Technical Specification

Chat Window / LLM Interface

Cursor-Style AI Assistant Integration

Specification Document

January 2026
Version 1.0

Abstract

This document provides a comprehensive technical specification for implementing a Cursor-style chat window and LLM interface within an existing TypeScript application. The chat window supports both local LLM inference (via Ollama or similar) and cloud API providers, features a collapsible panel positioned to the right of the Inspector panel, and provides context-aware AI assistance for development workflows.

Contents

1 Overview

1.1 Purpose

This specification defines the architecture, interfaces, and implementation requirements for an integrated AI chat assistant panel. The system provides:

- Real-time conversational AI assistance within the application
- Context-aware responses utilizing application state and selected elements
- Support for multiple LLM backends (local and API-based)
- Streaming response rendering with code syntax highlighting
- Persistent conversation history with session management

1.2 Scope

The chat window component integrates into the existing application layout as a collapsible right-side panel adjacent to the Inspector panel. It operates independently while maintaining awareness of the application's current context.

1.3 Design Goals

1. **Stability**: Rock-solid operation with graceful degradation on failures
2. **Performance**: Non-blocking UI with efficient streaming response handling
3. **Flexibility**: Provider-agnostic LLM integration supporting local and cloud backends
4. **Accessibility**: WCAG 2.1 AA compliance for all interactive elements
5. **Developer Experience**: Clean TypeScript APIs with comprehensive type safety

2 Architecture

2.1 High-Level Component Diagram

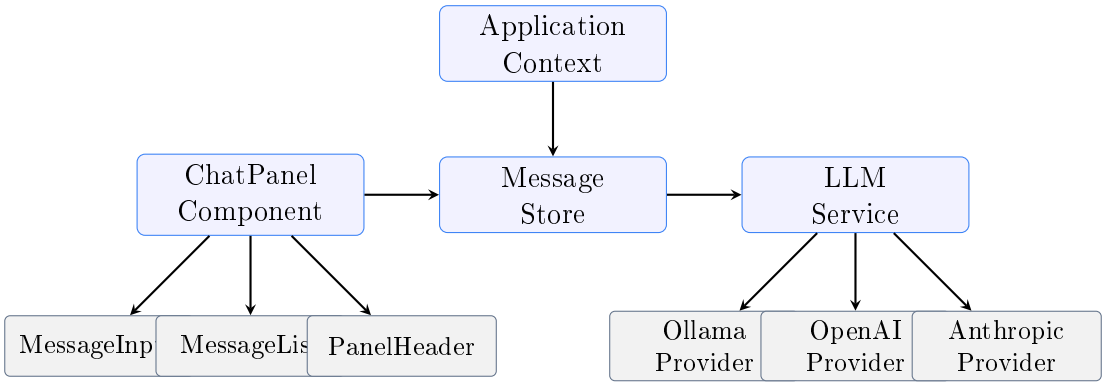

Figure 1: High-level architecture showing component relationships

2.2 Module Structure

```
src/
  chat/
    components/
      ChatPanel.tsx           # Main panel container
      MessageList.tsx         # Scrollable message display
```

```
      MessageItem.tsx          # Individual message rendering
      MessageInput.tsx         # Input area with actions
      CodeBlock.tsx            # Syntax-highlighted code
      PanelHeader.tsx          # Title bar with controls
      ModelSelector.tsx        # LLM provider/model picker
      ContextIndicator.tsx     # Shows active context
    hooks/
      useChatStore.ts          # Zustand store hook
      useStreamingResponse.ts  # SSE/streaming handler
      useLLMConnection.ts      # Provider connection state
      useAutoScroll.ts         # Smart scroll behavior
    services/
      LLMService.ts            # Provider abstraction layer
      providers/
        OllamaProvider.ts      # Local Ollama integration
        OpenAIProvider.ts      # OpenAI API provider
        AnthropicProvider.ts   # Anthropic API provider
        BaseProvider.ts        # Abstract base class
      ContextBuilder.ts        # Application context aggregator
      MessageFormatter.ts      # Markdown/code processing
    store/
      chatStore.ts             # Conversation state management
      settingsStore.ts         # User preferences
    types/
      index.ts                 # Core type definitions
      messages.ts              # Message-related types
      providers.ts             # LLM provider types
      context.ts               # Application context types
    utils/
      tokenCounter.ts          # Token estimation utilities
      codeParser.ts            # Code block extraction
      streamParser.ts          # SSE stream parsing
    constants.ts               # Configuration constants
    index.ts                   # Public API exports
```

3 TypeScript Type Definitions

3.1 Core Message Types

```typescript
// types/messages.ts

export type MessageRole = 'user' | 'assistant' | 'system';

export type MessageStatus =
  | 'pending'      // Awaiting send
  | 'streaming'    // Currently receiving
  | 'complete'     // Finished successfully
  | 'error'        // Failed with error
  | 'cancelled';   // User cancelled

export interface CodeBlock {
  readonly id: string;
  readonly language: string;
  readonly code: string;
  readonly startLine?: number;
  readonly filename?: string;
```

```typescript
}

export interface MessageAttachment {
  readonly id: string;
  readonly type: 'file' | 'selection' | 'image' | 'context';
  readonly name: string;
  readonly content: string;
  readonly mimeType?: string;
  readonly metadata?: Record<string, unknown>;
}

export interface ChatMessage {
  readonly id: string;
  readonly role: MessageRole;
  readonly content: string;
  readonly timestamp: Date;
  readonly status: MessageStatus;
  readonly codeBlocks: readonly CodeBlock[];
  readonly attachments: readonly MessageAttachment[];
  readonly tokenCount?: number;
  readonly modelId?: string;
  readonly error?: ChatError;
  readonly metadata?: MessageMetadata;
}

export interface MessageMetadata {
  readonly duration?: number;        // Response time in ms
  readonly promptTokens?: number;
  readonly completionTokens?: number;
  readonly totalTokens?: number;
  readonly finishReason?: string;
}

export interface ChatError {
  readonly code: string;
  readonly message: string;
  readonly recoverable: boolean;
  readonly details?: unknown;
}

export interface Conversation {
  readonly id: string;
  readonly title: string;
  readonly messages: readonly ChatMessage[];
  readonly createdAt: Date;
  readonly updatedAt: Date;
  readonly modelId: string;
  readonly systemPrompt?: string;
  readonly contextConfig: ContextConfig;
}
```

