-group">
            <label htmlFor="password">Password</label>
            <input
              type="password"
              id="password"
              name="password"
              value={credentials.password}
              onChange={handleChange}
              placeholder="Enter your password"
              required
            />
          </div>

          {error && (
            <div className="error-message">
              <span>⚠</span> {error}
            </div>
          )}

          <button
            type="submit"
            className="btn btn-primary login-btn"
            disabled={loading}
          >
            {loading ? (
```

```
                <>
                  <span className="spinner"></span>
                  Authenticating...
                </>
              ) : (
                <>
                  <span>🔐 </span>
                  Sign In
                </>
              )}
            </button>
          </form>

          <div className="login-help">
            <h4>🚀  Quick Access</h4>
            <div className="quick-login-buttons">
              <button
                type="button"
                className="btn btn-secondary quick-btn"
                onClick={() => handleQuickLogin('finance', 'fin')}
              >
                <span>💰 </span>
                Finance User
              </button>
              <button
                type="button"
                className="btn btn-secondary quick-btn"
                onClick={() => handleQuickLogin('ministry', 'min')}
              >
                <span>🏛️ </span>
                Ministry User
              </button>
            </div>
            <div className="credentials-info">
              <p><strong>Finance:</strong> username: finance, password:
fin</p>
              <p><strong>Ministry:</strong> username: ministry, password:
min</p>
            </div>
          </div>
        </div>
      </div>
    );
};

export default Login;
```

📄 ProposalForm.js

```
import React, { useState, useEffect } from 'react';
import { proposalAPI, categoryAPI, ministryAPI } from './api';
import { useUser } from './UserContext';

const ProposalForm = ({ onCreated }) => {
  const { user } = useUser();
  const [categories, setCategories] = useState([]);
  const [form, setForm] = useState({
    ministry_name: '',
    category_id: '',
    title: '',
    description: '',
    requested_amount: ''
  });
  const [error, setError] = useState(null);
  const [defaultMinistrySet, setDefaultMinistrySet] = useState(false);

  const onChange = (e) => {
    const { name, value } = e.target;
    setForm(prev => ({ ...prev, [name]: value }));
  };

  const onSubmit = async (e) => {
    e.preventDefault();
    setError(null);
    // Client-side validation
    if (!form.ministry_name || !form.title || !form.category_id ||
!form.requested_amount) {
      setError('Please fill all required fields.');
      return;
    }
    const amount = parseFloat(form.requested_amount);
    if (isNaN(amount) || amount <= 0) {
      setError('Requested amount must be a number greater than 0.');
      return;
    }
    try {
      await proposalAPI.create({
        ministry_name: form.ministry_name,
        category_id: parseInt(form.category_id, 10),
        title: form.title,
        description: form.description || null,
        requested_amount: amount,
      });
      setForm({ ministry_name: '', category_id: '', title: '',
description: '', requested_amount: '' });
      setDefaultMinistrySet(false); // Reset flag so default ministry can
be set again
      onCreated && onCreated();
    } catch (e) {
      // Handle different error formats
      let errorMessage = 'Failed to submit proposal.';
```

```
      if (e?.response?.data) {
        const errorData = e.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    }
  };

  useEffect(() => {
    const reload = async () => {
      try {
        const cats = await categoryAPI.getAll();
        setCategories(cats);
      } catch (e) {
        console.error('Failed to load categories:', e);
      }
    };

    reload(); // Load categories immediately
    const handler = () => reload();
    window.addEventListener('categories-updated', handler);
    return () => window.removeEventListener('categories-updated',
handler);
  }, []);

  // Set default ministry name when user loads (only once)
  useEffect(() => {
    if (user?.ministry?.name && !defaultMinistrySet) {
      setForm(prev => ({ ...prev, ministry_name: user.ministry.name }));
      setDefaultMinistrySet(true);
    }
  }, [user, defaultMinistrySet]);

  return (
```

```
    <div className="form-container">
      <h3>Submit Proposal</h3>
      {error && <div className="error">{error}</div>}
      <form onSubmit={onSubmit}>
        <div className="form-group">
          <label>Ministry</label>
          <input
            name="ministry_name"
            value={form.ministry_name}
            onChange={onChange}
            placeholder={user?.ministry?.name || "Enter ministry name"}
            required
          />
        </div>
        <div className="form-group">
          <label>Category</label>
          <select name="category_id" value={form.category_id} onChange=
{onChange} required>
            <option value="">Select category</option>
            {categories.map(c => (
              <option key={c.id} value={c.id}>{c.name}</option>
            ))}
          </select>
        </div>
        <div className="form-group">
          <label>Title</label>
          <input name="title" value={form.title} onChange={onChange}
placeholder="Project title" required />
        </div>
        <div className="form-group">
          <label>Description</label>
          <input name="description" value={form.description} onChange=
{onChange} placeholder="Optional details" />
        </div>
        <div className="form-group">
          <label>Requested Amount</label>
          <input name="requested_amount" type="number" min="1" value=
{form.requested_amount} onChange={onChange} required />
        </div>
        <div className="form-actions">
          <button className="btn btn-primary"
type="submit">Submit</button>
        </div>
      </form>
    </div>
  );
};

export default ProposalForm;
```

ProposalsList.js

```
import React, { useEffect, useState } from 'react';
import { proposalAPI, categoryAPI } from './api';
import ApprovalDialog from './ApprovalDialog';
import EditProposalDialog from './EditProposalDialog';
import { useUser } from './UserContext';

const ProposalsList = () => {
  const [proposals, setProposals] = useState([]);
  const [categories, setCategories] = useState([]);
  const [filters, setFilters] = useState({ category_id: '', min_amount:
'', max_amount: '' });
  const [error, setError] = useState(null);
  const [selectedProposal, setSelectedProposal] = useState(null);
  const { user } = useUser();
  const [deletingId, setDeletingId] = useState(null);
  const [editingProposal, setEditingProposal] = useState(null);

  const load = async () => {
    try {
      const params = { status: 'Pending' }; // Always filter for pending
proposals
      if (filters.category_id) params.category_id = filters.category_id;
      if (filters.min_amount) params.min_amount =
parseFloat(filters.min_amount);
      if (filters.max_amount) params.max_amount =
parseFloat(filters.max_amount);
      const data = await proposalAPI.getAll(params);
      setProposals(data);
      setError(null);
    } catch (e) {
      // Handle different error formats
      let errorMessage = 'Failed to load proposals';

      if (e?.response?.data) {
        const errorData = e.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
```

```
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    }
  };

  const loadCategories = async () => {
    try {
      const data = await categoryAPI.getAll();
      setCategories(data);
    } catch(e) {
      // ignore category loading errors
    }
  }
  const handleApprove = async (proposalId, data) => {
    try {
      await proposalAPI.approve(proposalId, data);
      setSelectedProposal(null);
      load(); // Reload proposals
    } catch (err) {
      // Handle different error formats
      let errorMessage = 'Failed to approve proposal';

      if (err?.response?.data) {
        const errorData = err.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    }
  };
```

```
  const handleReject = async (proposalId, data) => {
    try {
      await proposalAPI.reject(proposalId, data);
      setSelectedProposal(null);
      load(); // Reload proposals
    } catch (err) {
      // Handle different error formats
      let errorMessage = 'Failed to reject proposal';

      if (err?.response?.data) {
        const errorData = err.response.data;

        // Handle validation error array
        if (Array.isArray(errorData.detail)) {
          errorMessage = errorData.detail.map(err => err.msg ||
err.message || String(err)).join(', ');
        }
        // Handle single validation error object
        else if (errorData.detail && typeof errorData.detail === 'object')
{
          errorMessage = errorData.detail.msg || errorData.detail.message
|| String(errorData.detail);
        }
        // Handle string error
        else if (typeof errorData.detail === 'string') {
          errorMessage = errorData.detail;
        }
        // Handle other error formats
        else if (typeof errorData === 'string') {
          errorMessage = errorData;
        }
      }

      setError(errorMessage);
    }
  };

  const handleDelete = async (proposal) => {
    if (proposal.status !== 'Pending') {
      alert('Only pending proposals can be deleted.');
      return;
    }
    const reason = window.prompt('Please provide a reason for deleting
this proposal:');
    if (!reason || !reason.trim()) {
      return; // require a reason
    }
    try {
      setDeletingId(proposal.id);
      await proposalAPI.delete(proposal.id, { reason });
      await load();
    } catch (e) {
      let errorMessage = 'Failed to delete proposal';
      if (e?.response?.data) {
```

```
          const errorData = e.response.data;
          if (typeof errorData.detail === 'string') errorMessage =
errorData.detail;
          else if (typeof errorData === 'string') errorMessage = errorData;
        }
        setError(errorMessage);
      } finally {
        setDeletingId(null);
      }
    };


    useEffect(() => {
      // eslint-disable-next-line react-hooks/exhaustive-deps
      load();
      loadCategories();
    }, []);

    const onChange = (e) => {
      const { name, value } = e.target;
      setFilters(prev => ({ ...prev, [name]: value }));
    };

    const onApplyFilters = () => {
      load();
    };

    const onClearFilters = () => {
      setFilters({ category_id: '', min_amount: '', max_amount: '' });
      load();
    };

    // Edit validation logic
    const canEditProposal = (proposal) => {
      if (!user || user.role !== 'ministry') return false;
      if (proposal.status !== 'Pending') return false;

      // Check ministry match
      if (user.ministry !== proposal.ministry) return false;

      return true;
    };

    const handleEdit = (proposal) => {
      setEditingProposal(proposal);
    };

    const handleEditSubmit = async (formData) => {
      try {
        await proposalAPI.update(editingProposal.id, formData);
        setEditingProposal(null);
        load(); // Reload proposals
      } catch (err) {
        let errorMessage = 'Failed to update proposal';
```

```
      if (err?.response?.data?.detail) {
        errorMessage = err.response.data.detail;
      }
      setError(errorMessage);
    }
  };

  const getStatusBadge = (status) => {
    const statusConfig = {
      'Pending': { icon: '⏳', class: 'status-pending', color: '#f59e0b'
},
      'Approved': { icon: '✅', class: 'status-approved', color: '#10b981'
},
      'Rejected': { icon: '❌', class: 'status-rejected', color: '#ef4444'
}
    };

    const config = statusConfig[status] || { icon: '?', class: 'status-
unknown', color: '#6b7280' };

    return (
      <span className={`status-badge ${config.class}`} style={{
backgroundColor: `${config.color}20`, color: config.color }}>
        <span className="status-icon">{config.icon}</span>
        {status}
      </span>
    );
  };

  const formatCurrency = (amount) => {
    return new Intl.NumberFormat('en-US', {
      style: 'currency',
      currency: 'USD',
      minimumFractionDigits: 0
    }).format(amount || 0);
  };

  const formatDate = (dateString) => {
    if (!dateString) return '-';
    return new Date(dateString).toLocaleDateString('en-US', {
      year: 'numeric',
      month: 'short',
      day: 'numeric',
      hour: '2-digit',
      minute: '2-digit'
    });
  };

  return (
    <>
      <div className="card">
        <h2>Pending Proposals</h2>
        <p className="page-description">Review and manage pending budget
proposals</p>
```

```
        {error && <div className="error-message">{error}</div>}

        <div className="filters-section">
          <h3>🔍 Filters</h3>
          <div className="filters-grid">
            <div className="form-group">
              <label htmlFor="ministry">Ministry</label>
              <input
                type="text"
                id="ministry"
                name="ministry"
                value={filters.ministry}
                onChange={onChange}
                placeholder="Filter by ministry"
              />
            </div>

            <div className="form-group">
              <label htmlFor="min_amount">Min Amount</label>
              <input
                type="text"
                id="min_amount"
                name="min_amount"
                value={filters.min_amount}
                onChange={onChange}
                placeholder="e.g., 100000"
                pattern="[0-9]*"
                inputMode="numeric"
              />
            </div>

            <div className="form-group">
              <label htmlFor="max_amount">Max Amount</label>
              <input
                type="text"
                id="max_amount"
                name="max_amount"
                value={filters.max_amount}
                onChange={onChange}
                placeholder="e.g., 5000000"
                pattern="[0-9]*"
                inputMode="numeric"
              />
            </div>

            <div className="form-group">
              <label htmlFor="category_id">Category</label>
              <select name="category_id" value={filters.category_id}
onChange={onChange}>
                <option value="">All Categories</option>
                {categories.map(c => (
                  <option key={c.id} value={c.id}>{c.name}</option>
                ))}
              </select>
```

```
            </div>
          </div>

          <div className="filter-actions">
            <button className="btn btn-primary" onClick={onApplyFilters}>
              <span>🔍 </span>
              Apply Filters
            </button>
            <button className="btn btn-secondary" onClick=
{onClearFilters}>
              <span>🔄 </span>
              Clear Filters
            </button>
          </div>
        </div>

        <div className="table-section">
          <div className="table-header">
            <h3>📋 All Proposals</h3>
          </div>

