

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	3
1.1	Purpose	3
1.2	Scope	3
1.3	Design Goals	3
2	Architecture	3
2.1	High-Level Component Diagram	3
2.2	Module Structure	3
3	TypeScript Type Definitions	4
3.1	Core Message Types	4
3.2	LLM Provider Types	5
3.3	Application Context Types	7
3.4	Store Types	8
4	LLM Service Implementation	9
4.1	Abstract Provider Base	9
4.2	Ollama Provider Implementation	10
4.3	LLM Service Facade	13
5	UI Component Specifications	15
5.1	Panel Layout	16
5.2	Message List Component	17
5.3	Message Input Component	18
5.4	Code Block Rendering	20

6 State Management	22
6.1 Zustand Store Implementation	22
7 Context Building	27
7.1 Context Builder Service	27
8 Configuration	29
8.1 Default Configuration	29
9 Accessibility Requirements	30
9.1 WCAG 2.1 AA Compliance	30
10 Error Handling	31
10.1 Error Categories and Recovery	31
11 Performance Considerations	32
11.1 Optimization Strategies	32
12 Security Considerations	32
12.1 Security Requirements	32
13 Testing Strategy	33
13.1 Test Coverage Requirements	33
14 Implementation Phases	33
14.1 Development Roadmap	33
15 Appendix A: CSS Architecture	34
16 Appendix B: Keyboard Shortcuts	36

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

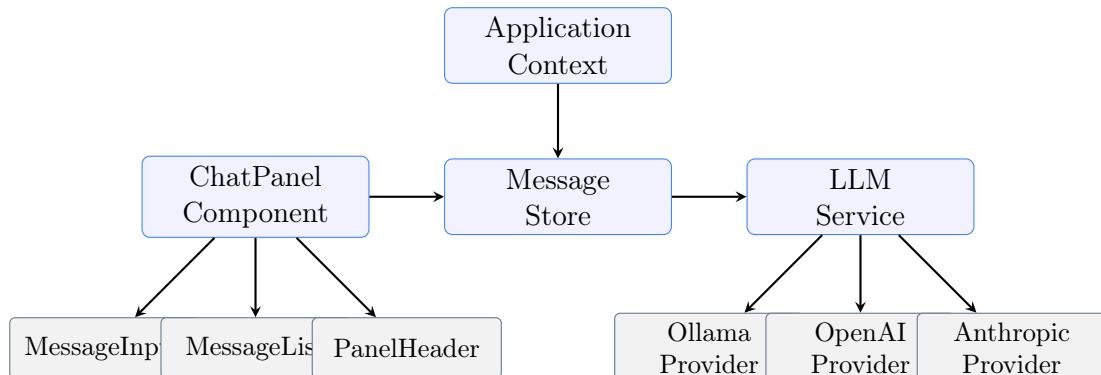


Figure 1: High-level architecture showing component relationships

2.2 Module Structure

```

1  src/
2    chat/
3      components/
4        ChatPanel.tsx          # Main panel container
5        MessageList.tsx        # Scrollable message display
  
```

```

6      MessageItem.tsx          # Individual message rendering
7      MessageInput.tsx        # Input area with actions
8      CodeBlock.tsx          # Syntax-highlighted code
9      PanelHeader.tsx        # Title bar with controls
10     ModelSelector.tsx      # LLM provider/model picker
11     ContextIndicator.tsx   # Shows active context
12     hooks/
13       useChatStore.ts       # Zustand store hook
14       useStreamingResponse.ts # SSE/streaming handler
15       useLLMConnection.ts   # Provider connection state
16       useAutoScroll.ts     # Smart scroll behavior
17     services/
18       LLMService.ts         # Provider abstraction layer
19     providers/
20       OllamaProvider.ts    # Local Ollama integration
21       OpenAIProvider.ts    # OpenAI API provider
22       AnthropicProvider.ts # Anthropic API provider
23       BaseProvider.ts      # Abstract base class
24       ContextBuilder.ts    # Application context aggregator
25       MessageFormatter.ts  # Markdown/code processing
26     store/
27       chatStore.ts          # Conversation state management
28       settingsStore.ts     # User preferences
29     types/
30       index.ts              # Core type definitions
31       messages.ts           # Message-related types
32       providers.ts          # LLM provider types
33       context.ts            # Application context types
34     utils/
35       tokenCounter.ts       # Token estimation utilities
36       codeParser.ts         # Code block extraction
37       streamParser.ts       # SSE stream parsing
38       constants.ts          # Configuration constants
39       index.ts              # Public API exports

```

3 TypeScript Type Definitions

3.1 Core Message Types

```

1 // types/messages.ts
2
3 export type MessageRole = 'user' | 'assistant' | 'system';
4
5 export type MessageStatus =
6   | 'pending'           // Awaiting send
7   | 'streaming'         // Currently receiving
8   | 'complete'          // Finished successfully
9   | 'error'             // Failed with error
10  | 'cancelled';       // User cancelled
11
12 export interface CodeBlock {
13   readonly id: string;
14   readonly language: string;
15   readonly code: string;
16   readonly startLine?: number;
17   readonly filename?: string;

```

```

18 }
19
20 export interface MessageAttachment {
21   readonly id: string;
22   readonly type: 'file' | 'selection' | 'image' | 'context';
23   readonly name: string;
24   readonly content: string;
25   readonly mimeType?: string;
26   readonly metadata?: Record<string, unknown>;
27 }
28
29 export interface ChatMessage {
30   readonly id: string;
31   readonly role: MessageRole;
32   readonly content: string;
33   readonly timestamp: Date;
34   readonly status: MessageStatus;
35   readonly codeBlocks: readonly CodeBlock[];
36   readonly attachments: readonly MessageAttachment[];
37   readonly tokenCount?: number;
38   readonly modelId?: string;
39   readonly error?: ChatError;
40   readonly metadata?: MessageMetadata;
41 }
42
43 export interface MessageMetadata {
44   readonly duration?: number; // Response time in ms
45   readonly promptTokens?: number;
46   readonly completionTokens?: number;
47   readonly totalTokens?: number;
48   readonly finishReason?: string;
49 }
50
51 export interface ChatError {
52   readonly code: string;
53   readonly message: string;
54   readonly recoverable: boolean;
55   readonly details?: unknown;
56 }
57
58 export interface Conversation {
59   readonly id: string;
60   readonly title: string;
61   readonly messages: readonly ChatMessage[];
62   readonly createdAt: Date;
63   readonly updatedAt: Date;
64   readonly modelId: string;
65   readonly systemPrompt?: string;
66   readonly contextConfig: ContextConfig;
67 }

