🚀 SHAKTI 2.0 - Additional Files (Part 2)

Continuing Testing & Additional Files...

41. src/tests/nlpService.test.js (Continued)

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');
  });

  test('should match identity intent', () => {
    const result = findTrainedResponse('who are you', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('identity');
  });

  test('should handle filler words', () => {
    const result = findTrainedResponse('hey actually hello friend', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('greeting');
  });

  test('should return null for unknown input', () => {
    const result = findTrainedResponse('xyzabc123', trainedDataset);
    expect(result).toBeNull();
  });

  test('should match wellbeing intent', () => {
    const result = findTrainedResponse('how are you', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('wellbeing');
  });

  test('should match gratitude intent', () => {
    const result = findTrainedResponse('thank you so much', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('gratitude');
  });

  test('should match compliment intent', () => {
    const result = findTrainedResponse('you are amazing', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('compliment');
  });

  test('should match insult intent', () => {
    const result = findTrainedResponse('you are stupid', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('insult');
  });

  test('should match love intent', () => {
    const result = findTrainedResponse('i love you', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('love');
  });

  test('should match confusion intent', () => {
    const result = findTrainedResponse('i dont understand', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('confusion');
  });

  test('should be case insensitive', () => {
    const result = findTrainedResponse('HELLO', trainedDataset);
    expect(result).not.toBeNull();
    expect(result.intent).toBe('greeting');
  });
});
```

---

42. src/tests/stateSpace.test.js

javascript

```
import {
  updateStateSpace,
  generateSemanticEmbedding,
  analyzeSentiment,
  extractKeyEntities,
  calculateAttentionWeights
} from '../utils/stateSpace';

describe('State Space Utils', () => {
  const initialState = {
    contextHistory: [],
    semanticMemory: [],
    attentionWeights: []
  };

  test('should update state space with new conversation', () => {
    const newState = updateStateSpace(initialState, 'hello', 'Hi there!');

    expect(newState.contextHistory.length).toBe(1);
    expect(newState.contextHistory[0].user).toBe('hello');
    expect(newState.contextHistory[0].assistant).toBe('Hi there!');
  });

  test('should limit context history to 10 items', () => {
    let state = initialState;
    for (let i = 0; i < 15; i++) {
      state = updateStateSpace(state, 'message ' + i, 'response ' + i);
    }

    expect(state.contextHistory.length).toBe(10);
  });

  test('should generate semantic embedding', () => {
    const embedding = generateSemanticEmbedding('hello world how are you');

    expect(embedding.length).toBe(5);
    expect(embedding.keywords).toBeInstanceOf(Array);
    expect(['positive', 'negative', 'neutral']).toContain(embedding.sentiment);
  });

  test('should analyze positive sentiment', () => {
    const sentiment = analyzeSentiment('I love this amazing wonderful product');
    expect(sentiment).toBe('positive');
```

```javascript
  });

  test('should analyze negative sentiment', () => {
    const sentiment = analyzeSentiment('This is terrible awful bad');
    expect(sentiment).toBe('negative');
  });

  test('should analyze neutral sentiment', () => {
    const sentiment = analyzeSentiment('The weather is cloudy today');
    expect(sentiment).toBe('neutral');
  });

  test('should extract key entities', () => {
    const entities = extractKeyEntities('Hello testing entities', 'Response message');

    expect(entities).toBeInstanceOf(Array);
    expect(entities.length).toBeLessThanOrEqual(3);
  });

  test('should calculate attention weights', () => {
    const history = [
      { user: 'msg1', assistant: 'resp1' },
      { user: 'msg2', assistant: 'resp2' },
      { user: 'msg3', assistant: 'resp3' }
    ];

    const weights = calculateAttentionWeights(history);

    expect(weights.length).toBe(3);
    expect(weights[2].weight).toBeGreaterThan(weights[0].weight);
  });
});
```

43. src/tests/apiService.test.js

javascript