3.2 LLM Provider Types

```typescript
// types/providers.ts

export type ProviderType = 'ollama' | 'openai' | 'anthropic' | '
    custom';
```

```typescript

export interface LLMModel {
  readonly id: string;
  readonly name: string;
  readonly provider: ProviderType;
  readonly contextWindow: number;
  readonly maxOutputTokens: number;
  readonly supportsStreaming: boolean;
  readonly supportsVision: boolean;
  readonly costPerInputToken?: number;
  readonly costPerOutputToken?: number;
}

export interface ProviderConfig {
  readonly type: ProviderType;
  readonly enabled: boolean;
  readonly baseUrl: string;
  readonly apiKey?: string;
  readonly defaultModel: string;
  readonly timeout: number;
  readonly maxRetries: number;
  readonly customHeaders?: Record<string, string>;
}

export interface OllamaConfig extends ProviderConfig {
  readonly type: 'ollama';
  readonly baseUrl: string;  // Default: http://localhost:11434
  readonly keepAlive?: string;
  readonly numGpu?: number;
  readonly numThread?: number;
}

export interface OpenAIConfig extends ProviderConfig {
  readonly type: 'openai';
  readonly apiKey: string;
  readonly organization?: string;
  readonly baseUrl: string;  // Default: https://api.openai.com/v1
}

export interface AnthropicConfig extends ProviderConfig {
  readonly type: 'anthropic';
  readonly apiKey: string;
  readonly baseUrl: string;  // Default: https://api.anthropic.com
  readonly apiVersion: string;
}

export type AnyProviderConfig =
  | OllamaConfig
  | OpenAIConfig
  | AnthropicConfig;

export interface CompletionRequest {
  readonly messages: readonly ChatMessage[];
  readonly model: string;
  readonly temperature?: number;
  readonly maxTokens?: number;
  readonly topP?: number;
  readonly frequencyPenalty?: number;
```

```
  readonly presencePenalty?: number;
  readonly stop?: readonly string[];
  readonly stream?: boolean;
  readonly signal?: AbortSignal;
}

export interface CompletionResponse {
  readonly id: string;
  readonly content: string;
  readonly model: string;
  readonly finishReason: 'stop' | 'length' | 'error';
  readonly usage: TokenUsage;
}

export interface TokenUsage {
  readonly promptTokens: number;
  readonly completionTokens: number;
  readonly totalTokens: number;
}

export interface StreamChunk {
  readonly id: string;
  readonly delta: string;
  readonly finishReason?: string;
}

export type StreamCallback = (chunk: StreamChunk) => void;
export type ErrorCallback = (error: ChatError) => void;
```

3.3 Application Context Types

```
// types/context.ts

export interface SelectionContext {
  readonly type: 'element' | 'text' | 'code' | 'range';
  readonly content: string;
  readonly elementIds?: readonly string[];
  readonly metadata?: Record<string, unknown>;
}

export interface FileContext {
  readonly path: string;
  readonly name: string;
  readonly content: string;
  readonly language?: string;
  readonly selection?: {
    readonly start: number;
    readonly end: number;
  };
}

export interface ApplicationContext {
  readonly selection: SelectionContext | null;
  readonly activeFile: FileContext | null;
  readonly openFiles: readonly FileContext[];
  readonly projectInfo: ProjectInfo | null;
  readonly customContext: Record<string, unknown>;
```

```
}

export interface ProjectInfo {
  readonly name: string;
  readonly type: string;
  readonly rootPath: string;
  readonly dependencies?: Record<string, string>;
}

export interface ContextConfig {
  readonly includeSelection: boolean;
  readonly includeActiveFile: boolean;
  readonly includeOpenFiles: boolean;
  readonly includeProjectInfo: boolean;
  readonly maxContextTokens: number;
  readonly customInstructions?: string;
}
```

3.4 Store Types

```
// types/store.ts

export interface ChatState {
  // Conversations
  readonly conversations: Map<string, Conversation>;
  readonly activeConversationId: string | null;

  // UI State
  readonly isPanelOpen: boolean;
  readonly isPanelCollapsed: boolean;
  readonly isLoading: boolean;
  readonly error: ChatError | null;

  // Provider State
  readonly activeProvider: ProviderType;
  readonly activeModel: string;
  readonly providerStatus: Map<ProviderType, ProviderStatus>;

  // Context
  readonly contextConfig: ContextConfig;
  readonly applicationContext: ApplicationContext;
}

export interface ProviderStatus {
  readonly connected: boolean;
  readonly lastChecked: Date;
  readonly availableModels: readonly LLMModel[];
  readonly error?: string;
}

export interface ChatActions {
  // Panel
  togglePanel(): void;
  setCollapsed(collapsed: boolean): void;

  // Conversations
  createConversation(title?: string): string;
```

```
  deleteConversation(id: string): void;
  setActiveConversation(id: string): void;
  clearConversation(id: string): void;

  // Messages
  sendMessage(content: string, attachments?: MessageAttachment[]):
     Promise<void>;
  cancelStreaming(): void;
  retryMessage(messageId: string): Promise<void>;
  deleteMessage(messageId: string): void;

  // Provider
  setProvider(provider: ProviderType): void;
  setModel(modelId: string): void;
  refreshProviderStatus(provider: ProviderType): Promise<void>;

  // Context
  updateContextConfig(config: Partial<ContextConfig>): void;
  refreshApplicationContext(): void;
}

export type ChatStore = ChatState & ChatActions;
```

4 LLM Service Implementation

4.1 Abstract Provider Base

```
// services/providers/BaseProvider.ts

export abstract class BaseProvider {
  protected config: ProviderConfig;
  protected abortController: AbortController | null = null;

  constructor(config: ProviderConfig) {
    this.config = config;
  }

  abstract get name(): string;
  abstract get type(): ProviderType;

  abstract connect(): Promise<boolean>;
  abstract disconnect(): Promise<void>;
  abstract isConnected(): boolean;

  abstract listModels(): Promise<LLMModel[]>;

  abstract complete(
    request: CompletionRequest
  ): Promise<CompletionResponse>;

  abstract stream(
    request: CompletionRequest,
    onChunk: StreamCallback,
    onError: ErrorCallback
  ): Promise<void>;

