--- frontend\.env ---

```
VITE_API_URL=http://localhost:5000/api
```

--- frontend\eslint.config.js ---

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import { defineConfig, globalIgnores } from 'eslint/config'

export default defineConfig([
  globalIgnores(['dist']),
  {
    files: ['**/*.{js,jsx}'],
    extends: [
      js.configs.recommended,
      reactHooks.configs['recommended-latest'],
      reactRefresh.configs.vite,
    ],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
      parserOptions: {
        ecmaVersion: 'latest',
        ecmaFeatures: { jsx: true },
        sourceType: 'module',
      },
    },
    rules: {
      'no-unused-vars': ['error', { varsIgnorePattern: '^[A-Z_]' }],
    },
  },
])
```

--- frontend\index.html ---

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>gigconnect-frontend</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

--- frontend\package.json ---

```
{
  "name": "gigconnect-frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.15.0",
    "axios": "^1.5.0",
    "socket.io-client": "^4.7.2"
  },
  "devDependencies": {
    "@types/react": "^18.2.15",
    "@types/react-dom": "^18.2.7",
    "@vitejs/plugin-react": "^4.0.3",
    "eslint": "^8.45.0",
    "eslint-plugin-react": "^7.32.2",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.3",
    "vite": "^4.4.5"
  }
}
```

--- frontend\README.md ---

```
# React + Vite

This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.

Currently, two official plugins are available:

- [@vitejs/plugin-react](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react) uses [Ba
- [@vitejs/plugin-react-swc](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react-swc)

## React Compiler

The React Compiler is not enabled on this template because of its impact on dev & build performances. To add i

## Expanding the ESLint configuration

If you are developing a production application, we recommend using TypeScript with type-aware lint rules enabl
```

--- frontend\vite.config.js ---

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:5000',
        changeOrigin: true
      }
    }
  }
})
```

--- frontend\src\App.css ---

```
/* Global Styles */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: #f8f9fa;
  color: #2c3e50;
  line-height: 1.6;
}

.app {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}

.main-content {
  flex: 1;
  padding-top: 80px; /* Account for fixed navbar */
}
```

```css
/* Common Component Styles */
.btn-primary {
  background: #3498db;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 6px;
  cursor: pointer;
  font-weight: 600;
  text-decoration: none;
  display: inline-block;
  text-align: center;
  transition: all 0.3s ease;
  font-size: 14px;
}

.btn-primary:hover {
  background: #2980b9;
  transform: translateY(-1px);
}

.btn-secondary {
  background: #95a5a6;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 6px;
  cursor: pointer;
  font-weight: 600;
  transition: all 0.3s ease;
}

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

.error-message {
  background: #e74c3c;
  color: white;
  padding: 12px 16px;
  border-radius: 6px;
  margin-bottom: 20px;
  text-align: center;
}

.loading {
  text-align: center;
  padding: 40px;
  font-size: 18px;
  color: #7f8c8d;
}

/* Form Styles */
.form-group {
  margin-bottom: 20px;
}

.form-group label {
  display: block;
  margin-bottom: 8px;
  font-weight: 600;
  color: #2c3e50;
}

.form-group input,
.form-group textarea,
.form-group select {
  width: 100%;
  padding: 12px 16px;
  border: 2px solid #ecf0f1;
  border-radius: 6px;
  font-size: 16px;
  transition: border-color 0.3s ease;
  font-family: inherit;
}
```

```
.form-group input:focus,
.form-group textarea:focus,
.form-group select:focus {
  outline: none;
  border-color: #3498db;
}

.form-group textarea {
  resize: vertical;
  min-height: 100px;
}

/* Card Styles */
.card {
  background: white;
  border-radius: 10px;
  padding: 25px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  margin-bottom: 20px;
}

/* Utility Classes */
.text-center {
  text-align: center;
}

.text-left {
  text-align: left;
}

.text-right {
  text-align: right;
}

.mt-1 { margin-top: 10px; }
.mt-2 { margin-top: 20px; }
.mt-3 { margin-top: 30px; }
.mb-1 { margin-bottom: 10px; }
.mb-2 { margin-bottom: 20px; }
.mb-3 { margin-bottom: 30px; }

.hidden {
  display: none;
}

/* Responsive */
@media (max-width: 768px) {
  .main-content {
    padding-top: 70px;
  }

  .card {
    padding: 20px;
    margin: 10px;
  }
}

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

.fade-in {
  animation: fadeIn 0.5s ease-out;
}
```

--- frontend\src\App.jsx ---

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider, useAuth } from './contexts/AuthContext';
import Navbar from './components/common/Navbar';
import Login from './pages/auth/Login';
import Register from './pages/auth/Register';
import Dashboard from './pages/Dashboard';
import GigFeed from './pages/GigFeed';
import GigDetails from './pages/GigDetails';
import Profile from './pages/Profile';
import Messages from './pages/Messages';
import CreateGig from './pages/CreateGig';

function ProtectedRoute({ children }) {
  const { user } = useAuth();
  return user ? children : <Navigate to="/login" />;
}

function App() {
  return (
    <AuthProvider>
      <Router>
        <div className="app">
          <Navbar />
          <main className="main-content">
            <Routes>
              <Route path="/login" element={<Login />} />
              <Route path="/register" element={<Register />} />
              <Route path="/" element={<ProtectedRoute><Dashboard /></ProtectedRoute>} />
              <Route path="/gigs" element={<ProtectedRoute><GigFeed /></ProtectedRoute>} />
              <Route path="/gigs/create" element={<ProtectedRoute><CreateGig /></ProtectedRoute>} />
              <Route path="/gigs/:id" element={<ProtectedRoute><GigDetails /></ProtectedRoute>} />
              <Route path="/profile" element={<ProtectedRoute><Profile /></ProtectedRoute>} />
              <Route path="/messages" element={<ProtectedRoute><Messages /></ProtectedRoute>} />
            </Routes>
          </main>
        </div>
      </Router>
    </AuthProvider>
  );
}

export default App;
```

--- frontend\src\index.css ---

```
:root {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  line-height: 1.5;
  font-weight: 400;

  color-scheme: light;
  color: #2c3e50;
  background-color: #ffffff;

  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

* {
  box-sizing: border-box;
}

html {
  scroll-behavior: smooth;
}

body {
  margin: 0;
  padding: 0;
  min-height: 100vh;
  background: #f8f9fa;
}
```

```css
#root {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}

h1, h2, h3, h4, h5, h6 {
  color: #2c3e50;
  margin-bottom: 0.5em;
}

p {
  margin-bottom: 1em;
}

a {
  color: #3498db;
  text-decoration: none;
  transition: color 0.3s ease;
}

a:hover {
  color: #2980b9;
}

button {
  font-family: inherit;
  cursor: pointer;
}

input, textarea, select {
  font-family: inherit;
}

/* Scrollbar Styling */
::-webkit-scrollbar {
  width: 8px;
}

::-webkit-scrollbar-track {
  background: #f1f1f1;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb {
  background: #c1c1c1;
  border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
  background: #a8a8a8;
}

/* Focus styles for accessibility */
button:focus-visible,
input:focus-visible,
textarea:focus-visible,
select:focus-visible {
  outline: 2px solid #3498db;
  outline-offset: 2px;
}

/* Loading animation */
@keyspin spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}

.loading-spinner {
  border: 3px solid #f3f3f3;
  border-top: 3px solid #3498db;
  border-radius: 50%;
  width: 20px;
  height: 20px;
  animation: spin 1s linear infinite;
  display: inline-block;
  margin-right: 10px;
}
```

--- frontend\src\main.jsx ---

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './styles/App.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

--- frontend\src\components\SearchFilters.jsx ---

```
import React, { useState } from 'react';
import '../../styles/SearchFilters.css';

const SearchFilters = ({ onFilter, type = 'gigs' }) => {
  const [filters, setFilters] = useState({
    category: '',
    skill: '',
    location: '',
    minBudget: '',
    maxBudget: '',
    minRate: '',
    maxRate: ''
  });

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

  const handleSubmit = (e) => {
    e.preventDefault();
    onFilter(filters);
  };

  const handleReset = () => {
    setFilters({
      category: '',
      skill: '',
      location: '',
      minBudget: '',
      maxBudget: '',
      minRate: '',
      maxRate: ''
    });
    onFilter({});
  };

  return (
    <div className="search-filters">
      <form onSubmit={handleSubmit} className="filters-form">
        <div className="filter-row">
          <div className="filter-group">
            <label>Location</label>
            <input
              type="text"
              name="location"
              value={filters.location}
              onChange={handleChange}
              placeholder="Enter location"
            />
          </div>
          {type === 'gigs' ? (
            <>
              <div className="filter-group">
                <label>Category</label>
                <input
                  type="text"
                  name="category"
                  value={filters.category}
                  onChange={handleChange}
                  placeholder="e.g., Web Development"
                />
              </div>
              <div className="filter-group">
                <label>Skill</label>
                <input
                  type="text"
                  name="skill"
                  value={filters.skill}
                  onChange={handleChange}
                  placeholder="e.g., React"
                />
              </div>
            </>
          ) : (
            <div className="filter-group">
              <label>Skill</label>
              <input
                type="text"
                name="skill"
                value={filters.skill}
                onChange={handleChange}
                placeholder="e.g., React"
              />
            </div>
          )}
        </div>

        <div className="filter-row">
          {type === 'gigs' ? (
            <>
              <div className="filter-group">
                <label>Min Budget ($)</label>
                <input
                  type="number"
                  name="minBudget"
                  value={filters.minBudget}
                  onChange={handleChange}
                  placeholder="0"
                />
              </div>
              <div className="filter-group">
                <label>Max Budget ($)</label>
                <input
                  type="number"
                  name="maxBudget"
                  value={filters.maxBudget}
                  onChange={handleChange}
                  placeholder="1000"
                />
              </div>
            </>
          ) : (
            <>
              <div className="filter-group">
                <label>Min Rate ($/hr)</label>
                <input
                  type="number"
                  name="minRate"
                  value={filters.minRate}
                  onChange={handleChange}
                  placeholder="0"
                />
              </div>
              <div className="filter-group">
```

```
                <label>Max Rate ($/hr)</label>
                <input
                  type="number"
                  name="maxRate"
                  value={filters.maxRate}
                  onChange={handleChange}
                  placeholder="100"
                />
              </div>
            </>
          )}
        </div>

        <div className="filter-actions">
          <button type="submit" className="btn-primary">Apply Filters</button>
          <button type="button" onClick={handleReset} className="btn-secondary">
            Reset
          </button>
        </div>
      </form>
    </div>
  );
};

export default SearchFilters;
```

--- frontend\src\components\common\Navbar.jsx ---

```
import React from 'react';
import { Link, useLocation, useNavigate } from 'react-router-dom';
import { useAuth } from '../../contexts/AuthContext';
import '../../styles/Navbar.css';

function Navbar() {
  const { user, logout } = useAuth();
  const location = useLocation();
  const navigate = useNavigate();

  const handleLogout = () => {
    logout();
    navigate('/login');
  };

  return (
    <nav className="navbar">
      <div className="navbar-container">
        <Link to="/" className="navbar-brand">
          GigConnect
        </Link>

        <div className="navbar-menu">
          {user ? (
            <>
              <Link
                to="/gigs"
                className={`nav-link ${location.pathname === '/gigs' ? 'active' : ''}`}
              >
                Find Gigs
              </Link>
              {user.userType === 'client' && (
                <Link
                  to="/gigs/create"
                  className={`nav-link ${location.pathname === '/gigs/create' ? 'active' : ''}`}
                >
                  Post Gig
                </Link>
              )}
              <Link
                to="/messages"
                className={`nav-link ${location.pathname === '/messages' ? 'active' : ''}`}
              >
                Messages
              </Link>
              <Link
                to="/profile"
                className={`nav-link ${location.pathname === '/profile' ? 'active' : ''}`}
              >
                Profile
              </Link>
              <span className="user-welcome">Hello, {user.firstName}</span>
              <button onClick={handleLogout} className="logout-btn">
                Logout
              </button>
            </>
          ) : (
            <>
              <Link to="/login" className="nav-link">Login</Link>
              <Link to="/register" className="nav-link">Register</Link>
            </>
          )}
        </div>
      </div>
    </nav>
  );
}

export default Navbar;
```

--- frontend\src\contexts\AuthContext.jsx ---

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

const AuthContext = createContext();

export function useAuth() {
  return useContext(AuthContext);
}

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

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      authService.getProfile()
        .then(userData => setUser(userData))
        .catch(() => {
          localStorage.removeItem('token');
          setUser(null);
        })
        .finally(() => setLoading(false));
    } else {
      setLoading(false);
    }
  }, []);

  const login = async (email, password) => {
    const response = await authService.login(email, password);
    setUser(response.user);
    localStorage.setItem('token', response.token);
    return response;
  };

  const register = async (userData) => {
    const response = await authService.register(userData);
    setUser(response.user);
    localStorage.setItem('token', response.token);
    return response;
  };

  const logout = () => {
    setUser(null);
    localStorage.removeItem('token');
  };

  const value = {
    user,
    login,
    register,
    logout,
    loading
  };

  return (
    <AuthContext.Provider value={value}>
      {!loading && children}
    </AuthContext.Provider>
  );
}
```

--- frontend\src\pages\CreateGig.jsx ---

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';
import { gigService } from '../services/gigService';
import '../styles/CreateGig.css';

function CreateGig() {
  const [formData, setFormData] = useState({
    title: '',
    description: '',
    skillsRequired: '',
    budget: '',
    location: '',
    duration: ''
  });
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  const { user } = useAuth();
  const navigate = useNavigate();

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

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

    if (user.userType !== 'client') {
      setError('Only clients can create gigs');
      setLoading(false);
      return;
    }

    try {
      await gigService.createGig(formData);
      navigate('/gigs');
    } catch (err) {
      setError(err.response?.data?.message || 'Failed to create gig');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="create-gig">
      <div className="create-gig-header">
        <h1>Post a New Gig</h1>
        <p>Find the perfect freelancer for your project</p>
      </div>

      <div className="create-gig-form">
        {error && <div className="error-message">{error}</div>}

        <form onSubmit={handleSubmit}>
          <div className="form-group">
            <label htmlFor="title">Gig Title *</label>
            <input
              type="text"
              id="title"
              name="title"
              value={formData.title}
              onChange={handleChange}
              required
              placeholder="e.g., Website Developer Needed"
            />
          </div>

          <div className="form-group">
            <label htmlFor="description">Description *</label>
            <textarea
              id="description"
              name="description"
              value={formData.description}
              onChange={handleChange}
              required
              rows="5"
              placeholder="Describe the gig in detail..."
            />
          </div>

          <div className="form-group">
            <label htmlFor="skillsRequired">Required Skills (comma separated) *</label>
            <input
              type="text"
              id="skillsRequired"
              name="skillsRequired"
              value={formData.skillsRequired}
              onChange={handleChange}
              required
              placeholder="e.g., React, Node.js, MongoDB"
            />
          </div>

          <div className="form-row">
            <div className="form-group">
              <label htmlFor="budget">Budget ($) *</label>
              <input
                type="number"
                id="budget"
                name="budget"
                value={formData.budget}
                onChange={handleChange}
                required
                min="1"
                placeholder="500"
              />
            </div>

            <div className="form-group">
              <label htmlFor="duration">Duration *</label>
              <select
                id="duration"
                name="duration"
                value={formData.duration}
                onChange={handleChange}
                required
              >
                <option value="">Select Duration</option>
                <option value="1 week">1 week</option>
                <option value="2 weeks">2 weeks</option>
                <option value="1 month">1 month</option>
                <option value="2 months">2 months</option>
                <option value="3+ months">3+ months</option>
              </select>
            </div>
          </div>

          <div className="form-group">
            <label htmlFor="location">Location *</label>
            <input
```

```
              type="text"
              id="location"
              name="location"
              value={formData.location}
              onChange={handleChange}
              required
              placeholder="e.g., New York, NY"
            />
          </div>