```
import { apiProviders } from '../services/apiService';

describe('API Service', () => {
  test('should have all required providers', () => {
    const expectedProviders = ['gemini', 'openai', 'anthropic', 'huggingface', 'cohere', 'custom'];

    expectedProviders.forEach(provider => {
      expect(apiProviders).toHaveProperty(provider);
    });
  });

  test('each provider should have required properties', () => {
    Object.values(apiProviders).forEach(provider => {
      expect(provider).toHaveProperty('name');
      expect(provider).toHaveProperty('endpoint');
      expect(provider).toHaveProperty('requiresKey');
      expect(provider).toHaveProperty('free');
      expect(provider).toHaveProperty('description');
    });
  });

  test('gemini should be marked as free', () => {
    expect(apiProviders.gemini.free).toBe(true);
  });

  test('openai should be marked as paid', () => {
    expect(apiProviders.openai.free).toBe(false);
  });

  test('custom provider should have empty endpoint', () => {
    expect(apiProviders.custom.endpoint).toBe('');
  });
});
```

---

44. jest.config.js

```
javascript
```

```
module.exports = {
  testEnvironment: 'jsdom',
  setupFilesAfterEnv: ['<rootDir>/src/setupTests.js'],
  moduleNameMapper: {
    '\\.(css|less|scss|sass)$': 'identity-obj-proxy',
  },
  collectCoverageFrom: [
    'src/**/*.{js,jsx}',
    '!src/index.jsx',
    '!src/reportWebVitals.js',
  ],
  coverageThreshold: {
    global: {
      branches: 70,
      functions: 70,
      lines: 70,
      statements: 70,
    },
  },
};
```

---

45. src/setupTests.js

```
javascript
```

```
import '@testing-library/jest-dom';

// Mock window.speechSynthesis
window.speechSynthesis = {
  speak: jest.fn(),
  cancel: jest.fn(),
  getVoices: jest.fn(() => []),
};

// Mock window.SpeechRecognition
window.SpeechRecognition = jest.fn().mockImplementation(() => ({
  start: jest.fn(),
  stop: jest.fn(),
  addEventListener: jest.fn(),
}));

window.webkitSpeechRecognition = window.SpeechRecognition;

// Mock localStorage
const localStorageMock = {
  getItem: jest.fn(),
  setItem: jest.fn(),
  removeItem: jest.fn(),
  clear: jest.fn(),
};
global.localStorage = localStorageMock;

// Mock clipboard
Object.assign(navigator, {
  clipboard: {
    writeText: jest.fn().mockResolvedValue(undefined),
    readText: jest.fn().mockResolvedValue(''),
  },
});
```

📱 PWA Support Files

46. public/manifest.json

```
json
```

```
{
  "short_name": "SHAKTI 2.0",
  "name": "SHAKTI 2.0 - AI Chatbot",
  "description": "Advanced AI Chatbot with State Space Fusion and Multi-API Support",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose": "any maskable"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose": "any maskable"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#7c3aed",
  "background_color": "#1e1b4b",
  "orientation": "portrait-primary",
  "categories": ["productivity", "utilities"],
  "lang": "en-US",
  "dir": "ltr"
}
```

---

47. public/service-worker.js

javascript

```
const CACHE_NAME = 'shakti-v2.0.0';
const urlsToCache = [
  '/',
  '/index.html',
  '/static/js/bundle.js',
  '/static/css/main.css',
  '/manifest.json'
];

// Install service worker
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('Opened cache');
        return cache.addAll(urlsToCache);
      })
  );
});

// Fetch event
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then((response) => {
        if (response) {
          return response;
        }
        return fetch(event.request);
      })
  );
});

// Activate and clean old caches
self.addEventListener('activate', (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

