```
// firebase.js - Firebase Configuration
import { initializeApp } from 'firebase/app';
import { getFirestore } from 'firebase/firestore';
import { getAuth } from 'firebase/auth';
import { getStorage } from 'firebase/storage';
const firebaseConfig = {
  // Replace with your Firebase config
  apiKey: "your-api-key",
  authDomain: "your-project.firebaseapp.com",
  projectId: "your-project-id",
  storageBucket: "your-project.appspot.com",
  messagingSenderId: "123456789",
  appId: “your-app-id”
};
const app = initializeApp(firebaseConfig);
export const db = getFirestore(app);
export const auth = getAuth(app);
export const storage = getStorage(app);
// firebaseService.js - Database Operations
import {
  collection,
  addDoc,
  getDocs,
  doc,
  updateDoc,
  deleteDoc,
  query,
  orderBy
} from 'firebase/firestore';
import { db } from './firebase';
export const FirebaseService = {
  // Portfolio Projects
  async addProject(projectData) {
    try {
      const docRef = await addDoc(collection(db, 'projects'), {
        ...projectData,
        createdAt: new Date()
      });
      return docRef.id;
    } catch (error) {
      console.error('Error adding project:', error);
      throw error;
    }
```

```
},
async getProjects() {
  try {
    const q = query(collection(db, 'projects'), orderBy('createdAt', 'desc'));
    const querySnapshot = await getDocs(q);
    return querySnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error('Error fetching projects:', error);
    throw error;
  }
},
async updateProject(projectId, updatedData) {
  try {
    const projectRef = doc(db, 'projects', projectId);
    await updateDoc(projectRef, {
      ...updatedData,
      updatedAt: new Date()
    });
  } catch (error) {
    console.error('Error updating project:', error);
    throw error;
  }
},
async deleteProject(projectId) {
  try {
    await deleteDoc(doc(db, 'projects', projectId));
  } catch (error) {
    console.error('Error deleting project:', error);
    throw error;
  }
},
// Testimonials
async addTestimonial(testimonialData) {
  try {
    const docRef = await addDoc(collection(db, 'testimonials'), {
      ...testimonialData,
      createdAt: new Date()
    });
    return docRef.id;
  } catch (error) {
    console.error('Error adding testimonial:', error);
```

```
    throw error;
  }
},
async getTestimonials() {
  try {
    const q = query(collection(db, 'testimonials'), orderBy('createdAt', 'desc'));
    const querySnapshot = await getDocs(q);
    return querySnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error('Error fetching testimonials:', error);
    throw error;
  }
},
async updateTestimonial(testimonialId, updatedData) {
  try {
    const testimonialRef = doc(db, 'testimonials', testimonialId);
    await updateDoc(testimonialRef, {
      ...updatedData,
      updatedAt: new Date()
    });
  } catch (error) {
    console.error('Error updating testimonial:', error);
    throw error;
  }
},
async deleteTestimonial(testimonialId) {
  try {
    await deleteDoc(doc(db, 'testimonials', testimonialId));
  } catch (error) {
    console.error('Error deleting testimonial:', error);
    throw error;
  }
},
// Contact Form Submissions
async addContactSubmission(contactData) {
  try {
    const docRef = await addDoc(collection(db, 'contacts'), {
      ...contactData,
      createdAt: new Date(),
      status: ‘unread’
    });
```

```
    return docRef.id;
  } catch (error) {
    console.error('Error adding contact submission:', error);
    throw error;
  }
},
async getContactSubmissions() {
  try {
    const q = query(collection(db, 'contacts'), orderBy('createdAt', 'desc'));
    const querySnapshot = await getDocs(q);
    return querySnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error('Error fetching contact submissions:', error);
    throw error;
  }
},
async updateContactStatus(contactId, status) {
  try {
    const contactRef = doc(db, 'contacts', contactId);
    await updateDoc(contactRef, {
      status,
      updatedAt: new Date()
    });
  } catch (error) {
    console.error('Error updating contact status:', error);
    throw error;
  }
};
// authService.js - Authentication Service
import { signInWithEmailAndPassword, signOut, onAuthStateChanged } from 'firebase/auth';
import { auth } from './firebase';
export const AuthService = {
  async signIn(email, password) {
    try {