          <button type="submit" disabled={loading} className="btn-primary">
            {loading ? 'Creating Gig...' : 'Create Gig'}
          </button>
        </form>
      </div>
    </div>
  );
}

export default CreateGig;
```

--- frontend\src\pages\Dashboard.jsx ---

```
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';
import { dashboardService } from '../services/dashboardService';
import { gigService } from '../services/gigService';
import '../styles/Dashboard.css';

function Dashboard() {
  const { user } = useAuth();
  const [stats, setStats] = useState({
    totalGigs: 0,
    activeGigs: 0,
    totalApplications: 0,
    pendingApplications: 0,
    earnings: 0,
    unreadMessages: 0
  });
  const [recentGigs, setRecentGigs] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchDashboardData();
  }, [user]);

  const fetchDashboardData = async () => {
  try {
    // Use real API calls with dashboardService
    const [statsData, recentGigsData] = await Promise.all([
      dashboardService.getStats(),        // Real stats from backend
      dashboardService.getRecentGigs(3)   // Real recent gigs
    ]);

    setStats(statsData);
    setRecentGigs(recentGigsData);
  } catch (error) {
    console.error('Failed to fetch dashboard data:', error);
  } finally {
    setLoading(false);
  }
};

  const getGreeting = () => {
    const hour = new Date().getHours();
    if (hour < 12) return 'Good morning';
    if (hour < 18) return 'Good afternoon';
    return 'Good evening';
  };

  if (loading) {
    return (
      <div className="dashboard-loading">
        <div className="loading-spinner"></div>
        <p>Loading your dashboard...</p>
      </div>
    );
  }

  return (
    <div className="dashboard">
      {/* Welcome Section */}
      <div className="dashboard-welcome">
        <div className="welcome-content">
          <h1>{getGreeting()}, {user?.firstName}! ■</h1>
          <p>
            {user?.userType === 'freelancer'
              ? 'Ready to find your next great opportunity?'
              : 'Ready to get your projects done by talented freelancers?'
            }
          </p>
        </div>
        <div className="welcome-actions">
          {user?.userType === 'client' ? (
            <Link to="/gigs/create" className="btn-primary">
              Post a New Gig
            </Link>
          ) : (
            <Link to="/gigs" className="btn-primary">
              Browse Available Gigs
            </Link>
          )}
        </div>
      </div>

      {/* Stats Overview */}
      <div className="stats-overview">
        <h2>Overview</h2>
        <div className="stats-grid">
          {user?.userType === 'freelancer' ? (
            <>
              <div className="stat-card">
                <div className="stat-icon applications">
                  <span>■</span>
                </div>
                <div className="stat-info">
                  <h3>{stats.totalApplications}</h3>
                  <p>Total Applications</p>
                </div>
                <div className="stat-trend">
                  <span className="trend-up">+2 this week</span>
                </div>
              </div>

              <div className="stat-card">
                <div className="stat-icon pending">
                  <span>■</span>
                </div>
                <div className="stat-info">
                  <h3>{stats.pendingApplications}</h3>
                  <p>Pending Reviews</p>
                </div>
                <div className="stat-trend">
                  <span className="trend-neutral">Waiting</span>
                </div>
              </div>

              <div className="stat-card">
                <div className="stat-card">
                  <div className="stat-icon earnings">
                    <span>■</span>
                  </div>
                  <div className="stat-info">
                    <h3>${stats.earnings}</h3>
                    <p>Total Earnings</p>
                  </div>
                  <div className="stat-trend">
                    <span className="trend-up">+15%</span>
                  </div>
                </div>
              </div>

              <div className="stat-card">
                <div className="stat-icon messages">
                  <span>■</span>
                </div>
                <div className="stat-info">
                  <h3>{stats.unreadMessages}</h3>
                  <p>Unread Messages</p>
                </div>
                <div className="stat-trend">
                  <span className="trend-up">New</span>
                </div>
              </div>
            </>
          ) : (
            <>
              <div className="stat-card">
                <div className="stat-icon gigs">
                  <span>■</span>
                </div>
                <div className="stat-info">
                  <h3>{stats.totalGigs}</h3>
                  <p>Total Gigs Posted</p>
                </div>
                <div className="stat-trend">
                  <span className="trend-up">+1 this month</span>
                </div>
              </div>

              <div className="stat-card">
                <div className="stat-icon active">
                  <span>■</span>
                </div>
                <div className="stat-info">
                  <h3>{stats.activeGigs}</h3>
                  <p>Active Gigs</p>
                </div>
                <div className="stat-trend">
                  <span className="trend-up">Looking good</span>
                </div>
              </div>

              <div className="stat-card">
                <div className="stat-icon applications">
                  <span>■</span>
                </div>
                <div className="stat-info">
                  <h3>{stats.totalApplications}</h3>
                  <p>Total Applications</p>
                </div>
                <div className="stat-trend">
                  <span className="trend-up">+5 today</span>
                </div>
              </div>

              <div className="stat-card">
                <div className="stat-icon messages">
                  <span>■</span>
                </div>
                <div className="stat-info">
                  <h3>{stats.unreadMessages}</h3>
                  <p>Unread Messages</p>
                </div>
                <div className="stat-trend">
                  <span className="trend-up">New</span>
                </div>
              </div>
            </>
          )}
        </div>
      </div>

      {/* Quick Actions & Recent Activity */}
```

```
        <div className="dashboard-content">
          {/* Quick Actions */}
          <div className="quick-actions-section">
            <h2>Quick Actions</h2>
            <div className="actions-grid">
              {user?.userType === 'freelancer' ? (
                <>
                  <Link to="/gigs" className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Find Gigs</h3>
                      <p>Browse available opportunities</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </Link>

                  <Link to="/profile" className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Update Profile</h3>
                      <p>Enhance your visibility</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </Link>

                  <Link to="/messages" className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Messages</h3>
                      <p>Connect with clients</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </Link>

                  <div className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Your Reviews</h3>
                      <p>Check your ratings</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </div>
                </>
              ) : (
                <>
                  <Link to="/gigs/create" className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Post a Gig</h3>
                      <p>Create new job posting</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </Link>

                  <Link to="/gigs" className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Find Freelancers</h3>
                      <p>Browse talented professionals</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </Link>

                  <Link to="/messages" className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Messages</h3>
                      <p>Manage conversations</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </Link>

                  <div className="action-card">
                    <div className="action-icon">■</div>
                    <div className="action-content">
                      <h3>Gig Analytics</h3>
                      <p>View performance metrics</p>
                    </div>
                    <div className="action-arrow">→</div>
                  </div>
                </>
              )}
            </div>
          </div>

          {/* Recent Gigs */}
          <div className="recent-activity-section">
            <div className="section-header">
              <h2>Recent Opportunities</h2>
              <Link to="/gigs" className="view-all-link">View All →</Link>
            </div>
            <div className="activity-list">
              {recentGigs.length > 0 ? (
                recentGigs.map(gig => (
                  <div key={gig._id} className="activity-item">
                    <div className="activity-icon">
                      {gig.client._id === user.id ? '■' : '■'}
                    </div>
                    <div className="activity-content">
                      <h4>{gig.title}</h4>
                      <p>
                        {gig.client._id === user.id
                          ? `You posted a new gig - ${gig.applications?.length || 0} applications`
                          : `New gig posted by ${gig.client.firstName} - $${gig.budget}`
                        }
                      </p>
                      <span className="activity-time">
                        {new Date(gig.createdAt).toLocaleDateString()}
                      </span>
                    </div>
                    <div className="activity-status">
                      <span className={`status-badge ${gig.status}`}>
                        {gig.status.replace('_', ' ')}
                      </span>
                    </div>
                  </div>
                ))
              ) : (
                <div className="no-activity">
                  <p>No recent activity found</p>
                  {user?.userType === 'client' ? (
                    <Link to="/gigs/create" className="btn-secondary">
                      Post Your First Gig
                    </Link>
                  ) : (
                    <Link to="/gigs" className="btn-secondary">
                      Browse Available Gigs
                    </Link>
                  )}
                </div>
              )}
            </div>
          </div>
        </div>

        {/* Performance Metrics */}
        <div className="performance-section">
          <h2>Your Performance</h2>
          <div className="performance-cards">
            <div className="performance-card">
              <div className="performance-metric">
                <span className="metric-value">{user?.rating || '0'}</span>
                <span className="metric-label">/ 5.0</span>
              </div>
              <div className="performance-info">
                <h3>Average Rating</h3>
                <p>Based on client feedback</p>
              </div>
              <div className="performance-stars">
                {'■'.repeat(Math.floor(user?.rating || 0))}
```

```
              {'■'.repeat(5 - Math.floor(user?.rating || 0))}
            </div>
          </div>

          <div className="performance-card">
            <div className="performance-metric">
              <span className="metric-value">
                {user?.userType === 'freelancer' ? stats.totalApplications : stats.totalGigs}
              </span>
              <span className="metric-label">Total</span>
            </div>
            <div className="performance-info">
              <h3>
                {user?.userType === 'freelancer' ? 'Applications' : 'Gigs Posted'}
              </h3>
              <p>All-time activity</p>
            </div>
            <div className="performance-progress">
              <div
                className="progress-bar"
                style={{ width: '75%' }}
              ></div>
            </div>
          </div>

          <div className="performance-card">
            <div className="performance-metric">
              <span className="metric-value">98%</span>
              <span className="metric-label">Rate</span>
            </div>
            <div className="performance-info">
              <h3>Response Rate</h3>
              <p>Messages responded to</p>
            </div>
            <div className="performance-trend trend-up">
              +2% from last month
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}

export default Dashboard;
```

--- frontend\src\pages\GigDetails.jsx ---

```
import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';
import { gigService } from '../services/gigService';
import '../styles/GigDetails.css';

function GigDetails() {
    const [gig, setGig] = useState(null);
    const [application, setApplication] = useState({
        proposal: '',
        bidAmount: '',
    });
    const [loading, setLoading] = useState(true);
    const [applying, setApplying] = useState(false);
    const [error, setError] = useState('');

    const { id } = useParams();
    const { user } = useAuth();
    const navigate = useNavigate();

    useEffect(() => {
        fetchGig();
    }, [id]);

    const fetchGig = async () => {
        try {
            const gigData = await gigService.getGig(id);
            setGig(gigData);
        } catch (err) {
            setError('Failed to load gig details');
        } finally {
            setLoading(false);
        }
    };

    const handleApply = async (e) => {
        e.preventDefault();
        setError('');
        setApplying(true);

        try {
            await gigService.applyToGig(id, application);
            alert('Application submitted successfully!');
            fetchGig(); // Refresh gig data
            setApplication({ proposal: '', bidAmount: '', });
        } catch (err) {
            setError(err.response?.data?.message || 'Failed to apply to gig');
        } finally {
            setApplying(false);
        }
    };

    const handleAcceptApplication = async (applicationId) => {
        if (window.confirm('Are you sure you want to accept this application?')) {
            try {
                await gigService.acceptApplication(id, applicationId);
                alert('Application accepted successfully!');
                fetchGig(); // Refresh gig data
            } catch (err) {
                setError(err.response?.data?.message || 'Failed to accept application');
            }
        }
    };

    const handleRejectApplication = async (applicationId) => {
        if (window.confirm('Are you sure you want to reject this application?')) {
            try {
                await gigService.rejectApplication(id, applicationId);
                alert('Application rejected successfully!');
                fetchGig(); // Refresh gig data
            } catch (err) {
                setError(err.response?.data?.message || 'Failed to reject application');
            }
        }
    };

    const hasApplied = gig?.applications?.some(
        app => app.freelancer._id === user?.id
    );

    if (loading) return <div className="loading">Loading gig details...</div>;
    if (!gig) return <div className="error-message">Gig not found</div>;

    return (
        <div className="gig-details">
            <div className="gig-details-header">
                <button onClick={() => navigate('/gigs')} className="back-btn">
                    ← Back to Gigs
                </button>

                <div className="gig-title-section">
                    <h1>{gig.title}</h1>
                    <div className="gig-meta">
                        <span className="budget">${gig.budget}</span>
                        <span className="location">{gig.location}</span>
                        <span className="duration">{gig.duration}</span>
                        <span className={`status ${gig.status}`}>{gig.status}</span>
                    </div>
                </div>
            </div>

            <div className="gig-content">
```

```
<div className="gig-info">
    <div className="info-card">
        <h3>Description</h3>
        <p>{gig.description}</p>
    </div>

    <div className="info-card">
        <h3>Required Skills</h3>
        <div className="skills-list">
            {gig.skillsRequired.map((skill, index) => (
                <span key={index} className="skill-tag">{skill}</span>
            ))}
        </div>
    </div>

    <div className="info-card">
        <h3>Client Information</h3>
        <div className="client-info">
            <p><strong>Name:</strong> {gig.client.firstName} {gig.client.lastName}</p>
            <p><strong>Rating:</strong> {gig.client.rating || 'No ratings yet'}</p>
        </div>
    </div>

    <div className="info-card">
        <h3>Applications ({gig.applications?.length || 0})</h3>
        {gig.applications?.map((app, index) => (
            <div key={app._id} className="application-item">
                <div className="application-header">
                    <strong>{app.freelancer.firstName} {app.freelancer.lastName}</strong>
                    <span className={`application-status ${app.status}`}>
                        {app.status}
                    </span>
                </div>
                <p><strong>Bid:</strong> ${app.bidAmount}</p>
                <p className="proposal">{app.proposal}</p>
                <div className="application-meta">
                    <span>Applied: {new Date(app.appliedAt).toLocaleDateString()}</span>
                    {user?.userType === 'client' && user?.id === gig.client._id && gig.status
                        <div className="application-actions">
                            <button
                                onClick={() => handleAcceptApplication(app._id)}
                                className="btn-primary btn-small"
                            >
                                Accept
                            </button>
                            <button
                                onClick={() => handleRejectApplication(app._id)}
                                className="btn-secondary btn-small"
                            >
                                Reject
                            </button>
                        </div>
                    )}
                </div>
            </div>
        ))}
    </div>
</div>

{user?.userType === 'freelancer' && gig.status === 'open' && (
    <div className="apply-section">
        <h3>Apply for this Gig</h3>
        {hasApplied ? (
            <div className="applied-message">
                ✓ You have already applied to this gig
            </div>
        ) : (
            <form onSubmit={handleApply} className="apply-form">
                {error && <div className="error-message">{error}</div>}

                <div className="form-group">
                    <label>Your Proposal *</label>
                    <textarea
                        value={application.proposal}
                        onChange={(e) => setApplication({
```

```
                                            ...application,
                                            proposal: e.target.value
                                        })}
                                        required
                                        rows="4"
                                        placeholder="Describe why you're the best fit for this gig..."
                                    />
                                </div>
                                <div className="form-group">
                                    <label>Your Bid Amount ($) *</label>
                                    <input
                                        type="number"
                                        value={application.bidAmount}
                                        onChange={(e) => setApplication({
                                            ...application,
                                            bidAmount: e.target.value
                                        })}
                                        required
                                        min="1"
                                        placeholder="Enter your bid"
                                    />
                                </div>
                                <button type="submit" disabled={applying} className="btn-primary">
                                    {applying ? 'Submitting...' : 'Submit Application'}
                                </button>
                            </form>
                        )}
                    </div>
                )}
            </div>
        </div>
    );
}

export default GigDetails;
```

--- frontend\src\pages\GigFeed.jsx ---

```
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import { gigService } from '../services/gigService';
import '../styles/GigFeed.css';

function GigFeed() {
  const [gigs, setGigs] = useState([]);
  const [loading, setLoading] = useState(true);
  const [filters, setFilters] = useState({
    search: '',
    skills: '',
    location: '',
    minPrice: '',
    maxPrice: ''
  });

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

  const fetchGigs = async (filterParams = {}) => {
    try {
      const gigsData = await gigService.getGigs(filterParams);
      setGigs(gigsData);
    } catch (error) {
      console.error('Failed to fetch gigs:', error);
    } finally {
      setLoading(false);
    }
  };