          {proposals.length === 0 ? (
            <div className="no-data">
              <div className="no-data-icon">📭 </div>
              <h3>No Proposals Found</h3>
              <p>No proposals match your current filters. Try adjusting
your search criteria.</p>
            </div>
          ) : (
            <div className="table-container">
              <table className="proposals-table">
                <thead>
                  <tr>
                    <th>ID</th>
                    <th>Ministry</th>
                    <th>Category</th>
                    <th>Title</th>
                    <th>Requested</th>
                    <th>Status</th>
                    <th>Approved</th>
                    <th>Decided At</th>
                    <th>Created</th>
                    <th>Actions</th>
                  </tr>
                </thead>
                <tbody>
                  {proposals.map(proposal => (
                    <tr key={proposal.id}>
                      <td>
                        <span className="proposal-id">#{proposal.id}
</span>
                      </td>
                      <td>
                        <span className="ministry-name">
```

```
{proposal.ministry?.name || 'Unknown'}</span>
                      </td>
                      <td>
                        <span className="category-name">
                          {categories.find(c => c.id ===
proposal.category_id)?.name || 'Unknown'}
                        </span>
                      </td>
                      <td>
                        <span className="proposal-title" title=
{proposal.title}>
                          {proposal.title}
                        </span>
                      </td>
                      <td>
                        <span className="amount requested">
{formatCurrency(proposal.requested_amount)}</span>
                      </td>
                      <td>{getStatusBadge(proposal.status)}</td>
                      <td>
                        <span className="amount approved">
                          {proposal.approved_amount ?
formatCurrency(proposal.approved_amount) : '-'}
                        </span>
                      </td>
                      <td>
                        <span className="date">
{formatDate(proposal.decided_at)}</span>
                      </td>
                      <td>
                        <span className="date">
{formatDate(proposal.created_at)}</span>
                      </td>
                      <td>
                        {proposal.status === 'Pending' && user?.role ===
'finance' && (
                          <button
                            className="btn btn-small btn-primary"
                            onClick={() => setSelectedProposal(proposal)}
                          >
                            Review
                          </button>
                        )}
                        {proposal.status === 'Pending' && user?.role ===
'ministry' && (
                          <div className="action-buttons">
                            {canEditProposal(proposal) && (
                              <button
                                className="btn btn-small btn-secondary"
                                onClick={() => handleEdit(proposal)}
                              >
                                Edit
                              </button>
                            )}
```

```
                              <button
                                className="btn btn-small btn-danger"
                                disabled={deletingId === proposal.id}
                                onClick={() => handleDelete(proposal)}
                              >
                                {deletingId === proposal.id ? 'Deleting…' :
'Delete'}
                              </button>
                            </div>
                          )}
                        </td>
                      </tr>
                    ))}
                  </tbody>
                </table>
              </div>
            )}
          </div>
        </div>

                    {selectedProposal && (
          <ApprovalDialog
            proposal={selectedProposal}
            categories={categories}
            onApprove={handleApprove}
            onReject={handleReject}
            onClose={() => setSelectedProposal(null)}
          />
        )}

        {editingProposal && (
          <EditProposalDialog
            proposal={editingProposal}
            categories={categories}
            onSubmit={handleEditSubmit}
            onClose={() => setEditingProposal(null)}
          />
        )}
      </>
    );
};

export default ProposalsList;
```

RoleGuard.js

```
import React from 'react';
import { useUser } from './UserContext';

const RoleGuard = ({ allowedRoles, children, fallback = null }) => {
```

```
  const { user } = useUser();

  if (!user) {
    return fallback;
  }

  if (!allowedRoles.includes(user.role)) {
    return fallback || (
      <div className="card">
        <div className="error-message">
          <span>🚫 </span> Access Denied
          <p>You don't have permission to access this feature.</p>
          <p>Required role: {allowedRoles.join(' or ')}</p>
          <p>Your role: {user.role}</p>
        </div>
      </div>
    );
  }

  return children;
};

export default RoleGuard;
```

Sidebar.js

```
import React from 'react';
import { useUser } from './UserContext';

const Sidebar = ({ activeTab, setActiveTab }) => {
  const { user } = useUser();

  const navigationItems = [
    {
      id: 'dashboard',
      label: 'Dashboard',
      icon: '📊 ',
      description: 'Financial Overview',
      roles: ['finance']
    },
    {
      id: 'categories',
      label: 'Categories',
      icon: '📁 ',
      description: 'Budget Categories',
      roles: ['finance']
    },
    {
      id: 'proposals',
      label: 'Proposals',
```

```
      icon: '📋',
      description: 'All Proposals',
      roles: ['finance', 'ministry']
    },
    {
      id: 'create-proposal',
      label: 'Create Proposal',
      icon: '➕',
      description: 'New Proposal',
      roles: ['ministry']
    },
    {
      id: 'upload-contract',
      label: 'Upload Contract',
      icon: '📄',
      description: 'Contract Upload',
      roles: ['ministry']
    },
    {
      id: 'history',
      label: 'History',
      icon: '📜',
      description: 'Transaction History',
      roles: ['finance', 'ministry']
    }
  ];

  const filteredItems = navigationItems.filter(item =>
    item.roles.includes(user?.role)
  );

    const handleItemClick = (itemId) => {
    // Prevent ministry users from accessing finance-only features
    if (itemId === 'dashboard' && user?.role !== 'finance') {
      return; // Don't allow access
    }
    if (itemId === 'categories' && user?.role !== 'finance') {
      return; // Don't allow access
    }
    if ((itemId === 'create-proposal' || itemId === 'upload-contract') &&
user?.role !== 'ministry') {
      return; // Don't allow access
    }

    setActiveTab(itemId);
  };

  return (
    <div className="sidebar">
      <div className="sidebar-header">
        <div className="sidebar-logo">
          <span className="logo-icon">🏛️</span>
          <div className="logo-text">
            <h3>GovTracker</h3>
```

```
            <p>Spending Portal</p>
          </div>
        </div>
      </div>

      <nav className="sidebar-nav">
        <div className="nav-section">
          <h4 className="nav-section-title">Navigation</h4>
          {filteredItems.map((item) => (
            <button
              key={item.id}
              className={`nav-item ${activeTab === item.id ? 'active' :
''}`}
              onClick={() => handleItemClick(item.id)}
              title={item.description}
            >
              <span className="nav-icon">{item.icon}</span>
              <div className="nav-content">
                <span className="nav-label">{item.label}</span>
                <span className="nav-description">{item.description}
</span>
              </div>
              {activeTab === item.id && (
                <div className="nav-indicator"></div>
              )}
            </button>
          ))}
        </div>

        <div className="nav-section">
          <h4 className="nav-section-title">User Info</h4>
          <div className="user-card">
            <div className="user-avatar">
              <span>{user?.role === 'finance' ? '💰 ' : '🏢 '}</span>
            </div>
            <div className="user-details">
              <span className="user-name">{user?.username}</span>
              <span className="user-role">{user?.role === 'finance' ?
'Finance Ministry' : 'Ministry User'}</span>
            </div>
          </div>
        </div>
      </nav>
    </div>
  );
};

export default Sidebar;
```

UserContext.js

```
import React, { createContext, useContext, useState, useEffect } from
'react';
import { authAPI } from './api';

const UserContext = createContext();

export const useUser = () => {
  const context = useContext(UserContext);
  if (!context) {
    throw new Error('useUser must be used within a UserProvider');
  }
  return context;
};

export const UserProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const loadUser = async () => {
      const token = localStorage.getItem('authToken');
      const savedUser = localStorage.getItem('user');

      if (token && savedUser) {
        try {
          // Validate the token by calling the /auth/me endpoint
          const currentUser = await authAPI.getMe();
          setUser(currentUser);
        } catch (error) {
          // Token is invalid, clear it
          console.log('Token validation failed, clearing auth data');
          localStorage.removeItem('authToken');
          localStorage.removeItem('user');
          setUser(null);
        }
      } else {
        // No saved auth data
        setUser(null);
      }
      setLoading(false);
    };

    loadUser();
  }, []);

  const login = (userData) => {
    setUser(userData);
  };

  const logout = () => {
    setUser(null);
    localStorage.removeItem('authToken');
    localStorage.removeItem('user');
```

```
  };

  const isFinance = () => user?.role === 'finance';
  const isMinistry = () => user?.role === 'ministry';

  const value = {
    user,
    login,
    logout,
    isFinance,
    isMinistry,
    loading
  };

  return (
    <UserContext.Provider value={value}>
      {children}
    </UserContext.Provider>
  );
};
```

api.js

```
import axios from 'axios';

const API_BASE_URL = 'http://localhost:8000';

const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Add a request interceptor to include the auth token
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('authToken');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

// Ministry API functions
export const ministryAPI = {
```

```
  getAll: async () => {
    const response = await api.get('/ministries');
    return response.data;
  },
  create: async (ministry) => {
    const response = await api.post('/ministries', ministry);
    return response.data;
  },
  findOrCreate: async (ministryName) => {
    const response = await api.post('/ministries/find-or-create', null, {
      params: { ministry_name: ministryName }
    });
    return response.data;
  }
};

// Category API functions
export const categoryAPI = {
  getAll: async () => {
    const response = await api.get('/categories');
    return response.data;
  },
  create: async (category) => {
    const response = await api.post('/categories', category);
    return response.data;
  },
  update: async (id, category) => {
    const response = await api.put(`/categories/${id}`, category);
    return response.data;
  },
  delete: async (id) => {
    await api.delete(`/categories/${id}`);
  }
};

// Proposal API functions
export const proposalAPI = {
  getAll: async (filters = {}) => {
    const query = new URLSearchParams(filters).toString();
    const response = await api.get(`/proposals${query ? `?${query}` :
''}`);
    return response.data;
  },
  create: async (proposal) => {
    const response = await api.post('/proposals', proposal);
    return response.data;
  },
  getById: async (id) => {
    const response = await api.get(`/proposals/${id}`);
    return response.data;
  },
  update: async (id, proposal) => {
    const response = await api.put(`/proposals/${id}`, proposal);
    return response.data;
```

```
  },
  delete: async (id, data) => {
    // send reason in body with axios.delete
    const response = await api.delete(`/proposals/${id}`, { data });
    return response.data;
  },
  approve: async (id, data) => {
    const response = await api.post(`/proposals/${id}/approve`, data);
    return response.data;
  },
  reject: async (id, data) => {
    const response = await api.post(`/proposals/${id}/reject`, data);
    return response.data;
  }
};

// Upload API functions
export const uploadAPI = {
  parse: async (file) => {
    const form = new FormData();
    form.append('file', file);
    const response = await api.post('/contracts/parse', form, {
      headers: { 'Content-Type': 'multipart/form-data' }
    });
    return response.data;
  }
};

// Dashboard API functions
export const dashboardAPI = {
  getSummary: async () => {
    const response = await api.get('/dashboard/summary');
    return response.data;
  }
};

// Auth API functions
export const authAPI = {
  login: async (username, password) => {
    const response = await api.post('/auth/login', { username, password
});
    return response.data;
  },
  getMe: async () => {
    const response = await api.get('/auth/me');
    return response.data;
  },
};

// History API functions
export const historyAPI = {
  list: async (filters = {}) => {
    const response = await api.get('/proposals', { params: filters });
    return response.data;
```

```
  },
};
```

errorUtils.js

```
// Error handling utility functions

export const extractErrorMessage = (error) => {
  // Handle different error formats
  let errorMessage = 'An error occurred';

  if (error?.response?.data) {
    const errorData = error.response.data;

    // Handle validation error array
    if (Array.isArray(errorData.detail)) {
      errorMessage = errorData.detail.map(err => err.msg || err.message ||
String(err)).join(', ');
    }
    // Handle single validation error object
    else if (errorData.detail && typeof errorData.detail === 'object') {
      errorMessage = errorData.detail.msg || errorData.detail.message ||
String(errorData.detail);
    }
    // Handle string error
    else if (typeof errorData.detail === 'string') {
      errorMessage = errorData.detail;
    }
    // Handle other error formats
    else if (typeof errorData === 'string') {
      errorMessage = errorData;
    }
  } else if (typeof error === 'string') {
    errorMessage = error;
  } else if (error?.message) {
    errorMessage = error.message;
  }

  return errorMessage;
};

export const handleApiError = (error, defaultMessage = 'An error
occurred') => {
  console.error('API Error:', error);
  return extractErrorMessage(error) || defaultMessage;
};
```

index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

proposalsApi.js

```
import api from './api';

export const proposalAPI = {
  getAll: async (params = {}) => {
    const response = await api.get('/proposals', { params });
    return response.data;
  },
  getById: async (id) => {
    const response = await api.get(`/proposals/${id}`);
```

```
    return response.data;
  },
  create: async (data) => {
    const response = await api.post('/proposals', data);
    return response.data;
  },
  update: async (id, data) => {
    const response = await api.put(`/proposals/${id}`, data);
    return response.data;
  },
  delete: async (id) => {
    const response = await api.delete(`/proposals/${id}`);
    return response.data;
  },
};
```

reportWebVitals.js