      const userCredential = await signInWithEmailAndPassword(auth, email, password);
      return userCredential.user;
    } catch (error) {
      console.error('Sign in error:', error);
      throw error;
    }
```

```
  },
  async signOut() {
    try {
      await signOut(auth);
    } catch (error) {
      console.error('Sign out error:', error);
      throw error;
    }
  },
  onAuthStateChanged(callback) {
    return onAuthStateChanged(auth, callback);
  }
};
// AdminPanel.jsx - Admin Dashboard Component
import React, { useState, useEffect } from 'react';
import { FirebaseService } from './firebaseService';
import { AuthService } from './authService';
import { Plus, Edit, Trash2, Eye, LogOut, Upload } from 'lucide-react';
const AdminPanel = () => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [activeTab, setActiveTab] = useState('projects');
  const [projects, setProjects] = useState([]);
  const [testimonials, setTestimonials] = useState([]);
  const [contacts, setContacts] = useState([]);
  const [showModal, setShowModal] = useState(false);
  const [editingItem, setEditingItem] = useState(null);
  // Login form state
  const [loginData, setLoginData] = useState({ email: '', password: '' });

// Project form state
const [projectForm, setProjectForm] = useState({
  title: '',
  description: '',
  image: '',
  category: '',
  link: ‘’
});
// Testimonial form state
const [testimonialForm, setTestimonialForm] = useState({
  name: '',
  company: '',
  text: '',
  rating: 5
```

```
});
useEffect(() => {
  const unsubscribe = AuthService.onAuthStateChanged((user) => {
    setUser(user);
    setLoading(false);
  });
  return () => unsubscribe();
}, []);
useEffect(() => {
  if (user) {
    loadData();
  }
}, [user, activeTab]);
const loadData = async () => {
  try {
    if (activeTab === 'projects') {
      const projectsData = await FirebaseService.getProjects();
      setProjects(projectsData);
    } else if (activeTab === 'testimonials') {
      const testimonialsData = await FirebaseService.getTestimonials();
      setTestimonials(testimonialsData);
    } else if (activeTab === 'contacts') {
      const contactsData = await FirebaseService.getContactSubmissions();
      setContacts(contactsData);
    }
  } catch (error) {
    console.error('Error loading data:', error);
  }
};
const handleLogin = async (e) => {
  e.preventDefault();
  try {
    await AuthService.signIn(loginData.email, loginData.password);
  } catch (error) {
    alert('Login failed: ' + error.message);
  }
};
const handleLogout = async () => {
  try {
    await AuthService.signOut();
  } catch (error) {
    console.error('Logout error:', error);
  }
};
```

```
const handleSaveProject = async (e) => {
  e.preventDefault();
  try {
    if (editingItem) {
      await FirebaseService.updateProject(editingItem.id, projectForm);
    } else {
      await FirebaseService.addProject(projectForm);
    }
    setShowModal(false);
    setEditingItem(null);
    setProjectForm({ title: '', description: '', image: '', category: '', link: '' });
    loadData();
  } catch (error) {
    alert('Error saving project: ' + error.message);
  }
};
const handleSaveTestimonial = async (e) => {
  e.preventDefault();
  try {
    if (editingItem) {
      await FirebaseService.updateTestimonial(editingItem.id, testimonialForm);
    } else {
      await FirebaseService.addTestimonial(testimonialForm);
    }
    setShowModal(false);
    setEditingItem(null);
    setTestimonialForm({ name: '', company: '', text: '', rating: 5 });
    loadData();
  } catch (error) {
    alert('Error saving testimonial: ' + error.message);
  }
};
const handleDelete = async (type, id) => {
  if (window.confirm('Are you sure you want to delete this item?')) {
    try {
      if (type === 'project') {
        await FirebaseService.deleteProject(id);
      } else if (type === 'testimonial') {
        await FirebaseService.deleteTestimonial(id);
      }
      loadData();
    } catch (error) {
      alert('Error deleting item: ' + error.message);
    }
```

```
  }
};
const handleContactStatusUpdate = async (contactId, status) => {
  try {
    await FirebaseService.updateContactStatus(contactId, status);
    loadData();
  } catch (error) {
    alert('Error updating contact status: ' + error.message);
  }
};
if (loading) {
  return <div className="min-h-screen flex items-center justify-center">