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

```
};

const handleSearch = (e) => {
  e.preventDefault();
  fetchGigs(filters);
};

const clearFilters = () => {
  setFilters({
    search: '',
    skills: '',
    location: '',
    minPrice: '',
    maxPrice: ''
  });
  fetchGigs();
};

if (loading) {
  return <div className="loading">Loading gigs...</div>;
}

return (
  <div className="gig-feed">
    <div className="gig-feed-header">
      <h1>Find Local Gigs</h1>
      <p>Discover opportunities in your area</p>
    </div>

    {/* Search and Filters */}
    <div className="filters-section">
      <form onSubmit={handleSearch} className="filters-form">
        <div className="filter-row">
          <input
            type="text"
            name="search"
            placeholder="Search gigs..."
            value={filters.search}
            onChange={handleFilterChange}
            className="search-input"
          />

          <input
            type="text"
            name="skills"
            placeholder="Filter by skills..."
            value={filters.skills}
            onChange={handleFilterChange}
            className="filter-input"
          />

          <input
            type="text"
            name="location"
            placeholder="Filter by location..."
            value={filters.location}
            onChange={handleFilterChange}
            className="filter-input"
          />
        </div>

        <div className="filter-row">
          <input
            type="number"
            name="minPrice"
            placeholder="Min budget"
            value={filters.minPrice}
            onChange={handleFilterChange}
            className="price-input"
          />

          <input
            type="number"
            name="maxPrice"
            placeholder="Max budget"
```

```
              value={filters.maxPrice}
              onChange={handleFilterChange}
              className="price-input"
            />

            <button type="submit" className="btn-primary">Search</button>
            <button type="button" onClick={clearFilters} className="btn-secondary">
              Clear
            </button>
          </div>
        </form>
      </div>

      {/* Gigs Grid */}
      <div className="gigs-grid">
        {gigs.length === 0 ? (
          <div className="no-gigs">
            <h3>No gigs found</h3>
            <p>Try adjusting your search filters or check back later</p>
            <Link to="/" className="btn-primary">Back to Dashboard</Link>
          </div>
        ) : (
          gigs.map(gig => (
            <div key={gig._id} className="gig-card">
              <div className="gig-card-header">
                <h3>{gig.title}</h3>
                <span className="budget">${gig.budget}</span>
              </div>

              <p className="gig-description">
                {gig.description.length > 100
                  ? `${gig.description.substring(0, 100)}...`
                  : gig.description
                }
              </p>

              <div className="gig-meta">
                <span className="location">■ {gig.location}</span>
                <span className="duration">■■ {gig.duration}</span>
              </div>

              <div className="skills-list">
                {gig.skillsRequired.slice(0, 3).map((skill, index) => (
                  <span key={index} className="skill-tag">{skill}</span>
                ))}
                {gig.skillsRequired.length > 3 && (
                  <span className="skill-tag">+{gig.skillsRequired.length - 3} more</span>
                )}
              </div>

              <div className="gig-card-footer">
                <span className="client">
                  Posted by {gig.client.firstName} {gig.client.lastName}
                </span>
                <Link to={`/gigs/${gig._id}`} className="view-details-btn">
                  View Details
                </Link>
              </div>
            </div>
          ))
        )}
      </div>
    </div>
  );
}

export default GigFeed;
```

--- frontend\src\pages\Messages.jsx ---

```
import React, { useState, useEffect, useRef } from 'react';
import { useAuth } from '../contexts/AuthContext';
import { messageService } from '../services/messageService';
import io from 'socket.io-client';
import '../styles/Messages.css';
```

```
function Messages() {
    const [conversations, setConversations] = useState([]);
    const [contacts, setContacts] = useState([]);
    const [activeConversation, setActiveConversation] = useState(null);
    const [messages, setMessages] = useState([]);
    const [newMessage, setNewMessage] = useState('');
    const [searchQuery, setSearchQuery] = useState('');
    const [activeTab, setActiveTab] = useState('conversations');
    const [loading, setLoading] = useState(true);
    const [socket, setSocket] = useState(null);
    const messagesEndRef = useRef(null);

    const { user } = useAuth();

    useEffect(() => {
        // Initialize socket connection
        const newSocket = io('http://localhost:5000');
        setSocket(newSocket);

        loadInitialData();

        return () => newSocket.close();
    }, []);

    useEffect(() => {
        if (socket && activeConversation) {
            socket.emit('join_conversation', activeConversation.id);

            socket.on('receive_message', (data) => {
                setMessages(prev => [...prev, data]);
            });
        }

        return () => {
            if (socket) {
                socket.off('receive_message');
            }
        };
    }, [socket, activeConversation]);

    useEffect(() => {
        scrollToBottom();
    }, [messages]);

    const loadInitialData = async () => {
        try {
            const [conversationsData, contactsData] = await Promise.all([
                messageService.getConversations(),
                messageService.getContacts()
            ]);

            setConversations(conversationsData);
            setContacts(contactsData);
        } catch (error) {
            console.error('Failed to load initial data:', error);
        } finally {
            setLoading(false);
        }
    };

    const searchContacts = async (query) => {
        if (!query.trim()) {
            const contactsData = await messageService.getContacts();
            setContacts(contactsData);
            return;
        }

        try {
            const searchResults = await messageService.searchContacts(query);
            setContacts(searchResults);
        } catch (error) {
            console.error('Search failed:', error);
        }
    };
```

```
   const startNewConversation = async (contact) => {
    try {
        console.log('Starting conversation with:', contact._id, contact.firstName);

        // Use the NEW createConversation method
        const conversation = await messageService.createConversation(contact._id);
        console.log('Conversation created:', conversation);

        setConversations(prev => {
            const exists = prev.find(conv => conv.id === conversation.id);
            if (!exists) {
                return [conversation, ...prev];
            }
            return prev;
        });

        setActiveConversation(conversation);
        setActiveTab('conversations');

        // Load messages for the new conversation
        const messagesData = await messageService.getMessages(conversation.id);
        setMessages(messagesData);
    } catch (error) {
        console.error('Failed to start conversation:', error);
        console.error('Error details:', error.response?.data);
        alert('Failed to start conversation: ' + (error.response?.data?.message || error.message));
    }
};

    const loadConversation = async (conversation) => {
        setActiveConversation(conversation);
        try {
            const messagesData = await messageService.getMessages(conversation.id);
            setMessages(messagesData);
        } catch (error) {
            console.error('Failed to load messages:', error);
        }
    };

    const handleSendMessage = async (e) => {
        e.preventDefault();
        if (!newMessage.trim() || !activeConversation) return;

        const messageData = {
            conversationId: activeConversation.id,
            receiverId: activeConversation.participant._id,
            content: newMessage.trim()
        };

        try {
            if (socket) {
                socket.emit('send_message', {
                    ...messageData,
                    senderId: user.id
                });
            }

            // Add message locally immediately
            const tempMessage = {
                _id: Date.now().toString(),
                senderId: { _id: user.id, firstName: user.firstName, lastName: user.lastName },
                content: newMessage.trim(),
                createdAt: new Date().toISOString(),
                read: false
            };

            setMessages(prev => [...prev, tempMessage]);
            setNewMessage('');

            // Also save to database
            await messageService.sendMessage(messageData);

        } catch (error) {
            console.error('Failed to send message:', error);
            alert('Failed to send message. Please try again.');
        }
    };

    const scrollToBottom = () => {
        messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
    };

    const filteredContacts = contacts.filter(contact =>
        `${contact.firstName} ${contact.lastName}`.toLowerCase().includes(searchQuery.toLowerCase()) ||
        contact.skills?.some(skill => skill.toLowerCase().includes(searchQuery.toLowerCase()))
    );

    if (loading) {
        return (
            <div className="messages-loading">
                <div className="loading-spinner"></div>
                <p>Loading messages...</p>
            </div>
        );
    }

    return (
        <div className="messages">
            <div className="messages-header">
                <h1>Messages</h1>
                <p>Communicate with {user?.userType === 'client' ? 'freelancers' : 'clients'}</p>
            </div>

            <div className="messages-container">
                {/* Sidebar */}
                <div className="messages-sidebar">
                    <div className="sidebar-tabs">
                        <button
                            className={`tab-button ${activeTab === 'conversations' ? 'active' : ''}`}
                            onClick={() => setActiveTab('conversations')}
                        >
                            Conversations
                        </button>
                        <button
                            className={`tab-button ${activeTab === 'contacts' ? 'active' : ''}`}
                            onClick={() => setActiveTab('contacts')}
                        >
                            {user?.userType === 'client' ? 'Freelancers' : 'Clients'} ({contacts.length})
                        </button>
                    </div>

                    {activeTab === 'conversations' ? (
                        <div className="conversations-list">
                            {conversations.length === 0 ? (
                                <div className="empty-state">
                                    <p>No conversations yet</p>
                                    <small>Start a conversation from the Contacts tab</small>
                                </div>
                            ) : (
                                conversations.map(conversation => (
                                    <div
                                        key={conversation.id}
                                        className={`conversation-item ${
                                            activeConversation?.id === conversation.id ? 'active' : ''
                                        }`}
                                        onClick={() => loadConversation(conversation)}
                                    >
                                        <div className="conversation-avatar">
                                            {conversation.participant.name.charAt(0)}
                                        </div>
                                        <div className="conversation-info">
                                            <div className="conversation-header">
                                                <span className="participant-name">
                                                    {conversation.participant.name}
                                                </span>
                                                <span className="message-time">
                                                    {new Date(conversation.timestamp).toLocaleDateString()}
                                                </span>
                                            </div>
                                            <p className="last-message">{conversation.lastMessage}</p>
                                            <span className="user-type-badge">
                                                {conversation.participant.type}
                                            </span>
                                        </div>
                                    </div>
                                ))
                            )}
                        </div>
                    ) : (
                        <div className="contacts-list">
                            <div className="contacts-search">
                                <input
                                    type="text"
                                    placeholder={`Search ${user?.userType === 'client' ? 'freelancers' : 'clie
                                    value={searchQuery}
                                    onChange={(e) => {
                                        setSearchQuery(e.target.value);
                                        searchContacts(e.target.value);
                                    }}
                                    className="search-input"
                                />
                            </div>
                            {filteredContacts.length === 0 ? (
                                <div className="empty-state">
                                    <p>No {user?.userType === 'client' ? 'freelancers' : 'clients'} found</p>
                                </div>
                            ) : (
                                filteredContacts.map(contact => (
                                    <div
                                        key={contact._id}
                                        className="contact-item"
                                        onClick={() => startNewConversation(contact)}
                                    >
                                        <div className="contact-avatar">
                                            {contact.firstName.charAt(0)}
                                        </div>
                                        <div className="contact-info">
                                            <div className="contact-header">
                                                <span className="contact-name">
                                                    {contact.firstName} {contact.lastName}
                                                </span>
                                                <span className="contact-rating">
                                                    {contact.rating || 'New'}
                                                </span>
                                            </div>
                                            <p className="contact-skills">
                                                {contact.skills?.slice(0, 3).join(', ')}
                                                {contact.skills?.length > 3 && '...'}
                                            </p>
                                            <span className="user-type-badge">
                                                {contact.userType}
                                            </span>
                                        </div>
                                        <div className="contact-action">
                                            <button className="start-chat-btn">■</button>
                                        </div>
                                    </div>
                                ))
                            )}
                        </div>
                    )}
                </div>

                {/* Chat Area */}
                <div className="chat-area">
                    {activeConversation ? (
                        <>
                            <div className="chat-header">
                                <div className="chat-partner">
                                    <div className="partner-avatar">
                                        {activeConversation.participant.name.charAt(0)}
                                    </div>
                                    <div>
                                        <h4>{activeConversation.participant.name}</h4>
```

```
                                        <span className="partner-type">
                                            {activeConversation.participant.type}
                                        </span>
                                    </div>
                                </div>
                            </div>

                            <div className="messages-list">
                                {messages.map(message => (
                                    <div
                                        key={message._id}
                                        className={`message ${
                                            message.senderId._id === user.id ? 'sent' : 'received'
                                        }`}
                                    >
                                        <div className="message-content">
                                            <p>{message.content}</p>
                                            <span className="message-time">
                                                {new Date(message.createdAt).toLocaleTimeString()}
                                            </span>
                                        </div>
                                    </div>
                                ))}
                                <div ref={messagesEndRef} />
                            </div>

                            <form onSubmit={handleSendMessage} className="message-input-form">
                                <input
                                    type="text"
                                    value={newMessage}
                                    onChange={(e) => setNewMessage(e.target.value)}
                                    placeholder="Type your message..."
                                    className="message-input"
                                />
                                <button type="submit" className="send-button">
                                    Send
                                </button>
                            </form>
                        </>
                    ) : (
                        <div className="no-conversation">
                            <h3>Select a conversation to start messaging</h3>
                            <p>
                                {activeTab === 'conversations'
                                    ? 'Choose from your existing conversations'
                                    : 'Or browse contacts to start a new conversation'
                                }
                            </p>
                        </div>
                    )}
                </div>
            </div>
        </div>
    );
}

export default Messages;
```

--- frontend\src\pages\Profile.jsx ---

```
import React, { useState, useEffect } from 'react';
import { useAuth } from '../contexts/AuthContext';
import { authService } from '../services/authService';
import '../styles/Profile.css';

function Profile() {
  const { user, logout } = useAuth();
  const [profile, setProfile] = useState(null);
  const [loading, setLoading] = useState(true);
  const [editing, setEditing] = useState(false);
  const [formData, setFormData] = useState({
    bio: '',
    hourlyRate: '',
    skills: '',
    location: ''
  });

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

  const fetchProfile = async () => {
    try {
      const profileData = await authService.getProfile();
      setProfile(profileData);
      setFormData({
        bio: profileData.bio || '',
        hourlyRate: profileData.hourlyRate || '',
        skills: profileData.skills?.join(', ') || '',
        location: profileData.location || ''
      });
    } catch (error) {
      console.error('Failed to fetch profile:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleSave = async () => {
    try {
      // In a real app, you would have an update profile endpoint
      const updatedData = {
        ...formData,
        skills: formData.skills.split(',').map(skill => skill.trim())
      };
      console.log('Profile updated:', updatedData);

      setEditing(false);
      fetchProfile(); // Refresh profile data
      alert('Profile updated successfully!');
    } catch (error) {
      console.error('Failed to update profile:', error);
      alert('Failed to update profile');
    }
  };

  const handleCancel = () => {
    setFormData({
      bio: profile.bio || '',
      hourlyRate: profile.hourlyRate || '',
      skills: profile.skills?.join(', ') || '',
      location: profile.location || ''
    });
    setEditing(false);
  };

  if (loading) return <div className="loading">Loading profile...</div>;
  if (!profile) return <div className="error-message">Failed to load profile</div>;

  return (
    <div className="profile">
      <div className="profile-header">
        <h1>Your Profile</h1>
        <p>Manage your account information and preferences</p>
      </div>

      <div className="profile-content">
        <div className="profile-card">
          <div className="card-header">
            <h2>Personal Information</h2>
            {!editing && (
              <button
                onClick={() => setEditing(true)}
                className="edit-btn"
              >
                Edit Profile
              </button>
            )}
          </div>
```

```
<div className="profile-info">
  <div className="info-item">
    <label>Name:</label>
    <span>{profile.firstName} {profile.lastName}</span>
  </div>

  <div className="info-item">
    <label>Email:</label>
    <span>{profile.email}</span>
  </div>

  <div className="info-item">
    <label>Account Type:</label>
    <span className={`account-type ${profile.userType}`}>
      {profile.userType}
    </span>
  </div>

  <div className="info-item">
    <label>Rating:</label>
    <span className="rating">
      {profile.rating > 0 ? `■ ${profile.rating}/5` : 'No ratings yet'}
    </span>
  </div>

  {profile.userType === 'freelancer' && (
    <>
      <div className="info-item">
        <label>Hourly Rate:</label>
        {editing ? (
          <input
            type="number"
            value={formData.hourlyRate}
            onChange={(e) => setFormData({
              ...formData,
              hourlyRate: e.target.value
            })}
            placeholder="Enter your hourly rate"
            className="edit-input"
          />
        ) : (
          <span>{profile.hourlyRate ? `$${profile.hourlyRate}/hr` : 'Not set'}</span>
        )}
      </div>

      <div className="info-item">
        <label>Skills:</label>
        {editing ? (
          <input
            type="text"
            value={formData.skills}
            onChange={(e) => setFormData({
              ...formData,
              skills: e.target.value
            })}
            placeholder="e.g., React, Node.js, Design"
            className="edit-input"
          />
        ) : (
          <div className="skills-display">
            {profile.skills?.map((skill, index) => (
              <span key={index} className="skill-tag">{skill}</span>
            ))}
            {(!profile.skills || profile.skills.length === 0) && 'No skills added'}
          </div>
        )}
      </div>
    </>
  )}

  <div className="info-item">
    <label>Location:</label>
    {editing ? (
      <input
        type="text"
```

```
            value={formData.location}
            onChange={(e) => setFormData({
              ...formData,
              location: e.target.value
            })}
            placeholder="e.g., New York, NY"
            className="edit-input"
          />
        ) : (
          <span>{profile.location}</span>
        )}
      </div>

      <div className="info-item bio-item">
        <label>Bio:</label>
        {editing ? (
          <textarea
            value={formData.bio}
            onChange={(e) => setFormData({
              ...formData,
              bio: e.target.value
            })}
            placeholder="Tell us about yourself..."
            className="edit-textarea"
            rows="4"
          />
        ) : (
          <span>{profile.bio || 'No bio added yet'}</span>
        )}
      </div>
    </div>

    {editing && (
      <div className="edit-actions">
        <button onClick={handleSave} className="btn-primary">
          Save Changes
        </button>
        <button onClick={handleCancel} className="btn-secondary">
          Cancel
        </button>
      </div>
    )}
  </div>

  {/* Reviews Section */}
  {profile.reviews && profile.reviews.length > 0 && (
    <div className="profile-card">
      <h2>Reviews ({profile.reviews.length})</h2>
      <div className="reviews-list">
        {profile.reviews.map((review, index) => (
          <div key={index} className="review-item">
            <div className="review-header">
              <span className="reviewer">
                {review.user?.firstName || 'Anonymous'}
              </span>
              <span className="review-rating">
                {'■'.repeat(review.rating)}
              </span>
            </div>
            <p className="review-comment">{review.comment}</p>
            <span className="review-date">
              {new Date(review.createdAt).toLocaleDateString()}
            </span>
          </div>
        ))}
      </div>
    </div>
  )}

  <div className="profile-actions">
    <button onClick={logout} className="logout-btn">
      Logout
    </button>
  </div>
</div>
```

```
    </div>
  );
}

export default Profile;
```