```
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB
}) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

setupTests.js

```
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';
```

theme.css

```
/* Modern Government Spending Tracker Theme */

:root {
  /* Modern Color Palette */
  --color-primary: #2563eb;
  --color-primary-dark: #1d4ed8;
  --color-primary-light: #3b82f6;
  --color-secondary: #059669;
  --color-secondary-dark: #047857;
  --color-accent: #7c3aed;
  --color-accent-light: #8b5cf6;

  /* Neutral Colors */
  --color-background: #f8fafc;
  --color-background-card: #ffffff;
  --color-background-light: #f1f5f9;
  --color-background-dark: #0f172a;
  --color-text-primary: #1e293b;
  --color-text-secondary: #64748b;
  --color-text-muted: #94a3b8;
  --color-border: #e2e8f0;
  --color-border-light: #f1f5f9;

  /* Status Colors */
  --color-success: #10b981;
  --color-success-light: #d1fae5;
  --color-warning: #f59e0b;
  --color-warning-light: #fef3c7;
  --color-error: #ef4444;
  --color-error-light: #fee2e2;
  --color-info: #3b82f6;
  --color-info-light: #dbeafe;

  /* Spacing */
  --spacing-xs: 0.25rem;
  --spacing-sm: 0.5rem;
  --spacing-md: 1rem;
  --spacing-lg: 1.5rem;
  --spacing-xl: 2rem;
  --spacing-2xl: 3rem;

  /* Typography */
  --font-family: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe UI',
'Roboto', sans-serif;
  --font-size-xs: 0.75rem;
  --font-size-sm: 0.875rem;
  --font-size-base: 1rem;
  --font-size-lg: 1.125rem;
  --font-size-xl: 1.25rem;
  --font-size-2xl: 1.5rem;
  --font-size-3xl: 1.875rem;
  --font-size-4xl: 2.25rem;
```

```
  /* Border Radius */
  --border-radius: 0.5rem;
  --border-radius-sm: 0.25rem;
  --border-radius-lg: 0.75rem;
  --border-radius-xl: 1rem;
  --border-radius-full: 9999px;

  /* Shadows */
  --shadow-sm: 0 1px 2px 0 rgb(0 0 0 / 0.05);
  --shadow: 0 1px 3px 0 rgb(0 0 0 / 0.1), 0 1px 2px -1px rgb(0 0 0 / 0.1);
  --shadow-md: 0 4px 6px -1px rgb(0 0 0 / 0.1), 0 2px 4px -2px rgb(0 0 0 /
0.1);
  --shadow-lg: 0 10px 15px -3px rgb(0 0 0 / 0.1), 0 4px 6px -4px rgb(0 0 0
/ 0.1);
  --shadow-xl: 0 20px 25px -5px rgb(0 0 0 / 0.1), 0 8px 10px -6px rgb(0 0
0 / 0.1);

  /* Transitions */
  --transition: all 0.2s cubic-bezier(0.4, 0, 0.2, 1);
  --transition-slow: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

/* Global Styles */
* {
  box-sizing: border-box;
}

body {
  margin: 0;
  padding: 0;
  font-family: var(--font-family);
  font-size: var(--font-size-base);
  line-height: 1.6;
  color: var(--color-text-primary);
  background: var(--color-background);
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

.App {
  min-height: 100vh;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  background-attachment: fixed;
}

/* Header Styles */
.app-header {
  background: rgba(255, 255, 255, 0.95);
  backdrop-filter: blur(10px);
  border-bottom: 1px solid var(--color-border);
  box-shadow: var(--shadow-sm);
  position: sticky;
  top: 0;
  z-index: 100;
```

```
  padding: var(--spacing-md) 0;
}

.header-content {
  display: flex;
  justify-content: space-between;
  align-items: center;
  width: 100%;
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 var(--spacing-lg);
}

.app-header h1 {
  margin: 0;
  font-size: var(--font-size-2xl);
  font-weight: 700;
  background: linear-gradient(135deg, var(--color-primary), var(--color-
accent));
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  background-clip: text;
}

.user-info {
  display: flex;
  align-items: center;
  gap: var(--spacing-md);
}

.user-role {
  background: linear-gradient(135deg, var(--color-primary), var(--color-
primary-dark));
  color: white;
  padding: var(--spacing-xs) var(--spacing-md);
  border-radius: var(--border-radius-full);
  font-size: var(--font-size-sm);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
  box-shadow: var(--shadow-sm);
}

.user-name {
  color: var(--color-text-secondary);
  font-weight: 600;
  font-size: var(--font-size-sm);
}

/* Container */
.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: var(--spacing-xl) var(--spacing-lg);
```

```
  display: grid;
  gap: var(--spacing-xl);
}

/* Card Styles */
.card {
  background: var(--color-background-card);
  border-radius: var(--border-radius-lg);
  box-shadow: var(--shadow-lg);
  padding: var(--spacing-xl);
  border: 1px solid var(--color-border-light);
  transition: var(--transition);
  position: relative;
  overflow: hidden;
}

.card::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  height: 4px;
  background: linear-gradient(90deg, var(--color-primary), var(--color-
accent));
}

.card:hover {
  transform: translateY(-2px);
  box-shadow: var(--shadow-xl);
}

.card h2 {
  margin: 0 0 var(--spacing-lg) 0;
  font-size: var(--font-size-xl);
  font-weight: 700;
  color: var(--color-text-primary);
  display: flex;
  align-items: center;
  gap: var(--spacing-sm);
}

.card h2::before {
  content: '';
  width: 4px;
  height: 24px;
  background: linear-gradient(135deg, var(--color-primary), var(--color-
accent));
  border-radius: var(--border-radius-sm);
}

/* Button Styles */
.btn {
  display: inline-flex;
```

```
  align-items: center;
  justify-content: center;
  gap: var(--spacing-sm);
  padding: var(--spacing-sm) var(--spacing-lg);
  border: none;
  border-radius: var(--border-radius);
  font-size: var(--font-size-sm);
  font-weight: 600;
  text-decoration: none;
  cursor: pointer;
  transition: var(--transition);
  position: relative;
  overflow: hidden;
}

.btn::before {
  content: '';
  position: absolute;
  top: 0;
  left: -100%;
  width: 100%;
  height: 100%;
  background: linear-gradient(90deg, transparent, rgba(255, 255, 255,
0.2), transparent);
  transition: var(--transition-slow);
}

.btn:hover::before {
  left: 100%;
}

.btn-primary {
  background: linear-gradient(135deg, var(--color-primary), var(--color-
primary-dark));
  color: white;
  box-shadow: var(--shadow-md);
}

.btn-primary:hover {
  transform: translateY(-1px);
  box-shadow: var(--shadow-lg);
}

.btn-secondary {
  background: var(--color-background-light);
  color: var(--color-text-primary);
  border: 1px solid var(--color-border);
}

.btn-secondary:hover {
  background: var(--color-border-light);
  transform: translateY(-1px);
}
```

```
.btn-small {
  padding: var(--spacing-xs) var(--spacing-md);
  font-size: var(--font-size-xs);
}

.btn:disabled {
  opacity: 0.6;
  cursor: not-allowed;
  transform: none !important;
}

/* Form Styles */
.form-group {
  margin-bottom: var(--spacing-lg);
}

.form-group label {
  display: block;
  margin-bottom: var(--spacing-sm);
  font-weight: 600;
  color: var(--color-text-primary);
  font-size: var(--font-size-sm);
}

.form-group input,
.form-group select,
.form-group textarea {
  width: 100%;
  padding: var(--spacing-md);
  border: 2px solid var(--color-border);
  border-radius: var(--border-radius);
  font-size: var(--font-size-base);
  font-family: var(--font-family);
  transition: var(--transition);
  background: var(--color-background-card);
}

.form-group input:focus,
.form-group select:focus,
.form-group textarea:focus {
  outline: none;
  border-color: var(--color-primary);
  box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.1);
}

.form-actions {
  display: flex;
  gap: var(--spacing-md);
  margin-top: var(--spacing-xl);
  padding-top: var(--spacing-lg);
  border-top: 1px solid var(--color-border-light);
}

/* Table Styles */
```

```
.categories-table,
.proposals-table {
  width: 100%;
  border-collapse: collapse;
  margin-top: var(--spacing-lg);
  background: var(--color-background-card);
  border-radius: var(--border-radius);
  overflow: hidden;
  box-shadow: var(--shadow-sm);
}

.categories-table th,
.proposals-table th {
  background: var(--color-background-light);
  padding: var(--spacing-md);
  text-align: left;
  font-weight: 600;
  color: var(--color-text-primary);
  font-size: var(--font-size-sm);
  text-transform: uppercase;
  letter-spacing: 0.05em;
  border-bottom: 2px solid var(--color-border);
}

.categories-table td,
.proposals-table td {
  padding: var(--spacing-md);
  border-bottom: 1px solid var(--color-border-light);
  vertical-align: middle;
}

.categories-table tr:hover,
.proposals-table tr:hover {
  background: var(--color-background-light);
}

/* Status Badges */
.status-badge {
  display: inline-flex;
  align-items: center;
  padding: var(--spacing-xs) var(--spacing-sm);
  border-radius: var(--border-radius-full);
  font-size: var(--font-size-xs);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
}

.status-pending {
  background: var(--color-warning-light);
  color: var(--color-warning);
}

.status-approved {
```

```
  background: var(--color-success-light);
  color: var(--color-success);
}

.status-rejected {
  background: var(--color-error-light);
  color: var(--color-error);
}

/* KPI Summary */
.kpi-summary {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: var(--spacing-lg);
  margin-bottom: var(--spacing-xl);
}

.kpi-item {
  background: linear-gradient(135deg, var(--color-background-light), var(-
-color-background-card));
  padding: var(--spacing-lg);
  border-radius: var(--border-radius-lg);
  text-align: center;
  border: 1px solid var(--color-border-light);
  transition: var(--transition);
  position: relative;
  overflow: hidden;
}

.kpi-item::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  height: 3px;
  background: linear-gradient(90deg, var(--color-primary), var(--color-
accent));
}

.kpi-item:hover {
  transform: translateY(-2px);
  box-shadow: var(--shadow-md);
}

.kpi-item h3 {
  margin: 0 0 var(--spacing-sm) 0;
  color: var(--color-text-secondary);
  font-size: var(--font-size-sm);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
}
```

```css
.kpi-item p {
  margin: 0;
  color: var(--color-primary);
  font-size: var(--font-size-2xl);
  font-weight: 700;
}

/* Charts */
.charts-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(400px, 1fr));
  gap: var(--spacing-xl);
}

.chart-section {
  background: var(--color-background-light);
  padding: var(--spacing-lg);
  border-radius: var(--border-radius-lg);
  border: 1px solid var(--color-border-light);
  transition: var(--transition);
}

.chart-section:hover {
  box-shadow: var(--shadow-md);
}

.chart-section h3 {
  margin: 0 0 var(--spacing-lg) 0;
  color: var(--color-text-primary);
  font-size: var(--font-size-lg);
  font-weight: 600;
  text-align: center;
}

.chart-container {
  position: relative;
  height: 200px;
  width: 100%;
  background: var(--color-background-card);
  border-radius: var(--border-radius);
  padding: var(--spacing-md);
  box-shadow: var(--shadow-sm);
}

/* Login Styles */
.login-container {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  padding: var(--spacing-lg);
}
```

```
.login-card {
  background: rgba(255, 255, 255, 0.95);
  backdrop-filter: blur(20px);
  padding: var(--spacing-2xl);
  border-radius: var(--border-radius-xl);
  box-shadow: var(--shadow-xl);
  width: 100%;
  max-width: 400px;
  text-align: center;
  border: 1px solid rgba(255, 255, 255, 0.2);
  position: relative;
  overflow: hidden;
}

.login-card::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  height: 4px;
  background: linear-gradient(90deg, var(--color-primary), var(--color-
accent));
}

.login-card h2 {
  color: var(--color-text-primary);
  margin-bottom: var(--spacing-sm);
  font-size: var(--font-size-3xl);
  font-weight: 700;
  background: linear-gradient(135deg, var(--color-primary), var(--color-
accent));
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  background-clip: text;
}

.login-card h3 {
  color: var(--color-text-secondary);
  margin-bottom: var(--spacing-xl);
  font-weight: 500;
  font-size: var(--font-size-lg);
}

.login-help {
  margin-top: var(--spacing-xl);
  padding: var(--spacing-lg);
  background: var(--color-background-light);
  border-radius: var(--border-radius);
  text-align: left;
  border: 1px solid var(--color-border-light);
}

.login-help h4 {
```

```
  margin: 0 0 var(--spacing-md) 0;
  color: var(--color-text-primary);
  font-size: var(--font-size-base);
  font-weight: 600;
}

.login-help p {
  margin: var(--spacing-sm) 0;
  font-size: var(--font-size-sm);
  color: var(--color-text-secondary);
}

.login-help strong {
  color: var(--color-primary);
  font-weight: 600;
}

/* Loading and Error States */
.loading {
  display: flex;
  justify-content: center;
  align-items: center;
  padding: var(--spacing-xl);
  color: var(--color-text-secondary);
  font-style: italic;
  font-size: var(--font-size-lg);
}

.error-message {
  background: var(--color-error-light);
  color: var(--color-error);
  padding: var(--spacing-md);
  border-radius: var(--border-radius);
  border: 1px solid var(--color-error);
  margin: var(--spacing-md) 0;
  font-weight: 500;
}

/* Animations */
@keyframes fadeIn {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

@keyframes slideIn {
  from {
    opacity: 0;
    transform: translateX(-20px);
```

```
  }
  to {
    opacity: 1;
    transform: translateX(0);
  }
}

.card {
  animation: fadeIn 0.6s ease-out;
}

.login-card {
  animation: slideIn 0.8s ease-out;
}

/* Responsive Design */
@media (max-width: 768px) {
  .header-content {
    flex-direction: column;
    gap: var(--spacing-md);
    text-align: center;
  }

  .user-info {
    flex-direction: column;
    gap: var(--spacing-sm);
  }

  .container {
    padding: var(--spacing-lg) var(--spacing-md);
    gap: var(--spacing-lg);
  }

  .card {
    padding: var(--spacing-lg);
  }

  .charts-grid {
    grid-template-columns: 1fr;
  }

  .kpi-summary {
    grid-template-columns: 1fr;
  }

  .login-card {
    padding: var(--spacing-xl);
    margin: var(--spacing-md);
  }

  .form-actions {
    flex-direction: column;
  }
```

```
  .btn {
    width: 100%;
  }
}

@media (max-width: 480px) {
  .app-header h1 {
    font-size: var(--font-size-xl);
  }

  .card h2 {
    font-size: var(--font-size-lg);
  }

  .kpi-item p {
    font-size: var(--font-size-xl);
  }
}


/* Enhanced Login Styles */
.login-header {
  margin-bottom: var(--spacing-xl);
}

.login-header p {
  color: var(--color-text-secondary);
  margin: var(--spacing-sm) 0 0 0;
  font-size: var(--font-size-sm);
}

.login-form {
  margin-bottom: var(--spacing-xl);
}

.login-btn {
  width: 100%;
  padding: var(--spacing-md) var(--spacing-lg);
  font-size: var(--font-size-base);
  font-weight: 600;
  margin-top: var(--spacing-md);
}

.quick-login-buttons {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: var(--spacing-sm);
  margin-bottom: var(--spacing-md);
}

.quick-btn {
  padding: var(--spacing-sm) var(--spacing-md);
  font-size: var(--font-size-sm);
  display: flex;
```

```
  align-items: center;
  justify-content: center;
  gap: var(--spacing-xs);
}

.credentials-info {
  background: var(--color-background-card);
  padding: var(--spacing-md);
  border-radius: var(--border-radius);
  border: 1px solid var(--color-border-light);
}

.credentials-info p {
  margin: var(--spacing-xs) 0;
  font-size: var(--font-size-xs);
  color: var(--color-text-secondary);
}

/* Spinner Animation */
.spinner {
  display: inline-block;
  width: 16px;
  height: 16px;
  border: 2px solid rgba(255, 255, 255, 0.3);
  border-radius: 50%;
  border-top-color: white;
  animation: spin 1s ease-in-out infinite;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

/* Enhanced Button Hover Effects */
.btn:hover {
  transform: translateY(-2px);
}

.btn:active {