```

3.2 LLM Provider Types

```

1 // types/providers.ts
2
3 export type ProviderType = 'ollama' | 'openai' | 'anthropic' | 'custom';

```

```

4
5 export interface LLMMModel {
6   readonly id: string;
7   readonly name: string;
8   readonly provider: ProviderType;
9   readonly contextWindow: number;
10  readonly maxOutputTokens: number;
11  readonly supportsStreaming: boolean;
12  readonly supportsVision: boolean;
13  readonly costPerInputToken?: number;
14  readonly costPerOutputToken?: number;
15 }
16
17 export interface ProviderConfig {
18   readonly type: ProviderType;
19   readonly enabled: boolean;
20   readonly baseUrl: string;
21   readonly apiKey?: string;
22   readonly defaultModel: string;
23   readonly timeout: number;
24   readonly maxRetries: number;
25   readonly customHeaders?: Record<string, string>;
26 }
27
28 export interface OllamaConfig extends ProviderConfig {
29   readonly type: 'ollama';
30   readonly baseUrl: string; // Default: http://localhost:11434
31   readonly keepAlive?: string;
32   readonly numGpu?: number;
33   readonly numThread?: number;
34 }
35
36 export interface OpenAIConfig extends ProviderConfig {
37   readonly type: 'openai';
38   readonly apiKey: string;
39   readonly organization?: string;
40   readonly baseUrl: string; // Default: https://api.openai.com/v1
41 }
42
43 export interface AnthropicConfig extends ProviderConfig {
44   readonly type: 'anthropic';
45   readonly apiKey: string;
46   readonly baseUrl: string; // Default: https://api.anthropic.com
47   readonly apiVersion: string;
48 }
49
50 export type AnyProviderConfig =
51   | OllamaConfig
52   | OpenAIConfig
53   | AnthropicConfig;
54
55 export interface CompletionRequest {
56   readonly messages: readonly ChatMessage[];
57   readonly model: string;
58   readonly temperature?: number;
59   readonly maxTokens?: number;
60   readonly topP?: number;
61   readonly frequencyPenalty?: number;

```

```

62   readonly presencePenalty?: number;
63   readonly stop?: readonly string[];
64   readonly stream?: boolean;
65   readonly signal?: AbortSignal;
66 }
67
68 export interface CompletionResponse {
69   readonly id: string;
70   readonly content: string;
71   readonly model: string;
72   readonly finishReason: 'stop' | 'length' | 'error';
73   readonly usage: TokenUsage;
74 }
75
76 export interface TokenUsage {
77   readonly promptTokens: number;
78   readonly completionTokens: number;
79   readonly totalTokens: number;
80 }
81
82 export interface StreamChunk {
83   readonly id: string;
84   readonly delta: string;
85   readonly finishReason?: string;
86 }
87
88 export type StreamCallback = (chunk: StreamChunk) => void;
89 export type ErrorCallback = (error: ChatError) => void;

```

3.3 Application Context Types

```

1 // types/context.ts
2
3 export interface SelectionContext {
4   readonly type: 'element' | 'text' | 'code' | 'range';
5   readonly content: string;
6   readonly elementIds?: readonly string[];
7   readonly metadata?: Record<string, unknown>;
8 }
9
10 export interface FileContext {
11   readonly path: string;
12   readonly name: string;
13   readonly content: string;
14   readonly language?: string;
15   readonly selection?: {
16     readonly start: number;
17     readonly end: number;
18   };
19 }
20
21 export interface ApplicationContext {
22   readonly selection: SelectionContext | null;
23   readonly activeFile: FileContext | null;
24   readonly openFiles: readonly FileContext[];
25   readonly projectInfo: ProjectInfo | null;
26   readonly customContext: Record<string, unknown>;

```

```

27 }
28
29 export interface ProjectInfo {
30   readonly name: string;
31   readonly type: string;
32   readonly rootPath: string;
33   readonly dependencies?: Record<string, string>;
34 }
35
36 export interface ContextConfig {
37   readonly includeSelection: boolean;
38   readonly includeActiveFile: boolean;
39   readonly includeOpenFiles: boolean;
40   readonly includeProjectInfo: boolean;
41   readonly maxContextTokens: number;
42   readonly customInstructions?: string;
43 }

```

3.4 Store Types

```

1 // types/store.ts
2
3 export interface ChatState {
4   // Conversations
5   readonly conversations: Map<string, Conversation>;
6   readonly activeConversationId: string | null;
7
8   // UI State
9   readonly isPanelOpen: boolean;
10  readonly isPanelCollapsed: boolean;
11  readonly isLoading: boolean;
12  readonly error: ChatError | null;
13
14  // Provider State
15  readonly activeProvider: ProviderType;
16  readonly activeModel: string;
17  readonly providerStatus: Map<ProviderType, ProviderStatus>;
18
19  // Context
20  readonly contextConfig: ContextConfig;
21  readonly applicationContext: ApplicationContext;
22 }
23
24 export interface ProviderStatus {
25   readonly connected: boolean;
26   readonly lastChecked: Date;
27   readonly availableModels: readonly LLMModel[];
28   readonly error?: string;
29 }
30
31 export interface ChatActions {
32   // Panel
33   togglePanel(): void;
34   setCollapsed(collapsed: boolean): void;
35
36   // Conversations
37   createConversation(title?: string): string;