```

```
  cancel(): void {
    if (this.abortController) {
      this.abortController.abort();
      this.abortController = null;
    }
  }

  protected createAbortController(): AbortController {
    this.cancel();
    this.abortController = new AbortController();
    return this.abortController;
  }

  protected async fetchWithRetry(
    url: string,
    options: RequestInit,
    retries = this.config.maxRetries
  ): Promise<Response> {
    let lastError: Error | null = null;

    for (let i = 0; i <= retries; i++) {
      try {
        const response = await fetch(url, {
          ...options,
          signal: this.abortController?.signal,
        });

        if (response.ok) return response;

        if (response.status >= 500 && i < retries) {
          await this.delay(Math.pow(2, i) * 1000);
          continue;
        }

        throw new Error(`HTTP ${response.status}: ${response.
           statusText}`);
      } catch (error) {
        lastError = error as Error;
        if (error instanceof Error && error.name === 'AbortError') {
          throw error;
        }
        if (i < retries) {
          await this.delay(Math.pow(2, i) * 1000);
        }
      }
    }

    throw lastError;
  }

  private delay(ms: number): Promise<void> {
    return new Promise(resolve => setTimeout(resolve, ms));
  }
}
```

4.2 Ollama Provider Implementation

```typescript
// services/providers/OllamaProvider.ts

import { BaseProvider } from './BaseProvider';

export class OllamaProvider extends BaseProvider {
  private connected = false;

  get name(): string { return 'Ollama'; }
  get type(): ProviderType { return 'ollama'; }

  async connect(): Promise<boolean> {
    try {
      const response = await fetch(`${this.config.baseUrl}/api/tags`);
      this.connected = response.ok;
      return this.connected;
    } catch {
      this.connected = false;
      return false;
    }
  }

  async disconnect(): Promise<void> {
    this.connected = false;
  }

  isConnected(): boolean {
    return this.connected;
  }

  async listModels(): Promise<LLMModel[]> {
    const response = await this.fetchWithRetry(
      `${this.config.baseUrl}/api/tags`,
      { method: 'GET' }
    );

    const data = await response.json();

    return data.models.map((model: any) => ({
      id: model.name,
      name: model.name,
      provider: 'ollama',
      contextWindow: model.details?.parameter_size || 4096,
      maxOutputTokens: 4096,
      supportsStreaming: true,
      supportsVision: model.details?.families?.includes('clip') ??
        false,
    }));
  }

  async complete(request: CompletionRequest): Promise<
    CompletionResponse> {
    const controller = this.createAbortController();

    const response = await this.fetchWithRetry(
      `${this.config.baseUrl}/api/chat`,
      {
```

```
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          model: request.model,
          messages: this.formatMessages(request.messages),
          stream: false,
          options: {
            temperature: request.temperature ?? 0.7,
            num_predict: request.maxTokens ?? 2048,
            top_p: request.topP ?? 0.9,
            stop: request.stop,
          },
        }),
        signal: controller.signal,
      }
    );

    const data = await response.json();

    return {
      id: crypto.randomUUID(),
      content: data.message.content,
      model: request.model,
      finishReason: data.done ? 'stop' : 'length',
      usage: {
        promptTokens: data.prompt_eval_count ?? 0,
        completionTokens: data.eval_count ?? 0,
        totalTokens: (data.prompt_eval_count ?? 0) + (data.eval_count ?? 0),
      },
    };
  }

  async stream(
    request: CompletionRequest,
    onChunk: StreamCallback,
    onError: ErrorCallback
  ): Promise<void> {
    const controller = this.createAbortController();

    try {
      const response = await fetch(`${this.config.baseUrl}/api/chat`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          model: request.model,
          messages: this.formatMessages(request.messages),
          stream: true,
          options: {
            temperature: request.temperature ?? 0.7,
            num_predict: request.maxTokens ?? 2048,
            top_p: request.topP ?? 0.9,
            stop: request.stop,
          },
        }),
        signal: controller.signal,
      });
```

```
111
112       if (!response.ok) {
113         throw new Error(`HTTP ${response.status}`);
114       }
115
116       const reader = response.body?.getReader();
117       if (!reader) throw new Error('No response body');
118
119       const decoder = new TextDecoder();
120       let buffer = '';
121
122       while (true) {
123         const { done, value } = await reader.read();
124         if (done) break;
125
126         buffer += decoder.decode(value, { stream: true });
127         const lines = buffer.split('\n');
128         buffer = lines.pop() ?? '';
129
130         for (const line of lines) {
131           if (!line.trim()) continue;
132
133           try {
134             const data = JSON.parse(line);
135             onChunk({
136               id: crypto.randomUUID(),
137               delta: data.message?.content ?? '',
138               finishReason: data.done ? 'stop' : undefined,
139             });
140           } catch (e) {
141             // Skip malformed JSON
142           }
143         }
144       }
145     } catch (error) {
146       if (error instanceof Error && error.name !== 'AbortError') {
147         onError({
148           code: 'STREAM_ERROR',
149           message: error.message,
150           recoverable: true,
151         });
152       }
153     }
154   }
155
156   private formatMessages(messages: readonly ChatMessage[]): any[] {
157     return messages.map(msg => ({
158       role: msg.role,
159       content: msg.content,
160     }));
161   }
162 }
```