--- frontend\src\pages\auth\Login.jsx ---

```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../../contexts/AuthContext';
import '../../styles/Auth.css';

function Login() {
  const [formData, setFormData] = useState({
    email: '',
    password: ''
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const { login } = useAuth();
  const navigate = useNavigate();

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

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

    try {
      await login(formData.email, formData.password);
      navigate('/');
    } catch (err) {
      setError(err.response?.data?.message || 'Login failed. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="auth-container">
      <div className="auth-card">
        <h2>Login to GigConnect</h2>
        {error && <div className="error-message">{error}</div>}

        <form onSubmit={handleSubmit} className="auth-form">
          <div className="form-group">
            <label htmlFor="email">Email</label>
            <input
              type="email"
              id="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              required
            />
          </div>

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

          <button
            type="submit"
            disabled={loading}
            className="auth-button"
          >
            {loading ? 'Logging in...' : 'Login'}
          </button>
        </form>

        <p className="auth-link">
          Don't have an account? <Link to="/register">Sign up</Link>
        </p>
      </div>
    </div>
  );
}

export default Login;
```

--- frontend\src\pages\auth\Register.jsx ---

```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../../contexts/AuthContext';
import '../../styles/Auth.css';

function Register() {
  const [formData, setFormData] = useState({
    firstName: '',
    lastName: '',
    email: '',
    password: '',
    confirmPassword: '',
    userType: 'freelancer',
    skills: '',
    location: ''
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const { register } = useAuth();
  const navigate = useNavigate();

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

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

    if (formData.password !== formData.confirmPassword) {
      setError('Passwords do not match');
      return;
    }

    if (formData.password.length < 6) {
      setError('Password must be at least 6 characters long');
      return;
    }

    setLoading(true);

    try {
      await register(formData);
      navigate('/');
    } catch (err) {
      setError(err.response?.data?.message || 'Registration failed. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="auth-container">
      <div className="auth-card">
        <h2>Join GigConnect</h2>
        {error && <div className="error-message">{error}</div>}

        <form onSubmit={handleSubmit} className="auth-form">
          <div className="form-row">
            <div className="form-group">
              <label htmlFor="firstName">First Name</label>
              <input
                type="text"
                id="firstName"
                name="firstName"
                value={formData.firstName}
                onChange={handleChange}
                required
              />
            </div>
            <div className="form-group">
              <label htmlFor="lastName">Last Name</label>
              <input
                type="text"
                id="lastName"
                name="lastName"
                value={formData.lastName}
                onChange={handleChange}
                required
              />
            </div>
          </div>

          <div className="form-group">
            <label htmlFor="email">Email</label>
            <input
              type="email"
              id="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              required
            />
          </div>

          <div className="form-row">
            <div className="form-group">
              <label htmlFor="password">Password</label>
              <input
                type="password"
                id="password"
                name="password"
                value={formData.password}
                onChange={handleChange}
                required
                minLength="6"
              />
            </div>
            <div className="form-group">
              <label htmlFor="confirmPassword">Confirm Password</label>
              <input
                type="password"
                id="confirmPassword"
                name="confirmPassword"
                value={formData.confirmPassword}
                onChange={handleChange}
                required
              />
            </div>
          </div>

          <div className="form-group">
            <label htmlFor="userType">I want to join as a:</label>
            <select
              id="userType"
              name="userType"
              value={formData.userType}
              onChange={handleChange}
            >
              <option value="freelancer">Freelancer</option>
              <option value="client">Client</option>
            </select>
          </div>

          {formData.userType === 'freelancer' && (
            <div className="form-group">
              <label htmlFor="skills">Skills (comma separated)</label>
              <input
                type="text"
                id="skills"
                name="skills"
                value={formData.skills}
                onChange={handleChange}
                placeholder="e.g., Web Development, Graphic Design, Writing"
              />
            </div>
          )}

          <div className="form-group">
            <label htmlFor="location">Location</label>
            <input
              type="text"
              id="location"
              name="location"
              value={formData.location}
              onChange={handleChange}
              required
              placeholder="e.g., New York, NY"
            />
          </div>

          <button
            type="submit"
            disabled={loading}
            className="auth-button"
          >
            {loading ? 'Creating Account...' : 'Create Account'}
          </button>
        </form>

        <p className="auth-link">
          Already have an account? <Link to="/login">Login</Link>
        </p>
      </div>
    </div>
  );
}

export default Register;
```

--- frontend\src\services\api.js ---

```
import axios from 'axios';

const API_URL = import.meta.env.VITE_API_URL;

const api = axios.create({
  baseURL: API_URL,
});

api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export default api;
```

--- frontend\src\services\authService.js ---

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

export const authService = {
  async login(email, password) {
    const response = await api.post('/auth/login', { email, password });
    return response.data;
  },

  async register(userData) {
    const response = await api.post('/auth/register', userData);
    return response.data;
  },

  async getProfile() {
    const response = await api.get('/auth/profile');
    return response.data;
  }
};
```

--- frontend\src\services\dashboardService.js ---

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

export const dashboardService = {
  /**
   * Get dashboard statistics for the current user
   * Returns role-specific stats (client/freelancer)
   */
  async getStats() {
    try {
      const response = await api.get('/users/dashboard/stats');
      return response.data;
    } catch (error) {
      console.error('Error fetching dashboard stats:', error);
      throw new Error(
        error.response?.data?.message ||
        'Failed to load dashboard statistics. Please try again.'
      );
    }
  },

  /**
   * Get recent gigs for the dashboard
   * Shows latest opportunities or posted gigs
   */
  async getRecentGigs(limit = 3) {
    try {
      const response = await api.get(`/gigs?limit=${limit}`);
      return response.data;
    } catch (error) {
      console.error('Error fetching recent gigs:', error);
      throw new Error(
        error.response?.data?.message ||
        'Failed to load recent gigs. Please try again.'
      );
    }
  },

  /**
   * Get user's performance metrics
   * Includes rating, response rate, completion rate, etc.
   */
  async getPerformanceMetrics() {
    try {
      // This could be expanded with a dedicated performance endpoint
      // For now, we'll calculate from existing data
      const userResponse = await api.get('/auth/profile');
      const user = userResponse.data;

      const stats = await this.getStats();

      return {
        rating: user.rating || 0,
        totalProjects: user.userType === 'freelancer' ?
          stats.activeGigs : stats.totalGigs,
        responseRate: 98, // Mock data - could be calculated from message responses
        completionRate: 95 // Mock data - could be calculated from completed gigs
      };
    } catch (error) {
      console.error('Error fetching performance metrics:', error);
      throw new Error(
        error.response?.data?.message ||
        'Failed to load performance metrics. Please try again.'
      );
    }
  },

  /**
   * Get quick overview data for dashboard widgets
   * Combines multiple data points for efficient loading
   */
  async getDashboardOverview() {
    try {
      const [stats, recentGigs] = await Promise.all([
        this.getStats(),
        this.getRecentGigs(5)
      ]);

      return {
        stats,
        recentGigs,
        lastUpdated: new Date().toISOString()
      };
    } catch (error) {
      console.error('Error fetching dashboard overview:', error);
      throw new Error(
        error.response?.data?.message ||
        'Failed to load dashboard overview. Please try again.'
      );
    }
  },

  /**
   * Get activity feed for the dashboard
   * Shows recent applications, messages, gig updates, etc.
   */
  async getActivityFeed(limit = 10) {
    try {
      // For now, we'll use recent gigs as activity
      // In a real app, you'd have a dedicated activity endpoint
      const recentGigs = await this.getRecentGigs(limit);

      // Transform gigs into activity items
      const activities = recentGigs.map(gig => ({
        id: gig._id,
        type: 'gig_posted',
        title: gig.title,
        description: `New gig posted by ${gig.client.firstName}`,
        timestamp: gig.createdAt,
        metadata: {
          budget: gig.budget,
          location: gig.location,
          status: gig.status
        }
      }));

      return activities;
    } catch (error) {
      console.error('Error fetching activity feed:', error);
      throw new Error(
        error.response?.data?.message ||
        'Failed to load activity feed. Please try again.'
      );
    }
  },

  /**
   * Get earnings summary for freelancers
   * Shows current month earnings, pending payments, etc.
   */
  async getEarningsSummary() {
    try {
      const stats = await this.getStats();

      return {
        totalEarnings: stats.earnings || 0,
        pendingEarnings: stats.earnings * 0.2, // Mock data - 20% pending
        thisMonth: stats.earnings * 0.3, // Mock data - 30% from current month
        lastMonth: stats.earnings * 0.7 // Mock data - 70% from previous month
      };
    } catch (error) {
      console.error('Error fetching earnings summary:', error);
      throw new Error(
        error.response?.data?.message ||
        'Failed to load earnings summary. Please try again.'
      );
    }
  },

  /**
   * Get notifications for the dashboard
   * Alerts, messages, application updates, etc.
   */
  async getNotifications() {
    try {
      // Mock notifications - in real app, fetch from notifications endpoint
      return [
        {
          id: 1,
          type: 'message',
          title: 'New Message',
          description: 'You have a new message from John Client',
          timestamp: new Date(Date.now() - 1000 * 60 * 30).toISOString(), // 30 mins ago
          read: false,
          action: '/messages'
        },
        {
          id: 2,
          type: 'application',
          title: 'Application Update',
          description: 'Your application for "Website Development" was viewed',
          timestamp: new Date(Date.now() - 1000 * 60 * 60 * 2).toISOString(), // 2 hours ago
          read: true,
          action: '/gigs'
        },
        {
          id: 3,
          type: 'system',
          title: 'Welcome to GigConnect!',
          description: 'Complete your profile to get more opportunities',
          timestamp: new Date(Date.now() - 1000 * 60 * 60 * 24).toISOString(), // 1 day ago
          read: true,
          action: '/profile'
        }
      ];
    } catch (error) {
      console.error('Error fetching notifications:', error);
      throw new Error(
        error.response?.data?.message ||
        'Failed to load notifications. Please try again.'
      );
    }
  }
};

export default dashboardService;
```

--- frontend\src\services\gigService.js ---

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

export const gigService = {
    /**
     * Get all gigs with optional filters
     */
    async getGigs(filters = {}) {
        try {
            const params = new URLSearchParams();
            if (filters.search) params.append('search', filters.search);
            if (filters.skills) params.append('skills', filters.skills);
            if (filters.location) params.append('location', filters.location);
            if (filters.minPrice) params.append('minPrice', filters.minPrice);
            if (filters.maxPrice) params.append('maxPrice', filters.maxPrice);
            if (filters.limit) params.append('limit', filters.limit);
            if (filters.status) params.append('status', filters.status);

            const response = await api.get(`/gigs?${params}`);
            return response.data;
        } catch (error) {
            console.error('Error fetching gigs:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load gigs. Please try again.'
            );
        }
    },

    /**
     * Get single gig by ID
     */
    async getGig(id) {
        try {
            const response = await api.get(`/gigs/${id}`);
            return response.data;
        } catch (error) {
            console.error('Error fetching gig:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load gig details. Please try again.'
            );
        }
    },

    /**
     * Create new gig
     */
    async createGig(gigData) {
        try {
            const response = await api.post('/gigs', gigData);
            return response.data;
        } catch (error) {
            console.error('Error creating gig:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to create gig. Please try again.'
            );
        }
    },
    /**
     * Apply to a gig
     */
    async applyToGig(gigId, applicationData) {
        try {
            const response = await api.post(`/gigs/${gigId}/apply`, applicationData);
            return response.data;
        } catch (error) {
            console.error('Error applying to gig:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to submit application. Please try again.'
            );
        }
    },

    /**
     * Accept application
     */
    async acceptApplication(gigId, applicationId) {
        try {
            const response = await api.put(`/gigs/${gigId}/accept-application`, { applicationId });
            return response.data;
        } catch (error) {
            console.error('Error accepting application:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to accept application. Please try again.'
            );
        }
    },

    /**
     * Reject application
     */
    async rejectApplication(gigId, applicationId) {
        try {
            const response = await api.put(`/gigs/${gigId}/reject-application`, { applicationId });
            return response.data;
        } catch (error) {
            console.error('Error rejecting application:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to reject application. Please try again.'
            );
        }
    },

    /**
     * Get freelancer's applications
     */
    async getUserApplications() {
        try {
            const response = await api.get('/gigs/user/applications');
            return response.data;
        } catch (error) {
            console.error('Error fetching applications:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load your applications. Please try again.'
            );
        }
    },

    /**
     * Get user's posted gigs (for clients)
     */
    async getMyGigs() {
        try {
            const allGigs = await this.getGigs();
            // Filter gigs posted by current user
            // This would be more efficient with a dedicated endpoint
            return allGigs.filter(gig => gig.client._id === 'current-user-id');
        } catch (error) {
            console.error('Error fetching my gigs:', error);
```

```
            throw new Error(
                error.response?.data?.message ||
                'Failed to load your gigs. Please try again.'
            );
        }
    }
};

export default gigService;
```

--- frontend\src\services\messageService.js ---

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

export const messageService = {
    // Get all contacts for messaging
    async getContacts() {
        try {
            const response = await api.get('/users/contacts/list');
            return response.data;
        } catch (error) {
            console.error('Error fetching contacts:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load contacts. Please try again.'
            );
        }
    },

    // Search contacts
    async searchContacts(query) {
        try {
            const response = await api.get(`/users/contacts/search?query=${encodeURIComponent(query)}`);
            return response.data;
        } catch (error) {
            console.error('Error searching contacts:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to search contacts. Please try again.'
            );
        }
    },

    // Create new conversation with a user
    async createConversation(userId) {
        try {
            const response = await api.post(`/messages/conversation/${userId}`);
            return response.data;
        } catch (error) {
            console.error('Error creating conversation:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to start conversation. Please try again.'
            );
        }
    },

    // Get existing conversation by ID
    async getConversation(conversationId) {
        try {
            const response = await api.get(`/messages/conversation/${conversationId}`);
            return response.data;
        } catch (error) {
            console.error('Error getting conversation:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load conversation. Please try again.'
            );
        }
    },

    // Get all conversations between two users
    async getUserConversations(userId) {
        try {
            const response = await api.get(`/messages/user/${userId}/conversations`);
            return response.data;
        } catch (error) {
            console.error('Error getting user conversations:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load conversation history. Please try again.'
            );
        }
    },

    // Get user's conversations
    async getConversations() {
        try {
            const response = await api.get('/messages/conversations');
            return response.data;
        } catch (error) {
            console.error('Error fetching conversations:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load conversations. Please try again.'
            );
        }
    },

    // Get messages for a conversation
    async getMessages(conversationId) {
        try {
            const response = await api.get(`/messages/${conversationId}`);
            return response.data;
        } catch (error) {
            console.error('Error fetching messages:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to load messages. Please try again.'
            );
        }
    },

    // Send message
    async sendMessage(messageData) {
        try {
            const response = await api.post('/messages', messageData);
            return response.data;
        } catch (error) {
            console.error('Error sending message:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to send message. Please try again.'
            );
        }
    },

    // Mark messages as read
    async markAsRead(conversationId) {
        try {
            const response = await api.put(`/messages/${conversationId}/read`);
            return response.data;
        } catch (error) {
            console.error('Error marking messages as read:', error);
            throw new Error(
                error.response?.data?.message ||
                'Failed to mark messages as read. Please try again.'
            );
        }
    }
};

export default messageService;
```