  transform: translateY(0);
}

/* Enhanced Card Animations */
.card {
  animation: fadeInUp 0.6s ease-out;
}

@keyframes fadeInUp {
  from {
    opacity: 0;
    transform: translateY(30px);
  }
  to {
    opacity: 1;
```

```
    transform: translateY(0);
  }
}

/* Staggered Animation for Multiple Cards */
.card:nth-child(1) { animation-delay: 0.1s; }
.card:nth-child(2) { animation-delay: 0.2s; }
.card:nth-child(3) { animation-delay: 0.3s; }
.card:nth-child(4) { animation-delay: 0.4s; }

/* Enhanced Form Focus States */
.form-group input:focus,
.form-group select:focus,
.form-group textarea:focus {
  transform: translateY(-1px);
  box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.1), var(--shadow-md);
}

/* Enhanced Table Hover Effects */
.categories-table tr,
.proposals-table tr {
  transition: var(--transition);
}

.categories-table tr:hover,
.proposals-table tr:hover {
  background: linear-gradient(90deg, var(--color-background-light), var(--
color-background-card));
  transform: scale(1.01);
}

/* Enhanced Status Badges */
.status-badge {
  position: relative;
  overflow: hidden;
}

.status-badge::before {
  content: '';
  position: absolute;
  top: 0;
  left: -100%;
  width: 100%;
  height: 100%;
  background: linear-gradient(90deg, transparent, rgba(255, 255, 255,
0.2), transparent);
  transition: var(--transition-slow);
}

.status-badge:hover::before {
  left: 100%;
}

/* Enhanced KPI Items */
```

```
.kpi-item {
  position: relative;
  overflow: hidden;
}

.kpi-item::after {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: linear-gradient(135deg, rgba(37, 99, 235, 0.05), rgba(124,
58, 237, 0.05));
  opacity: 0;
  transition: var(--transition);
}

.kpi-item:hover::after {
  opacity: 1;
}

/* Enhanced Chart Containers */
.chart-container {
  position: relative;
  overflow: hidden;
}

.chart-container::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: linear-gradient(135deg, rgba(37, 99, 235, 0.02), rgba(124,
58, 237, 0.02));
  pointer-events: none;
}

/* Loading States */
.loading {
  position: relative;
}

.loading::after {
  content: '';
  position: absolute;
  top: 50%;
  left: 50%;
  width: 20px;
  height: 20px;
  margin: -10px 0 0 -10px;
  border: 2px solid var(--color-border);
```

```
  border-radius: 50%;
  border-top-color: var(--color-primary);
  animation: spin 1s linear infinite;
}

/* Enhanced Error Messages */
.error-message {
  display: flex;
  align-items: center;
  gap: var(--spacing-sm);
  animation: shake 0.5s ease-in-out;
}

@keyframes shake {
  0%, 100% { transform: translateX(0); }
  25% { transform: translateX(-5px); }
  75% { transform: translateX(5px); }
}

/* Enhanced Header */
.app-header {
  position: relative;
  overflow: hidden;
}

.app-header::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: linear-gradient(135deg, rgba(255, 255, 255, 0.1), rgba(255,
255, 255, 0.05));
  pointer-events: none;
}

/* Enhanced User Info */
.user-info {
  position: relative;
  z-index: 1;
}

.user-role {
  position: relative;
  overflow: hidden;
}

.user-role::before {
  content: '';
  position: absolute;
  top: 0;
  left: -100%;
  width: 100%;
```

```
  height: 100%;
  background: linear-gradient(90deg, transparent, rgba(255, 255, 255,
0.2), transparent);
  transition: var(--transition-slow);
}

.user-role:hover::before {
  left: 100%;
}

/* Responsive Enhancements */
@media (max-width: 768px) {
  .quick-login-buttons {
    grid-template-columns: 1fr;
  }

  .login-card {
    margin: var(--spacing-md);
    padding: var(--spacing-lg);
  }

  .card {
    padding: var(--spacing-lg);
  }

  .kpi-summary {
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
  }
}

@media (max-width: 480px) {
  .login-card {
    padding: var(--spacing-md);
  }

  .app-header h1 {
    font-size: var(--font-size-lg);
  }

  .kpi-item p {
    font-size: var(--font-size-lg);
  }
}


/* Enhanced Header Styles */
.header-left {
  display: flex;
  flex-direction: column;
  gap: var(--spacing-xs);
}

.header-subtitle {
  margin: 0;
```

```
  color: var(--color-text-secondary);
  font-size: var(--font-size-sm);
  font-weight: 500;
  opacity: 0.8;
}

.user-details {
  display: flex;
  flex-direction: column;
  align-items: flex-end;
  gap: var(--spacing-xs);
}

.user-icon {
  margin-right: var(--spacing-xs);
}

.logout-btn {
  display: flex;
  align-items: center;
  gap: var(--spacing-xs);
  padding: var(--spacing-xs) var(--spacing-md);
  font-size: var(--font-size-xs);
  border-radius: var(--border-radius-full);
  background: rgba(255, 255, 255, 0.1);
  border: 1px solid rgba(255, 255, 255, 0.2);
  color: var(--color-text-primary);
  transition: var(--transition);
}

.logout-btn:hover {
  background: rgba(255, 255, 255, 0.2);
  transform: translateY(-1px);
}

.role-icon {
  margin-right: var(--spacing-xs);
}

/* Enhanced User Role Badge */
.user-role {
  display: flex;
  align-items: center;
  gap: var(--spacing-xs);
  padding: var(--spacing-xs) var(--spacing-md);
  border-radius: var(--border-radius-full);
  font-size: var(--font-size-xs);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
  box-shadow: var(--shadow-sm);
  transition: var(--transition);
}
```

```
.user-role:hover {
  transform: translateY(-1px);
  box-shadow: var(--shadow-md);
}

/* Enhanced User Name */
.user-name {
  color: var(--color-text-primary);
  font-weight: 600;
  font-size: var(--font-size-sm);
  display: flex;
  align-items: center;
  gap: var(--spacing-xs);
}

/* Responsive Header */
@media (max-width: 768px) {
  .header-content {
    flex-direction: column;
    gap: var(--spacing-md);
    text-align: center;
  }

  .user-details {
    align-items: center;
  }

  .user-info {
    flex-direction: column;
    gap: var(--spacing-sm);
  }

  .header-left {
    align-items: center;
  }

  .header-subtitle {
    text-align: center;
  }
}

@media (max-width: 480px) {
  .app-header h1 {
    font-size: var(--font-size-lg);
  }

  .header-subtitle {
    font-size: var(--font-size-xs);
  }

  .user-role {
    font-size: var(--font-size-xs);
    padding: var(--spacing-xs) var(--spacing-sm);
  }
```

```
}


/* Enhanced Dashboard Styles */
.dashboard-card {
  background: linear-gradient(135deg, var(--color-background-card), var(--
color-background-light));
}

.card-header {
  margin-bottom: var(--spacing-xl);
  text-align: center;
}

.card-header p {
  margin: var(--spacing-sm) 0 0 0;
  color: var(--color-text-secondary);
  font-size: var(--font-size-sm);
  font-weight: 500;
}

/* Enhanced KPI Items */
.kpi-item {
  position: relative;
  overflow: hidden;
  text-align: center;
  padding: var(--spacing-lg);
  background: linear-gradient(135deg, var(--color-background-card), var(--
color-background-light));
  border: 1px solid var(--color-border-light);
  border-radius: var(--border-radius-lg);
  transition: var(--transition);
  box-shadow: var(--shadow-sm);
}

.kpi-item::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  height: 4px;
  background: linear-gradient(90deg, var(--color-primary), var(--color-
accent));
}

.kpi-item:hover {
  transform: translateY(-4px);
  box-shadow: var(--shadow-lg);
}

.kpi-icon {
  font-size: var(--font-size-2xl);
  margin-bottom: var(--spacing-sm);
```

```
  display: block;
}

.kpi-item h3 {
  margin: 0 0 var(--spacing-sm) 0;
  color: var(--color-text-secondary);
  font-size: var(--font-size-sm);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
}

.kpi-item p {
  margin: 0;
  color: var(--color-primary);
  font-size: var(--font-size-2xl);
  font-weight: 700;
  font-family: 'Inter', monospace;
}

/* Enhanced Chart Sections */
.chart-section {
  background: linear-gradient(135deg, var(--color-background-light), var(-
-color-background-card));
  padding: var(--spacing-lg);
  border-radius: var(--border-radius-lg);
  border: 1px solid var(--color-border-light);
  transition: var(--transition);
  box-shadow: var(--shadow-sm);
}

.chart-section:hover {
  transform: translateY(-2px);
  box-shadow: var(--shadow-md);
}

.chart-header {
  text-align: center;
  margin-bottom: var(--spacing-lg);
}

.chart-header h3 {
  margin: 0 0 var(--spacing-xs) 0;
  color: var(--color-text-primary);
  font-size: var(--font-size-lg);
  font-weight: 600;
}

.chart-header p {
  margin: 0;
  color: var(--color-text-secondary);
  font-size: var(--font-size-sm);
  font-weight: 500;
}
```

```
.chart-container {
  position: relative;
  height: 200px;
  width: 100%;
  background: var(--color-background-card);
  border-radius: var(--border-radius);
  padding: var(--spacing-md);
  box-shadow: var(--shadow-sm);
  border: 1px solid var(--color-border-light);
}

/* Enhanced Loading States */
.loading {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  padding: var(--spacing-2xl);
  color: var(--color-text-secondary);
  font-style: italic;
  font-size: var(--font-size-lg);
  gap: var(--spacing-md);
}

.loading-spinner {
  width: 32px;
  height: 32px;
  border: 3px solid var(--color-border);
  border-radius: 50%;
  border-top-color: var(--color-primary);
  animation: spin 1s linear infinite;
}

/* Enhanced Error Messages */
.error-message {
  display: flex;
  align-items: center;
  gap: var(--spacing-sm);
  padding: var(--spacing-lg);
  background: var(--color-error-light);
  color: var(--color-error);
  border-radius: var(--border-radius);
  border: 1px solid var(--color-error);
  margin: var(--spacing-lg) 0;
  font-weight: 500;
  animation: shake 0.5s ease-in-out;
}

/* Enhanced Button Styles */
.btn {
  position: relative;
  overflow: hidden;
  transition: var(--transition);
```

```
}

.btn::after {
  content: '';
  position: absolute;
  top: 50%;
  left: 50%;
  width: 0;
  height: 0;
  background: rgba(255, 255, 255, 0.2);
  border-radius: 50%;
  transform: translate(-50%, -50%);
  transition: var(--transition-slow);
}

.btn:hover::after {
  width: 300px;
  height: 300px;
}

/* Enhanced Form Styles */
.form-group input:focus,
.form-group select:focus,
.form-group textarea:focus {
  transform: translateY(-1px);
  box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.1), var(--shadow-md);
  border-color: var(--color-primary);
}

/* Enhanced Table Styles */
.categories-table,
.proposals-table {
  border-radius: var(--border-radius-lg);
  overflow: hidden;
  box-shadow: var(--shadow-md);
  border: 1px solid var(--color-border-light);
}

.categories-table th,
.proposals-table th {
  background: linear-gradient(135deg, var(--color-background-light), var(-
-color-background-card));
  font-weight: 700;
  text-transform: uppercase;
  letter-spacing: 0.05em;
  font-size: var(--font-size-xs);
  color: var(--color-text-primary);
  border-bottom: 2px solid var(--color-border);
}

.categories-table td,
.proposals-table td {
  transition: var(--transition);
}
```

```
.categories-table tr:hover,
.proposals-table tr:hover {
  background: linear-gradient(90deg, var(--color-background-light), var(--
color-background-card));
  transform: scale(1.01);
}

/* Enhanced Status Badges */
.status-badge {
  position: relative;
  overflow: hidden;
  padding: var(--spacing-xs) var(--spacing-md);
  border-radius: var(--border-radius-full);
  font-size: var(--font-size-xs);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
  transition: var(--transition);
}

.status-badge::before {
  content: '';
  position: absolute;
  top: 0;
  left: -100%;
  width: 100%;
  height: 100%;
  background: linear-gradient(90deg, transparent, rgba(255, 255, 255,
0.2), transparent);
  transition: var(--transition-slow);
}

.status-badge:hover::before {
  left: 100%;
}

.status-badge:hover {
  transform: translateY(-1px);
  box-shadow: var(--shadow-sm);
}

/* Responsive Enhancements */
@media (max-width: 768px) {
  .kpi-summary {
    grid-template-columns: repeat(2, 1fr);
    gap: var(--spacing-md);
  }

  .charts-grid {
    grid-template-columns: 1fr;
  }

  .chart-container {
```

```
    height: 180px;
  }

  .kpi-item {
    padding: var(--spacing-md);
  }

  .kpi-item p {
    font-size: var(--font-size-xl);
  }
}

@media (max-width: 480px) {
  .kpi-summary {
    grid-template-columns: 1fr;
  }

  .kpi-item p {
    font-size: var(--font-size-lg);
  }

  .chart-container {
    height: 160px;
  }
}


/* Sidebar and Layout Styles */
.app-layout {
  display: flex;
  min-height: calc(100vh - 80px);
}

.sidebar {
  width: 280px;
  background: linear-gradient(180deg, var(--color-background-card), var(--
color-background-light));
  border-right: 1px solid var(--color-border);
  box-shadow: var(--shadow-lg);
  position: sticky;
  top: 80px;
  height: calc(100vh - 80px);
  overflow-y: auto;
  z-index: 10;
  transition: var(--transition);
}

.sidebar-header {
  padding: var(--spacing-xl);
  border-bottom: 1px solid var(--color-border-light);
  background: linear-gradient(135deg, var(--color-primary), var(--color-
accent));
  color: white;
}
```

```
.sidebar-logo {
  display: flex;
  align-items: center;
  gap: var(--spacing-md);
}

.logo-icon {
  font-size: var(--font-size-2xl);
  background: rgba(255, 255, 255, 0.2);
  padding: var(--spacing-sm);
  border-radius: var(--border-radius);
  backdrop-filter: blur(10px);
}

.logo-text h3 {
  margin: 0;
  font-size: var(--font-size-lg);
  font-weight: 700;
  color: white;
}

.logo-text p {
  margin: 0;
  font-size: var(--font-size-sm);
  color: rgba(255, 255, 255, 0.8);
  font-weight: 500;
}

.sidebar-nav {
  padding: var(--spacing-lg) 0;
}

.nav-section {
  margin-bottom: var(--spacing-xl);
}

.nav-section-title {
  margin: 0 0 var(--spacing-md) 0;
  padding: 0 var(--spacing-lg);
  font-size: var(--font-size-xs);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.1em;
  color: var(--color-text-secondary);
}

.nav-item {
  display: flex;
  align-items: center;
  width: 100%;
  padding: var(--spacing-md) var(--spacing-lg);
  border: none;
  background: none;
```

```
  cursor: pointer;
  transition: var(--transition);
  position: relative;
  text-align: left;
  gap: var(--spacing-md);
}

.nav-item:hover {
  background: var(--color-background-light);
  transform: translateX(4px);
}

.nav-item.active {
  background: linear-gradient(90deg, var(--color-primary), var(--color-
primary-light));
  color: white;
  box-shadow: var(--shadow-md);
}

.nav-item.active .nav-label {
  color: white;
  font-weight: 600;
}

.nav-item.active .nav-description {
  color: rgba(255, 255, 255, 0.8);
}

.nav-icon {
  font-size: var(--font-size-lg);
  width: 24px;
  text-align: center;
  flex-shrink: 0;
}

.nav-content {
  display: flex;
  flex-direction: column;
  gap: var(--spacing-xs);
  flex: 1;
}

.nav-label {
  font-size: var(--font-size-sm);
  font-weight: 600;
  color: var(--color-text-primary);
  transition: var(--transition);
}

.nav-description {
  font-size: var(--font-size-xs);
  color: var(--color-text-secondary);
  transition: var(--transition);
}
```

```
.nav-indicator {
  position: absolute;
  right: 0;
  top: 50%;
  transform: translateY(-50%);
  width: 4px;
  height: 20px;
  background: white;
  border-radius: var(--border-radius-sm) 0 0 var(--border-radius-sm);
  box-shadow: var(--shadow-sm);
}

/* User Card in Sidebar */
.user-card {
  display: flex;
  align-items: center;
  gap: var(--spacing-md);
  padding: var(--spacing-md) var(--spacing-lg);
  margin: 0 var(--spacing-lg);
  background: var(--color-background-light);
  border-radius: var(--border-radius);
  border: 1px solid var(--color-border-light);
  transition: var(--transition);
}

.user-card:hover {