4.3 LLM Service Facade

```
1 // services/LLMService.ts
2
```

```typescript
import { OllamaProvider } from './providers/OllamaProvider';
import { OpenAIProvider } from './providers/OpenAIProvider';
import { AnthropicProvider } from './providers/AnthropicProvider';

export class LLMService {
  private providers: Map<ProviderType, BaseProvider> = new Map();
  private activeProvider: BaseProvider | null = null;

  constructor(configs: AnyProviderConfig[]) {
    for (const config of configs) {
      if (!config.enabled) continue;

      const provider = this.createProvider(config);
      if (provider) {
        this.providers.set(config.type, provider);
      }
    }
  }

  private createProvider(config: AnyProviderConfig): BaseProvider |
     null {
    switch (config.type) {
      case 'ollama':
        return new OllamaProvider(config);
      case 'openai':
        return new OpenAIProvider(config);
      case 'anthropic':
        return new AnthropicProvider(config);
      default:
        return null;
    }
  }

  async initialize(): Promise<Map<ProviderType, boolean>> {
    const results = new Map<ProviderType, boolean>();

    for (const [type, provider] of this.providers) {
      const connected = await provider.connect();
      results.set(type, connected);
    }

    // Set first connected provider as active
    for (const [type, connected] of results) {
      if (connected) {
        this.activeProvider = this.providers.get(type) ?? null;
        break;
      }
    }

    return results;
  }

  setActiveProvider(type: ProviderType): boolean {
    const provider = this.providers.get(type);
    if (provider?.isConnected()) {
      this.activeProvider = provider;
      return true;
    }
```

```
60      return false;
61    }
62
63    getActiveProvider(): BaseProvider | null {
64      return this.activeProvider;
65    }
66
67    async complete(request: CompletionRequest): Promise<
          CompletionResponse> {
68      if (!this.activeProvider) {
69        throw new Error('No active LLM provider');
70      }
71      return this.activeProvider.complete(request);
72    }
73
74    async stream(
75      request: CompletionRequest,
76      onChunk: StreamCallback,
77      onError: ErrorCallback
78    ): Promise<void> {
79      if (!this.activeProvider) {
80        throw new Error('No active LLM provider');
81      }
82      return this.activeProvider.stream(request, onChunk, onError);
83    }
84
85    cancel(): void {
86      this.activeProvider?.cancel();
87    }
88
89    async listModels(type?: ProviderType): Promise<LLMModel[]> {
90      if (type) {
91        const provider = this.providers.get(type);
92        return provider?.listModels() ?? [];
93      }
94
95      const allModels: LLMModel[] = [];
96      for (const provider of this.providers.values()) {
97        if (provider.isConnected()) {
98          allModels.push(...await provider.listModels());
99        }
100     }
101     return allModels;
102   }
103 }
```

5 UI Component Specifications

5.1 Panel Layout

Layout Requirements

- Position: Right side of application, adjacent to Inspector panel
- Default width: 400px (configurable: 300px–600px)
- Collapsible: Minimize to 48px icon strip
- Resizable: Horizontal drag handle on left edge
- Z-index: Above main canvas, below modals (z-index: 100)
- Responsive: Collapse automatically below 1200px viewport width

```tsx
// components/ChatPanel.tsx

import { useState, useCallback, useRef } from 'react';
import { useChatStore } from '../hooks/useChatStore';

interface ChatPanelProps {
  defaultWidth?: number;
  minWidth?: number;
  maxWidth?: number;
  collapsedWidth?: number;
}

export const ChatPanel: React.FC<ChatPanelProps> = ({
  defaultWidth = 400,
  minWidth = 300,
  maxWidth = 600,
  collapsedWidth = 48,
}) => {
  const {
    isPanelOpen,
    isPanelCollapsed,
    isLoading,
    togglePanel,
    setCollapsed,
  } = useChatStore();

  const [width, setWidth] = useState(defaultWidth);
  const panelRef = useRef<HTMLDivElement>(null);
  const resizing = useRef(false);

  const handleResizeStart = useCallback((e: React.MouseEvent) => {
    e.preventDefault();
    resizing.current = true;

    const startX = e.clientX;
    const startWidth = width;

    const handleMouseMove = (e: MouseEvent) => {
      if (!resizing.current) return;
      const delta = startX - e.clientX;
      const newWidth = Math.min(maxWidth, Math.max(minWidth,
          startWidth + delta));
      setWidth(newWidth);
    };

    const handleMouseUp = () => {
```

```
      resizing.current = false;
      document.removeEventListener('mousemove', handleMouseMove);
      document.removeEventListener('mouseup', handleMouseUp);
    };

    document.addEventListener('mousemove', handleMouseMove);
    document.addEventListener('mouseup', handleMouseUp);
  }, [width, minWidth, maxWidth]);

  if (!isPanelOpen) return null;

  const panelWidth = isPanelCollapsed ? collapsedWidth : width;

  return (
    <aside
      ref={panelRef}
      className="chat-panel"
      style={{ width: panelWidth }}
      aria-label="AI Chat Assistant"
      role="complementary"
    >
      {/* Resize Handle */}
      {!isPanelCollapsed && (
        <div
          className="chat-panel__resize-handle"
          onMouseDown={handleResizeStart}
          role="separator"
          aria-orientation="vertical"
          aria-label="Resize chat panel"
        />
      )}

      {/* Panel Content */}
      {isPanelCollapsed ? (
        <CollapsedPanel onExpand={() => setCollapsed(false)} />
      ) : (
        <>
          <PanelHeader onCollapse={() => setCollapsed(true)} />
          <MessageList />
          <MessageInput disabled={isLoading} />
        </>
      )}
    </aside>
  );
};
```

5.2 Message List Component

```
// components/MessageList.tsx

import { useRef, useEffect, useCallback } from 'react';
import { useChatStore } from '../hooks/useChatStore';
import { useAutoScroll } from '../hooks/useAutoScroll';
import { MessageItem } from './MessageItem';

export const MessageList: React.FC = () => {
  const { activeConversation } = useChatStore();
```

```
  const containerRef = useRef<HTMLDivElement>(null);
  const { scrollToBottom, isAtBottom } = useAutoScroll(containerRef);

  const messages = activeConversation?.messages ?? [];

  // Auto-scroll on new messages when at bottom
  useEffect(() => {
    if (isAtBottom) {
      scrollToBottom();
    }
  }, [messages.length, isAtBottom, scrollToBottom]);

  if (messages.length === 0) {
    return (
      <div className="message-list message-list--empty">
        <div className="message-list__placeholder">
          <IconSparkles size={48} />
          <h3>Start a conversation</h3>
          <p>Ask questions, get help with code, or discuss your
             project.</p>
        </div>
      </div>
    );
  }

  return (
    <div
      ref={containerRef}
      className="message-list"
      role="log"
      aria-live="polite"
      aria-label="Conversation messages"
    >
      {messages.map((message, index) => (
        <MessageItem
          key={message.id}
          message={message}
          isLast={index === messages.length - 1}
        />
      ))}
    </div>
  );
};
```