    <div className="text-xl">Loading...</div>
  </div>;
}
if (!user) {
  return (
    <div className="min-h-screen bg-gray-100 flex items-center justify-center">
      <div className="bg-white p-8 rounded-2xl shadow-lg w-full max-w-md">
        <h1 className="text-2xl font-bold text-center mb-6">Admin Login</h1>
        <form onSubmit={handleLogin} className="space-y-4">
          <input
            type="email"
            placeholder="Email"
            value={loginData.email}
            onChange={(e) => setLoginData({...loginData, email: e.target.value})}
            className="w-full p-3 border rounded-lg focus:ring-2 focus:ring-blue-500"
            required
          />
          <input
            type="password"
            placeholder="Password"
            value={loginData.password}
            onChange={(e) => setLoginData({...loginData, password: e.target.value})}
            className="w-full p-3 border rounded-lg focus:ring-2 focus:ring-blue-500"
            required
          />
          <button
            type="submit"
            className="w-full bg-blue-600 text-white p-3 rounded-lg hover:bg-blue-700 transition-colors"
          >
            Sign In
          </button>
        </form>
```

```
      </div>
    </div>
  );
}
return (
  <div className="min-h-screen bg-gray-100">
    <nav className="bg-white shadow-sm p-4">
      <div className="max-w-7xl mx-auto flex justify-between items-center">
        <h1 className="text-2xl font-bold">Admin Panel</h1>
        <button
          onClick={handleLogout}
          className="flex items-center space-x-2 bg-red-600 text-white px-4 py-2 rounded-lg hover:bg-red-700 transition-colors"
        >
          <LogOut className="w-4 h-4" />
          <span>Logout</span>
        </button>
      </div>
    </nav>
    <div className="max-w-7xl mx-auto p-6">
      {/* Tabs */}
      <div className="bg-white rounded-lg shadow-sm mb-6">
        <div className="flex border-b">
          {['projects', 'testimonials', 'contacts'].map((tab) => (
            <button
              key={tab}
              onClick={() => setActiveTab(tab)}
              className={`px-6 py-3 font-medium capitalize ${
                activeTab === tab
                  ? 'text-blue-600 border-b-2 border-blue-600'
                  : 'text-gray-500 hover:text-gray-700'
              }`}
            >
              {tab}
            </button>
          ))}
        </div>
      </div>
      {/* Projects Tab */}
      {activeTab === 'projects' && (
        <div className="bg-white rounded-lg shadow-sm p-6">
          <div className="flex justify-between items-center mb-6">
            <h2 className="text-xl font-bold">Portfolio Projects</h2>
            <button
              onClick={() => {
```

```
      setShowModal(true);
      setEditingItem(null);
      setProjectForm({ title: '', description: '', image: '', category: '', link: '' });
    }}
    className="bg-blue-600 text-white px-4 py-2 rounded-lg hover:bg-blue-700 transition-colors flex items-center space-x-2"
  >
    <Plus className="w-4 h-4" />
    <span>Add Project</span>
  </button>
</div>

<div className="grid gap-4">
  {projects.map((project) => (
    <div key={project.id} className="border rounded-lg p-4 flex justify-between items-center">
      <div>
        <h3 className="font-semibold">{project.title}</h3>
        <p className="text-gray-600 text-sm">{project.category}</p>
        <p className="text-gray-500 text-sm mt-1">{project.description}</p>
      </div>
      <div className="flex space-x-2">
        <button
          onClick={() => {
            setEditingItem(project);
            setProjectForm(project);
            setShowModal(true);
          }}
          className="p-2 text-blue-600 hover:bg-blue-50 rounded"
        >
          <Edit className="w-4 h-4" />
        </button>
        <button
          onClick={() => handleDelete('project', project.id)}
          className="p-2 text-red-600 hover:bg-red-50 rounded"
        >
          <Trash2 className="w-4 h-4" />
        </button>
      </div>
    </div>
  ))}
</div>
)}
{/* Testimonials Tab */}
{activeTab === 'testimonials' && (
```

```
<div className="bg-white rounded-lg shadow-sm p-6">
  <div className="flex justify-between items-center mb-6">
    <h2 className="text-xl font-bold">Testimonials</h2>