```

```

38     deleteConversation(id: string): void;
39     setActiveConversation(id: string): void;
40     clearConversation(id: string): void;
41
42     // Messages
43     sendMessage(content: string, attachments?: MessageAttachment[]): Promise<void>;
44     cancelStreaming(): void;
45     retryMessage(messageId: string): Promise<void>;
46     deleteMessage(messageId: string): void;
47
48     // Provider
49     setProvider(provider: ProviderType): void;
50     setModel(modelId: string): void;
51     refreshProviderStatus(provider: ProviderType): Promise<void>;
52
53     // Context
54     updateContextConfig(config: Partial<ContextConfig>): void;
55     refreshApplicationContext(): void;
56 }
57
58 export type ChatStore = ChatState & ChatActions;

```

4 LLM Service Implementation

4.1 Abstract Provider Base

```

1 // services/providers/BaseProvider.ts
2
3 export abstract class BaseProvider {
4   protected config: ProviderConfig;
5   protected abortController: AbortController | null = null;
6
7   constructor(config: ProviderConfig) {
8     this.config = config;
9   }
10
11  abstract get name(): string;
12  abstract get type(): ProviderType;
13
14  abstract connect(): Promise<boolean>;
15  abstract disconnect(): Promise<void>;
16  abstract isConnected(): boolean;
17
18  abstract listModels(): Promise<LLMModel[]>;
19
20  abstract complete(
21    request: CompletionRequest
22  ): Promise<CompletionResponse>;
23
24  abstract stream(
25    request: CompletionRequest,
26    onChunk: StreamCallback,
27    onError: ErrorCallback
28  ): Promise<void>;
29

```

```

30     cancel(): void {
31         if (this.abortController) {
32             this.abortController.abort();
33             this.abortController = null;
34         }
35     }
36
37     protected createAbortController(): AbortController {
38         this.cancel();
39         this.abortController = new AbortController();
40         return this.abortController;
41     }
42
43     protected async fetchWithRetry(
44         url: string,
45         options: RequestInit,
46         retries = this.config.maxRetries
47     ): Promise<Response> {
48         let lastError: Error | null = null;
49
50         for (let i = 0; i <= retries; i++) {
51             try {
52                 const response = await fetch(url, {
53                     ...options,
54                     signal: this.abortController?.signal,
55                 });
56
57                 if (response.ok) return response;
58
59                 if (response.status >= 500 && i < retries) {
60                     await this.delay(Math.pow(2, i) * 1000);
61                     continue;
62                 }
63
64                 throw new Error(`HTTP ${response.status}: ${response.statusText}`);
65             } catch (error) {
66                 lastError = error as Error;
67                 if (error instanceof Error && error.name === 'AbortError') {
68                     throw error;
69                 }
70                 if (i < retries) {
71                     await this.delay(Math.pow(2, i) * 1000);
72                 }
73             }
74         }
75
76         throw lastError;
77     }
78
79     private delay(ms: number): Promise<void> {
80         return new Promise(resolve => setTimeout(resolve, ms));
81     }
82 }

```

4.2 Ollama Provider Implementation

```
1 // services/providers/OllamaProvider.ts
2
3 import { BaseProvider } from './BaseProvider';
4
5 export class OllamaProvider extends BaseProvider {
6     private connected = false;
7
8     get name(): string { return 'Ollama'; }
9     get type(): ProviderType { return 'ollama'; }
10
11    async connect(): Promise<boolean> {
12        try {
13            const response = await fetch(`${this.config.baseUrl}/api/tags`)
14            ;
15            this.connected = response.ok;
16            return this.connected;
17        } catch {
18            this.connected = false;
19            return false;
20        }
21    }
22
23    async disconnect(): Promise<void> {
24        this.connected = false;
25    }
26
27    isConnected(): boolean {
28        return this.connected;
29    }
30
31    async listModels(): Promise<LLMModel[]> {
32        const response = await this.fetchWithRetry(
33            `${this.config.baseUrl}/api/tags`,
34            { method: 'GET' }
35        );
36
37        const data = await response.json();
38
39        return data.models.map((model: any) => ({
40            id: model.name,
41            name: model.name,
42            provider: 'ollama',
43            contextWindow: model.details?.parameter_size || 4096,
44            maxOutputTokens: 4096,
45            supportsStreaming: true,
46            supportsVision: model.details?.families?.includes('clip') ??
47                false,
48        }));
49    }
50
51    async complete(request: CompletionRequest): Promise<
52        CompletionResponse> {
53        const controller = this.createAbortController();
54        const response = await this.fetchWithRetry(
55            `${this.config.baseUrl}/api/chat`,
56            {
57
```

```
55     method: 'POST',
56     headers: { 'Content-Type': 'application/json' },
57     body: JSON.stringify({
58       model: request.model,
59       messages: this.formatMessages(request.messages),
60       stream: false,
61       options: {
62         temperature: request.temperature ?? 0.7,
63         num_predict: request.maxTokens ?? 2048,
64         top_p: request.topP ?? 0.9,
65         stop: request.stop,
66       },
67     }),
68     signal: controller.signal,
69   }
70 );
71
72 const data = await response.json();
73
74 return {
75   id: crypto.randomUUID(),
76   content: data.message.content,
77   model: request.model,
78   finishReason: data.done ? 'stop' : 'length',
79   usage: {
80     promptTokens: data.prompt_eval_count ?? 0,
81     completionTokens: data.eval_count ?? 0,
82     totalTokens: (data.prompt_eval_count ?? 0) + (data.eval_count
83       ?? 0),
84   },
85 };
86
87 async stream(
88   request: CompletionRequest,
89   onChunk: StreamCallback,
90   onError: ErrorCallback
91 ): Promise<void> {
92   const controller = this.createAbortController();
93
94   try {
95     const response = await fetch(`${this.config.baseUrl}/api/chat`,
96       {
97         method: 'POST',
98         headers: { 'Content-Type': 'application/json' },
99         body: JSON.stringify({
100           model: request.model,
101           messages: this.formatMessages(request.messages),
102           stream: true,
103           options: {
104             temperature: request.temperature ?? 0.7,
105             num_predict: request.maxTokens ?? 2048,
106             top_p: request.topP ?? 0.9,
107             stop: request.stop,
108           },
109         },
110       });
111 }
```

```

111     if (!response.ok) {
112         throw new Error(`HTTP ${response.status}`);
113     }
114
115     const reader = response.body?.getReader();
116     if (!reader) throw new Error('No response body');
117
118     const decoder = new TextDecoder();
119     let buffer = '';
120
121     while (true) {
122         const { done, value } = await reader.read();
123         if (done) break;
124
125         buffer += decoder.decode(value, { stream: true });
126         const lines = buffer.split('\n');
127         buffer = lines.pop() ?? '';
128
129         for (const line of lines) {
130             if (!line.trim()) continue;
131
132             try {
133                 const data = JSON.parse(line);
134                 onChunk({
135                     id: crypto.randomUUID(),
136                     delta: data.message?.content ?? '',
137                     finishReason: data.done ? 'stop' : undefined,
138                 });
139             } catch (e) {
140                 // Skip malformed JSON
141             }
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
```

```

3 import { OllamaProvider } from './providers/OllamaProvider';
4 import { OpenAIProvider } from './providers/OpenAIProvider';
5 import { AnthropicProvider } from './providers/AnthropicProvider';
6
7 export class LLMService {
8     private providers: Map<ProviderType, BaseProvider> = new Map();
9     private activeProvider: BaseProvider | null = null;
10
11    constructor(configs: AnyProviderConfig[]) {
12        for (const config of configs) {
13            if (!config.enabled) continue;
14
15            const provider = this.createProvider(config);
16            if (provider) {
17                this.providers.set(config.type, provider);
18            }
19        }
20    }
21
22    private createProvider(config: AnyProviderConfig): BaseProvider | null {
23        switch (config.type) {
24            case 'ollama':
25                return new OllamaProvider(config);
26            case 'openai':
27                return new OpenAIProvider(config);
28            case 'anthropic':
29                return new AnthropicProvider(config);
30            default:
31                return null;
32        }
33    }
34
35    async initialize(): Promise<Map<ProviderType, boolean>> {
36        const results = new Map<ProviderType, boolean>();
37
38        for (const [type, provider] of this.providers) {
39            const connected = await provider.connect();
40            results.set(type, connected);
41        }
42
43        // Set first connected provider as active
44        for (const [type, connected] of results) {
45            if (connected) {
46                this.activeProvider = this.providers.get(type) ?? null;
47                break;
48            }
49        }
50
51        return results;
52    }
53
54    setActiveProvider(type: ProviderType): boolean {
55        const provider = this.providers.get(type);
56        if (provider?.isConnected()) {
57            this.activeProvider = provider;
58            return true;
59        }