--- frontend\src\services\userService.js ---

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

export const userService = {
  getProfile: (id) => {
    return api.get(`/users/profile/${id}`).then(res => res.data);
  },

  updateProfile: (profileData) => {
    return api.put('/users/profile', profileData).then(res => res.data);
  },

  searchFreelancers: (filters = {}) => {
    const params = new URLSearchParams();
    Object.keys(filters).forEach(key => {
      if (filters[key]) {
        params.append(key, filters[key]);
      }
    });
    return api.get(`/users/freelancers?${params}`).then(res => res.data);
  }
};
```

--- frontend\src\styles\App.css ---

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  line-height: 1.6;
  color: #333;
  background-color: #f8f9fa;
}

.app {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}

.main-content {
  flex: 1;
  padding: 20px;
  max-width: 1200px;
  margin: 0 auto;
  width: 100%;
}

.loading {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 200px;
  font-size: 18px;
  color: #666;
}

.error-message {
  background-color: #fee;
  color: #c33;
  padding: 12px;
  border-radius: 6px;
  margin-bottom: 16px;
  border: 1px solid #fcc;
}

.btn-primary {
  background-color: #3498db;
  color: white;
  padding: 12px 24px;
  border: none;
  border-radius: 6px;
  text-decoration: none;
  display: inline-block;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s;
}

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

.btn-secondary {
  background-color: #95a5a6;
  color: white;
  padding: 12px 24px;
  border: none;
  border-radius: 6px;
  text-decoration: none;
  display: inline-block;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s;
}

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

.skill-tag {
  background-color: #e1f5fe;
  color: #0277bd;
  padding: 4px 8px;
  border-radius: 4px;
  font-size: 12px;
  margin: 2px;
  display: inline-block;
}
```

--- frontend\src\styles\Auth.css ---

```
.auth-container {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: calc(100vh - 60px);
  padding: 20px;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
}

.auth-card {
  background: white;
  padding: 40px;
  border-radius: 10px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
  width: 100%;
  max-width: 500px;
}

.auth-card h2 {
  text-align: center;
  margin-bottom: 30px;
  color: #2c3e50;
  font-size: 28px;
}

.auth-form {
  display: flex;
  flex-direction: column;
  gap: 20px;
}

.form-row {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 15px;
}

.form-group {
  display: flex;
```

```css
  flex-direction: column;
}

.form-group label {
  margin-bottom: 8px;
  font-weight: 600;
  color: #555;
}

.form-group input,
.form-group select,
.form-group textarea {
  padding: 12px;
  border: 2px solid #e1e8ed;
  border-radius: 6px;
  font-size: 16px;
  transition: border-color 0.3s;
}

.form-group input:focus,
.form-group select:focus,
.form-group textarea:focus {
  outline: none;
  border-color: #3498db;
}

.auth-button {
  background: linear-gradient(135deg, #3498db, #2980b9);
  color: white;
  padding: 14px;
  border: none;
  border-radius: 6px;
  font-size: 16px;
  font-weight: 600;
  cursor: pointer;
  transition: transform 0.2s;
}

.auth-button:hover:not(:disabled) {
  transform: translateY(-2px);
}

.auth-button:disabled {
  opacity: 0.6;
  cursor: not-allowed;
  transform: none;
}

.auth-link {
  text-align: center;
  margin-top: 25px;
  color: #666;
}

.auth-link a {
  color: #3498db;
  text-decoration: none;
  font-weight: 600;
}

.auth-link a:hover {
  text-decoration: underline;
}

@media (max-width: 768px) {
  .form-row {
    grid-template-columns: 1fr;
  }

  .auth-card {
    padding: 30px 20px;
  }
}
```

--- frontend\src\styles\CreateGig.css ---

```
.create-gig {
  padding: 20px 0;
  max-width: 800px;
  margin: 0 auto;
}

.create-gig-header {
  text-align: center;
  margin-bottom: 40px;
}

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

.create-gig-header p {
  color: #7f8c8d;
  font-size: 1.1rem;
}

.create-gig-form {
  background: white;
  padding: 40px;
  border-radius: 12px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

.form-group {
  margin-bottom: 25px;
}

.form-group label {
  display: block;
  margin-bottom: 8px;
  font-weight: 600;
  color: #2c3e50;
}

.form-group input,
.form-group textarea,
.form-group select {
  width: 100%;
  padding: 12px 16px;
  border: 2px solid #ecf0f1;
  border-radius: 8px;
  font-size: 16px;
  transition: border-color 0.3s ease;
  box-sizing: border-box;
}

.form-group input:focus,
.form-group textarea:focus,
.form-group select:focus {
  outline: none;
  border-color: #3498db;
}

.form-group textarea {
  resize: vertical;
  min-height: 120px;
}

.form-row {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 20px;
}

.error-message {
  background: #e74c3c;
  color: white;
  padding: 12px 16px;
  border-radius: 8px;
  margin-bottom: 20px;
  text-align: center;
}

.btn-primary {
  background: #3498db;
  color: white;
  border: none;
  padding: 14px 30px;
  border-radius: 8px;
  font-size: 16px;
  font-weight: 600;
  cursor: pointer;
  transition: background 0.3s ease;
  width: 100%;
}

.btn-primary:hover:not(:disabled) {
  background: #2980b9;
}

.btn-primary:disabled {
  background: #bdc3c7;
  cursor: not-allowed;
}

@media (max-width: 768px) {
  .create-gig {
    padding: 10px;
  }

  .create-gig-form {
    padding: 20px;
  }

  .form-row {
    grid-template-columns: 1fr;
    gap: 0;
  }
}
```

--- frontend\src\styles\Dashboard.css ---

```
.dashboard {
  padding: 20px;
  max-width: 1400px;
  margin: 0 auto;
  background: #f8fafc;
  min-height: calc(100vh - 80px);
}

/* Loading State */
.dashboard-loading {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 400px;
  color: #64748b;
}

.loading-spinner {
  border: 3px solid #f1f5f9;
  border-top: 3px solid #3b82f6;
  border-radius: 50%;
  width: 40px;
  height: 40px;
  animation: spin 1s linear infinite;
  margin-bottom: 20px;
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
```

```
/* Welcome Section */
.dashboard-welcome {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  padding: 40px;
  border-radius: 16px;
  margin-bottom: 30px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 10px 25px rgba(102, 126, 234, 0.3);
}

.welcome-content h1 {
  font-size: 2.5rem;
  margin-bottom: 10px;
  font-weight: 700;
}

.welcome-content p {
  font-size: 1.2rem;
  opacity: 0.9;
  margin: 0;
}

.welcome-actions .btn-primary {
  background: rgba(255, 255, 255, 0.2);
  border: 2px solid rgba(255, 255, 255, 0.3);
  color: white;
  padding: 14px 28px;
  border-radius: 12px;
  font-weight: 600;
  font-size: 1rem;
  backdrop-filter: blur(10px);
  transition: all 0.3s ease;
}

.welcome-actions .btn-primary:hover {
  background: rgba(255, 255, 255, 0.3);
  transform: translateY(-2px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.2);
}

/* Stats Overview */
.stats-overview {
  margin-bottom: 40px;
}

.stats-overview h2 {
  color: #1e293b;
  margin-bottom: 20px;
  font-size: 1.8rem;
  font-weight: 600;
}

.stats-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
  gap: 20px;
}

.stat-card {
  background: white;
  padding: 25px;
  border-radius: 12px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.05);
  border: 1px solid #e2e8f0;
  display: flex;
  align-items: center;
  transition: all 0.3s ease;
  position: relative;
  overflow: hidden;
}
```

```
.stat-card::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  height: 4px;
  background: linear-gradient(90deg, #3b82f6, #8b5cf6);
}

.stat-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 12px 25px rgba(0, 0, 0, 0.1);
}

.stat-icon {
  width: 60px;
  height: 60px;
  border-radius: 12px;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-right: 20px;
  font-size: 1.5rem;
}

.stat-icon.applications { background: #dbeafe; }
.stat-icon.pending { background: #fef3c7; }
.stat-icon.earnings { background: #dcfce7; }
.stat-icon.messages { background: #f3e8ff; }
.stat-icon.gigs { background: #ffe4e6; }
.stat-icon.active { background: #ffedd5; }

.stat-info h3 {
  font-size: 2rem;
  font-weight: 700;
  color: #1e293b;
  margin: 0 0 5px 0;
}

.stat-info p {
  color: #64748b;
  margin: 0;
  font-size: 0.9rem;
}

.stat-trend {
  margin-left: auto;
  text-align: right;
}

.trend-up {
  color: #10b981;
  font-weight: 600;
  font-size: 0.85rem;
}

.trend-down {
  color: #ef4444;
  font-weight: 600;
  font-size: 0.85rem;
}

.trend-neutral {
  color: #6b7280;
  font-weight: 600;
  font-size: 0.85rem;
}

/* Dashboard Content Layout */
.dashboard-content {
  display: grid;
  grid-template-columns: 1fr 400px;
  gap: 30px;
  margin-bottom: 40px;
```

```
}

/* Quick Actions */
.quick-actions-section h2 {
  color: #1e293b;
  margin-bottom: 20px;
  font-size: 1.8rem;
  font-weight: 600;
}

.actions-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 15px;
}

.action-card {
  background: white;
  padding: 20px;
  border-radius: 12px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
  border: 1px solid #e2e8f0;
  display: flex;
  align-items: center;
  text-decoration: none;
  color: inherit;
  transition: all 0.3s ease;
  position: relative;
}

.action-card:hover {
  transform: translateY(-3px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
  border-color: #3b82f6;
}

.action-icon {
  width: 50px;
  height: 50px;
  border-radius: 10px;
  background: #f1f5f9;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-right: 15px;
  font-size: 1.3rem;
}

.action-content {
  flex: 1;
}

.action-content h3 {
  font-size: 1.1rem;
  font-weight: 600;
  color: #1e293b;
  margin: 0 0 5px 0;
}

.action-content p {
  color: #64748b;
  margin: 0;
  font-size: 0.85rem;
}

.action-arrow {
  color: #94a3b8;
  font-size: 1.2rem;
  font-weight: 300;
  transition: transform 0.3s ease;
}

.action-card:hover .action-arrow {
  transform: translateX(5px);
  color: #3b82f6;
```

```css
}

/* Recent Activity */
.recent-activity-section {
  background: white;
  padding: 25px;
  border-radius: 12px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.05);
  border: 1px solid #e2e8f0;
  height: fit-content;
}

.section-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
}

.section-header h2 {
  color: #1e293b;
  margin: 0;
  font-size: 1.5rem;
  font-weight: 600;
}

.view-all-link {
  color: #3b82f6;
  text-decoration: none;
  font-weight: 500;
  font-size: 0.9rem;
  transition: color 0.3s ease;
}

.view-all-link:hover {
  color: #1d4ed8;
}

.activity-list {
  display: flex;
  flex-direction: column;
  gap: 15px;
}

.activity-item {
  display: flex;
  align-items: flex-start;
  padding: 15px;
  border-radius: 8px;
  background: #f8fafc;
  transition: background 0.3s ease;
}

.activity-item:hover {
  background: #f1f5f9;
}

.activity-icon {
  width: 40px;
  height: 40px;
  border-radius: 8px;
  background: #e2e8f0;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-right: 15px;
  flex-shrink: 0;
}

.activity-content {
  flex: 1;
}

.activity-content h4 {
  font-size: 1rem;
  font-weight: 600;
  color: #1e293b;
  margin: 0 0 5px 0;
}

.activity-content p {
  color: #64748b;
  margin: 0 0 5px 0;
  font-size: 0.85rem;
}

.activity-time {
  color: #94a3b8;
  font-size: 0.8rem;
}

.activity-status {
  margin-left: 15px;
}

.status-badge {
  padding: 4px 8px;
  border-radius: 12px;
  font-size: 0.75rem;
  font-weight: 600;
  text-transform: capitalize;
}

.status-badge.open {
  background: #d1fae5;
  color: #065f46;
}

.status-badge.in_progress {
  background: #fef3c7;
  color: #92400e;
}

.status-badge.completed {
  background: #e0e7ff;
  color: #3730a3;
}

.status-badge.cancelled {
  background: #fee2e2;
  color: #991b1b;
}

.no-activity {
  text-align: center;
  padding: 40px 20px;
  color: #64748b;
}

.no-activity p {
  margin-bottom: 15px;
}

.btn-secondary {
  background: #64748b;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 8px;
  text-decoration: none;
  font-size: 0.9rem;
  font-weight: 500;
  transition: background 0.3s ease;
  display: inline-block;
}

.btn-secondary:hover {
  background: #475569;
}
```

```
/* Performance Section */
.performance-section {
  margin-bottom: 40px;
}

.performance-section h2 {
  color: #1e293b;
  margin-bottom: 20px;
  font-size: 1.8rem;
  font-weight: 600;
}

.performance-cards {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 20px;
}

.performance-card {
  background: white;
  padding: 25px;
  border-radius: 12px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.05);
  border: 1px solid #e2e8f0;
  text-align: center;
}

.performance-metric {
  margin-bottom: 15px;
}

.metric-value {
  font-size: 2.5rem;
  font-weight: 700;
  color: #1e293b;
}

.metric-label {
  font-size: 1.2rem;
  color: #64748b;
  margin-left: 5px;
}

.performance-info h3 {
  font-size: 1.2rem;
  font-weight: 600;
  color: #1e293b;
  margin: 0 0 5px 0;
}

.performance-info p {
  color: #64748b;
  margin: 0 0 15px 0;
  font-size: 0.9rem;
}

.performance-stars {
  font-size: 1.2rem;
  color: #f59e0b;
}

.performance-progress {
  background: #e2e8f0;
  border-radius: 10px;
  height: 8px;
  margin: 15px 0;
  overflow: hidden;
}

.progress-bar {
  background: linear-gradient(90deg, #10b981, #3b82f6);
  height: 100%;
  border-radius: 10px;
  transition: width 0.3s ease;
}
```

```css
.performance-trend {
  font-size: 0.85rem;
  font-weight: 600;
  padding: 4px 8px;
  border-radius: 6px;
  display: inline-block;
}

.trend-up {
  background: #d1fae5;
  color: #065f46;
}

/* Responsive Design */
@media (max-width: 1024px) {
  .dashboard-content {
    grid-template-columns: 1fr;
  }

  .recent-activity-section {
    order: -1;
  }
}

@media (max-width: 768px) {
  .dashboard {
    padding: 15px;
  }

  .dashboard-welcome {
    flex-direction: column;
    text-align: center;
    gap: 20px;
    padding: 30px 20px;
  }

  .welcome-content h1 {
    font-size: 2rem;
  }

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

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

  .performance-cards {
    grid-template-columns: 1fr;
  }

  .stat-card {
    padding: 20px;
  }
}

@media (max-width: 480px) {
  .dashboard-welcome {
    padding: 20px 15px;
  }

  .welcome-content h1 {
    font-size: 1.7rem;
  }

  .stat-card {
    flex-direction: column;
    text-align: center;
    gap: 15px;
  }

  .stat-icon {
    margin-right: 0;
  }

  .stat-trend {
    margin-left: 0;
  }

  .activity-item {
    flex-direction: column;
    gap: 10px;
    text-align: center;
  }

  .activity-status {
    margin-left: 0;
  }
}
```