---

48. src/serviceWorkerRegistration.js

```
javascript
```

```
const isLocalhost = Boolean(
  window.location.hostname === 'localhost' ||
  window.location.hostname === '[::1]' ||
  window.location.hostname.match(/^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/)
);

export function register(config) {
  if ('serviceWorker' in navigator) {
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);
    if (publicUrl.origin !== window.location.origin) {
      return;
    }

    window.addEventListener('load', () => {
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

      if (isLocalhost) {
        checkValidServiceWorker(swUrl, config);
        navigator.serviceWorker.ready.then(() => {
          console.log('This web app is being served cache-first by a service worker.');
        });
      } else {
        registerValidSW(swUrl, config);
      }
    });
  }
}

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then((registration) => {
      registration.onupdatefound = () => {
        const installingWorker = registration.installing;
        if (installingWorker == null) {
          return;
        }
        installingWorker.onstatechange = () => {
          if (installingWorker.state === 'installed') {
            if (navigator.serviceWorker.controller) {
              console.log('New content is available; please refresh.');
              if (config && config.onUpdate) {
                config.onUpdate(registration);
              }
            } else {
              console.log('Content is cached for offline use.');
              if (config && config.onSuccess) {
                config.onSuccess(registration);
              }
            }
          }
        };
      };
    })
    .catch((error) => {
      console.error('Error during service worker registration:', error);
    });
}

function checkValidServiceWorker(swUrl, config) {
  fetch(swUrl, { headers: { 'Service-Worker': 'script' } })
    .then((response) => {
      const contentType = response.headers.get('content-type');
      if (response.status === 404 || (contentType != null && contentType.indexOf('javascript') === -1)) {
        navigator.serviceWorker.ready.then((registration) => {
          registration.unregister().then(() => {
            window.location.reload();
          });
        });
      } else {
        registerValidSW(swUrl, config);
      }
    })
    .catch(() => {
      console.log('No internet connection found. App is running in offline mode.');
    });
}

export function unregister() {
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.ready
      .then((registration) => {
        registration.unregister();
      })
      .catch((error) => {
        console.error(error.message);
      });
  }
}
```

🔐 Security & Error Handling

49. src/utils/errorHandler.js

```
javascript
```

```
// Centralized Error Handler

export class AppError extends Error {
  constructor(message, code, details = {}) {
    super(message);
    this.name = 'AppError';
    this.code = code;
    this.details = details;
    this.timestamp = new Date().toISOString();
  }
}

export const ErrorCodes = {
  API_ERROR: 'API_ERROR',
  NETWORK_ERROR: 'NETWORK_ERROR',
  AUTH_ERROR: 'AUTH_ERROR',
  VALIDATION_ERROR: 'VALIDATION_ERROR',
  RATE_LIMIT_ERROR: 'RATE_LIMIT_ERROR',
  UNKNOWN_ERROR: 'UNKNOWN_ERROR'
};

export const handleAPIError = (error, provider) => {
  console.error(`[${provider}] API Error:`, error);

  if (error.message.includes('401') || error.message.includes('403')) {
    return new AppError(
      'Invalid API key. Please check your credentials.',
      ErrorCodes.AUTH_ERROR,
      { provider }
    );
  }

  if (error.message.includes('429')) {
    return new AppError(
      'Rate limit exceeded. Please wait a moment and try again.',
      ErrorCodes.RATE_LIMIT_ERROR,
      { provider }
    );
  }

  if (error.message.includes('500') || error.message.includes('502') || error.message.includes('503')) {
    return new AppError(
      'API service is temporarily unavailable. Please try again later.',
      ErrorCodes.API_ERROR,
      { provider }
    );
  }

  if (error.name === 'TypeError' && error.message.includes('fetch')) {
    return new AppError(
      'Network error. Please check your internet connection.',
      ErrorCodes.NETWORK_ERROR,
      { provider }
    );
  }

  return new AppError(
    error.message || 'An unexpected error occurred.',
    ErrorCodes.UNKNOWN_ERROR,
    { provider, originalError: error.toString() }
  );
};

export const getErrorMessage = (error) => {
  if (error instanceof AppError) {
    return error.message;
  }

  if (error.response) {
    return `Server error: ${error.response.status}`;
  }

  if (error.request) {
    return 'No response from server. Please check your connection.';
  }

  return error.message || 'An unexpected error occurred.';
};

export const logError = (error, context = {}) => {
  const errorLog = {
    timestamp: new Date().toISOString(),
    error: error instanceof AppError ? {
      message: error.message,
      code: error.code,
      details: error.details
    } : {
      message: error.message,
      stack: error.stack
    },
    context
  };

  console.error('Error Log:', errorLog);

  // In production, you might send this to an error tracking service
  // sendToErrorTracking(errorLog);
};
```

