🚀 SHAKTI 2.0 - Additional Files & Advanced Features

📁 Additional Files

31. src/components/SettingsPanel.jsx

```jsx
```

```
import React from 'react';
import { X, Database, Cpu, Zap, Globe } from 'lucide-react';

const SettingsPanel = ({
  darkMode,
  stateSpace,
  apiProvider,
  providerName,
  onChangeProvider,
  onClose
}) => {
  const cardBg = darkMode ? 'bg-slate-800' : 'bg-white';
  const textPrimary = darkMode ? 'text-white' : 'text-gray-800';
  const textSecondary = darkMode ? 'text-slate-400' : 'text-gray-600';
  const accentColor = darkMode ? 'text-purple-400' : 'text-blue-600';
  const borderColor = darkMode ? 'border-purple-500/20' : 'border-blue-200';

  return (
    <div className={'px-6 py-4 ' + (darkMode ? 'bg-purple-900/30' : 'bg-blue-50') + ' ' + borderColor + ' border-b'}>
      <div className="flex items-center justify-between mb-4">
        <h3 className={'font-semibold text-lg ' + textPrimary}>⚙️ Settings & Statistics</h3>
        <button
          onClick={onClose}
          className={'p-1 rounded hover:bg-opacity-20 ' + (darkMode ? 'hover:bg-white' : 'hover:bg-black')}
        >
          <X className={'w-5 h-5 ' + textSecondary} />
        </button>
      </div>

      <div className="grid grid-cols-2 md:grid-cols-4 gap-4">
        {/* API Provider */}
        <div className={cardBg + ' p-4 rounded-lg ' + borderColor + ' border'}>
          <div className="flex items-center gap-2 mb-2">
            <Globe className={'w-4 h-4 ' + accentColor} />
            <span className={'text-xs font-medium ' + textSecondary}>API Provider</span>
          </div>
          <p className={'text-sm font-semibold ' + textPrimary}>{providerName}</p>
          <button
            onClick={onChangeProvider}
            className={'text-xs mt-2 ' + accentColor + ' hover:underline'}
          >
            Change Provider
          </button>
        </div>

        {/* Context History */}
        <div className={cardBg + ' p-4 rounded-lg ' + borderColor + ' border'}>
          <div className="flex items-center gap-2 mb-2">
            <Database className={'w-4 h-4 ' + accentColor} />
            <span className={'text-xs font-medium ' + textSecondary}>Context History</span>
          </div>
          <p className={'text-2xl font-bold ' + textPrimary}>{stateSpace.contextHistory.length}</p>
          <p className={'text-xs ' + textSecondary}>of 10 max states</p>
        </div>

        {/* Semantic Memory */}
        <div className={cardBg + ' p-4 rounded-lg ' + borderColor + ' border'}>
          <div className="flex items-center gap-2 mb-2">
            <Cpu className={'w-4 h-4 ' + accentColor} />
            <span className={'text-xs font-medium ' + textSecondary}>Semantic Memory</span>
          </div>
          <p className={'text-2xl font-bold ' + textPrimary}>{stateSpace.semanticMemory.length}</p>
          <p className={'text-xs ' + textSecondary}>of 20 max entities</p>
        </div>

        {/* Attention Weights */}
        <div className={cardBg + ' p-4 rounded-lg ' + borderColor + ' border'}>
          <div className="flex items-center gap-2 mb-2">
            <Zap className={'w-4 h-4 ' + accentColor} />
            <span className={'text-xs font-medium ' + textSecondary}>Attention Layers</span>
          </div>
          <p className={'text-2xl font-bold ' + textPrimary}>{stateSpace.attentionWeights.length}</p>
          <p className={'text-xs ' + textSecondary}>active weights</p>
        </div>
      </div>

      {/* Memory Details */}
      {stateSpace.semanticMemory.length > 0 && (
        <div className={'mt-4 p-3 rounded-lg ' + cardBg + ' ' + borderColor + ' border'}>
          <p className={'text-xs font-medium mb-2 ' + textSecondary}>Recent Memory Entities:</p>
          <div className="flex flex-wrap gap-2">
            {stateSpace.semanticMemory.slice(-10).map((item, idx) => (
              <span
                key={idx}
                className={'px-2 py-1 text-xs rounded-full ' + (darkMode ? 'bg-purple-600/30 text-purple-300' : 'bg-blue-100 text-
              >
                {item.entity}
```

```
            </span>
          ))}
        </div>
      </div>
    )}
  </div>
 );
};

export default SettingsPanel;
```