5.3 Message Input Component

```
// components/MessageInput.tsx

import { useState, useRef, useCallback, KeyboardEvent } from 'react';
import { useChatStore } from '../hooks/useChatStore';

interface MessageInputProps {
  disabled?: boolean;
  maxLength?: number;
}

export const MessageInput: React.FC<MessageInputProps> = ({
```

```
  disabled = false,
  maxLength = 32000,
}) => {
  const [input, setInput] = useState('');
  const [attachments, setAttachments] = useState<MessageAttachment[]>([]);
  const textareaRef = useRef<HTMLTextAreaElement>(null);

  const { sendMessage, isLoading, cancelStreaming } = useChatStore();

  const handleSubmit = useCallback(async () => {
    const trimmed = input.trim();
    if (!trimmed || disabled || isLoading) return;

    setInput('');
    setAttachments([]);

    await sendMessage(trimmed, attachments);
  }, [input, attachments, disabled, isLoading, sendMessage]);

  const handleKeyDown = useCallback((e: KeyboardEvent<HTMLTextAreaElement>) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault();
      handleSubmit();
    }
  }, [handleSubmit]);

  // Auto-resize textarea
  useEffect(() => {
    const textarea = textareaRef.current;
    if (textarea) {
      textarea.style.height = 'auto';
      textarea.style.height = `${Math.min(textarea.scrollHeight, 200)}px`;
    }
  }, [input]);

  return (
    <div className="message-input">
      {/* Attachments Preview */}
      {attachments.length > 0 && (
        <AttachmentList
          attachments={attachments}
          onRemove={(id) => setAttachments(a => a.filter(x => x.id !== id))}
        />
      )}

      {/* Input Area */}
      <div className="message-input__field">
        <textarea
          ref={textareaRef}
          value={input}
          onChange={(e) => setInput(e.target.value)}
          onKeyDown={handleKeyDown}
          placeholder="Ask anything..."
          disabled={disabled}
```

```
            maxLength={maxLength}
            rows={1}
            aria-label="Message input"
          />

          {/* Actions */}
          <div className="message-input__actions">
            <AttachButton onAttach={(a) => setAttachments(prev => [...
                prev, a])} />

            {isLoading ? (
              <button
                onClick={cancelStreaming}
                className="message-input__cancel"
                aria-label="Cancel response"
              >
                <IconStop size={20} />
              </button>
            ) : (
              <button
                onClick={handleSubmit}
                disabled={!input.trim() || disabled}
                className="message-input__send"
                aria-label="Send message"
              >
                <IconSend size={20} />
              </button>
            )}
          </div>
        </div>

        {/* Character Count */}
        <div className="message-input__footer">
          <span className="message-input__count">
            {input.length} / {maxLength}
          </span>
          <ModelIndicator />
        </div>
      </div>
    );
};
```

5.4 Code Block Rendering

```
// components/CodeBlock.tsx

import { useState, useCallback } from 'react';
import { highlight, languages } from 'prismjs';
import 'prismjs/components/prism-typescript';
import 'prismjs/components/prism-javascript';
import 'prismjs/components/prism-python';
import 'prismjs/components/prism-css';
import 'prismjs/components/prism-json';

interface CodeBlockProps {
  code: string;
  language: string;
```

```
  filename?: string;
  showLineNumbers?: boolean;
  onCopy?: () => void;
  onInsert?: (code: string) => void;
}

export const CodeBlock: React.FC<CodeBlockProps> = ({
  code,
  language,
  filename,
  showLineNumbers = true,
  onCopy,
  onInsert,
}) => {
  const [copied, setCopied] = useState(false);

  const handleCopy = useCallback(async () => {
    await navigator.clipboard.writeText(code);
    setCopied(true);
    onCopy?.();
    setTimeout(() => setCopied(false), 2000);
  }, [code, onCopy]);

  const highlightedCode = highlight(
    code,
    languages[language] || languages.plaintext,
    language
  );

  return (
    <div className="code-block">
      {/* Header */}
      <div className="code-block__header">
        {filename && (
          <span className="code-block__filename">
            <IconFile size={14} />
            {filename}
          </span>
        )}
        <span className="code-block__language">{language}</span>

        <div className="code-block__actions">
          {onInsert && (
            <button
              onClick={() => onInsert(code)}
              className="code-block__action"
              title="Insert at cursor"
            >
              <IconCursorText size={16} />
            </button>
          )}
          <button
            onClick={handleCopy}
            className="code-block__action"
            title={copied ? 'Copied!' : 'Copy code'}
          >
            {copied ? <IconCheck size={16} /> : <IconCopy size={16}
               />}
```

```
            </button>
          </div>
        </div>

        {/* Code Content */}
        <pre className="code-block__content">
          <code
            className={`language-${language}`}
            dangerouslySetInnerHTML={{ __html: highlightedCode }}
          />
        </pre>
      </div>
    );
};
```