    <button
      onClick={() => {
        setShowModal(true);
        setEditingItem(null);
        setTestimonialForm({ name: '', company: '', text: '', rating: 5 });
      }}
      className="bg-blue-600 text-white px-4 py-2 rounded-lg hover:bg-blue-700 transition-colors flex items-center space-x-2"
    >
      <Plus className="w-4 h-4" />
      <span>Add Testimonial</span>
    </button>
  </div>

  <div className="grid gap-4">
    {testimonials.map((testimonial) => (
      <div key={testimonial.id} className="border rounded-lg p-4 flex justify-between items-start">
        <div>
          <h3 className="font-semibold">{testimonial.name}</h3>
          <p className="text-gray-600 text-sm">{testimonial.company}</p>
          <p className="text-gray-500 text-sm mt-2">"{testimonial.text}"</p>
          <p className="text-yellow-500 text-sm mt-1">★ {testimonial.rating}/5</p>
        </div>
        <div className="flex space-x-2">
          <button
            onClick={() => {
              setEditingItem(testimonial);
              setTestimonialForm(testimonial);
              setShowModal(true);
            }}
            className="p-2 text-blue-600 hover:bg-blue-50 rounded"
          >
            <Edit className="w-4 h-4" />
          </button>
          <button
            onClick={() => handleDelete('testimonial', testimonial.id)}
            className="p-2 text-red-600 hover:bg-red-50 rounded"
          >
            <Trash2 className="w-4 h-4" />
          </button>
        </div>
      </div>
```

```
      ))}
    </div>
  </div>
)}
{/* Contacts Tab */}
{activeTab === 'contacts' && (
  <div className="bg-white rounded-lg shadow-sm p-6">
    <h2 className="text-xl font-bold mb-6">Contact Submissions</h2>
    <div className="grid gap-4">
      {contacts.map((contact) => (
        <div key={contact.id} className="border rounded-lg p-4">
          <div className="flex justify-between items-start mb-3">
            <div>
              <h3 className="font-semibold">{contact.name}</h3>
              <p className="text-gray-600 text-sm">{contact.email}</p>
            </div>
            <span className={`px-2 py-1 rounded text-xs ${
              contact.status === 'read'
                ? 'bg-green-100 text-green-800'
                : 'bg-yellow-100 text-yellow-800'
            }`}>
              {contact.status}
            </span>
          </div>
          <p className="text-gray-700 mb-3">{contact.message}</p>
          <div className="flex justify-between items-center text-sm text-gray-500">
            <span>{new Date(contact.createdAt?.toDate()).toLocaleDateString()}</span>
            <button
              onClick={() => handleContactStatusUpdate(
                contact.id,
                contact.status === 'unread' ? 'read' : 'unread'
              )}
              className="text-blue-600 hover:underline"
            >
              Mark as {contact.status === 'unread' ? 'Read' : 'Unread'}
            </button>
          </div>
        </div>
      ))}
    </div>
  </div>
)}
</div>
{/* Modal for adding/editing projects and testimonials */}
```

```
{showModal && (
 <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
  <div className="bg-white rounded-lg p-6 w-full max-w-md max-h-screen overflow-y-auto">
   <h3 className="text-lg font-bold mb-4">
    {editingItem ? 'Edit' : 'Add'} {activeTab.slice(0, -1)}
   </h3>

   {activeTab === 'projects' ? (
    <form onSubmit={handleSaveProject} className="space-y-4">
     <input
      type="text"
      placeholder="Project Title"
      value={projectForm.title}
      onChange={(e) => setProjectForm({...projectForm, title: e.target.value})}
      className="w-full p-3 border rounded-lg"
      required
     />
     <input
      type="text"
      placeholder="Category"
      value={projectForm.category}
      onChange={(e) => setProjectForm({...projectForm, category: e.target.value})}
      className="w-full p-3 border rounded-lg"
      required
     />
     <textarea
      placeholder="Description"
      value={projectForm.description}
      onChange={(e) => setProjectForm({...projectForm, description: e.target.value})}
      className="w-full p-3 border rounded-lg"
      rows="3"
      required
     />
     <input
      type="url"
      placeholder="Image URL"
      value={projectForm.image}
      onChange={(e) => setProjec
```