---

32. src/components/WelcomeScreen.jsx

```
jsx
```

```
import React from 'react';
import { Zap, MessageSquare, Brain, Shield, Sparkles } from 'lucide-react';

const WelcomeScreen = ({ darkMode }) => {
  const textPrimary = darkMode ? 'text-white' : 'text-gray-800';
  const textSecondary = darkMode ? 'text-slate-400' : 'text-gray-600';
  const accentColor = darkMode ? 'text-purple-400' : 'text-blue-600';
  const cardBg = darkMode ? 'bg-slate-800/50' : 'bg-white/50';

  const features = [
    { icon: Brain, title: 'State Space Fusion', desc: 'Advanced memory management' },
    { icon: MessageSquare, title: '2000+ Patterns', desc: 'Trained conversations' },
    { icon: Shield, title: '90% Accuracy', desc: 'On trained intents' },
    { icon: Sparkles, title: 'Multi-API', desc: '6 provider options' }
  ];

  return (
    <div className="flex flex-col items-center justify-center h-full p-8 text-center">
      <Zap className={'w-20 h-20 mb-6 ' + accentColor + ' opacity-50'} />
      <h2 className={'text-2xl font-bold mb-2 ' + textPrimary}>Welcome to SHAKTI 2.0</h2>
      <p className={'mb-8 max-w-md ' + textSecondary}>
        Your advanced AI assistant powered by State Space Fusion and Mamba Architecture
      </p>

      <div className="grid grid-cols-2 gap-4 max-w-lg">
        {features.map((feature, idx) => (
          <div key={idx} className={cardBg + ' p-4 rounded-xl backdrop-blur-sm'}>
            <feature.icon className={'w-8 h-8 mb-2 mx-auto ' + accentColor} />
            <h3 className={'font-semibold text-sm ' + textPrimary}>{feature.title}</h3>
            <p className={'text-xs ' + textSecondary}>{feature.desc}</p>
          </div>
        ))}
      </div>

      <div className={'mt-8 p-4 rounded-lg max-w-md ' + (darkMode ? 'bg-purple-900/30' : 'bg-blue-50')}>
        <p className={'text-sm ' + textSecondary}>
          💡 <strong>Try saying:</strong> "Hello", "Who are you?", "What can you do?", or ask any question!
        </p>
      </div>
    </div>
  );
};
```

```
export default WelcomeScreen;
```