```

```
60     return false;
61 }
62
63     getActiveProvider(): BaseProvider | null {
64         return this.activeProvider;
65     }
66
67     async complete(request: CompletionRequest): Promise<
68         CompletionResponse> {
69         if (!this.activeProvider) {
70             throw new Error('No active LLM provider');
71         }
72         return this.activeProvider.complete(request);
73     }
74
75     async stream(
76         request: CompletionRequest,
77         onChunk: StreamCallback,
78         onError: ErrorCallback
79     ): Promise<void> {
80         if (!this.activeProvider) {
81             throw new Error('No active LLM provider');
82         }
83         return this.activeProvider.stream(request, onChunk, onError);
84     }
85
86     cancel(): void {
87         this.activeProvider?.cancel();
88     }
89
90     async listModels(type?: ProviderType): Promise<LLMModel[]> {
91         if (type) {
92             const provider = this.providers.get(type);
93             return provider?.listModels() ?? [];
94         }
95
96         const allModels: LLMModel[] = [];
97         for (const provider of this.providers.values()) {
98             if (provider.isConnected()) {
99                 allModels.push(...await provider.listModels());
100            }
101        }
102        return allModels;
103    }
104 }
```

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

```

1 // components/ChatPanel.tsx
2
3 import { useState, useCallback, useRef } from 'react';
4 import { useChatStore } from '../hooks/useChatStore';
5
6 interface ChatPanelProps {
7   defaultWidth?: number;
8   minWidth?: number;
9   maxWidth?: number;
10  collapsedWidth?: number;
11 }
12
13 export const ChatPanel: React.FC<ChatPanelProps> = ({{
14   defaultWidth = 400,
15   minWidth = 300,
16   maxWidth = 600,
17   collapsedWidth = 48,
18 }) => {
19   const {
20     isPanelOpen,
21     isPanelCollapsed,
22     isLoading,
23     togglePanel,
24     setCollapsed,
25   } = useChatStore();
26
27   const [width,setWidth] = useState(defaultWidth);
28   const panelRef = useRef<HTMLDivElement>(null);
29   const resizing = useRef(false);
30
31   const handleResizeStart = useCallback((e: React.MouseEvent) => {
32     e.preventDefault();
33     resizing.current = true;
34
35     const startX = e.clientX;
36     const startWidth = width;
37
38     const handleMouseMove = (e: MouseEvent) => {
39       if (!resizing.current) return;
40       const delta = startX - e.clientX;
41       const newWidth = Math.min(maxWidth, Math.max(minWidth,
42         startWidth + delta));
43       setWidth(newWidth);
44     };
45
46     const handleMouseUp = () => {

```

```

46     resizing.current = false;
47     document.removeEventListener('mousemove', handleMouseMove);
48     document.removeEventListener('mouseup', handleMouseUp);
49   };
50
51   document.addEventListener('mousemove', handleMouseMove);
52   document.addEventListener('mouseup', handleMouseUp);
53 }, [width, minWidth, maxWidth]);
54
55 if (!isPanelOpen) return null;
56
57 const panelWidth = isPanelCollapsed ? collapsedWidth : width;
58
59 return (
60   <aside
61     ref={panelRef}
62     className="chat-panel"
63     style={{ width: panelWidth }}
64     aria-label="AI Chat Assistant"
65     role="complementary"
66   >
67     {/* Resize Handle */}
68     {!isPanelCollapsed && (
69       <div
70         className="chat-panel__resize-handle"
71         onMouseDown={handleResizeStart}
72         role="separator"
73         aria-orientation="vertical"
74         aria-label="Resize chat panel"
75       />
76     )}
77
78     {/* Panel Content */}
79     {isPanelCollapsed ? (
80       <CollapsedPanel onExpand={() => setCollapsed(false)} />
81     ) : (
82       <>
83         <PanelHeader onCollapse={() => setCollapsed(true)} />
84         <MessageList />
85         <MessageInput disabled={isLoading} />
86       </>
87     )
88   </aside>
89 );
90 };