--- frontend\src\styles\GigDetails.css ---

```
/* Add these styles to your existing GigDetails.css */

.application-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 8px;
}

.application-status {
    padding: 4px 8px;
    border-radius: 12px;
    font-size: 12px;
    font-weight: 600;
    text-transform: capitalize;
}

.application-status.pending {
    background: #fff3cd;
    color: #856404;
}

.application-status.accepted {
    background: #d1ecf1;
    color: #0c5460;
}

.application-status.rejected {
    background: #f8d7da;
    color: #721c24;
}

.application-meta {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-top: 10px;
    font-size: 12px;
    color: #6c757d;
}

.application-actions {
    display: flex;
    gap: 8px;
}

.btn-small {
    padding: 6px 12px;
    font-size: 12px;
}

.btn-primary.btn-small {
    background: #28a745;
}

.btn-primary.btn-small:hover {
    background: #218838;
}

.btn-secondary.btn-small {
    background: #dc3545;
}

.btn-secondary.btn-small:hover {
    background: #c82333;
}
```

--- frontend\src\styles\GigFeed.css ---

```
.gig-feed {
  padding: 20px 0;
  max-width: 1200px;
  margin: 0 auto;
}

.gig-feed-header {
  text-align: center;
  margin-bottom: 40px;
}

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

.gig-feed-header p {
  color: #7f8c8d;
  font-size: 1.1rem;
}

.filters-section {
  background: white;
  padding: 25px;
  border-radius: 12px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  margin-bottom: 30px;
}

.filters-form {
  display: flex;
  flex-direction: column;
  gap: 15px;
}

.filter-row {
  display: flex;
  gap: 15px;
  align-items: center;
  flex-wrap: wrap;
}

.search-input,
.filter-input,
.price-input {
  padding: 12px 16px;
  border: 2px solid #ecf0f1;
  border-radius: 8px;
  font-size: 14px;
  flex: 1;
  min-width: 150px;
}

.search-input:focus,
.filter-input:focus,
.price-input:focus {
  outline: none;
  border-color: #3498db;
}

.btn-primary {
  background: #3498db;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 8px;
  cursor: pointer;
  font-weight: 600;
  transition: background 0.3s ease;
}

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

.btn-secondary {
  background: #95a5a6;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 8px;
  cursor: pointer;
  font-weight: 600;
  transition: background 0.3s ease;
}

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

.gigs-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(350px, 1fr));
  gap: 25px;
  margin-top: 30px;
}

.gig-card {
  background: white;
  border-radius: 12px;
  padding: 25px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
  border: 1px solid #ecf0f1;
}

.gig-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 15px rgba(0, 0, 0, 0.15);
}

.gig-card-header {
  display: flex;
  justify-content: space-between;
  align-items: flex-start;
  margin-bottom: 15px;
}

.gig-card-header h3 {
  color: #2c3e50;
  margin: 0;
  font-size: 1.3rem;
  flex: 1;
  margin-right: 15px;
}

.budget {
  background: #27ae60;
  color: white;
  padding: 6px 12px;
  border-radius: 20px;
  font-weight: 600;
  font-size: 14px;
  white-space: nowrap;
}
```

```css
.gig-description {
  color: #7f8c8d;
  line-height: 1.5;
  margin-bottom: 15px;
}

.gig-meta {
  display: flex;
  gap: 15px;
  margin-bottom: 15px;
  flex-wrap: wrap;
}

.location, .duration {
  display: flex;
  align-items: center;
  gap: 5px;
  color: #7f8c8d;
  font-size: 14px;
}

.skills-list {
  display: flex;
  flex-wrap: wrap;
  gap: 8px;
  margin-bottom: 15px;
}

.skill-tag {
  background: #ecf0f1;
  color: #2c3e50;
  padding: 4px 10px;
  border-radius: 20px;
  font-size: 12px;
  font-weight: 500;
}

.gig-card-footer {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-top: 15px;
  padding-top: 15px;
  border-top: 1px solid #ecf0f1;
}

.client {
  color: #7f8c8d;
  font-size: 14px;
}

.view-details-btn {
  background: #3498db;
  color: white;
  padding: 8px 16px;
  border-radius: 6px;
  text-decoration: none;
  font-size: 14px;
  font-weight: 500;
  transition: background 0.3s ease;
}

.view-details-btn:hover {
  background: #2980b9;
}

.no-gigs {
  grid-column: 1 / -1;
  text-align: center;
  padding: 60px 20px;
  color: #7f8c8d;
  background: white;
  border-radius: 12px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
```

```
.no-gigs h3 {
  margin-bottom: 10px;
  color: #2c3e50;
}

.loading {
  text-align: center;
  padding: 40px;
  font-size: 18px;
  color: #7f8c8d;
}

@media (max-width: 768px) {
  .gig-feed {
    padding: 10px;
  }

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

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

  .search-input,
  .filter-input,
  .price-input {
    width: 100%;
  }
}
```

--- frontend\src\styles\Messages.css ---

```
.messages {
  padding: 20px 0;
  height: calc(100vh - 80px);
  max-width: 1400px;
  margin: 0 auto;
}

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

.messages-header h1 {
  color: #2c3e50;
  margin-bottom: 10px;
  font-size: 2.2rem;
}

.messages-header p {
  color: #7f8c8d;
  font-size: 1.1rem;
}

.messages-container {
  display: grid;
  grid-template-columns: 350px 1fr;
  gap: 0;
  height: 600px;
  background: white;
  border-radius: 12px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  overflow: hidden;
}

.conversations-list {
  border-right: 1px solid #ecf0f1;
  background: #f8f9fa;
  overflow-y: auto;
}
```

```
.conversations-list h3 {
  padding: 20px;
  margin: 0;
  color: #2c3e50;
  border-bottom: 1px solid #ecf0f1;
  background: white;
}

.conversation-item {
  display: flex;
  align-items: center;
  padding: 15px 20px;
  border-bottom: 1px solid #ecf0f1;
  cursor: pointer;
  transition: background 0.3s ease;
  background: white;
}

.conversation-item:hover {
  background: #f8f9fa;
}

.conversation-item.active {
  background: #3498db;
  color: white;
}

.conversation-item.active .participant-name,
.conversation-item.active .last-message,
.conversation-item.active .message-time {
  color: white;
}

.conversation-avatar {
  width: 50px;
  height: 50px;
  border-radius: 50%;
  background: #3498db;
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-weight: 600;
  font-size: 18px;
  margin-right: 15px;
  flex-shrink: 0;
}

.conversation-info {
  flex: 1;
  min-width: 0;
}

.conversation-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 5px;
}

.participant-name {
  font-weight: 600;
  color: #2c3e50;
  font-size: 14px;
}

.message-time {
  font-size: 12px;
  color: #7f8c8d;
}

.last-message {
  color: #7f8c8d;
  font-size: 13px;
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
  margin: 0;
}

.chat-area {
  display: flex;
  flex-direction: column;
  background: white;
}

.chat-header {
  padding: 20px;
  border-bottom: 1px solid #ecf0f1;
  background: white;
}

.chat-partner {
  display: flex;
  align-items: center;
  gap: 15px;
}

.partner-avatar {
  width: 50px;
  height: 50px;
  border-radius: 50%;
  background: #e74c3c;
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-weight: 600;
  font-size: 18px;
}

.partner-type {
  background: #ecf0f1;
  color: #7f8c8d;
  padding: 2px 8px;
  border-radius: 12px;
  font-size: 12px;
  text-transform: capitalize;
}

.messages-list {
  flex: 1;
  padding: 20px;
  overflow-y: auto;
  display: flex;
  flex-direction: column;
  gap: 15px;
  background: #f8f9fa;
}

.message {
  display: flex;
  max-width: 70%;
}

.message.sent {
  align-self: flex-end;
}

.message.received {
  align-self: flex-start;
}

.message-content {
  background: white;
  padding: 12px 16px;
  border-radius: 18px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  position: relative;
}
```

```css
.message.sent .message-content {
  background: #3498db;
  color: white;
}

.message-content p {
  margin: 0 0 5px 0;
  line-height: 1.4;
}

.message-time {
  font-size: 11px;
  color: #7f8c8d;
  display: block;
  text-align: right;
}

.message.sent .message-time {
  color: rgba(255, 255, 255, 0.8);
}

.message-input-form {
  display: flex;
  padding: 20px;
  gap: 10px;
  border-top: 1px solid #ecf0f1;
  background: white;
}

.message-input {
  flex: 1;
  padding: 12px 16px;
  border: 2px solid #ecf0f1;
  border-radius: 25px;
  font-size: 14px;
  outline: none;
  transition: border-color 0.3s ease;
}

.message-input:focus {
  border-color: #3498db;
}

.send-button {
  background: #3498db;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 25px;
  cursor: pointer;
  font-weight: 600;
  transition: background 0.3s ease;
}

.send-button:hover {
  background: #2980b9;
}

.no-conversation {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100%;
  color: #7f8c8d;
  text-align: center;
}

.no-conversation h3 {
  margin-bottom: 10px;
  color: #2c3e50;
}

@media (max-width: 968px) {
  .messages-container {
    grid-template-columns: 1fr;
    height: 500px;
  }

  .conversations-list {
    display: none;
  }

  .message {
    max-width: 85%;
  }
}

/* New styles for contacts and tabs */
.messages-sidebar {
  width: 350px;
  border-right: 1px solid #ecf0f1;
  background: #f8f9fa;
  display: flex;
  flex-direction: column;
}

.sidebar-tabs {
  display: flex;
  border-bottom: 1px solid #e2e8f0;
  background: white;
}

.tab-button {
  flex: 1;
  padding: 15px;
  border: none;
  background: none;
  cursor: pointer;
  font-weight: 500;
  transition: all 0.3s ease;
}

.tab-button.active {
  background: #3498db;
  color: white;
}

.tab-button:hover:not(.active) {
  background: #f1f5f9;
}

.contacts-list {
  flex: 1;
  overflow-y: auto;
}

.contacts-search {
  padding: 15px;
  background: white;
  border-bottom: 1px solid #e2e8f0;
}

.search-input {
  width: 100%;
  padding: 10px 12px;
  border: 2px solid #e2e8f0;
  border-radius: 8px;
  font-size: 14px;
}

.search-input:focus {
  outline: none;
  border-color: #3498db;
}

.contact-item {
  display: flex;
  align-items: center;
  padding: 15px;
  border-bottom: 1px solid #f1f5f9;
  cursor: pointer;
  transition: background 0.3s ease;
  background: white;
}

.contact-item:hover {
  background: #f8fafc;
}

.contact-avatar {
  width: 45px;
  height: 45px;
  border-radius: 50%;
  background: #3498db;
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-weight: 600;
  margin-right: 12px;
  flex-shrink: 0;
}

.contact-info {
  flex: 1;
  min-width: 0;
}

.contact-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 4px;
}

.contact-name {
  font-weight: 600;
  color: #2c3e50;
  font-size: 14px;
}

.contact-rating {
  font-size: 12px;
  color: #f39c12;
}

.contact-skills {
  color: #7f8c8d;
  font-size: 12px;
  margin: 0 0 4px 0;
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}

.user-type-badge {
  background: #ecf0f1;
  color: #7f8c8d;
  padding: 2px 6px;
  border-radius: 10px;
  font-size: 11px;
  text-transform: capitalize;
}

.contact-action {
  margin-left: 10px;
}

.start-chat-btn {
  background: #3498db;
  color: white;
  border: none;
  width: 35px;
  height: 35px;
  border-radius: 50%;
  cursor: pointer;
  display: flex;
  align-items: center;
  justify-content: center;
  transition: background 0.3s ease;
}

.start-chat-btn:hover {
  background: #2980b9;
}

.empty-state {
  text-align: center;
  padding: 40px 20px;
  color: #7f8c8d;
}

.empty-state p {
  margin-bottom: 5px;
}

.empty-state small {
  font-size: 12px;
}

.messages-loading {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 400px;
  color: #7f8c8d;
}
```

--- frontend\src\styles\Navbar.css ---

```
.navbar {
  background-color: #2c3e50;
  color: white;
  padding: 0 20px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  position: sticky;
  top: 0;
  z-index: 1000;
}

.navbar-container {
  display: flex;
  justify-content: space-between;
  align-items: center;
  max-width: 1200px;
  margin: 0 auto;
  height: 60px;
}

.navbar-brand {
  font-size: 24px;
  font-weight: bold;
  color: white;
  text-decoration: none;
}

.navbar-menu {
  display: flex;
  align-items: center;
  gap: 20px;
}

.nav-link {
  color: #ecf0f1;
  text-decoration: none;
  padding: 8px 16px;
  border-radius: 4px;
  transition: background-color 0.3s;
}

.nav-link:hover,
.nav-link.active {
  background-color: #34495e;
  color: white;
}

.user-welcome {
  color: #bdc3c7;
  font-size: 14px;
}

.logout-btn {
  background: none;
  border: 1px solid #e74c3c;
  color: #e74c3c;
  padding: 8px 16px;
  border-radius: 4px;
  cursor: pointer;
  transition: all 0.3s;
}

.logout-btn:hover {
  background-color: #e74c3c;
  color: white;
}
```

--- frontend\src\styles\Profile.css ---

```
.profile {
  padding: 20px 0;
  max-width: 800px;
  margin: 0 auto;
}

.profile-header {
  text-align: center;
  margin-bottom: 40px;
}

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

.profile-header p {
  color: #7f8c8d;
  font-size: 1.1rem;
}

.profile-content {
  display: flex;
  flex-direction: column;
  gap: 25px;
}

.profile-card {
  background: white;
  padding: 30px;
  border-radius: 12px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

.card-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 25px;
  border-bottom: 2px solid #ecf0f1;
  padding-bottom: 15px;
}
```

```
.card-header h2 {
  color: #2c3e50;
  margin: 0;
}

.edit-btn {
  background: #3498db;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 6px;
  cursor: pointer;
  font-weight: 500;
  transition: background 0.3s ease;
}

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

.profile-info {
  display: flex;
  flex-direction: column;
  gap: 20px;
}

.info-item {
  display: flex;
  justify-content: space-between;
  align-items: flex-start;
  padding: 15px 0;
  border-bottom: 1px solid #ecf0f1;
}

.info-item:last-child {
  border-bottom: none;
}

.info-item label {
  font-weight: 600;
  color: #2c3e50;
  min-width: 120px;
}

.info-item span {
  color: #7f8c8d;
  text-align: right;
  flex: 1;
}

.account-type {
  background: #3498db;
  color: white;
  padding: 6px 12px;
  border-radius: 20px;
  font-size: 12px;
  text-transform: capitalize;
  display: inline-block;
}

.account-type.client {
  background: #e74c3c;
}

.account-type.freelancer {
  background: #27ae60;
}

.rating {
  color: #f39c12;
  font-weight: 600;
}

.skills-display {
  display: flex;
```

```css
  flex-wrap: wrap;
  gap: 8px;
  justify-content: flex-end;
}

.skill-tag {
  background: #ecf0f1;
  color: #2c3e50;
  padding: 6px 12px;
  border-radius: 20px;
  font-size: 12px;
  font-weight: 500;
}

.bio-item {
  align-items: flex-start;
}

.bio-item span {
  line-height: 1.5;
}

.edit-input, .edit-textarea {
  padding: 10px 12px;
  border: 2px solid #ecf0f1;
  border-radius: 6px;
  font-size: 14px;
  width: 100%;
  box-sizing: border-box;
  font-family: inherit;
}

.edit-input:focus, .edit-textarea:focus {
  outline: none;
  border-color: #3498db;
}

.edit-textarea {
  resize: vertical;
  min-height: 100px;
}

.edit-actions {
  display: flex;
  gap: 15px;
  justify-content: flex-end;
  margin-top: 25px;
  padding-top: 20px;
  border-top: 1px solid #ecf0f1;
}

.btn-primary {
  background: #27ae60;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 6px;
  cursor: pointer;
  font-weight: 600;
  transition: background 0.3s ease;
}

.btn-primary:hover {
  background: #219a52;
}

.btn-secondary {
  background: #95a5a6;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 6px;
  cursor: pointer;
  font-weight: 600;
  transition: background 0.3s ease;
```

```css
}

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

.reviews-list {
  display: flex;
  flex-direction: column;
  gap: 20px;
}

.review-item {
  background: #f8f9fa;
  padding: 20px;
  border-radius: 8px;
  border-left: 4px solid #3498db;
}

.review-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}

.reviewer {
  font-weight: 600;
  color: #2c3e50;
}

.review-rating {
  color: #f39c12;
  font-size: 14px;
}

.review-comment {
  color: #7f8c8d;
  line-height: 1.5;
  margin: 0 0 10px 0;
}

.review-date {
  font-size: 12px;
  color: #bdc3c7;
}

.profile-actions {
  text-align: center;
  margin-top: 30px;
}

.logout-btn {
  background: #e74c3c;
  color: white;
  border: none;
  padding: 12px 30px;
  border-radius: 6px;
  cursor: pointer;
  font-weight: 600;
  transition: background 0.3s ease;
}

.logout-btn:hover {
  background: #c0392b;
}

.loading, .error-message {
  text-align: center;
  padding: 40px;
  font-size: 18px;
  color: #7f8c8d;
}

.error-message {
  color: #e74c3c;
  background: #f8d7da;
  padding: 20px;
  border-radius: 8px;
  margin: 20px 0;
}

@media (max-width: 768px) {
  .profile {
    padding: 10px;
  }

  .profile-card {
    padding: 20px;
  }

  .info-item {
    flex-direction: column;
    align-items: flex-start;
    gap: 8px;
  }

  .info-item span {
    text-align: left;
  }

  .skills-display {
    justify-content: flex-start;
  }

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

  .card-header {
    flex-direction: column;
    gap: 15px;
    align-items: flex-start;
  }
}
```