---

33. src/components/ExportChat.jsx

```
jsx
```

```
import React from 'react';
import { Download, FileText, FileJson } from 'lucide-react';

const ExportChat = ({ messages, darkMode, onClose }) => {
  const cardBg = darkMode ? 'bg-slate-800' : 'bg-white';
  const textPrimary = darkMode ? 'text-white' : 'text-gray-800';
  const textSecondary = darkMode ? 'text-slate-400' : 'text-gray-600';
  const btnBg = darkMode ? 'bg-purple-600 hover:bg-purple-700' : 'bg-blue-600 hover:bg-blue-700';

  const exportAsText = () => {
    const content = messages.map(msg =>
      (msg.role === 'user' ? 'You: ' : 'SHAKTI: ') + msg.content
    ).join('\n\n');

    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = 'shakti-chat-' + new Date().toISOString().slice(0,10) + '.txt';
    a.click();
    URL.revokeObjectURL(url);
    onClose();
  };

  const exportAsJSON = () => {
    const content = JSON.stringify({
      exportDate: new Date().toISOString(),
      chatbot: 'SHAKTI 2.0',
      messages: messages
    }, null, 2);

    const blob = new Blob([content], { type: 'application/json' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = 'shakti-chat-' + new Date().toISOString().slice(0,10) + '.json';
    a.click();
    URL.revokeObjectURL(url);
    onClose();
  };

  return (
    <div className="fixed inset-0 bg-black/50 flex items-center justify-center z-50 p-4">
```

```
      <div className={cardBg + ' rounded-2xl p-6 max-w-sm w-full shadow-2xl'}>
        <div className="flex items-center gap-3 mb-6">
          <Download className={'w-6 h-6 ' + (darkMode ? 'text-purple-400' : 'text-blue-600')} />
          <h3 className={'text-lg font-bold ' + textPrimary}>Export Chat</h3>
        </div>

        <p className={'text-sm mb-6 ' + textSecondary}>
          Export your conversation history ({messages.length} messages)
        </p>

        <div className="space-y-3">
          <button
            onClick={exportAsText}
            className={'w-full p-4 rounded-lg flex items-center gap-3 transition-colors ' + (darkMode ? 'bg-slate-700 hover:bg-sl
          >
            <FileText className={'w-5 h-5 ' + (darkMode ? 'text-purple-400' : 'text-blue-600')} />
            <div className="text-left">
              <p className={'font-medium ' + textPrimary}>Text File (.txt)</p>
              <p className={'text-xs ' + textSecondary}>Simple readable format</p>
            </div>
          </button>

          <button
            onClick={exportAsJSON}
            className={'w-full p-4 rounded-lg flex items-center gap-3 transition-colors ' + (darkMode ? 'bg-slate-700 hover:bg-sl
          >
            <FileJson className={'w-5 h-5 ' + (darkMode ? 'text-purple-400' : 'text-blue-600')} />
            <div className="text-left">
              <p className={'font-medium ' + textPrimary}>JSON File (.json)</p>
              <p className={'text-xs ' + textSecondary}>Structured data format</p>
            </div>
          </button>
        </div>

        <button
          onClick={onClose}
          className={'w-full mt-4 p-3 rounded-lg text-sm font-medium ' + textSecondary + ' ' + (darkMode ? 'hover:bg-slate-70
        >
          Cancel
        </button>
      </div>
    </div>
  );
};
```

```
export default ExportChat;
```

34. src/components/QuickReplies.jsx

```jsx
import React from 'react';

const QuickReplies = ({ onSelect, darkMode }) => {
  const quickReplies = [
    'Hello!',
    'Who are you?',
    'What can you do?',
    'Tell me a joke',
    'How are you?',
    'Help me'
  ];

  const btnStyle = darkMode
    ? 'bg-purple-600/20 hover:bg-purple-600/40 text-purple-300 border-purple-500/30'
    : 'bg-blue-100 hover:bg-blue-200 text-blue-700 border-blue-200';

  return (
    <div className="flex flex-wrap gap-2 px-6 py-3">
      {quickReplies.map((reply, idx) => (
        <button
          key={idx}
          onClick={() => onSelect(reply)}
          className={'px-3 py-1.5 text-sm rounded-full border transition-colors ' + btnStyle}
        >
          {reply}
        </button>
      ))}
    </div>
  );
};

export default QuickReplies;
```

35. src/components/TypingIndicator.jsx

```jsx
import React from 'react';

const TypingIndicator = ({ darkMode }) => {
  const bgColor = darkMode ? 'bg-slate-800' : 'bg-white';
  const dotColor = darkMode ? 'bg-purple-400' : 'bg-blue-600';
  const borderColor = darkMode ? 'border-slate-700' : 'border-blue-100';

  return (
    <div className="flex justify-start">
      <div className={bgColor + ' ' + borderColor + ' border px-4 py-3 rounded-2xl ' + (darkMode ? '' : 'shadow-sm')}>
        <div className="flex items-center gap-1">
          <span className={'text-xs mr-2 ' + (darkMode ? 'text-slate-400' : 'text-gray-500')}>
            SHAKTI is typing
          </span>
          <div className="flex gap-1">
            {[0, 1, 2].map(i => (
              <div
                key={i}
                className={'w-2 h-2 rounded-full animate-bounce ' + dotColor}
                style={{ animationDelay: i * 150 + 'ms' }}
              />
            ))}
          </div>
        </div>
      </div>
    </div>
  );
};

export default TypingIndicator;
```