6 State Management

6.1 Zustand Store Implementation

```
// store/chatStore.ts

import { create } from 'zustand';
import { devtools, persist } from 'zustand/middleware';
import { immer } from 'zustand/middleware/immer';

export const useChatStore = create<ChatStore>()(
  devtools(
    persist(
      immer((set, get) => ({
        // Initial State
        conversations: new Map(),
        activeConversationId: null,
        isPanelOpen: false,
        isPanelCollapsed: false,
        isLoading: false,
        error: null,
        activeProvider: 'ollama',
        activeModel: 'llama3.1:8b',
        providerStatus: new Map(),
        contextConfig: {
          includeSelection: true,
          includeActiveFile: true,
          includeOpenFiles: false,
          includeProjectInfo: true,
          maxContextTokens: 8000,
        },
        applicationContext: {
          selection: null,
          activeFile: null,
          openFiles: [],
          projectInfo: null,
          customContext: {},
        },

        // Panel Actions
        togglePanel: () => set(state => {
```

```
38            state.isPanelOpen = !state.isPanelOpen;
39          }),
40
41          setCollapsed: (collapsed) => set(state => {
42            state.isPanelCollapsed = collapsed;
43          }),
44
45          // Conversation Actions
46          createConversation: (title) => {
47            const id = crypto.randomUUID();
48            const conversation: Conversation = {
49              id,
50              title: title ?? `Chat ${new Date().toLocaleDateString()}`
                  ,
51              messages: [],
52              createdAt: new Date(),
53              updatedAt: new Date(),
54              modelId: get().activeModel,
55              contextConfig: get().contextConfig,
56            };
57
58            set(state => {
59              state.conversations.set(id, conversation);
60              state.activeConversationId = id;
61            });
62
63            return id;
64          },
65
66          deleteConversation: (id) => set(state => {
67            state.conversations.delete(id);
68            if (state.activeConversationId === id) {
69              const remaining = Array.from(state.conversations.keys());
70              state.activeConversationId = remaining[0] ?? null;
71            }
72          }),
73
74          setActiveConversation: (id) => set(state => {
75            if (state.conversations.has(id)) {
76              state.activeConversationId = id;
77            }
78          }),
79
80          clearConversation: (id) => set(state => {
81            const conv = state.conversations.get(id);
82            if (conv) {
83              conv.messages = [];
84              conv.updatedAt = new Date();
85            }
86          }),
87
88          // Message Actions
89          sendMessage: async (content, attachments = []) => {
90            const { activeConversationId, activeModel } = get();
91
92            let conversationId = activeConversationId;
93            if (!conversationId) {
94              conversationId = get().createConversation();
```

```
95            }
96
97            // Create user message
98            const userMessage: ChatMessage = {
99              id: crypto.randomUUID(),
100             role: 'user',
101             content,
102             timestamp: new Date(),
103             status: 'complete',
104             codeBlocks: [],
105             attachments,
106           };
107
108           // Create placeholder assistant message
109           const assistantMessage: ChatMessage = {
110             id: crypto.randomUUID(),
111             role: 'assistant',
112             content: '',
113             timestamp: new Date(),
114             status: 'streaming',
115             codeBlocks: [],
116             attachments: [],
117             modelId: activeModel,
118           };
119
120           // Add messages to conversation
121           set(state => {
122             const conv = state.conversations.get(conversationId!);
123             if (conv) {
124               conv.messages = [...conv.messages, userMessage,
                      assistantMessage];
125               conv.updatedAt = new Date();
126             }
127             state.isLoading = true;
128             state.error = null;
129           });
130
131           // Stream response
132           try {
133             const llmService = getLLMService();
134             const conv = get().conversations.get(conversationId);
135
136             await llmService.stream(
137               {
138                 messages: conv?.messages ?? [],
139                 model: activeModel,
140                 temperature: 0.7,
141                 maxTokens: 4096,
142               },
143               // On chunk
144               (chunk) => {
145                 set(state => {
146                   const conv = state.conversations.get(conversationId
                          !);
147                   if (conv) {
148                     const msgIndex = conv.messages.findIndex(
149                       m => m.id === assistantMessage.id
150                     );
```

```
                    if (msgIndex >= 0) {
                      const msg = conv.messages[msgIndex];
                      conv.messages[msgIndex] = {
                        ...msg,
                        content: msg.content + chunk.delta,
                        status: chunk.finishReason ? 'complete' : '
                            streaming',
                      };
                    }
                  }
                });
              },
              // On error
              (error) => {
                set(state => {
                  state.error = error;
                  const conv = state.conversations.get(conversationId
                      !);
                  if (conv) {
                    const msgIndex = conv.messages.findIndex(
                      m => m.id === assistantMessage.id
                    );
                    if (msgIndex >= 0) {
                      conv.messages[msgIndex] = {
                        ...conv.messages[msgIndex],
                        status: 'error',
                        error,
                      };
                    }
                  }
                });
              }
            );
          } finally {
            set(state => {
              state.isLoading = false;
            });
          }
        },

        cancelStreaming: () => {
          getLLMService().cancel();
          set(state => {
            state.isLoading = false;
            const convId = state.activeConversationId;
            if (convId) {
              const conv = state.conversations.get(convId);
              if (conv) {
                const lastMsg = conv.messages[conv.messages.length -
                    1];
                if (lastMsg?.status === 'streaming') {
                  conv.messages[conv.messages.length - 1] = {
                    ...lastMsg,
                    status: 'cancelled',
                  };
                }
              }
            }
```

```
            });
          },

          retryMessage: async (messageId) => {
            // Implementation for retry logic
          },

          deleteMessage: (messageId) => set(state => {
            const convId = state.activeConversationId;
            if (convId) {
              const conv = state.conversations.get(convId);
              if (conv) {
                conv.messages = conv.messages.filter(m => m.id !==
                    messageId);
              }
            }
          }),

          // Provider Actions
          setProvider: (provider) => set(state => {
            state.activeProvider = provider;
          }),

          setModel: (modelId) => set(state => {
            state.activeModel = modelId;
          }),

          refreshProviderStatus: async (provider) => {
            // Implementation for provider status refresh
          },

          // Context Actions
          updateContextConfig: (config) => set(state => {
            state.contextConfig = { ...state.contextConfig, ...config
                };
          }),

          refreshApplicationContext: () => {
            // Hook into application state to get current context
          },
        })),
        {
          name: 'chat-storage',
          partialize: (state) => ({
            conversations: Array.from(state.conversations.entries()),
            activeProvider: state.activeProvider,
            activeModel: state.activeModel,
            contextConfig: state.contextConfig,
          }),
        }
      ),
      { name: 'ChatStore' }
    )
);
```