--- backend\.env ---

```
PORT=5000
MONGODB_URI=mongodb://localhost:27017/gigconnect
JWT_SECRET=gigconnect_super_secure_jwt_secret_2024
FRONTEND_URL=http://localhost:3000
```

--- backend\package.json ---

```
{
  "name": "gigconnect-backend",
  "version": "1.0.0",
  "description": "GigConnect Backend API",
  "main": "server.js",
  "type": "module",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^7.5.0",
    "razorpay": "^2.9.6",
    "socket.io": "^4.7.2"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

--- backend\server.js ---

```
import express from 'express';
import mongoose from 'mongoose';
import cors from 'cors';
import http from 'http';
import { Server } from 'socket.io';
import dotenv from 'dotenv';

// Load env variables
dotenv.config();

const app = express();
const server = http.createServer(app);
const io = new Server(server, {
    cors: {
        origin: process.env.FRONTEND_URL,
        methods: ['GET', 'POST']
    }
});

// Middleware
app.use(cors());
app.use(express.json());

// Routes
import authRoutes from './routes/auth.js';
import gigRoutes from './routes/gigs.js';
import userRoutes from './routes/users.js';
import messageRoutes from './routes/messages.js';

app.use('/api/auth', authRoutes);
app.use('/api/gigs', gigRoutes);
app.use('/api/users', userRoutes);
app.use('/api/messages', messageRoutes);

// Test route
app.get('/api/test', (req, res) => {
    res.json({
        message: 'GigConnect API is running!',
        timestamp: new Date().toISOString(),
        database: mongoose.connection.readyState === 1 ? 'Connected' : 'Disconnected'
    });
});

// Health check
app.get('/api/health', (req, res) => {
    res.json({
        status: 'OK',
        database: mongoose.connection.readyState === 1 ? 'Connected' : 'Disconnected',
        uptime: process.uptime()
    });
});

// MongoDB Connection with better error handling
const connectDB = async () => {
    try {
        await mongoose.connect(process.env.MONGODB_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        });
        console.log('✓ Local MongoDB connected successfully');
        console.log('✓ Database: ${mongoose.connection.db.databaseName}');
    } catch (error) {
        console.error('✗ MongoDB connection failed:', error.message);
        console.log('✗ Make sure MongoDB is running on localhost:27017');
        console.log('✗ Run: net start MongoDB (as Administrator)');
        process.exit(1);
    }
};

// Socket.io for real-time messaging
io.on('connection', (socket) => {
    console.log('User connected:', socket.id);

    // Join user to their personal room for notifications
    socket.on('join_user', (userId) => {
        socket.join(`user_${userId}`);
        console.log(`User ${userId} joined their room`);
    });

    socket.on('join_conversation', (conversationId) => {
        socket.join(conversationId);
        console.log(`User ${socket.id} joined conversation ${conversationId}`);
    });

    socket.on('send_message', async (data) => {
        try {
            // Save message to database
            const Message = (await import('./models/Message.js')).default;
            const Conversation = (await import('./models/Conversation.js')).default;

            let conversation = await Conversation.findById(data.conversationId);

            if (!conversation) {
                // Create new conversation if it doesn't exist
                conversation = new Conversation({
                    participants: [data.senderId, data.receiverId],
                    lastMessage: data.content,
                    lastMessageAt: new Date()
                });
                await conversation.save();
                data.conversationId = conversation._id;
            } else {
                // Update conversation last message
                conversation.lastMessage = data.content;
                conversation.lastMessageAt = new Date();
                await conversation.save();
            }

            const message = new Message({
                conversationId: data.conversationId,
                senderId: data.senderId,
                receiverId: data.receiverId,
                content: data.content
            });
            await message.save();
            await message.populate('senderId', 'firstName lastName');

            // Add the saved message data to the emission
            const messageData = {
                ...data,
                _id: message._id,
                createdAt: message.createdAt,
                senderId: {
                    _id: message.senderId._id,
                    firstName: message.senderId.firstName,
                    lastName: message.senderId.lastName
                }
            };

            // Emit to all users in the conversation
            io.to(data.conversationId).emit('receive_message', messageData);

            // Notify the receiver if they're not in the conversation
            socket.to(`user_${data.receiverId}`).emit('new_message_notification', {
                conversationId: data.conversationId,
                message: data.content,
                sender: message.senderId
            });
        } catch (error) {
            console.error('Socket message error:', error);
            socket.emit('message_error', { error: 'Failed to send message' });
        }
    });

    socket.on('typing_start', (data) => {
        socket.to(data.conversationId).emit('user_typing', {
            userId: data.userId,
            typing: true
        });
    });
    socket.on('typing_stop', (data) => {
        socket.to(data.conversationId).emit('user_typing', {
            userId: data.userId,
            typing: false
        });
    });

    socket.on('disconnect', () => {
        console.log('User disconnected:', socket.id);
    });
});

// Database connection events
mongoose.connection.on('connected', () => {
    console.log('Mongoose connected to MongoDB');
});

mongoose.connection.on('error', (err) => {
    console.error('Mongoose connection error:', err);
});

mongoose.connection.on('disconnected', () => {
    console.log('Mongoose disconnected');
});

// Start server
const startServer = async () => {
    await connectDB();

    const PORT = process.env.PORT || 5000;
    server.listen(PORT, () => {
        console.log(`Server running on port ${PORT}`);
        console.log(`Frontend: ${process.env.FRONTEND_URL}`);
        console.log(`Health: http://localhost:${PORT}/api/health`);
        console.log(`Test API: http://localhost:${PORT}/api/test`);
    });
};

// Handle graceful shutdown
process.on('SIGTERM', () => {
    console.log('SIGTERM received, shutting down gracefully');
    server.close(() => {
        mongoose.connection.close();
        process.exit(0);
    });
});

startServer();
```

--- backend\middleware\auth.js ---

```
import jwt from 'jsonwebtoken';

const auth = (req, res, next) => {
  const token = req.header('Authorization')?.replace('Bearer ', '');

  if (!token) {
    return res.status(401).json({ message: 'No token, authorization denied' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.userId = decoded.userId;
    next();
  } catch (error) {
    res.status(401).json({ message: 'Token is not valid' });
  }
};

export default auth;
```

--- backend\models\Conversation.js ---

```
import mongoose from 'mongoose';

const ConversationSchema = new mongoose.Schema({
    participants: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true
    }],
    lastMessage: {
        type: String,
        default: ''
    },
    lastMessageAt: {
        type: Date,
        default: Date.now
    }
}, {
    timestamps: true
});

// ONLY THIS INDEX - NO UNIQUE INDEX
// This index is for better performance when sorting by last message
ConversationSchema.index({ lastMessageAt: -1 });

// Method to get the other participant in the conversation
ConversationSchema.methods.getOtherParticipant = function(userId) {
    return this.participants.find(participant => !participant._id.equals(userId));
};

export default mongoose.model('Conversation', ConversationSchema);
```

--- backend\models\Gig.js ---

```
import mongoose from 'mongoose';

const ApplicationSchema = new mongoose.Schema({
  freelancer: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  proposal: {
    type: String,
    required: true
  },
  bidAmount: {
    type: Number,
    required: true
  },
  status: {
    type: String,
    enum: ['pending', 'accepted', 'rejected'],
    default: 'pending'
  },
  appliedAt: {
    type: Date,
    default: Date.now
  }
});

const GigSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    trim: true
  },
  description: {
    type: String,
    required: true
  },
  client: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  skillsRequired: [{
    type: String,
    trim: true
  }],
  budget: {
    type: Number,
    required: true
  },
  location: {
    type: String,
    required: true
  },
  duration: {
    type: String,
    required: true
  },
  status: {
    type: String,
    enum: ['open', 'in_progress', 'completed', 'cancelled'],
    default: 'open'
  },
  applications: [ApplicationSchema]
}, {
  timestamps: true
});

export default mongoose.model('Gig', GigSchema);
```

--- backend\models\Message.js ---

```
import mongoose from 'mongoose';

const MessageSchema = new mongoose.Schema({
  conversationId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Conversation',
    required: true
  },
  senderId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  receiverId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  content: {
    type: String,
    required: true,
    trim: true
  },
  read: {
    type: Boolean,
    default: false
  }
}, {
  timestamps: true
});

// Create indexes for better performance
MessageSchema.index({ conversationId: 1, createdAt: 1 });
MessageSchema.index({ senderId: 1 });
MessageSchema.index({ receiverId: 1 });

export default mongoose.model('Message', MessageSchema);
```

--- backend\models\User.js ---

```
import mongoose from 'mongoose';

const ReviewSchema = new mongoose.Schema({
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  rating: {
    type: Number,
    required: true,
    min: 1,
    max: 5
  },
  comment: {
    type: String,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

const UserSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: [true, 'First name is required'],
    trim: true,
    maxlength: [50, 'First name cannot be more than 50 characters']
  },
  lastName: {
    type: String,
    required: [true, 'Last name is required'],
    trim: true,
    maxlength: [50, 'Last name cannot be more than 50 characters']
  },
  email: {
    type: String,
    required: [true, 'Email is required'],
    unique: true,
    lowercase: true,
    match: [/^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/, 'Please enter a valid email']
  },
  password: {
    type: String,
    required: [true, 'Password is required'],
    minlength: [6, 'Password must be at least 6 characters']
  },
  userType: {
    type: String,
    enum: {
      values: ['client', 'freelancer'],
      message: 'User type must be either client or freelancer'
    },
    required: true
  },
  skills: [{
    type: String,
    trim: true
  }],
  location: {
    type: String,
    required: [true, 'Location is required'],
    trim: true
  },
  profileCompleted: {
    type: Boolean,
    default: false
  },
  rating: {
    type: Number,
    default: 0,
    min: 0,
    max: 5
  },
  reviews: [ReviewSchema],
  bio: {
    type: String,
    default: '',
    maxlength: [500, 'Bio cannot be more than 500 characters']
  },
  hourlyRate: {
    type: Number,
    default: 0,
    min: 0
  }
}, {
  timestamps: true
});

// Create indexes for better performance
UserSchema.index({ email: 1 });
UserSchema.index({ userType: 1 });
UserSchema.index({ location: 1 });
UserSchema.index({ skills: 1 });

export default mongoose.model('User', UserSchema);
```

--- backend\routes\auth.js ---

```
import express from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import User from '../models/User.js';

const router = express.Router();

// Register
router.post('/register', async (req, res) => {
  try {
    const { firstName, lastName, email, password, userType, skills, location } = req.body;

    // Check if user exists
    let user = await User.findOne({ email });
    if (user) {
      return res.status(400).json({ message: 'User already exists with this email' });
    }

    // Create new user
    user = new User({
      firstName,
      lastName,
      email,
      password,
      userType,
      skills: userType === 'freelancer' ? skills.split(',').map(skill => skill.trim()) : [],
      location
    });

    // Hash password
    const salt = await bcrypt.genSalt(10);
    user.password = await bcrypt.hash(password, salt);

    await user.save();

    // Create JWT token
    const payload = { userId: user.id };
    const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '7d' });

    res.status(201).json({
      token,
      user: {
        id: user.id,
        firstName: user.firstName,
        lastName: user.lastName,
        email: user.email,
        userType: user.userType,
        skills: user.skills,
        location: user.location,
        rating: user.rating
      }
    });
  } catch (error) {
    console.error('Registration error:', error);
    res.status(500).json({ message: 'Server error during registration' });
  }
});

// Login
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;

    // Check if user exists
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: 'Invalid email or password' });
    }

    // Check password
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: 'Invalid email or password' });
    }

    // Create JWT token
    const payload = { userId: user.id };
    const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '7d' });

    res.json({
      token,
      user: {
        id: user.id,
        firstName: user.firstName,
        lastName: user.lastName,
        email: user.email,
        userType: user.userType,
        skills: user.skills,
        location: user.location,
        rating: user.rating
      }
    });
  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({ message: 'Server error during login' });
  }
});

// Get user profile
router.get('/profile', async (req, res) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');
    if (!token) {
      return res.status(401).json({ message: 'No token provided' });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.userId).select('-password');

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    res.json(user);
  } catch (error) {
    console.error('Profile error:', error);
    res.status(500).json({ message: 'Server error fetching profile' });
  }
});

export default router;
```

--- backend\routes\gigs.js ---

```
import express from 'express';
import Gig from '../models/Gig.js';
import auth from '../middleware/auth.js';
```

```
const router = express.Router();

// Get all gigs with filters
router.get('/', async (req, res) => {
    try {
        const { search, skills, location, minPrice, maxPrice, limit } = req.query;

        let filter = { status: 'open' };

        if (search) {
            filter.$or = [
                { title: { $regex: search, $options: 'i' } },
                { description: { $regex: search, $options: 'i' } }
            ];
        }

        if (skills) {
            filter.skillsRequired = { $in: skills.split(',') };
        }

        if (location) {
            filter.location = { $regex: location, $options: 'i' };
        }

        if (minPrice || maxPrice) {
            filter.budget = {};
            if (minPrice) filter.budget.$gte = parseInt(minPrice);
            if (maxPrice) filter.budget.$lte = parseInt(maxPrice);
        }

        let query = Gig.find(filter)
            .populate('client', 'firstName lastName rating')
            .sort({ createdAt: -1 });

        if (limit) {
            query = query.limit(parseInt(limit));
        }

        const gigs = await query;

        res.json(gigs);
    } catch (error) {
        console.error('Get gigs error:', error);
        res.status(500).json({ message: 'Server error fetching gigs' });
    }
});

// Create new gig
router.post('/', auth, async (req, res) => {
    try {
        const { title, description, skillsRequired, budget, location, duration } = req.body;

        const gig = new Gig({
            title,
            description,
            client: req.userId,
            skillsRequired: Array.isArray(skillsRequired) ? skillsRequired : skillsRequired.split(',').map(ski
            budget: parseInt(budget),
            location,
            duration
        });
        await gig.save();
        await gig.populate('client', 'firstName lastName');

        res.status(201).json(gig);
    } catch (error) {
        console.error('Create gig error:', error);
        res.status(500).json({ message: 'Server error creating gig' });
    }
});

// Get single gig
router.get('/:id', async (req, res) => {
    try {
        const gig = await Gig.findById(req.params.id)
```

```
            .populate('client', 'firstName lastName rating')
            .populate('applications.freelancer', 'firstName lastName skills rating');

        if (!gig) {
            return res.status(404).json({ message: 'Gig not found' });
        }

        res.json(gig);
    } catch (error) {
        console.error('Get gig error:', error);
        res.status(500).json({ message: 'Server error fetching gig' });
    }
});

// Apply to gig
router.post('/:id/apply', auth, async (req, res) => {
    try {
        const { proposal, bidAmount } = req.body;

        const gig = await Gig.findById(req.params.id);
        if (!gig) {
            return res.status(404).json({ message: 'Gig not found' });
        }

        if (gig.status !== 'open') {
            return res.status(400).json({ message: 'This gig is no longer accepting applications' });
        }