---

36. src/services/analyticsService.js

```javascript
```

```
// Analytics Service for tracking usage

class AnalyticsService {
  constructor() {
    this.sessionId = this.generateSessionId();
    this.startTime = Date.now();
    this.messageCount = 0;
    this.trainedHits = 0;
    this.apiCalls = 0;
    this.intentsUsed = {};
  }

  generateSessionId() {
    return 'session_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9);
  }

  trackMessage(isUser) {
    this.messageCount++;
    return {
      sessionId: this.sessionId,
      messageCount: this.messageCount,
      timestamp: Date.now()
    };
  }

  trackTrainedResponse(intent) {
    this.trainedHits++;
    this.intentsUsed[intent] = (this.intentsUsed[intent] || 0) + 1;
    return {
      type: 'trained',
      intent: intent,
      trainedHits: this.trainedHits
    };
  }

  trackAPICall(provider, success) {
    this.apiCalls++;
    return {
      type: 'api',
      provider: provider,
      success: success,
      apiCalls: this.apiCalls
    };
  }

  getSessionStats() {
    const duration = Date.now() - this.startTime;
    return {
      sessionId: this.sessionId,
      duration: duration,
      durationFormatted: this.formatDuration(duration),
      messageCount: this.messageCount,
      trainedHits: this.trainedHits,
      apiCalls: this.apiCalls,
      trainedPercentage: this.messageCount > 0
        ? Math.round((this.trainedHits / (this.trainedHits + this.apiCalls)) * 100)
        : 0,
      topIntents: this.getTopIntents()
    };
  }

  getTopIntents() {
    return Object.entries(this.intentsUsed)
      .sort((a, b) => b[1] - a[1])
      .slice(0, 5)
      .map(([intent, count]) => ({ intent, count }));
  }

  formatDuration(ms) {
    const seconds = Math.floor(ms / 1000);
    const minutes = Math.floor(seconds / 60);
    const hours = Math.floor(minutes / 60);

    if (hours > 0) return hours + 'h ' + (minutes % 60) + 'm';
    if (minutes > 0) return minutes + 'm ' + (seconds % 60) + 's';
    return seconds + 's';
  }

  reset() {
    this.sessionId = this.generateSessionId();
    this.startTime = Date.now();
    this.messageCount = 0;
    this.trainedHits = 0;
    this.apiCalls = 0;
    this.intentsUsed = {};
  }
}
```

```
export const analytics = new AnalyticsService();
export default AnalyticsService;
```

---

37. src/services/voiceService.js

javascript

```
// Voice Input/Output Service

class VoiceService {
  constructor() {
    this.recognition = null;
    this.synthesis = window.speechSynthesis;
    this.isListening = false;
    this.isSupported = this.checkSupport();
  }

  checkSupport() {
    return {
      speechRecognition: 'webkitSpeechRecognition' in window || 'SpeechRecognition' in window,
      speechSynthesis: 'speechSynthesis' in window
    };
  }

  initRecognition() {
    if (!this.isSupported.speechRecognition) {
      console.warn('Speech recognition not supported');
      return false;
    }

    const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;
    this.recognition = new SpeechRecognition();
    this.recognition.continuous = false;
    this.recognition.interimResults = false;
    this.recognition.lang = 'en-US';

    return true;
  }

  startListening(onResult, onError) {
    if (!this.recognition && !this.initRecognition()) {
      onError('Speech recognition not supported');
      return;
    }

    this.recognition.onresult = (event) => {
      const transcript = event.results[0][0].transcript;
      onResult(transcript);
      this.isListening = false;
    };
    this.recognition.onerror = (event) => {
      onError(event.error);
      this.isListening = false;
    };

    this.recognition.onend = () => {
      this.isListening = false;
    };

    try {
      this.recognition.start();
      this.isListening = true;
    } catch (error) {
      onError(error.message);
    }
  }

  stopListening() {
    if (this.recognition && this.isListening) {
      this.recognition.stop();
      this.isListening = false;
    }
  }

  speak(text, options = {}) {
    if (!this.isSupported.speechSynthesis) {
      console.warn('Speech synthesis not supported');
      return;
    }

    // Cancel any ongoing speech
    this.synthesis.cancel();

    const utterance = new SpeechSynthesisUtterance(text);
    utterance.rate = options.rate || 1;
    utterance.pitch = options.pitch || 1;
    utterance.volume = options.volume || 1;
    utterance.lang = options.lang || 'en-US';

    // Get voices and set preferred voice
    const voices = this.synthesis.getVoices();
    const preferredVoice = voices.find(v => v.lang.startsWith('en') && v.name.includes('Female'));
    if (preferredVoice) {
      utterance.voice = preferredVoice;
    }

    this.synthesis.speak(utterance);
  }

  stopSpeaking() {
    if (this.synthesis) {
      this.synthesis.cancel();
    }
  }

  getVoices() {
    return this.synthesis ? this.synthesis.getVoices() : [];
  }
}

export const voiceService = new VoiceService();
export default VoiceService;
```