```

5.2 Message List Component

```

1 // components/MessageList.tsx
2
3 import { useRef, useEffect, useCallback } from 'react';
4 import { useChatStore } from '../hooks/useChatStore';
5 import { useAutoScroll } from '../hooks/useAutoScroll';
6 import { MessageItem } from './MessageItem';
7
8 export const MessageList: React.FC = () => {
9   const { activeConversation } = useChatStore();

```

```

10  const containerRef = useRef<HTMLDivElement>(null);
11  const { scrollToBottom, isAtBottom } = useAutoScroll(containerRef);
12
13  const messages = activeConversation?.messages ?? [];
14
15  // Auto-scroll on new messages when at bottom
16  useEffect(() => {
17    if (isAtBottom) {
18      scrollToBottom();
19    }
20  }, [messages.length, isAtBottom, scrollToBottom]);
21
22  if (messages.length === 0) {
23    return (
24      <div className="message-list message-list--empty">
25        <div className="message-list__placeholder">
26          <IconSparkles size={48} />
27          <h3>Start a conversation</h3>
28          <p>Ask questions, get help with code, or discuss your
29            project.</p>
30        </div>
31      </div>
32    );
33
34  return (
35    <div
36      ref={containerRef}
37      className="message-list"
38      role="log"
39      aria-live="polite"
40      aria-label="Conversation messages"
41    >
42      {messages.map((message, index) => (
43        <MessageItem
44          key={message.id}
45          message={message}
46          isLast={index === messages.length - 1}
47        />
48      )));
49    </div>
50  );
51};

```

5.3 Message Input Component

```

1 // components/MessageInput.tsx
2
3 import { useState, useRef, useCallback, KeyboardEvent } from 'react';
4 import { useChatStore } from '../hooks/useChatStore';
5
6 interface MessageInputProps {
7   disabled?: boolean;
8   maxLength?: number;
9 }
10
11 export const MessageInput: React.FC<MessageInputProps> = ({
```

```

12     disabled = false,
13     maxLength = 32000,
14   }) => {
15     const [input, setInput] = useState('');
16     const [attachments, setAttachments] = useState<MessageAttachment>(
17       [] as []);
18     const textareaRef = useRef<HTMLTextAreaElement>(null);
19
20     const { sendMessage, isLoading, cancelStreaming } = useChatStore();
21
22     const handleSubmit = useCallback(async () => {
23       const trimmed = input.trim();
24       if (!trimmed || disabled || isLoading) return;
25
26       setInput('');
27       setAttachments([]);
28
29       await sendMessage(trimmed, attachments);
30     }, [input, attachments, disabled, isLoading, sendMessage]);
31
32     const handleKeyDown = useCallback((e: KeyboardEvent<
33       HTMLTextAreaElement>) => {
34       if (e.key === 'Enter' && !e.shiftKey) {
35         e.preventDefault();
36         handleSubmit();
37       }
38     }, [handleSubmit]);
39
40     // Auto-resize textarea
41     useEffect(() => {
42       const textarea = textareaRef.current;
43       if (textarea) {
44         textarea.style.height = 'auto';
45         textarea.style.height = `${Math.min(textarea.scrollHeight, 200)}px`;
46       }
47     }, [input]);
48
49     return (
50       <div className="message-input">
51         {/* Attachments Preview */}
52         {attachments.length > 0 && (
53           <AttachmentList
54             attachments={attachments}
55             onRemove={(id) => setAttachments(a => a.filter(x => x.id !== id))}>
56           />
57         )}
58
59         {/* Input Area */}
60         <div className="message-input__field">
61           <textarea
62             ref={textareaRef}
63             value={input}
64             onChange={(e) => setInput(e.target.value)}
65             onKeyDown={handleKeyDown}
66             placeholder="Ask anything..."
67             disabled={disabled}>
68           </textarea>
69         </div>
70       </div>
71     );
72   }
73
74   const [input, setInput] = useState('');
75   const [attachments, setAttachments] = useState<MessageAttachment>(
76     [] as []);
77   const textareaRef = useRef<HTMLTextAreaElement>(null);
78
79   const { sendMessage, isLoading, cancelStreaming } = useChatStore();
80
81   const handleSubmit = useCallback(async () => {
82     const trimmed = input.trim();
83     if (!trimmed || disabled || isLoading) return;
84
85     setInput('');
86     setAttachments([]);
87
88     await sendMessage(trimmed, attachments);
89   }, [input, attachments, disabled, isLoading, sendMessage]);
90
91   const handleKeyDown = useCallback((e: KeyboardEvent<
92     HTMLTextAreaElement>) => {
93     if (e.key === 'Enter' && !e.shiftKey) {
94       e.preventDefault();
95       handleSubmit();
96     }
97   }, [handleSubmit]);
98
99   useEffect(() => {
100     const textarea = textareaRef.current;
101     if (textarea) {
102       textarea.style.height = 'auto';
103       textarea.style.height = `${Math.min(textarea.scrollHeight, 200)}px`;
104     }
105   }, [input]);
106
107   return (
108     <div className="message-input">
109       <div>
110         <AttachmentList
111           attachments={attachments}
112           onRemove={(id) => setAttachments(a => a.filter(x => x.id !== id))}>
113         />
114       </div>
115       <div>
116         <div>
117           <input
118             type="text"
119             value={input}
120             onChange={(e) => setInput(e.target.value)}
121             onKeyDown={handleKeyDown}
122             placeholder="Ask anything...">
123           </input>
124         </div>
125         <div>
126           <button>Send</button>
127         </div>
128       </div>
129     </div>
130   );
131 }
132
133 export default ChatInput;