  background: var(--color-background-card);
  box-shadow: var(--shadow-sm);
}

.user-avatar {
  width: 40px;
  height: 40px;
  background: linear-gradient(135deg, var(--color-primary), var(--color-
accent));
  border-radius: var(--border-radius-full);
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: var(--font-size-lg);
  color: white;
  box-shadow: var(--shadow-sm);
}

.user-details {
  display: flex;
  flex-direction: column;
  gap: var(--spacing-xs);
}

.user-details .user-name {
  font-size: var(--font-size-sm);
  font-weight: 600;
```

```
  color: var(--color-text-primary);
}

.user-details .user-role {
  font-size: var(--font-size-xs);
  color: var(--color-text-secondary);
  font-weight: 500;
}

/* Main Content Area */
.main-content {
  flex: 1;
  background: var(--color-background);
  min-height: calc(100vh - 80px);
  overflow-y: auto;
}

.content-container {
  padding: var(--spacing-xl);
  max-width: 1200px;
  margin: 0 auto;
}

/* Loading Container */
.loading-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  gap: var(--spacing-lg);
}

.loading-container .loading-spinner {
  width: 48px;
  height: 48px;
  border: 4px solid rgba(255, 255, 255, 0.3);
  border-radius: 50%;
  border-top-color: white;
  animation: spin 1s linear infinite;
}

.loading-container span {
  font-size: var(--font-size-lg);
  font-weight: 500;
}

/* Responsive Design */
@media (max-width: 1024px) {
  .sidebar {
    width: 240px;
  }
```

```
  .content-container {
    padding: var(--spacing-lg);
  }
}

@media (max-width: 768px) {
  .app-layout {
    flex-direction: column;
  }

  .sidebar {
    width: 100%;
    height: auto;
    position: static;
    order: 2;
  }

  .main-content {
    order: 1;
  }

  .sidebar-nav {
    display: flex;
    overflow-x: auto;
    padding: var(--spacing-md) 0;
  }

  .nav-section {
    margin-bottom: 0;
    margin-right: var(--spacing-lg);
    min-width: 200px;
  }

  .nav-section-title {
    display: none;
  }

  .nav-item {
    flex-direction: column;
    text-align: center;
    padding: var(--spacing-sm) var(--spacing-md);
    min-width: 80px;
  }

  .nav-content {
    align-items: center;
  }

  .nav-description {
    display: none;
  }

  .user-card {
```

```
    display: none;
  }

  .content-container {
    padding: var(--spacing-md);
  }
}

@media (max-width: 480px) {
  .sidebar-header {
    padding: var(--spacing-lg);
  }

  .logo-text h3 {
    font-size: var(--font-size-base);
  }

  .logo-text p {
    font-size: var(--font-size-xs);
  }

  .nav-item {
    padding: var(--spacing-xs) var(--spacing-sm);
    min-width: 60px;
  }

  .nav-icon {
    font-size: var(--font-size-base);
  }

  .nav-label {
    font-size: var(--font-size-xs);
  }
}

/* Sidebar Scrollbar */
.sidebar::-webkit-scrollbar {
  width: 4px;
}

.sidebar::-webkit-scrollbar-track {
  background: var(--color-background-light);
}

.sidebar::-webkit-scrollbar-thumb {
  background: var(--color-border);
  border-radius: var(--border-radius-sm);
}

.sidebar::-webkit-scrollbar-thumb:hover {
  background: var(--color-text-secondary);
}

/* Animation for sidebar items */
```

```
.nav-item {
  animation: slideInLeft 0.3s ease-out;
}

@keyframes slideInLeft {
  from {
    opacity: 0;
    transform: translateX(-20px);
  }
  to {
    opacity: 1;
    transform: translateX(0);
  }
}

/* Staggered animation for nav items */
.nav-item:nth-child(1) { animation-delay: 0.1s; }
.nav-item:nth-child(2) { animation-delay: 0.2s; }
.nav-item:nth-child(3) { animation-delay: 0.3s; }
.nav-item:nth-child(4) { animation-delay: 0.4s; }
.nav-item:nth-child(5) { animation-delay: 0.5s; }
.nav-item:nth-child(6) { animation-delay: 0.6s; }


/* Simplified Header Styles */
.header-actions {
  display: flex;
  align-items: center;
  gap: var(--spacing-md);
}

.logout-btn {
  display: flex;
  align-items: center;
  gap: var(--spacing-xs);
  padding: var(--spacing-sm) var(--spacing-md);
  font-size: var(--font-size-sm);
  border-radius: var(--border-radius-full);
  background: rgba(255, 255, 255, 0.1);
  border: 1px solid rgba(255, 255, 255, 0.2);
  color: var(--color-text-primary);
  transition: var(--transition);
}

.logout-btn:hover {
  background: rgba(255, 255, 255, 0.2);
  transform: translateY(-1px);
}

/* Responsive header */
@media (max-width: 768px) {
  .header-content {
    flex-direction: column;
    gap: var(--spacing-md);
```

```
    text-align: center;
  }

  .header-left {
    align-items: center;
  }

  .header-subtitle {
    text-align: center;
  }
}

@media (max-width: 480px) {
  .app-header h1 {
    font-size: var(--font-size-lg);
  }

  .header-subtitle {
    font-size: var(--font-size-xs);
  }
}


/* History Page Styles */
.history-card {
  background: linear-gradient(135deg, var(--color-background-card), var(--
color-background-light));
}

/* History Statistics */
.history-stats {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
  gap: var(--spacing-lg);
  margin-bottom: var(--spacing-xl);
}

.stat-card {
  background: linear-gradient(135deg, var(--color-background-light), var(-
-color-background-card));
  padding: var(--spacing-lg);
  border-radius: var(--border-radius-lg);
  border: 1px solid var(--color-border-light);
  display: flex;
  align-items: center;
  gap: var(--spacing-md);
  transition: var(--transition);
  position: relative;
  overflow: hidden;
}

.stat-card::before {
  content: '';
  position: absolute;
```

```
  top: 0;
  left: 0;
  right: 0;
  height: 3px;
  background: linear-gradient(90deg, var(--color-primary), var(--color-
accent));
}

.stat-card:hover {
  transform: translateY(-2px);
  box-shadow: var(--shadow-md);
}

.stat-icon {
  font-size: var(--font-size-2xl);
  background: linear-gradient(135deg, var(--color-primary), var(--color-
accent));
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  background-clip: text;
}

.stat-content h3 {
  margin: 0 0 var(--spacing-xs) 0;
  color: var(--color-text-secondary);
  font-size: var(--font-size-sm);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
}

.stat-content p {
  margin: 0;
  color: var(--color-primary);
  font-size: var(--font-size-xl);
  font-weight: 700;
  font-family: 'Inter', monospace;
}

/* Filters Section */
.filters-section {
  background: var(--color-background-light);
  padding: var(--spacing-lg);
  border-radius: var(--border-radius-lg);
  border: 1px solid var(--color-border-light);
  margin-bottom: var(--spacing-xl);
}

.filters-section h3 {
  margin: 0 0 var(--spacing-lg) 0;
  color: var(--color-text-primary);
  font-size: var(--font-size-lg);
  font-weight: 600;
  display: flex;
```

```
  align-items: center;
  gap: var(--spacing-sm);
}

.filters-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
}

.filter-actions {
  display: flex;
  gap: var(--spacing-md);
  padding-top: var(--spacing-md);
  border-top: 1px solid var(--color-border-light);
}

/* Table Section */
.table-section {
  background: var(--color-background-light);
  border-radius: var(--border-radius-lg);
  border: 1px solid var(--color-border-light);
  overflow: hidden;
}

.table-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: var(--spacing-lg);
  background: linear-gradient(135deg, var(--color-background-card), var(--
color-background-light));
  border-bottom: 1px solid var(--color-border-light);
}

.table-header h3 {
  margin: 0;
  color: var(--color-text-primary);
  font-size: var(--font-size-lg);
  font-weight: 600;
  display: flex;
  align-items: center;
  gap: var(--spacing-sm);
}

.table-actions {
  display: flex;
  gap: var(--spacing-sm);
}

.table-container {
  overflow-x: auto;
  max-height: 600px;
```

```
  overflow-y: auto;
}

/* Enhanced Proposals Table */
.proposals-table {
  width: 100%;
  border-collapse: collapse;
  background: var(--color-background-card);
}

.proposals-table th {
  background: linear-gradient(135deg, var(--color-background-light), var(-
-color-background-card));
  padding: var(--spacing-md);
  text-align: left;
  font-weight: 700;
  color: var(--color-text-primary);
  font-size: var(--font-size-sm);
  text-transform: uppercase;
  letter-spacing: 0.05em;
  border-bottom: 2px solid var(--color-border);
  position: sticky;
  top: 0;
  z-index: 10;
}

.proposals-table td {
  padding: var(--spacing-md);
  border-bottom: 1px solid var(--color-border-light);
  vertical-align: middle;
  transition: var(--transition);
}

.proposals-table tr:hover {
  background: linear-gradient(90deg, var(--color-background-light), var(--
color-background-card));
  transform: scale(1.01);
}

.proposals-table tr:hover td {
  background: transparent;
}

/* Table Cell Styling */
.proposal-id {
  font-family: 'Inter', monospace;
  font-weight: 600;
  color: var(--color-primary);
  background: var(--color-primary-light);
  background: linear-gradient(135deg, rgba(37, 99, 235, 0.1), rgba(37, 99,
235, 0.05));
  padding: var(--spacing-xs) var(--spacing-sm);
  border-radius: var(--border-radius-sm);
  font-size: var(--font-size-xs);
```

```
}

.ministry-name {
  font-weight: 600;
  color: var(--color-text-primary);
}

.category-name {
  background: var(--color-secondary-light);
  background: linear-gradient(135deg, rgba(5, 150, 105, 0.1), rgba(5, 150,
105, 0.05));
  color: var(--color-secondary);
  padding: var(--spacing-xs) var(--spacing-sm);
  border-radius: var(--border-radius-sm);
  font-size: var(--font-size-xs);
  font-weight: 600;
}

.proposal-title {
  font-weight: 500;
  color: var(--color-text-primary);
  max-width: 200px;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
  display: block;
}

.amount {
  font-family: 'Inter', monospace;
  font-weight: 600;
  font-size: var(--font-size-sm);
}

.amount.requested {
  color: var(--color-warning);
}

.amount.approved {
  color: var(--color-success);
}

.date {
  font-size: var(--font-size-sm);
  color: var(--color-text-secondary);
  font-family: 'Inter', monospace;
}

/* Enhanced Status Badges */
.status-badge {
  display: inline-flex;
  align-items: center;
  gap: var(--spacing-xs);
  padding: var(--spacing-xs) var(--spacing-sm);
```

```
  border-radius: var(--border-radius-full);
  font-size: var(--font-size-xs);
  font-weight: 600;
  text-transform: uppercase;
  letter-spacing: 0.05em;
  transition: var(--transition);
}

.status-badge:hover {
  transform: translateY(-1px);
  box-shadow: var(--shadow-sm);
}

.status-icon {
  font-size: var(--font-size-sm);
}

/* No Data State */
.no-data {
  text-align: center;
  padding: var(--spacing-2xl);
  color: var(--color-text-secondary);
}

.no-data-icon {
  font-size: var(--font-size-4xl);
  margin-bottom: var(--spacing-lg);
  opacity: 0.5;
}

.no-data h3 {
  margin: 0 0 var(--spacing-md) 0;
  color: var(--color-text-primary);
  font-size: var(--font-size-xl);
  font-weight: 600;
}

.no-data p {
  margin: 0;
  font-size: var(--font-size-base);
  color: var(--color-text-secondary);
}

/* Loading State */
.loading {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  padding: var(--spacing-2xl);
  color: var(--color-text-secondary);
  font-style: italic;
  font-size: var(--font-size-lg);
  gap: var(--spacing-md);
```

```
}

.loading-spinner {
  width: 32px;
  height: 32px;
  border: 3px solid var(--color-border);
  border-radius: 50%;
  border-top-color: var(--color-primary);
  animation: spin 1s linear infinite;
}

/* Responsive Design */
@media (max-width: 1024px) {
  .history-stats {
    grid-template-columns: repeat(3, 1fr);
  }

  .filters-grid {
    grid-template-columns: repeat(2, 1fr);
  }
}

@media (max-width: 768px) {
  .history-stats {
    grid-template-columns: repeat(2, 1fr);
    gap: var(--spacing-md);
  }

  .filters-grid {
    grid-template-columns: 1fr;
  }

  .filter-actions {
    flex-direction: column;
  }

  .table-header {
    flex-direction: column;
    gap: var(--spacing-md);
    align-items: stretch;
  }

  .table-actions {
    justify-content: center;
  }

  .proposals-table {
    font-size: var(--font-size-sm);
  }

  .proposals-table th,
  .proposals-table td {
    padding: var(--spacing-sm);
  }
```

```
}

@media (max-width: 480px) {
  .history-stats {
    grid-template-columns: 1fr;
  }

  .stat-card {
    padding: var(--spacing-md);
  }

  .stat-content p {
    font-size: var(--font-size-lg);
  }

  .filters-section {
    padding: var(--spacing-md);
  }

  .table-header {
    padding: var(--spacing-md);
  }

  .proposal-title {
    max-width: 150px;
  }
}

/* Table Scrollbar */
.table-container::-webkit-scrollbar {
  width: 6px;
  height: 6px;
}

.table-container::-webkit-scrollbar-track {
  background: var(--color-background-light);
}

.table-container::-webkit-scrollbar-thumb {
  background: var(--color-border);
  border-radius: var(--border-radius-sm);
}

.table-container::-webkit-scrollbar-thumb:hover {
  background: var(--color-text-secondary);
}


/* Modal Styles */
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
```

```
  bottom: 0;
  background: rgba(0, 0, 0, 0.5);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1000;
  backdrop-filter: blur(4px);
}

.modal {
  background: var(--color-background-card);
  border-radius: var(--border-radius-lg);
  padding: var(--spacing-xl);
  max-width: 500px;
  width: 90%;
  max-height: 80vh;
  overflow-y: auto;
  box-shadow: var(--shadow-xl);
  border: 1px solid var(--color-border-light);
  position: relative;
}

.modal h3 {
  margin: 0 0 var(--spacing-lg) 0;
  color: var(--color-text-primary);
  font-size: var(--font-size-xl);
  font-weight: 600;
  text-align: center;
}

.modal p {
  margin: var(--spacing-sm) 0;
  color: var(--color-text-secondary);
  font-size: var(--font-size-sm);
}

.modal .form-group {
  margin-bottom: var(--spacing-lg);
}

.modal .form-group label {
  display: block;
  margin-bottom: var(--spacing-sm);
  font-weight: 600;
  color: var(--color-text-primary);
  font-size: var(--font-size-sm);
}

.modal .form-group input,
.modal .form-group textarea {
  width: 100%;
  padding: var(--spacing-md);
  border: 2px solid var(--color-border);
  border-radius: var(--border-radius);
```

```
  font-size: var(--font-size-base);
  font-family: var(--font-family);
  transition: var(--transition);
  background: var(--color-background-card);
}

.modal .form-group input:focus,
.modal .form-group textarea:focus {
  outline: none;
  border-color: var(--color-primary);
  box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.1);
}

.modal .form-group input:disabled,
.modal .form-group textarea:disabled {
  background: var(--color-background-light);
  color: var(--color-text-secondary);
  cursor: not-allowed;
}

.modal .form-actions {
  display: flex;
  gap: var(--spacing-md);
  margin-top: var(--spacing-xl);
  padding-top: var(--spacing-lg);
  border-top: 1px solid var(--color-border-light);
}

.modal .btn {
  flex: 1;
  padding: var(--spacing-md) var(--spacing-lg);
  font-size: var(--font-size-sm);
  font-weight: 600;
}

.modal .btn-danger {
  background: linear-gradient(135deg, var(--color-error), #dc2626);
  color: white;
  border: none;
}

.modal .btn-danger:hover {
  background: linear-gradient(135deg, #dc2626, #b91c1c);
  transform: translateY(-1px);
}

.modal .btn-danger:disabled {
  background: var(--color-text-secondary);
  cursor: not-allowed;
  transform: none;
}

/* Responsive modal */
@media (max-width: 768px) {
```

```
  .modal {
    width: 95%;
    padding: var(--spacing-lg);
    margin: var(--spacing-md);
  }

  .modal .form-actions {
    flex-direction: column;
  }

  .modal .btn {
    width: 100%;
  }
}
```