---

38. src/components/VoiceButton.jsx

jsx

```
import React, { useState } from 'react';
import { Mic, MicOff, Volume2, VolumeX } from 'lucide-react';
import { voiceService } from '../services/voiceService';

const VoiceButton = ({ onVoiceInput, darkMode }) => {
  const [isListening, setIsListening] = useState(false);
  const [error, setError] = useState(null);

  const accentColor = darkMode ? 'text-purple-400' : 'text-blue-600';
  const btnBg = darkMode
    ? 'bg-slate-700 hover:bg-slate-600'
    : 'bg-blue-100 hover:bg-blue-200';
  const activeBg = darkMode
    ? 'bg-purple-600 animate-pulse'
    : 'bg-blue-600 animate-pulse';

  const handleVoiceClick = () => {
    if (isListening) {
      voiceService.stopListening();
      setIsListening(false);
    } else {
      setError(null);
      setIsListening(true);
      voiceService.startListening(
        (transcript) => {
          onVoiceInput(transcript);
          setIsListening(false);
        },
        (err) => {
          setError(err);
          setIsListening(false);
        }
      );
    }
  };

  if (!voiceService.isSupported.speechRecognition) {
    return null;
  }

  return (
    <button
      onClick={handleVoiceClick}
      className={'p-3 rounded-lg transition-all ' + (isListening ? activeBg + ' text-white' : btnBg + ' ' + accentColor)}
      title={isListening ? 'Stop listening' : 'Start voice input'}
    >
      {isListening ? <MicOff className="w-5 h-5" /> : <Mic className="w-5 h-5" />}
    </button>
  );
};

export default VoiceButton;
```