```

```

66         maxLength={maxLength}
67         rows={1}
68         aria-label="Message input"
69     />
70
71     {/* Actions */}
72     <div className="message-input__actions">
73       <AttachButton onAttach={(a) => setAttachments(prev => [...
74         prev, a])} />
75
76     {isLoading ? (
77       <button
78         onClick={cancelStreaming}
79         className="message-input__cancel"
80         aria-label="Cancel response"
81       >
82         <IconStop size={20} />
83       </button>
84     ) : (
85       <button
86         onClick={handleSubmit}
87         disabled={!input.trim() || disabled}
88         className="message-input__send"
89         aria-label="Send message"
90       >
91         <IconSend size={20} />
92       </button>
93     )}
94   </div>
95 </div>
96
97     {/* Character Count */}
98     <div className="message-input__footer">
99       <span className="message-input__count">
100         {input.length} / {maxLength}
101       </span>
102       <ModelIndicator />
103     </div>
104   );
105 };

```

5.4 Code Block Rendering

```

1 // components/CodeBlock.tsx
2
3 import { useState, useCallback } from 'react';
4 import { highlight, languages } from 'prismjs';
5 import 'prismjs/components/prism-typescript';
6 import 'prismjs/components/prism-javascript';
7 import 'prismjs/components/prism-python';
8 import 'prismjs/components/prism-css';
9 import 'prismjs/components/prism-json';
10
11 interface CodeBlockProps {
12   code: string;
13   language: string;

```

```

14     filename?: string;
15     showLineNumbers?: boolean;
16     onCopy?: () => void;
17     onInsert?: (code: string) => void;
18   }
19
20   export const CodeBlock: React.FC<CodeBlockProps> = ({
21     code,
22     language,
23     filename,
24     showLineNumbers = true,
25     onCopy,
26     onInsert,
27   }) => {
28     const [copied, setCopied] = useState(false);
29
30     const handleCopy = useCallback(async () => {
31       await navigator.clipboard.writeText(code);
32       setCopied(true);
33       onCopy?.();
34       setTimeout(() => setCopied(false), 2000);
35     }, [code, onCopy]);
36
37     const highlightedCode = highlight(
38       code,
39       languages[language] || languages.plaintext,
40       language
41     );
42
43     return (
44       <div className="code-block">
45         {/* Header */}
46         <div className="code-block__header">
47           {filename && (
48             <span className="code-block__filename">
49               <IconFile size={14} />
50               {filename}
51             </span>
52           )}
53           <span className="code-block__language">{language}</span>
54
55           <div className="code-block__actions">
56             {onInsert && (
57               <button
58                 onClick={() => onInsert(code)}
59                 className="code-block__action"
60                 title="Insert at cursor"
61               >
62                 <IconCursorText size={16} />
63               </button>
64             )}
65             <button
66               onClick={handleCopy}
67               className="code-block__action"
68               title={copied ? 'Copied!' : 'Copy code'}
69             >
70               {copied ? <IconCheck size={16} /> : <IconCopy size={16} />}
71             </button>
72           </div>
73         </div>
74       </div>
75     );
76   }
77 }
78
79 
```

```

71         </button>
72     </div>
73   </div>
74
75   /* Code Content */
76   <pre className="code-block__content">
77     <code
78       className={'language-${language}'}
79       dangerouslySetInnerHTML={{ __html: highlightedCode }}
80     />
81   </pre>
82 </div>
83 );
84 };

```

6 State Management

6.1 Zustand Store Implementation

```

1 // store/chatStore.ts
2
3 import { create } from 'zustand';
4 import { devtools, persist } from 'zustand/middleware';
5 import { immer } from 'zustand/middleware/immer';
6
7 export const useChatStore = create<ChatStore>()(
8   devtools(
9     persist(
10       immer((set, get) => ({
11         // Initial State
12         conversations: new Map(),
13         activeConversationId: null,
14         isPanelOpen: false,
15         isPanelCollapsed: false,
16         isLoading: false,
17         error: null,
18         activeProvider: 'ollama',
19         activeModel: 'llama3.1:8b',
20         providerStatus: new Map(),
21         contextConfig: {
22           includeSelection: true,
23           includeActiveFile: true,
24           includeOpenFiles: false,
25           includeProjectInfo: true,
26           maxContextTokens: 8000,
27         },
28         applicationContext: {
29           selection: null,
30           activeFile: null,
31           openFiles: [],
32           projectInfo: null,
33           customContext: {},
34         },
35         // Panel Actions
36         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,
125                    assistantMessage];
126                conv.updatedAt = new Date();
127            }
128            state.isLoading = true;
129            state.error = null;
130        });
131
132        // Stream response
133        try {
134            const llmService = getLLMService();
135            const conv = get().conversations.get(conversationId);
136
137            await llmService.stream(
138            {
139                messages: conv?.messages ?? [],
140                model: activeModel,
141                temperature: 0.7,
142                maxTokens: 4096,
143            },
144            // On chunk
145            (chunk) => {
146                set(state => {
147                    const conv = state.conversations.get(conversationId
148                        !);
149                    if (conv) {
150                        const msgIndex = conv.messages.findIndex(
151                            m => m.id === assistantMessage.id
152                        );

```

```

151         if (msgIndex >= 0) {
152             const msg = conv.messages[msgIndex];
153             conv.messages[msgIndex] = {
154                 ...msg,
155                 content: msg.content + chunk.delta,
156                 status: chunk.finishReason ? 'complete' : 'streaming',
157             };
158         }
159     });
160 },
161 // On error
162 (error) => {
163     set(state => {
164         state.error = error;
165         const conv = state.conversations.get(conversationId
166             !);
167         if (conv) {
168             const msgIndex = conv.messages.findIndex(
169                 m => m.id === assistantMessage.id
170             );
171             if (msgIndex >= 0) {
172                 conv.messages[msgIndex] = {
173                     ...conv.messages[msgIndex],
174                     status: 'error',
175                     error,
176                 };
177             }
178         }
179     });
180 );
181 } finally {
182     set(state => {
183         state.isLoading = false;
184     });
185 }
186 }
187 },
188 cancelStreaming: () => {
189     getLLMService().cancel();
190     set(state => {
191         state.isLoading = false;
192         const convId = state.activeConversationId;
193         if (convId) {
194             const conv = state.conversations.get(convId);
195             if (conv) {
196                 const lastMsg = conv.messages[conv.messages.length -
197                     1];
198                 if (lastMsg?.status === 'streaming') {
199                     conv.messages[conv.messages.length - 1] = {
200                         ...lastMsg,
201                         status: 'cancelled',
202                     };
203                 }
204             }
205         }
206     });
207 }