7 Context Building

7.1 Context Builder Service

```
// services/ContextBuilder.ts

export class ContextBuilder {
  private config: ContextConfig;
  private tokenCounter: TokenCounter;

  constructor(config: ContextConfig) {
    this.config = config;
    this.tokenCounter = new TokenCounter();
  }

  build(context: ApplicationContext): string {
    const sections: string[] = [];
    let totalTokens = 0;

    // System instructions
    const systemPrompt = this.buildSystemPrompt();
    sections.push(systemPrompt);
    totalTokens += this.tokenCounter.count(systemPrompt);

    // Selection context
    if (this.config.includeSelection && context.selection) {
      const selectionCtx = this.formatSelection(context.selection);
      const tokens = this.tokenCounter.count(selectionCtx);

      if (totalTokens + tokens <= this.config.maxContextTokens) {
        sections.push(selectionCtx);
        totalTokens += tokens;
      }
    }

    // Active file context
    if (this.config.includeActiveFile && context.activeFile) {
      const fileCtx = this.formatFile(context.activeFile);
      const tokens = this.tokenCounter.count(fileCtx);

      if (totalTokens + tokens <= this.config.maxContextTokens) {
        sections.push(fileCtx);
        totalTokens += tokens;
      }
    }

    // Project info
    if (this.config.includeProjectInfo && context.projectInfo) {
      const projectCtx = this.formatProject(context.projectInfo);
      const tokens = this.tokenCounter.count(projectCtx);

      if (totalTokens + tokens <= this.config.maxContextTokens) {
        sections.push(projectCtx);
        totalTokens += tokens;
      }
    }

    // Custom instructions
```

```
    if (this.config.customInstructions) {
      sections.push(`\n## Custom Instructions\n${this.config.
          customInstructions}`);
    }

    return sections.join('\n\n');
  }

  private buildSystemPrompt(): string {
    return `You are an AI assistant integrated into a design/
        development application.
Your role is to help users with:
- Understanding and modifying their designs
- Writing and debugging code
- Explaining concepts and best practices
- Answering questions about the application

Always be concise and helpful. When providing code, use appropriate
    syntax highlighting.
If you're unsure about something, ask for clarification.`;
  }

  private formatSelection(selection: SelectionContext): string {
    return `## Current Selection
Type: ${selection.type}
${selection.elementIds ? `Elements: ${selection.elementIds.join(', ')
    }` : ''}

\`\`\`
${selection.content}
\`\`\``;
  }

  private formatFile(file: FileContext): string {
    const lang = file.language ?? this.detectLanguage(file.name);
    let content = file.content;

    // Highlight selection if present
    if (file.selection) {
      content = this.highlightSelection(content, file.selection);
    }

    return `## Active File: ${file.name}
Path: ${file.path}

\`\`\`${lang}
${content}
\`\`\``;
  }

  private formatProject(project: ProjectInfo): string {
    return `## Project Information
Name: ${project.name}
Type: ${project.type}
Path: ${project.rootPath}`;
  }

  private detectLanguage(filename: string): string {
```

```
    const ext = filename.split('.').pop()?.toLowerCase();
    const langMap: Record<string, string> = {
      ts: 'typescript',
      tsx: 'typescript',
      js: 'javascript',
      jsx: 'javascript',
      py: 'python',
      css: 'css',
      json: 'json',
      html: 'html',
    };
    return langMap[ext ?? ''] ?? 'plaintext';
  }

  private highlightSelection(
    content: string,
    selection: { start: number; end: number }
  ): string {
    const before = content.slice(0, selection.start);
    const selected = content.slice(selection.start, selection.end);
    const after = content.slice(selection.end);
    return `${before}/* >>> SELECTED START >>> */${selected}/* <<<
       SELECTED END <<< */${after}`;
  }
}
```

8 Configuration

8.1 Default Configuration

```
// constants.ts

export const DEFAULT_PROVIDERS: AnyProviderConfig[] = [
  {
    type: 'ollama',
    enabled: true,
    baseUrl: 'http://localhost:11434',
    defaultModel: 'llama3.1:8b',
    timeout: 120000,
    maxRetries: 3,
  },
  {
    type: 'openai',
    enabled: false,
    baseUrl: 'https://api.openai.com/v1',
    apiKey: '',
    defaultModel: 'gpt-4-turbo',
    timeout: 60000,
    maxRetries: 3,
  },
  {
    type: 'anthropic',
    enabled: false,
    baseUrl: 'https://api.anthropic.com',
    apiKey: '',
    apiVersion: '2024-01-01',
```

```
    defaultModel: 'claude-3-sonnet-20240229',
    timeout: 60000,
    maxRetries: 3,
  },
];

export const PANEL_CONFIG = {
  defaultWidth: 400,
  minWidth: 300,
  maxWidth: 600,
  collapsedWidth: 48,
  autoCollapseBreakpoint: 1200,
} as const;

export const MESSAGE_CONFIG = {
  maxLength: 32000,
  maxAttachments: 10,
  maxAttachmentSize: 10 * 1024 * 1024, // 10MB
} as const;

export const CONTEXT_CONFIG: ContextConfig = {
  includeSelection: true,
  includeActiveFile: true,
  includeOpenFiles: false,
  includeProjectInfo: true,
  maxContextTokens: 8000,
};

export const SUPPORTED_CODE_LANGUAGES = [
  'typescript',
  'javascript',
  'python',
  'css',
  'html',
  'json',
  'markdown',
  'yaml',
  'sql',
  'bash',
  'rust',
  'go',
] as const;
```

9 Accessibility Requirements

9.1 WCAG 2.1 AA Compliance

Accessibility Requirements

1. **Keyboard Navigation**:
   - All interactive elements focusable via Tab
   - Escape closes panel/cancels streaming
   - Enter sends message (Shift+Enter for newline)
   - Arrow keys navigate conversation history

2. **Screen Reader Support**:
   - ARIA labels on all interactive elements
   - Live regions for streaming responses
   - Proper heading hierarchy
   - Descriptive button labels
3. **Visual Accessibility**:
   - Minimum contrast ratio 4.5:1 for text
   - Focus indicators visible (2px outline minimum)
   - No reliance on color alone for information
   - Resizable text up to 200%
4. **Motion/Animation**:
   - Respect `prefers-reduced-motion`
   - No auto-playing animations
   - Typing indicator can be disabled

10 Error Handling

10.1 Error Categories and Recovery

Error Code	Category	Recovery Strategy
`PROVIDER_OFFLINE`	Connection	Auto-retry with exponential backoff; fallback to alternative provider
`RATE_LIMITED`	API	Queue requests; display cooldown timer
`CONTEXT_TOO_LARGE`	Input	Truncate context; warn user
`MODEL_NOT_FOUND`	Configuration	Fallback to default model; prompt reconfiguration
`STREAM_INTERRUPTED`	Network	Offer retry; preserve partial response
`AUTH_FAILED`	API	Prompt for credential update
`TIMEOUT`	Network	Cancel request; offer retry
`INVALID_RESPONSE`	API	Log error; display generic message

```
// utils/errorHandler.ts

export function handleChatError(error: ChatError): ErrorRecovery {
  const strategies: Record<string, ErrorRecovery> = {
    PROVIDER_OFFLINE: {
      action: 'retry',
      message: 'Connection lost. Retrying...',
      retryDelay: 2000,
      maxRetries: 3,
      fallback: () => tryAlternativeProvider(),
    },
    RATE_LIMITED: {
      action: 'wait',
      message: 'Rate limited. Please wait...',
      retryDelay: 60000,
```

```
      maxRetries: 1,
    },
    CONTEXT_TOO_LARGE: {
      action: 'modify',
      message: 'Context too large. Reducing...',
      modification: () => truncateContext(),
    },
    STREAM_INTERRUPTED: {
      action: 'prompt',
      message: 'Response interrupted. Retry?',
      userAction: true,
    },
  };

  return strategies[error.code] ?? {
    action: 'display',
    message: error.message,
  };
}
```

11 Performance Considerations

11.1 Optimization Strategies

1. **Virtual Scrolling**: Implement windowed rendering for conversations with 100+ messages using `react-window` or similar.
2. **Debounced Input**: Debounce input handlers and auto-resize calculations (150ms).
3. **Lazy Code Highlighting**: Defer syntax highlighting for off-screen code blocks.
4. **Message Memoization**: Memoize `MessageItem` components to prevent unnecessary re-renders.
5. **Streaming Buffer**: Buffer streaming chunks (50ms) to reduce render frequency.
6. **IndexedDB Persistence**: Store conversation history in IndexedDB for conversations exceeding 1MB.