📁 backend

📄 auth.py

```
from datetime import datetime, timedelta
from typing import Optional
from jose import JWTError, jwt
import hashlib
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from sqlalchemy.orm import Session
from database import get_db, User as DBUser
from models import TokenData

# Configuration
SECRET_KEY = "your-secret-key-change-in-production"  # Change this in
production!
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

# Password hashing
# Simple password hashing for demo purposes

# JWT token scheme
security = HTTPBearer()

def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verify a password against its hash."""
    return hashlib.sha256(plain_password.encode()).hexdigest() ==
hashed_password

def get_password_hash(password: str) -> str:
    """Hash a password."""
    return hashlib.sha256(password.encode()).hexdigest()
```

```
def create_access_token(data: dict, expires_delta: Optional[timedelta] =
None):
    """Create a JWT access token."""
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

def authenticate_user(db: Session, username: str, password: str) ->
Optional[DBUser]:
    """Authenticate a user with username and password."""
    user = db.query(DBUser).filter(DBUser.username == username).first()
    if not user:
        return None
    if not verify_password(password, user.hashed_password):
        return None
    return user

def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security),
    db: Session = Depends(get_db)
) -> DBUser:
    """Get the current authenticated user from JWT token."""
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )

    try:
        token = credentials.credentials
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
        token_data = TokenData(username=username)
    except JWTError:
        raise credentials_exception

    user = db.query(DBUser).filter(DBUser.username ==
token_data.username).first()
    if user is None:
        raise credentials_exception
    return user

def require_role(required_role: str):
    """Decorator to require a specific role."""
    def role_checker(current_user: DBUser = Depends(get_current_user)) ->
DBUser:
```

```
        if current_user.role != required_role:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail=f"Access denied. Required role: {required_role}"
            )
        return current_user
    return role_checker

def require_finance_role(current_user: DBUser = Depends(get_current_user))
-> DBUser:
    """Require finance role."""
    return require_role("finance")(current_user)

def require_ministry_role(current_user: DBUser =
Depends(get_current_user)) -> DBUser:
    """Require ministry role."""
    return require_role("ministry")(current_user)
```