```

```

206         });
207     },
208
209     retryMessage: async (messageId) => {
210       // Implementation for retry logic
211     },
212
213     deleteMessage: (messageId) => set(state => {
214       const convId = state.activeConversationId;
215       if (convId) {
216         const conv = state.conversations.get(convId);
217         if (conv) {
218           conv.messages = conv.messages.filter(m => m.id !==
219             messageId);
220         }
221       },
222
223     // Provider Actions
224     setProvider: (provider) => set(state => {
225       state.activeProvider = provider;
226     }),
227
228     setModel: (modelId) => set(state => {
229       state.activeModel = modelId;
230     }),
231
232     refreshProviderStatus: async (provider) => {
233       // Implementation for provider status refresh
234     },
235
236     // Context Actions
237     updateContextConfig: (config) => set(state => {
238       state.contextConfig = { ...state.contextConfig, ...config
239         };
240     }),
241
242     refreshApplicationContext: () => {
243       // Hook into application state to get current context
244     },
245   ),
246   {
247     name: 'chat-storage',
248     partialize: (state) => ({
249       conversations: Array.from(state.conversations.entries()),
250       activeProvider: state.activeProvider,
251       activeModel: state.activeModel,
252       contextConfig: state.contextConfig,
253     }),
254   },
255   { name: 'ChatStore' }
256 )
257 );

```

7 Context Building

7.1 Context Builder Service

```

1 // services/ContextBuilder.ts
2
3 export class ContextBuilder {
4     private config: ContextConfig;
5     private tokenCounter: TokenCounter;
6
7     constructor(config: ContextConfig) {
8         this.config = config;
9         this.tokenCounter = new TokenCounter();
10    }
11
12    build(context: ApplicationContext): string {
13        const sections: string[] = [];
14        let totalTokens = 0;
15
16        // System instructions
17        const systemPrompt = this.buildSystemPrompt();
18        sections.push(systemPrompt);
19        totalTokens += this.tokenCounter.count(systemPrompt);
20
21        // Selection context
22        if (this.config.includeSelection && context.selection) {
23            const selectionCtx = this.formatSelection(context.selection);
24            const tokens = this.tokenCounter.count(selectionCtx);
25
26            if (totalTokens + tokens <= this.config.maxContextTokens) {
27                sections.push(selectionCtx);
28                totalTokens += tokens;
29            }
30        }
31
32        // Active file context
33        if (this.config.includeActiveFile && context.activeFile) {
34            const fileCtx = this.formatFile(context.activeFile);
35            const tokens = this.tokenCounter.count(fileCtx);
36
37            if (totalTokens + tokens <= this.config.maxContextTokens) {
38                sections.push(fileCtx);
39                totalTokens += tokens;
40            }
41        }
42
43        // Project info
44        if (this.config.includeProjectInfo && context.projectInfo) {
45            const projectCtx = this.formatProject(context.projectInfo);
46            const tokens = this.tokenCounter.count(projectCtx);
47
48            if (totalTokens + tokens <= this.config.maxContextTokens) {
49                sections.push(projectCtx);
50                totalTokens += tokens;
51            }
52        }
53
54        // Custom instructions

```

```

55     if (this.config.customInstructions) {
56       sections.push(`\n## Custom Instructions\n${this.config.
57         customInstructions}`);
58     }
59   return sections.join('\n\n');
60 }
61
62 private buildSystemPrompt(): string {
63   return 'You are an AI assistant integrated into a design/
64   development application.
65 Your role is to help users with:
66 - Understanding and modifying their designs
67 - Writing and debugging code
68 - Explaining concepts and best practices
69 - Answering questions about the application
70 Always be concise and helpful. When providing code, use appropriate
71 syntax highlighting.
72 If you're unsure about something, ask for clarification.';
73 }
74
75 private formatSelection(selection: SelectionContext): string {
76   return `## Current Selection
77 Type: ${selection.type}
78 ${selection.elementIds ? 'Elements: ${selection.elementIds.join(', )}' : ''}
79 \\
80 ${selection.content}
81 \\
82 }
83
84 private formatFile(file: FileContext): string {
85   const lang = file.language ?? this.detectLanguage(file.name);
86   let content = file.content;
87
88   // Highlight selection if present
89   if (file.selection) {
90     content = this.highlightSelection(content, file.selection);
91   }
92
93   return `## Active File: ${file.name}
94 Path: ${file.path}
95 \\
96 ${lang}
97 ${content}
98 \\
99 }
100
101 private formatProject(project: ProjectInfo): string {
102   return `## Project Information
103 Name: ${project.name}
104 Type: ${project.type}
105 Path: ${project.rootPath}`;
106 }
107
108 private detectLanguage(filename: string): string {

```

```

109     const ext = filename.split('.').pop()?.toLowerCase();
110     const langMap: Record<string, string> = {
111         ts: 'typescript',
112         tsx: 'typescript',
113         js: 'javascript',
114         jsx: 'javascript',
115         py: 'python',
116         css: 'css',
117         json: 'json',
118         html: 'html',
119     };
120     return langMap[ext ?? ''] ?? 'plaintext';
121 }
122
123 private highlightSelection(
124     content: string,
125     selection: { start: number; end: number }
126 ): string {
127     const before = content.slice(0, selection.start);
128     const selected = content.slice(selection.start, selection.end);
129     const after = content.slice(selection.end);
130     return `${before}/* >> SELECTED START >> */${selected}/* <<<
131             SELECTED END <<< */${after}`;
132 }