Performance Targets

- Panel open/close: < 100ms
- Message send to first token: < 500ms (local), < 2s (API)
- Scroll performance: 60fps with 1000+ messages
- Memory usage: < 50MB for typical session

12 Security Considerations

12.1 Security Requirements

1. **API Key Storage**: Store API keys in system keychain or encrypted storage; never in localStorage.
2. **Input Sanitization**: Sanitize all user input before display; use DOMPurify for HTML content.

3. **Content Security Policy**: Restrict inline scripts; whitelist trusted CDNs only.
4. **Rate Limiting**: Implement client-side rate limiting to prevent accidental API abuse.
5. **Context Filtering**: Filter sensitive data (passwords, tokens) from context before sending.

13 Testing Strategy

13.1 Test Coverage Requirements

Testing Requirements

1. **Unit Tests** ($>$ 80% coverage):
   - All provider implementations
   - Store actions and selectors
   - Context builder logic
   - Message formatting utilities
2. **Integration Tests**:
   - Provider connection flows
   - Streaming response handling
   - Error recovery scenarios
   - State persistence/restoration
3. **E2E Tests**:
   - Complete conversation flow
   - Panel collapse/expand
   - Model switching
   - Context attachment
4. **Accessibility Audit**:
   - Automated axe-core scanning
   - Manual screen reader testing
   - Keyboard-only navigation verification

14 Implementation Phases

14.1 Development Roadmap

1. **Phase 1 - Foundation** (Week 1-2):
   - Core type definitions
   - Zustand store setup
   - Basic panel layout
   - Message list/input components
2. **Phase 2 - LLM Integration** (Week 3-4):
   - Provider abstraction layer
   - Ollama provider implementation
   - Streaming response handling

- Basic error handling

3. **Phase 3 - Context & Features** (Week 5-6):
   - Context builder integration
   - Code block rendering
   - File attachments
   - Conversation persistence
4. **Phase 4 - Polish** (Week 7-8):
   - OpenAI/Anthropic providers
   - Accessibility compliance
   - Performance optimization
   - Comprehensive testing

15 Appendix A: CSS Architecture

```css
/* Chat Panel Container */
.chat-panel {
  position: fixed;
  right: 0;
  top: var(--header-height, 48px);
  bottom: 0;
  display: flex;
  flex-direction: column;
  background: var(--bg-primary);
  border-left: 1px solid var(--border-color);
  z-index: 100;
  transition: width 0.2s ease-out;
}

@media (prefers-reduced-motion: reduce) {
  .chat-panel {
    transition: none;
  }
}

/* Resize Handle */
.chat-panel__resize-handle {
  position: absolute;
  left: 0;
  top: 0;
  bottom: 0;
  width: 4px;
  cursor: ew-resize;
  background: transparent;
  transition: background 0.15s;
}

.chat-panel__resize-handle:hover,
.chat-panel__resize-handle:active {
  background: var(--color-primary);
}

/* Message List */
.message-list {
```

```
  flex: 1;
  overflow-y: auto;
  padding: 16px;
  scroll-behavior: smooth;
}

@media (prefers-reduced-motion: reduce) {
  .message-list {
    scroll-behavior: auto;
  }
}

.message-list--empty {
  display: flex;
  align-items: center;
  justify-content: center;
}

/* Message Item */
.message-item {
  margin-bottom: 16px;
  padding: 12px;
  border-radius: 8px;
}

.message-item--user {
  background: var(--bg-user-message);
  margin-left: 24px;
}

.message-item--assistant {
  background: var(--bg-assistant-message);
  margin-right: 24px;
}

/* Code Block */
.code-block {
  margin: 8px 0;
  border-radius: 6px;
  overflow: hidden;
  border: 1px solid var(--border-color);
}

.code-block__header {
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 8px 12px;
  background: var(--bg-secondary);
  border-bottom: 1px solid var(--border-color);
  font-size: 12px;
}

.code-block__content {
  margin: 0;
  padding: 12px;
  overflow-x: auto;
  font-family: var(--font-mono);
```

```
  font-size: 13px;
  line-height: 1.5;
}

/* Message Input */
.message-input {
  padding: 12px;
  border-top: 1px solid var(--border-color);
  background: var(--bg-primary);
}

.message-input__field {
  display: flex;
  align-items: flex-end;
  gap: 8px;
  padding: 8px 12px;
  border-radius: 8px;
  border: 1px solid var(--border-color);
  background: var(--bg-input);
}

.message-input__field:focus-within {
  border-color: var(--color-primary);
  box-shadow: 0 0 0 2px var(--color-primary-alpha);
}

.message-input__field textarea {
  flex: 1;
  border: none;
  background: transparent;
  resize: none;
  font-family: inherit;
  font-size: 14px;
  line-height: 1.5;
  max-height: 200px;
}

.message-input__field textarea:focus {
  outline: none;
}
```

styles/chat-panel.css

16 Appendix B: Keyboard Shortcuts

Shortcut	Action	Scope
`Cmd/Ctrl + Shift + L`	Toggle chat panel	Global
`Escape`	Close panel / Cancel streaming	Panel
`Enter`	Send message	Input
`Shift + Enter`	New line in message	Input
`Cmd/Ctrl + K`	Clear conversation	Panel
`Cmd/Ctrl + N`	New conversation	Panel
`Up Arrow`	Previous message (when input empty)	Input

Shortcut	Action	Scope
`Cmd/Ctrl + C`	Copy selected code	Code block