database.py

```
import sqlite3
from sqlalchemy import create_engine, Column, Integer, String, Float,
DateTime, ForeignKey, Boolean
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relationship
from datetime import datetime

# Database setup
DATABASE_URL = "sqlite:///./government_spending.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread":
False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
Base = declarative_base()

# Ministry model
class Ministry(Base):
    __tablename__ = "ministries"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, index=True, nullable=False)
    description = Column(String, nullable=True)
    created_at = Column(DateTime, default=datetime.utcnow)
    is_active = Column(Boolean, default=True)

    # Relationships
    users = relationship("User", back_populates="ministry")
    proposals = relationship("Proposal", back_populates="ministry")

# Category model
```

```
class Category(Base):
    __tablename__ = "categories"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, index=True, nullable=False)
    allocated_budget = Column(Float, nullable=False)
    remaining_budget = Column(Float, nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    proposals = relationship("Proposal", back_populates="category")


# User model (Authentication)
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, index=True, nullable=False)
    email = Column(String, unique=True, index=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    role = Column(String, nullable=False)  # "ministry" or "finance"
    ministry_id = Column(Integer, ForeignKey("ministries.id"),
nullable=True)  # Foreign key to Ministry
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    ministry = relationship("Ministry", back_populates="users")


# Proposal model (Phase 2)
class Proposal(Base):
    __tablename__ = "proposals"

    id = Column(Integer, primary_key=True, index=True)
    ministry_id = Column(Integer, ForeignKey("ministries.id"),
nullable=False)  # Foreign key to Ministry
    category_id = Column(Integer, ForeignKey("categories.id"),
nullable=False)
    title = Column(String, nullable=False)
    description = Column(String, nullable=True)
    requested_amount = Column(Float, nullable=False)
    status = Column(String, default="Pending", nullable=False)  #
Pending/Approved/Rejected
    approved_amount = Column(Float, nullable=True)
    decision_notes = Column(String, nullable=True)
    decided_at = Column(DateTime, nullable=True)
    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    ministry = relationship("Ministry", back_populates="proposals")
    category = relationship("Category", back_populates="proposals")
```

```
# Create tables

def create_tables():
    """Create all database tables including the new Ministry table."""
    from database import User as DBUser, Ministry as DBMinistry
    # Create all tables
    Base.metadata.create_all(bind=engine)

    # Create default ministries and users if they don't exist
    from auth import get_password_hash
    from sqlalchemy.orm import sessionmaker

    SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
    db = SessionLocal()

    try:
        # Create default ministry
        education_ministry = db.query(DBMinistry).filter(DBMinistry.name
== "Ministry of Education").first()
        if not education_ministry:
            education_ministry = DBMinistry(
                name="Ministry of Education",
                description="Government ministry responsible for education
policy and funding"
            )
            db.add(education_ministry)
            db.flush()  # Get the ID

        # Create default finance user
        finance_user = db.query(DBUser).filter(DBUser.username ==
"finance").first()
        if not finance_user:
            finance_user = DBUser(
                username="finance",
                email="finance@gov.com",
                hashed_password=get_password_hash("fin"),
                role="finance",
                ministry_id=None  # Finance users don't belong to a
ministry
            )
            db.add(finance_user)

        # Create default ministry user
        ministry_user = db.query(DBUser).filter(DBUser.username ==
"ministry").first()
        if not ministry_user:
            ministry_user = DBUser(
                username="ministry",
                email="ministry@gov.com",
                hashed_password=get_password_hash("min"),
                role="ministry",
                ministry_id=education_ministry.id  # Link to Ministry of
Education
```

```
                )
                db.add(ministry_user)

        db.commit()
        print("Default ministries and users created successfully")
    except Exception as e:
        print(f"Error creating default ministries and users: {e}")
        db.rollback()
    finally:
        db.close()

    # Lightweight migration for SQLite: add new columns if missing
    try:
        conn = sqlite3.connect(engine.url.database)
        cur = conn.cursor()
        cur.execute("PRAGMA table_info(proposals)")
        cols = {row[1] for row in cur.fetchall()}
        migrations = []
        if "approved_amount" not in cols:
            migrations.append("ALTER TABLE proposals ADD COLUMN
approved_amount REAL")
        if "decision_notes" not in cols:
            migrations.append("ALTER TABLE proposals ADD COLUMN
decision_notes TEXT")
        if "decided_at" not in cols:
            migrations.append("ALTER TABLE proposals ADD COLUMN decided_at
DATETIME")
        for sql in migrations:
            cur.execute(sql)
        if migrations:
            conn.commit()
    except Exception as e:
        # Best-effort; on failure, proceed (dev environment)
        pass
    finally:
        try:
            conn.close()
        except Exception:
            pass

# Database dependency

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

main.py

```
from fastapi import FastAPI, Depends, HTTPException, UploadFile, File
from datetime import timedelta
from auth import (
    authenticate_user, create_access_token, get_current_user,
    get_password_hash, require_finance_role, require_ministry_role
)
from models import UserCreate, UserLogin, Token, User as UserModel
from sqlalchemy.orm import Session
from typing import List
import uvicorn
import csv
import json
import io
from datetime import datetime
from fastapi.middleware.cors import CORSMiddleware

from database import get_db, create_tables, Category as DBCategory,
Proposal as DBProposal, User as DBUser, Ministry as DBMinistry
from models import Category, CategoryCreate, CategoryUpdate, Proposal,
ProposalCreate, ProposalUpdate, ProposalApprove, ProposalReject,
ProposalDelete, Ministry, MinistryCreate

# Create FastAPI app
app = FastAPI(title="Government Spending Tracker", version="1.0.0")
app.add_middleware(CORSMiddleware,
    allow_origins=["http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Create database tables on startup
@app.on_event("startup")
def startup_event():
    create_tables()

# ------------------ Authentication Endpoints ------------------

@app.post("/auth/register", response_model=UserModel)
def register_user(user_data: UserCreate, db: Session = Depends(get_db)):
    """Register a new user."""
    # Check if username already exists
    existing_user = db.query(DBUser).filter(DBUser.username ==
user_data.username).first()
    if existing_user:
        raise HTTPException(status_code=400, detail="Username already
registered")

    # Check if email already exists
    existing_email = db.query(DBUser).filter(DBUser.email ==
user_data.email).first()
    if existing_email:
        raise HTTPException(status_code=400, detail="Email already
```

```
registered")

    # Validate role
    if user_data.role not in ["ministry", "finance"]:
        raise HTTPException(status_code=400, detail="Role must be
'ministry' or 'finance'")

    # Validate ministry_id for ministry users
    if user_data.role == "ministry" and not user_data.ministry_id:
        raise HTTPException(status_code=400, detail="Ministry ID is
required for ministry users")

    # Validate ministry exists if provided
    if user_data.ministry_id:
        ministry = db.query(DBMinistry).filter(DBMinistry.id ==
user_data.ministry_id).first()
        if not ministry:
            raise HTTPException(status_code=400, detail="Invalid ministry
ID")

    # Create new user
    hashed_password = get_password_hash(user_data.password)
    db_user = DBUser(
        username=user_data.username,
        email=user_data.email,
        hashed_password=hashed_password,
        role=user_data.role,
        ministry_id=user_data.ministry_id
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

@app.post("/auth/login", response_model=Token)
def login_user(user_credentials: UserLogin, db: Session =
Depends(get_db)):
    """Login a user and return JWT token."""
    user = authenticate_user(db, user_credentials.username,
user_credentials.password)
    if not user:
        raise HTTPException(
            status_code=401,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=30)
    access_token = create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )
    return {
        "access_token": access_token,
        "token_type": "bearer",
        "user": user
```

```
    }

@app.get("/auth/me", response_model=UserModel)
def get_current_user_info(current_user: DBUser =
Depends(get_current_user), db: Session = Depends(get_db)):
    """Get current user information."""
    # Load ministry relationship if user has ministry_id
    if current_user.ministry_id:
        from sqlalchemy.orm import joinedload
        user_with_ministry =
db.query(DBUser).options(joinedload(DBUser.ministry)).filter(DBUser.id ==
current_user.id).first()
        return user_with_ministry
    return current_user



def startup_event():
    create_tables()

# CRUD endpoints for categories
@app.get("/categories", response_model=List[Category])
def get_categories(db: Session = Depends(get_db)):
    """Get all budget categories"""
    categories = db.query(DBCategory).all()
    return categories

@app.post("/categories", response_model=Category)
def create_category(category: CategoryCreate, db: Session =
Depends(get_db)):
    """Create a new budget category"""
    # Check if category name already exists
    existing = db.query(DBCategory).filter(DBCategory.name ==
category.name).first()
    if existing:
        raise HTTPException(status_code=400, detail="Category name already
exists")

    # Create new category with remaining_budget = allocated_budget
initially
    db_category = DBCategory(
        name=category.name,
        allocated_budget=category.allocated_budget,
        remaining_budget=category.allocated_budget
    )
    db.add(db_category)
    db.commit()
    db.refresh(db_category)
    return db_category

@app.get("/categories/{category_id}", response_model=Category)
def get_category(category_id: int, db: Session = Depends(get_db)):
    """Get a specific category by ID"""
    category = db.query(DBCategory).filter(DBCategory.id ==
```

```
    category_id).first()
    if not category:
        raise HTTPException(status_code=404, detail="Category not found")
    return category

@app.put("/categories/{category_id}", response_model=Category)
def update_category(category_id: int, category_update: CategoryUpdate, db:
Session = Depends(get_db)):
    """Update a category"""
    db_category = db.query(DBCategory).filter(DBCategory.id ==
category_id).first()
    if not db_category:
        raise HTTPException(status_code=404, detail="Category not found")

    # Update fields if provided
    if category_update.name is not None:
        # Check if new name already exists (excluding current category)
        existing = db.query(DBCategory).filter(
            DBCategory.name == category_update.name,
            DBCategory.id != category_id
        ).first()
        if existing:
            raise HTTPException(status_code=400, detail="Category name
already exists")
        db_category.name = category_update.name

    if category_update.allocated_budget is not None:
        # Update remaining budget proportionally
        old_allocated = db_category.allocated_budget
        new_allocated = category_update.allocated_budget
        ratio = new_allocated / old_allocated if old_allocated > 0 else 1
        db_category.remaining_budget = db_category.remaining_budget *
ratio
        db_category.allocated_budget = new_allocated

    db.commit()
    db.refresh(db_category)
    return db_category

@app.delete("/categories/{category_id}")
def delete_category(category_id: int, db: Session = Depends(get_db)):
    """Delete a category"""
    db_category = db.query(DBCategory).filter(DBCategory.id ==
category_id).first()
    if not db_category:
        raise HTTPException(status_code=404, detail="Category not found")

    db.delete(db_category)
    db.commit()
    return {"message": "Category deleted successfully"}

# ------------------ Ministry Endpoints ------------------

@app.get("/ministries", response_model=List[Ministry])
```

```
def get_ministries(db: Session = Depends(get_db)):
    """Get all ministries"""
    ministries = db.query(DBMinistry).filter(DBMinistry.is_active ==
True).all()
    return ministries

@app.post("/ministries", response_model=Ministry)
def create_ministry(ministry: MinistryCreate, db: Session =
Depends(get_db), current_user: DBUser = Depends(require_finance_role)):
    """Create a new ministry (Finance users only)"""
    # Check if ministry name already exists
    existing = db.query(DBMinistry).filter(DBMinistry.name ==
ministry.name).first()
    if existing:
        raise HTTPException(status_code=400, detail="Ministry name already
exists")

    db_ministry = DBMinistry(
        name=ministry.name,
        description=ministry.description
    )
    db.add(db_ministry)
    db.commit()
    db.refresh(db_ministry)
    return db_ministry

@app.post("/ministries/find-or-create", response_model=Ministry)
def find_or_create_ministry(ministry_name: str, db: Session =
Depends(get_db), current_user: DBUser = Depends(get_current_user)):
    """Find existing ministry or create new one if not found"""
    if not ministry_name or not ministry_name.strip():
        raise HTTPException(status_code=400, detail="Ministry name is
required")

    ministry_name = ministry_name.strip()

    # Check if ministry already exists
    existing =
db.query(DBMinistry).filter(DBMinistry.name.ilike(ministry_name)).first()
    if existing:
        return existing

    # Create new ministry
    db_ministry = DBMinistry(
        name=ministry_name,
        description=f"Ministry of {ministry_name}"
    )
    db.add(db_ministry)
    db.commit()
    db.refresh(db_ministry)
    return db_ministry

# Health check endpoint
@app.get("/")
```

```
def root():
    return {"message": "Government Spending Tracker API"}



# ------------------ Phase 2: Proposal Endpoints ------------------

@app.get("/proposals", response_model=List[Proposal])
def list_proposals(db: Session = Depends(get_db), ministry_id: int | None
= None, category_id: int | None = None, status: str | None = None):
    from sqlalchemy.orm import joinedload
    query = db.query(DBProposal).options(joinedload(DBProposal.ministry))
    if ministry_id:
        query = query.filter(DBProposal.ministry_id == ministry_id)
    if category_id:
        query = query.filter(DBProposal.category_id == category_id)
    if status:
        query = query.filter(DBProposal.status == status)
    return query.order_by(DBProposal.created_at.desc()).all()

@app.post("/proposals", response_model=Proposal)
def create_proposal(payload: ProposalCreate, db: Session =
Depends(get_db)):
    # Validate category exists
    category = db.query(DBCategory).filter(DBCategory.id ==
payload.category_id).first()
    if not category:
        raise HTTPException(status_code=400, detail="Category does not
exist")

    # Handle ministry - either by ID or name
    ministry = None
    if payload.ministry_id:
        # Find ministry by ID
        ministry = db.query(DBMinistry).filter(DBMinistry.id ==
payload.ministry_id).first()
        if not ministry:
            raise HTTPException(status_code=400, detail="Ministry does not
exist")
    elif payload.ministry_name:
        # Find or create ministry by name
        ministry_name = payload.ministry_name.strip()
        if not ministry_name:
            raise HTTPException(status_code=400, detail="Ministry name
cannot be empty")

        # Check if ministry exists (case-insensitive)
        ministry =
db.query(DBMinistry).filter(DBMinistry.name.ilike(ministry_name)).first()
        if not ministry:
            # Create new ministry
            ministry = DBMinistry(
                name=ministry_name,
                description=f"Ministry of {ministry_name}"
```