```

8 Configuration

8.1 Default Configuration

```

1 // constants.ts
2
3 export const DEFAULT_PROVIDERS: AnyProviderConfig[] = [
4     {
5         type: 'ollama',
6         enabled: true,
7         baseUrl: 'http://localhost:11434',
8         defaultModel: 'llama3.1:8b',
9         timeout: 120000,
10        maxRetries: 3,
11    },
12    {
13        type: 'openai',
14        enabled: false,
15        baseUrl: 'https://api.openai.com/v1',
16        apiKey: '',
17        defaultModel: 'gpt-4-turbo',
18        timeout: 60000,
19        maxRetries: 3,
20    },
21    {
22        type: 'anthropic',
23        enabled: false,
24        baseUrl: 'https://api.anthropic.com',
25        apiKey: '',
26        apiVersion: '2024-01-01',

```

```

27     defaultModel: 'claude-3-sonnet-20240229',
28     timeout: 60000,
29     maxRetries: 3,
30   },
31 ];
32
33 export const PANEL_CONFIG = {
34   defaultWidth: 400,
35   minWidth: 300,
36   maxWidth: 600,
37   collapsedWidth: 48,
38   autoCollapseBreakpoint: 1200,
39 } as const;
40
41 export const MESSAGE_CONFIG = {
42   maxLength: 32000,
43   maxAttachments: 10,
44   maxAttachmentSize: 10 * 1024 * 1024, // 10MB
45 } as const;
46
47 export const CONTEXT_CONFIG: ContextConfig = {
48   includeSelection: true,
49   includeActiveFile: true,
50   includeOpenFiles: false,
51   includeProjectInfo: true,
52   maxContextTokens: 8000,
53 };
54
55 export const SUPPORTED_CODE_LANGUAGES = [
56   'typescript',
57   'javascript',
58   'python',
59   'css',
60   'html',
61   'json',
62   'markdown',
63   'yaml',
64   'sql',
65   'bash',
66   'rust',
67   'go',
68 ] 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

```

1 // utils/errorHandler.ts
2
3 export function handleChatError(error: ChatError): ErrorRecovery {
4   const strategies: Record<string, ErrorRecovery> = {
5     PROVIDER_OFFLINE: {
6       action: 'retry',
7       message: 'Connection lost. Retrying...',
8       retryDelay: 2000,
9       maxRetries: 3,
10      fallback: () => tryAlternativeProvider(),
11    },
12    RATE_LIMITED: {
13      action: 'wait',
14      message: 'Rate limited. Please wait...',
15      retryDelay: 60000,

```

```

16     maxRetries: 1,
17   },
18   CONTEXT_TOO_LARGE: {
19     action: 'modify',
20     message: 'Context too large. Reducing...',
21     modification: () => truncateContext(),
22   },
23   STREAM_INTERRUPTED: {
24     action: 'prompt',
25     message: 'Response interrupted. Retry?',
26     userAction: true,
27   },
28 };
29
30 return strategies[error.code] ?? {
31   action: 'display',
32   message: error.message,
33 };
34 }

```

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

```

1  /* Chat Panel Container */
2  .chat-panel {
3    position: fixed;
4    right: 0;
5    top: var(--header-height, 48px);
6    bottom: 0;
7    display: flex;
8    flex-direction: column;
9    background: var(--bg-primary);
10   border-left: 1px solid var(--border-color);
11   z-index: 100;
12   transition: width 0.2s ease-out;
13 }
14
15 @media (prefers-reduced-motion: reduce) {
16   .chat-panel {
17     transition: none;
18   }
19 }
20
21 /* Resize Handle */
22 .chat-panel__resize-handle {
23   position: absolute;
24   left: 0;
25   top: 0;
26   bottom: 0;
27   width: 4px;
28   cursor: ew-resize;
29   background: transparent;
30   transition: background 0.15s;
31 }
32
33 .chat-panel__resize-handle:hover,
34 .chat-panel__resize-handle:active {
35   background: var(--color-primary);
36 }
37
38 /* Message List */
39 .message-list {

```

```
40     flex: 1;
41     overflow-y: auto;
42     padding: 16px;
43     scroll-behavior: smooth;
44   }
45
46 @media (prefers-reduced-motion: reduce) {
47   .message-list {
48     scroll-behavior: auto;
49   }
50 }
51
52 .message-list--empty {
53   display: flex;
54   align-items: center;
55   justify-content: center;
56 }
57
58 /* Message Item */
59 .message-item {
60   margin-bottom: 16px;
61   padding: 12px;
62   border-radius: 8px;
63 }
64
65 .message-item--user {
66   background: var(--bg-user-message);
67   margin-left: 24px;
68 }
69
70 .message-item--assistant {
71   background: var(--bg-assistant-message);
72   margin-right: 24px;
73 }
74
75 /* Code Block */
76 .code-block {
77   margin: 8px 0;
78   border-radius: 6px;
79   overflow: hidden;
80   border: 1px solid var(--border-color);
81 }
82
83 .code-block__header {
84   display: flex;
85   align-items: center;
86   justify-content: space-between;
87   padding: 8px 12px;
88   background: var(--bg-secondary);
89   border-bottom: 1px solid var(--border-color);
90   font-size: 12px;
91 }
92
93 .code-block__content {
94   margin: 0;
95   padding: 12px;
96   overflow-x: auto;
97   font-family: var(--font-mono);
```

```

98     font-size: 13px;
99     line-height: 1.5;
100    }
101
102   /* Message Input */
103   .message-input {
104     padding: 12px;
105     border-top: 1px solid var(--border-color);
106     background: var(--bg-primary);
107   }
108
109  .message-input__field {
110    display: flex;
111    align-items: flex-end;
112    gap: 8px;
113    padding: 8px 12px;
114    border-radius: 8px;
115    border: 1px solid var(--border-color);
116    background: var(--bg-input);
117  }
118
119  .message-input__field:focus-within {
120    border-color: var(--color-primary);
121    box-shadow: 0 0 0 2px var(--color-primary-alpha);
122  }
123
124  .message-input__field textarea {
125    flex: 1;
126    border: none;
127    background: transparent;
128    resize: none;
129    font-family: inherit;
130    font-size: 14px;
131    line-height: 1.5;
132    max-height: 200px;
133  }
134
135  .message-input__field textarea:focus {
136    outline: none;
137  }

```

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