        // Check if already applied
        const existingApplication = gig.applications.find(
            app => app.freelancer.toString() === req.userId
        );

        if (existingApplication) {
            return res.status(400).json({ message: 'You have already applied to this gig' });
        }

        gig.applications.push({
            freelancer: req.userId,
            proposal,
            bidAmount: parseInt(bidAmount)
        });

        await gig.save();

        res.json({
            message: 'Application submitted successfully!',
            gig: await gig.populate('applications.freelancer', 'firstName lastName rating')
        });
    } catch (error) {
        console.error('Apply to gig error:', error);
        res.status(500).json({ message: 'Server error applying to gig' });
    }
});

// Accept application
router.put('/:id/accept-application', auth, async (req, res) => {
    try {
        const { applicationId } = req.body;

        const gig = await Gig.findById(req.params.id);
        if (!gig) {
            return res.status(404).json({ message: 'Gig not found' });
        }

        // Check if user is the gig owner
        if (gig.client.toString() !== req.userId) {
            return res.status(403).json({ message: 'Not authorized' });
        }

        const application = gig.applications.id(applicationId);
        if (!application) {
            return res.status(404).json({ message: 'Application not found' });
        }
        // Accept this application and reject others
        gig.applications.forEach(app => {
            if (app._id.toString() === applicationId) {
                app.status = 'accepted';
            } else {
                app.status = 'rejected';
            }
        });

        gig.status = 'in_progress';
        await gig.save();

        res.json({
            message: 'Application accepted successfully!',
            gig: await gig.populate('applications.freelancer', 'firstName lastName rating')
        });
    } catch (error) {
        console.error('Accept application error:', error);
        res.status(500).json({ message: 'Server error accepting application' });
    }
});

// Reject application
router.put('/:id/reject-application', auth, async (req, res) => {
    try {
        const { applicationId } = req.body;

        const gig = await Gig.findById(req.params.id);
        if (!gig) {
            return res.status(404).json({ message: 'Gig not found' });
        }

        if (gig.client.toString() !== req.userId) {
            return res.status(403).json({ message: 'Not authorized' });
        }

        const application = gig.applications.id(applicationId);
        if (!application) {
            return res.status(404).json({ message: 'Application not found' });
        }

        application.status = 'rejected';
        await gig.save();

        res.json({
            message: 'Application rejected successfully!',
            gig: await gig.populate('applications.freelancer', 'firstName lastName rating')
        });
    } catch (error) {
        console.error('Reject application error:', error);
        res.status(500).json({ message: 'Server error rejecting application' });
    }
});

// Get freelancer's applications
router.get('/user/applications', auth, async (req, res) => {
    try {
        const gigs = await Gig.find({
            'applications.freelancer': req.userId
        })
        .populate('client', 'firstName lastName rating')
        .sort({ createdAt: -1 });

        // Extract applications for the current user
        const userApplications = gigs.map(gig => {
            const application = gig.applications.find(app =>
                app.freelancer.toString() === req.userId
            );
            return {
                gig: {
                    _id: gig._id,
                    title: gig.title,
                    budget: gig.budget,
                    location: gig.location,
                    status: gig.status,
                    client: gig.client
                },
                application: application
            };
        });

        res.json(userApplications);
    } catch (error) {
        console.error('Get applications error:', error);
        res.status(500).json({ message: 'Server error fetching applications' });
    }
});

export default router;
```

--- backend\routes\messages.js ---

```
import express from 'express';
import auth from '../middleware/auth.js';
import Conversation from '../models/Conversation.js';
import Message from '../models/Message.js';
import User from '../models/User.js';

const router = express.Router();

// Get user's conversations
router.get('/conversations', auth, async (req, res) => {
    try {
        const conversations = await Conversation.find({
            participants: req.userId
        })
        .populate('participants', 'firstName lastName userType email')
        .sort({ lastMessageAt: -1 });

        // Format response to include participant info
        const formattedConversations = conversations.map(conv => {
            const otherParticipant = conv.participants.find(
                participant => !participant._id.equals(req.userId)
            );

            return {
                id: conv._id,
                participant: {
                    _id: otherParticipant._id,
                    name: `${otherParticipant.firstName} ${otherParticipant.lastName}`,
                    type: otherParticipant.userType,
                    email: otherParticipant.email
                },
                lastMessage: conv.lastMessage,
                timestamp: conv.lastMessageAt,
                unreadCount: 0
            };
        });

        res.json(formattedConversations);
    } catch (error) {
        console.error('Get conversations error:', error);
        res.status(500).json({ message: 'Server error fetching conversations' });
    }
});

// Create new conversation with another user - FIXED ROUTE
router.post('/conversation/:userId', auth, async (req, res) => {
    try {
        const { userId } = req.params;
        console.log('Creating conversation between:', req.userId, 'and:', userId);

        // Check if user exists
        const otherUser = await User.findById(userId);
        if (!otherUser) {
            return res.status(404).json({ message: 'User not found' });
        }

        // Always create a new conversation
        const conversation = new Conversation({
            participants: [req.userId, userId],
            lastMessage: 'Conversation started',
            lastMessageAt: new Date()
        });

        await conversation.save();
        await conversation.populate('participants', 'firstName lastName userType email');

        const otherParticipant = conversation.participants.find(
            participant => !participant._id.equals(req.userId)
        );

        console.log('Conversation created successfully:', conversation._id);

        res.status(201).json({
            id: conversation._id,
            participant: {
                _id: otherParticipant._id,
                name: `${otherParticipant.firstName} ${otherParticipant.lastName}`,
                type: otherParticipant.userType
            }
        });
    } catch (error) {
        console.error('Create conversation error:', error);
        res.status(500).json({ message: 'Server error creating conversation: ' + error.message });
    }
});

// Get existing conversation by ID
router.get('/conversation/:conversationId', auth, async (req, res) => {
    try {
        const { conversationId } = req.params;

        const conversation = await Conversation.findById(conversationId)
            .populate('participants', 'firstName lastName userType email');

        if (!conversation) {
            return res.status(404).json({ message: 'Conversation not found' });
        }

        // Verify user is part of this conversation
        if (!conversation.participants.some(p => p._id.equals(req.userId))) {
            return res.status(403).json({ message: 'Access denied' });
        }

        const otherParticipant = conversation.participants.find(
            participant => !participant._id.equals(req.userId)
        );

        res.json({
            id: conversation._id,
            participant: {
                _id: otherParticipant._id,
                name: `${otherParticipant.firstName} ${otherParticipant.lastName}`,
                type: otherParticipant.userType
            }
        });
    } catch (error) {
        console.error('Get conversation error:', error);
        res.status(500).json({ message: 'Server error fetching conversation' });
    }
});

// Get messages for a conversation
router.get('/:conversationId', auth, async (req, res) => {
    try {
        const { conversationId } = req.params;

        // Verify user is part of this conversation
        const conversation = await Conversation.findById(conversationId);
        if (!conversation) {
            return res.status(404).json({ message: 'Conversation not found' });
        }

        if (!conversation.participants.includes(req.userId)) {
            return res.status(403).json({ message: 'Access denied' });
        }

        const messages = await Message.find({ conversationId })
            .populate('senderId', 'firstName lastName')
            .sort({ createdAt: 1 });

        res.json(messages);
    } catch (error) {
        console.error('Get messages error:', error);
        res.status(500).json({ message: 'Server error fetching messages' });
    }
});

// Send message
router.post('/', auth, async (req, res) => {
    try {
        const { conversationId, receiverId, content } = req.body;

        if (!content || !content.trim()) {
            return res.status(400).json({ message: 'Message content is required' });
        }

        let conversation;

        if (conversationId) {
            // Existing conversation
            conversation = await Conversation.findById(conversationId);
            if (!conversation) {
                return res.status(404).json({ message: 'Conversation not found' });
            }

            if (!conversation.participants.includes(req.userId)) {
                return res.status(403).json({ message: 'Access denied' });
            }
        } else if (receiverId) {
            // Create new conversation for new messages
            const receiver = await User.findById(receiverId);
            if (!receiver) {
                return res.status(404).json({ message: 'Receiver not found' });
            }

            conversation = new Conversation({
                participants: [req.userId, receiverId],
                lastMessage: content,
                lastMessageAt: new Date()
            });
            await conversation.save();
        } else {
            return res.status(400).json({ message: 'Either conversationId or receiverId is required' });
        }

        // Update conversation last message
        conversation.lastMessage = content;
        conversation.lastMessageAt = new Date();
        await conversation.save();

        // Create message
        const message = new Message({
            conversationId: conversation._id,
            senderId: req.userId,
            receiverId: receiverId || conversation.getOtherParticipant(req.userId)._id,
            content: content.trim()
        });

        await message.save();
        await message.populate('senderId', 'firstName lastName');

        res.status(201).json(message);
    } catch (error) {
        console.error('Send message error:', error);
        res.status(500).json({ message: 'Server error sending message' });
    }
});
```

```
// Get all conversations between two users
router.get('/user/:userId/conversations', auth, async (req, res) => {
    try {
        const { userId } = req.params;

        const conversations = await Conversation.find({
            participants: { $all: [req.userId, userId] }
        })
        .populate('participants', 'firstName lastName userType email')
        .sort({ lastMessageAt: -1 });

        res.json(conversations);
    } catch (error) {
        console.error('Get user conversations error:', error);
        res.status(500).json({ message: 'Server error fetching user conversations' });
    }
});

// Mark messages as read
router.put('/:conversationId/read', auth, async (req, res) => {
    try {
        const { conversationId } = req.params;

        await Message.updateMany(
            {
                conversationId,
                receiverId: req.userId,
                read: false
            },
            {
                $set: { read: true }
            }
        );

        res.json({ message: 'Messages marked as read' });
    } catch (error) {
        console.error('Mark messages read error:', error);
        res.status(500).json({ message: 'Server error marking messages as read' });
    }
});

export default router;
```

--- backend\routes\users.js ---

```
import express from 'express';
import User from '../models/User.js';
import Gig from '../models/Gig.js';
import auth from '../middleware/auth.js';

const router = express.Router();

// Get all freelancers
router.get('/freelancers', async (req, res) => {
  try {
    const { search, skills, location } = req.query;

    let filter = { userType: 'freelancer' };

    if (search) {
      filter.$or = [
        { firstName: { $regex: search, $options: 'i' } },
        { lastName: { $regex: search, $options: 'i' } },
        { bio: { $regex: search, $options: 'i' } }
      ];
    }

    if (skills) {
      filter.skills = { $in: skills.split(',') };
    }

    if (location) {
      filter.location = { $regex: location, $options: 'i' };
    }
    const freelancers = await User.find(filter)
      .select('-password')
      .sort({ rating: -1, createdAt: -1 });

    res.json(freelancers);
  } catch (error) {
    console.error('Get freelancers error:', error);
    res.status(500).json({ message: 'Server error fetching freelancers' });
  }
});

// Get user by ID
router.get('/:id', async (req, res) => {
  try {
    const user = await User.findById(req.params.id)
      .select('-password')
      .populate('reviews.user', 'firstName lastName');

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    res.json(user);
  } catch (error) {
    console.error('Get user error:', error);
    res.status(500).json({ message: 'Server error fetching user' });
  }
});

// Update user profile
router.put('/profile', auth, async (req, res) => {
  try {
    const { bio, hourlyRate, skills, location } = req.body;

    const user = await User.findById(req.userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Update fields
    if (bio !== undefined) user.bio = bio;
    if (hourlyRate !== undefined) user.hourlyRate = parseInt(hourlyRate) || 0;
    if (skills !== undefined) {
      user.skills = Array.isArray(skills)
        ? skills
        : skills.split(',').map(skill => skill.trim()).filter(skill => skill.length > 0);
    }
    if (location !== undefined) user.location = location;

    user.profileCompleted = true;

    await user.save();

    res.json({
      message: 'Profile updated successfully',
      user: {
        id: user._id,
        firstName: user.firstName,
        lastName: user.lastName,
        email: user.email,
        userType: user.userType,
        skills: user.skills,
        location: user.location,
        bio: user.bio,
        hourlyRate: user.hourlyRate,
        rating: user.rating,
        profileCompleted: user.profileCompleted
      }
    });
  } catch (error) {
    console.error('Update profile error:', error);
    res.status(500).json({ message: 'Server error updating profile' });
  }
});
```

```
// Get dashboard statistics
router.get('/dashboard/stats', auth, async (req, res) => {
  try {
    const user = await User.findById(req.userId);
    let stats = {};

    if (user.userType === 'freelancer') {
      // Freelancer stats
      const gigs = await Gig.find({
        'applications.freelancer': req.userId
      });

      const applications = gigs.flatMap(gig =>
        gig.applications.filter(app => app.freelancer.toString() === req.userId)
      );

      stats = {
        totalApplications: applications.length,
        pendingApplications: applications.filter(app => app.status === 'pending').length,
        earnings: applications
          .filter(app => app.status === 'accepted')
          .reduce((total, app) => total + app.bidAmount, 0),
        totalGigs: await Gig.countDocuments({ status: 'open' }), // Available gigs
        activeGigs: applications.filter(app => app.status === 'accepted').length,
        unreadMessages: 0 // You can implement this later
      };

    } else {
      // Client stats
      const clientGigs = await Gig.find({ client: req.userId });

      stats = {
        totalGigs: clientGigs.length,
        activeGigs: clientGigs.filter(gig =>
          gig.status === 'open' || gig.status === 'in_progress'
        ).length,
        totalApplications: clientGigs.reduce(
          (total, gig) => total + gig.applications.length, 0
        ),
        earnings: 0, // Not used for clients in your current design
        pendingApplications: 0, // Not used for clients
        unreadMessages: 0
      };
    }

    res.json(stats);
  } catch (error) {
    console.error('Dashboard stats error:', error);
    res.status(500).json({ message: 'Server error fetching dashboard stats' });
  }
});

// Add review to user
router.post('/:id/reviews', auth, async (req, res) => {
  try {
    const { rating, comment } = req.body;
    const userId = req.params.id;

    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Check if user already reviewed
    const existingReview = user.reviews.find(
      review => review.user.toString() === req.userId
    );

    if (existingReview) {
      return res.status(400).json({ message: 'You have already reviewed this user' });
    }

    // Add review
    user.reviews.push({
      user: req.userId,
      rating: parseInt(rating),
      comment
    });

    // Recalculate average rating
    const totalRating = user.reviews.reduce((sum, review) => sum + review.rating, 0);
    user.rating = totalRating / user.reviews.length;

    await user.save();
    await user.populate('reviews.user', 'firstName lastName');

    res.json({
      message: 'Review added successfully',
      reviews: user.reviews
    });
  } catch (error) {
    console.error('Add review error:', error);
    res.status(500).json({ message: 'Server error adding review' });
  }
});

// Get all users for messaging (contacts list)
router.get('/contacts/list', auth, async (req, res) => {
  try {
    const currentUser = await User.findById(req.userId);

    // For clients: show all freelancers
    // For freelancers: show all clients
    const filter = {
      _id: { $ne: req.userId } // Exclude current user
    };

    if (currentUser.userType === 'client') {
      filter.userType = 'freelancer';
    } else {
      filter.userType = 'client';
    }

    const contacts = await User.find(filter)
      .select('firstName lastName email userType skills location rating bio')
      .sort({ firstName: 1 });

    res.json(contacts);
  } catch (error) {
    console.error('Get contacts error:', error);
    res.status(500).json({ message: 'Server error fetching contacts' });
  }
});

// Search users for messaging
router.get('/contacts/search', auth, async (req, res) => {
  try {
    const { query } = req.query;
    const currentUser = await User.findById(req.userId);

    const filter = {
      _id: { $ne: req.userId },
      $or: [
        { firstName: { $regex: query, $options: 'i' } },
        { lastName: { $regex: query, $options: 'i' } },
        { skills: { $in: [new RegExp(query, 'i')] } }
      ]
    };

    if (currentUser.userType === 'client') {
      filter.userType = 'freelancer';
    } else {
      filter.userType = 'client';
    }

    const contacts = await User.find(filter)
      .select('firstName lastName email userType skills location rating bio')
      .limit(20)
      .sort({ rating: -1, firstName: 1 });
    res.json(contacts);
  } catch (error) {
    console.error('Search contacts error:', error);
    res.status(500).json({ message: 'Server error searching contacts' });
  }
});

export default router;
```