```
            )
            db.add(ministry)
            db.flush()  # Get the ID
    else:
        raise HTTPException(status_code=400, detail="Either ministry_id or
ministry_name is required")

    # Validate amount
    if payload.requested_amount is None or payload.requested_amount <= 0:
        raise HTTPException(status_code=400, detail="Requested amount must
be greater than 0")

    proposal = DBProposal(
        ministry_id=ministry.id,
        category_id=payload.category_id,
        title=payload.title,
        description=payload.description,
        requested_amount=payload.requested_amount,
        status="Pending",
    )
    db.add(proposal)
    db.commit()
    db.refresh(proposal)
    return proposal

@app.get("/proposals/{proposal_id}", response_model=Proposal)
def get_proposal(proposal_id: int, db: Session = Depends(get_db)):
    from sqlalchemy.orm import joinedload
    proposal =
db.query(DBProposal).options(joinedload(DBProposal.ministry)).filter(DBPro
posal.id == proposal_id).first()
    if not proposal:
        raise HTTPException(status_code=404, detail="Proposal not found")
    return proposal

@app.put("/proposals/{proposal_id}", response_model=Proposal)
def update_proposal(proposal_id: int, payload: ProposalUpdate, db: Session
= Depends(get_db)):
    proposal = db.query(DBProposal).filter(DBProposal.id ==
proposal_id).first()
    if not proposal:
        raise HTTPException(status_code=404, detail="Proposal not found")
    # Only edits in Pending state in Phase 2
    if proposal.status != "Pending":
        raise HTTPException(status_code=400, detail="Only pending
proposals can be edited in Phase 2")

    if payload.ministry is not None:
        proposal.ministry = payload.ministry
    if payload.category_id is not None:
        category = db.query(DBCategory).filter(DBCategory.id ==
payload.category_id).first()
        if not category:
            raise HTTPException(status_code=400, detail="Category does not
```

```
exist")
        proposal.category_id = payload.category_id
    if payload.title is not None:
        proposal.title = payload.title
    if payload.description is not None:
        proposal.description = payload.description
    if payload.requested_amount is not None:
        if payload.requested_amount <= 0:
            raise HTTPException(status_code=400, detail="Requested amount
must be greater than 0")
        proposal.requested_amount = payload.requested_amount
    # Ignore status changes in Phase 2 unless staying Pending
    if payload.status is not None and payload.status != "Pending":
        raise HTTPException(status_code=400, detail="Status changes are
not allowed in Phase 2")

    db.commit()
    db.refresh(proposal)
    return proposal

@app.delete("/proposals/{proposal_id}")
def delete_proposal(proposal_id: int, db: Session = Depends(get_db)):
    proposal = db.query(DBProposal).filter(DBProposal.id ==
proposal_id).first()
    if not proposal:
        raise HTTPException(status_code=404, detail="Proposal not found")
    if proposal.status != "Pending":
        raise HTTPException(status_code=400, detail="Only pending
proposals can be deleted in Phase 2")
    db.delete(proposal)
    db.commit()
    return {"message": "Proposal deleted successfully"}


# ------------------ Phase 3: Approval Endpoints ------------------

@app.post("/proposals/{proposal_id}/approve", response_model=Proposal)
def approve_proposal(proposal_id: int, body: ProposalApprove, db: Session
= Depends(get_db), current_user: DBUser = Depends(require_finance_role)):
    proposal = db.query(DBProposal).filter(DBProposal.id ==
proposal_id).first()
    if not proposal:
        raise HTTPException(status_code=404, detail="Proposal not found")
    if proposal.status != "Pending":
        raise HTTPException(status_code=400, detail="Only pending
proposals can be approved")
    if body.approved_amount is None or body.approved_amount <= 0:
        raise HTTPException(status_code=400, detail="approved_amount must
be > 0")
    if body.approved_amount > proposal.requested_amount:
        raise HTTPException(status_code=400, detail="approved_amount
exceeds requested amount")
    category = db.query(DBCategory).filter(DBCategory.id ==
proposal.category_id).with_for_update(nowait=False).first()
```

```
    if not category:
        raise HTTPException(status_code=400, detail="Category does not
exist")
    if body.approved_amount > category.remaining_budget:
        raise HTTPException(status_code=400, detail="Insufficient
remaining budget")
    # Apply decision atomically
    category.remaining_budget = category.remaining_budget -
body.approved_amount
    proposal.status = "Approved"
    proposal.approved_amount = body.approved_amount
    proposal.decision_notes = body.decision_notes
    proposal.decided_at = datetime.utcnow()
    db.commit()
    db.refresh(proposal)
    return proposal

@app.post("/proposals/{proposal_id}/reject", response_model=Proposal)
def reject_proposal(proposal_id: int, body: ProposalReject, db: Session =
Depends(get_db), current_user: DBUser = Depends(require_finance_role)):
    proposal = db.query(DBProposal).filter(DBProposal.id ==
proposal_id).first()
    if not proposal:
        raise HTTPException(status_code=404, detail="Proposal not found")
    if proposal.status != "Pending":
        raise HTTPException(status_code=400, detail="Only pending
proposals can be rejected")
    proposal.status = "Rejected"
    proposal.approved_amount = None
    proposal.decision_notes = body.decision_notes
    proposal.decided_at = datetime.utcnow()
    db.commit()
    db.refresh(proposal)
    return proposal


# ------------------ Phase 4: Contract Upload & Parsing ------------------

@app.post("/contracts/parse")
def parse_contract(file: UploadFile = File(...), db: Session =
Depends(get_db), current_user: DBUser = Depends(require_ministry_role)):
    filename = file.filename or ""
    content = file.file.read()
    drafts = []

    def map_category_id(category_name: str | None):
        if not category_name:
            return None
        q =
db.query(DBCategory).filter(DBCategory.name.ilike(category_name)).first()
        if q:
            return q.id
        q = db.query(DBCategory).filter(DBCategory.name.ilike(f"%
{category_name}%")).first()
```

```
        return q.id if q else None

    def map_ministry_id(ministry_name: str | None):
        if not ministry_name:
            return None
        q =
db.query(DBMinistry).filter(DBMinistry.name.ilike(ministry_name)).first()
        if q:
            return q.id
        q = db.query(DBMinistry).filter(DBMinistry.name.ilike(f"%
{ministry_name}%")).first()
        if q:
            return q.id
        # Create new ministry if not found
        new_ministry = DBMinistry(
            name=ministry_name,
            description=f"Ministry of {ministry_name}"
        )
        db.add(new_ministry)
        db.flush()
        return new_ministry.id

    def normalize_record(r: dict):
        ministry_name = r.get('ministry') or r.get('dept') or
r.get('ministry_name')
        category_name = r.get('category') or r.get('category_name') or
r.get('dept_category')
        title = r.get('title') or r.get('project') or r.get('subject')
        description = r.get('description') or r.get('details')
        amount = r.get('requested_amount')
        if amount in (None, ''):
            amount = r.get('amount') or r.get('value') or
r.get('requested')
        try:
            requested_amount = float(amount) if amount not in (None, '')
else None
        except Exception:
            requested_amount = None
        category_id = map_category_id(category_name)
        ministry_id = map_ministry_id(ministry_name)
        errors = []
        if not ministry_name:
            errors.append('missing ministry')
        if not title:
            errors.append('missing title')
        if requested_amount is None or requested_amount <= 0:
            errors.append('invalid amount')
        if category_id is None:
            errors.append('unknown category')
        dup = None
        if ministry_id and title and requested_amount is not None:
            dup = db.query(DBProposal).filter(
                DBProposal.ministry_id == ministry_id,
                DBProposal.title == title,
```

```
                DBProposal.requested_amount == requested_amount,
            ).first()
            if dup:
                errors.append('possible duplicate')
        return {
            'ministry_name': ministry_name,
            'ministry_id': ministry_id,
            'category_id': category_id,
            'category_name': category_name,
            'title': title,
            'description': description,
            'requested_amount': requested_amount,
            'errors': errors,
            'valid': len(errors) == 0,
        }

    try:
        if filename.lower().endswith('.json'):
            import json
            data = json.loads(content.decode('utf-8'))
            records = data if isinstance(data, list) else [data]
            for r in records:
                drafts.append(normalize_record(r))
        elif filename.lower().endswith('.csv'):
            import csv, io
            text = content.decode('utf-8')
            reader = csv.DictReader(io.StringIO(text))
            for r in reader:
                drafts.append(normalize_record(r))
        else:
            raise HTTPException(status_code=400, detail='Unsupported file
type. Use .json or .csv')
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=400, detail=f'Failed to parse
file: {str(e)}')

    return {'drafts': drafts}


# ------------------ Phase 5: Dashboard Summary ------------------

@app.get("/dashboard/summary")
def dashboard_summary(db: Session = Depends(get_db), current_user: DBUser
= Depends(require_finance_role)):
    # Per-category aggregates
    categories = db.query(DBCategory).all()
    # Sum approved amounts grouped by category_id
    from sqlalchemy import func
    approved_sums = dict(
        db.query(DBProposal.category_id,
func.coalesce(func.sum(DBProposal.approved_amount), 0.0))
          .filter(DBProposal.status == "Approved")
```

```
            .group_by(DBProposal.category_id)
            .all()
    )
    category_stats = []
    for c in categories:
        approved_total = float(approved_sums.get(c.id, 0.0) or 0.0)
        category_stats.append({
            "id": c.id,
            "name": c.name,
            "allocated_budget": float(c.allocated_budget),
            "remaining_budget": float(c.remaining_budget),
            "approved_total": approved_total,
        })

    # Per-ministry aggregates (requested vs approved)
    from sqlalchemy.orm import joinedload
    ministry_proposals =
db.query(DBMinistry).options(joinedload(DBMinistry.proposals)).all()

    ministry_stats = []
    for ministry in ministry_proposals:
        proposals = ministry.proposals
        requested_total = sum(p.requested_amount for p in proposals)
        approved_total = sum(p.approved_amount for p in proposals if
p.status == "Approved" and p.approved_amount)

        ministry_stats.append({
            "ministry_id": ministry.id,
            "ministry_name": ministry.name,
            "requested_total": float(requested_total),
            "approved_total": float(approved_total),
        })

    # Overall KPIs
    total_allocated = float(sum(c.allocated_budget for c in categories))
    total_remaining = float(sum(c.remaining_budget for c in categories))
    total_approved = float(sum(x[1] for x in approved_sums.items()))

    return {
        "categories": category_stats,
        "ministries": ministry_stats,
        "kpis": {
            "total_allocated": total_allocated,
            "total_remaining": total_remaining,
            "total_approved": total_approved,
        },
    }
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

models.py

```
from pydantic import BaseModel
from datetime import datetime
from typing import Optional, TYPE_CHECKING

# Pydantic models for API
class CategoryBase(BaseModel):
    name: str
    allocated_budget: float

class CategoryCreate(CategoryBase):
    pass

class CategoryUpdate(BaseModel):
    name: Optional[str] = None
    allocated_budget: Optional[float] = None

class Category(CategoryBase):
    id: int
    remaining_budget: float
    created_at: datetime

    class Config:
        from_attributes = True

# ------------------ Phase 2: Proposal Schemas ------------------

class ProposalBase(BaseModel):
    category_id: int
    title: str
    description: Optional[str] = None
    requested_amount: float

class ProposalCreate(ProposalBase):
    ministry_id: Optional[int] = None  # Either ministry_id or
ministry_name
    ministry_name: Optional[str] = None  # Either ministry_id or
ministry_name

class ProposalUpdate(BaseModel):
    ministry_id: Optional[int] = None
    category_id: Optional[int] = None
    title: Optional[str] = None
    description: Optional[str] = None
    requested_amount: Optional[float] = None
    status: Optional[str] = None  # still Pending in Phase 2

class Proposal(ProposalBase):
    id: int
    status: str
    approved_amount: float | None = None
    decision_notes: str | None = None
    decided_at: datetime | None = None
```

```
    created_at: datetime
    ministry: Optional['Ministry'] = None

    class Config:
        from_attributes = True


class ProposalApprove(BaseModel):
    approved_amount: float
    decision_notes: str | None = None

class ProposalReject(BaseModel):
    decision_notes: str | None = None

class ProposalDelete(BaseModel):
    reason: str  # Required reason for deletion




# Ministry Models
class MinistryBase(BaseModel):
    name: str
    description: Optional[str] = None

class MinistryCreate(MinistryBase):
    pass

class Ministry(MinistryBase):
    id: int
    is_active: bool
    created_at: datetime

    class Config:
        from_attributes = True

# Authentication Models
class UserBase(BaseModel):
    username: str
    email: str
    role: str  # "ministry" or "finance"
    ministry_id: Optional[int] = None

class UserCreate(UserBase):
    password: str

class UserLogin(BaseModel):
    username: str
    password: str

class User(UserBase):
    id: int
    is_active: bool
    created_at: datetime
```

```
    ministry: Optional[Ministry] = None  # Include ministry relationship

    class Config:
        from_attributes = True

class Token(BaseModel):
    access_token: str
    token_type: str
    user: User

class TokenData(BaseModel):
    username: Optional[str] = None
```