---

39. src/utils/helpers.js

javascript

```
// Utility helper functions

// Format timestamp to readable string
export const formatTime = (timestamp) => {
  const date = new Date(timestamp);
  return date.toLocaleTimeString('en-US', {
    hour: '2-digit',
    minute: '2-digit'
  });
};

// Format date to readable string
export const formatDate = (timestamp) => {
  const date = new Date(timestamp);
  return date.toLocaleDateString('en-US', {
    month: 'short',
    day: 'numeric',
    year: 'numeric'
  });
};

// Truncate text with ellipsis
export const truncate = (text, maxLength = 100) => {
  if (text.length <= maxLength) return text;
  return text.slice(0, maxLength).trim() + '...';
};

// Debounce function
export const debounce = (func, wait) => {
  let timeout;
  return function executedFunction(...args) {
    const later = () => {
      clearTimeout(timeout);
      func(...args);
    };
    clearTimeout(timeout);
    timeout = setTimeout(later, wait);
  };
};

// Throttle function
export const throttle = (func, limit) => {
  let inThrottle;
  return function executedFunction(...args) {
    if (!inThrottle) {
      func(...args);
      inThrottle = true;
      setTimeout(() => inThrottle = false, limit);
    }
  };
};

// Generate unique ID
export const generateId = () => {
  return 'id_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9);
};

// Copy text to clipboard
export const copyToClipboard = async (text) => {
  try {
    await navigator.clipboard.writeText(text);
    return true;
  } catch (err) {
    console.error('Failed to copy:', err);
    return false;
  }
};

// Check if string contains URL
export const containsURL = (text) => {
  const urlPattern = /(https?:\/\/[^\s]+)/g;
  return urlPattern.test(text);
};

// Extract URLs from text
export const extractURLs = (text) => {
  const urlPattern = /(https?:\/\/[^\s]+)/g;
  return text.match(urlPattern) || [];
};

// Sanitize HTML
export const sanitizeHTML = (text) => {
  const div = document.createElement('div');
  div.textContent = text;
  return div.innerHTML;
};
```

```
// Calculate reading time
export const calculateReadingTime = (text) => {
  const wordsPerMinute = 200;
  const words = text.trim().split(/\s+/).length;
  const minutes = Math.ceil(words / wordsPerMinute);
  return minutes;
};

// Deep clone object
export const deepClone = (obj) => {
  return JSON.parse(JSON.stringify(obj));
};

// Check if object is empty
export const isEmpty = (obj) => {
  return Object.keys(obj).length === 0;
};

// Local storage helpers
export const storage = {
  get: (key, defaultValue = null) => {
    try {
      const item = localStorage.getItem(key);
      return item ? JSON.parse(item) : defaultValue;
    } catch {
      return defaultValue;
    }
  },
  set: (key, value) => {
    try {
      localStorage.setItem(key, JSON.stringify(value));
      return true;
    } catch {
      return false;
    }
  },
  remove: (key) => {
    try {
      localStorage.removeItem(key);
      return true;
    } catch {
      return false;
    }
  },
  clear: () => {
    try {
      localStorage.clear();
      return true;
    } catch {
      return false;
    }
  }
};
```

---

40. src/components/MessageActions.jsx

```
jsx
```

```
import React, { useState } from 'react';
import { Copy, Check, Volume2, VolumeX } from 'lucide-react';
import { copyToClipboard } from '../utils/helpers';
import { voiceService } from '../services/voiceService';

const MessageActions = ({ message, darkMode }) => {
  const [copied, setCopied] = useState(false);
  const [speaking, setSpeaking] = useState(false);

  const iconColor = darkMode ? 'text-slate-500 hover:text-slate-300' : 'text-gray-400 hover:text-gray-600';

  const handleCopy = async () => {
    const success = await copyToClipboard(message.content);
    if (success) {
      setCopied(true);
      setTimeout(() => setCopied(false), 2000);
    }
  };

  const handleSpeak = () => {
    if (speaking) {
      voiceService.stopSpeaking();
      setSpeaking(false);
    } else {
      voiceService.speak(message.content);
      setSpeaking(true);
      // Reset after approximate speech duration
      setTimeout(() => setSpeaking(false), message.content.length * 50);
    }
  };

  return (
    <div className="flex gap-1 mt-1 opacity-0 group-hover:opacity-100 transition-opacity">
      <button
        onClick={handleCopy}
        className={'p-1 rounded transition-colors ' + iconColor}
        title="Copy message"
      >
        {copied ? <Check className="w-3 h-3 text-green-500" /> : <Copy className="w-3 h-3" />}
      </button>

      {voiceService.isSupported.speechSynthesis && message.role === 'assistant' && (
        <button
```

```
          onClick={handleSpeak}
          className={'p-1 rounded transition-colors ' + iconColor}
          title={speaking ? 'Stop speaking' : 'Read aloud'}
        >
          {speaking ? <VolumeX className="w-3 h-3" /> : <Volume2 className="w-3 h-3" />}
        </button>
      )}
    </div>
  );
};

export default MessageActions;
```

🧪 Testing Files

41. src/tests/nlpService.test.js

```javascript
import { findTrainedResponse } from '../services/nlpService';
import { trainedDataset } from '../data/trainedDataset';

describe('NLP Service', () => {
  test('should match greeting intent', () => {
    const result = findTrainedResponse('hello', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('greeting');
  });

  test('should match farewell intent', () => {
    const result = findTrainedResponse('goodbye', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('farewell
```