---

50. src/utils/validation.js

javascript

```
// Input Validation Utils

export const validateAPIKey = (key, provider) => {
  if (!key || typeof key !== 'string') {
    return { valid: false, error: 'API key is required' };
  }

  const trimmedKey = key.trim();

  if (trimmedKey.length < 10) {
    return { valid: false, error: 'API key is too short' };
  }

  // Provider-specific validation
  switch (provider) {
    case 'gemini':
      if (!trimmedKey.startsWith('AI')) {
        return { valid: false, error: 'Gemini API keys typically start with "AI"' };
      }
      break;
    case 'openai':
      if (!trimmedKey.startsWith('sk-')) {
        return { valid: false, error: 'OpenAI API keys start with "sk-"' };
      }
      break;
    case 'anthropic':
      if (!trimmedKey.startsWith('sk-ant-')) {
        return { valid: false, error: 'Anthropic API keys start with "sk-ant-"' };
      }
      break;
    case 'huggingface':
      if (!trimmedKey.startsWith('hf_')) {
        return { valid: false, error: 'Hugging Face tokens start with "hf_"' };
      }
      break;
  }

  return { valid: true, error: null };
};

export const validateMessage = (message) => {
  if (!message || typeof message !== 'string') {
    return { valid: false, error: 'Message is required' };
  }

  const trimmedMessage = message.trim();

  if (trimmedMessage.length === 0) {
    return { valid: false, error: 'Message cannot be empty' };
  }

  if (trimmedMessage.length > 5000) {
    return { valid: false, error: 'Message is too long (max 5000 characters)' };
  }

  return { valid: true, error: null, sanitized: trimmedMessage };
};

export const validateEndpoint = (url) => {
  if (!url || typeof url !== 'string') {
    return { valid: false, error: 'Endpoint URL is required' };
  }

  try {
    const parsedUrl = new URL(url.trim());

    if (!['http:', 'https:'].includes(parsedUrl.protocol)) {
      return { valid: false, error: 'URL must use HTTP or HTTPS' };
    }

    return { valid: true, error: null, sanitized: url.trim() };
  } catch {
    return { valid: false, error: 'Invalid URL format' };
  }
};

export const sanitizeInput = (input) => {
  if (typeof input !== 'string') return '';

  return input
    .trim()
    .replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, '')
    .replace(/<[^>]*>/g, '')
    .slice(0, 5000);
};
```

📊 Complete File List Summary

#	File	Category	Description
31	SettingsPanel.jsx	Component	Settings & stats panel
32	WelcomeScreen.jsx	Component	Welcome screen UI
33	ExportChat.jsx	Component	Export chat history
34	QuickReplies.jsx	Component	Quick reply buttons
35	TypingIndicator.jsx	Component	Enhanced typing animation
36	analyticsService.js	Service	Usage analytics tracking
37	voiceService.js	Service	Voice input/output
38	VoiceButton.jsx	Component	Voice input button
39	helpers.js	Utils	Helper functions
40	MessageActions.jsx	Component	Copy/speak actions
41	nlpService.test.js	Test	NLP service tests
42	stateSpace.test.js	Test	State space tests
43	apiService.test.js	Test	API service tests
44	jest.config.js	Config	Jest configuration
45	setupTests.js	Test	Test setup file
46	manifest.json	PWA	PWA manifest
47	service-worker.js	PWA	Service worker
48	serviceWorkerRegistration.js	PWA	SW registration
49	errorHandler.js	Utils	Error handling
50	validation.js	Utils	Input validation

🎯 Total Project Files: 50+

Categories:

- **Core Files**: 4
- **Components**: 15
- **Services**: 4
- **Data**: 2
- **Utils**: 4

- **Hooks**: 2
- **Config**: 4
- **Tests**: 4
- **PWA**: 3
- **Deployment**: 8

🚀 Final Setup Commands

```bash
# Complete setup script
cd shakti-2.0

# Install all dependencies
npm install lucide-react
npm install -D tailwindcss postcss autoprefixer
npm install -D jest @testing-library/react @testing-library/jest-dom identity-obj-proxy

# Initialize configs
npx tailwindcss init -p

# Create all directories
mkdir -p src/{components,services,data,utils,hooks,config,styles,__tests__}
mkdir -p public

# Run tests
npm test

# Build for production
npm run build

# Deploy
vercel deploy --prod
```

🎉 SHAKTI 2.0 is now complete with 50+ files!

Features included:

- ✅ 2000+ trained patterns

- ✅ Multi-API support (6 providers)
- ✅ Dark/Light themes
- ✅ Voice input/output
- ✅ Chat export
- ✅ Analytics tracking
- ✅ PWA support
- ✅ Offline capability
- ✅ Unit tests
- ✅ Error handling
- ✅ Input validation