

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	2
1.1	Purpose	2
1.2	Scope	2
1.3	Design Goals	2
2	Architecture	2
2.1	High-Level Component Diagram	2
2.2	Module Structure	2
3	TypeScript Type Definitions	3
3.1	Core Message Types	3
3.2	LLM Provider Types	4
3.3	Application Context Types	6
3.4	Store Types	7
4	LLM Service Implementation	8
4.1	Abstract Provider Base	8
4.2	Ollama Provider Implementation	9
4.3	LLM Service Facade	12
5	UI Component Specifications	14
5.1	Panel Layout	15
5.2	Message List Component	16
5.3	Message Input Component	17
5.4	Code Block Rendering	19

6 State Management	21
6.1 Zustand Store Implementation	21
7 Context Building	26
7.1 Context Builder Service	26
8 Configuration	28
8.1 Default Configuration	28
9 Accessibility Requirements	29
9.1 WCAG 2.1 AA Compliance	29
10 Error Handling	30
10.1 Error Categories and Recovery	30
11 Performance Considerations	31
11.1 Optimization Strategies	31
12 Security Considerations	31
12.1 Security Requirements	31
13 Testing Strategy	32
13.1 Test Coverage Requirements	32
14 Implementation Phases	32
14.1 Development Roadmap	32
15 Appendix A: CSS Architecture	33
16 Appendix B: Keyboard Shortcuts	35

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

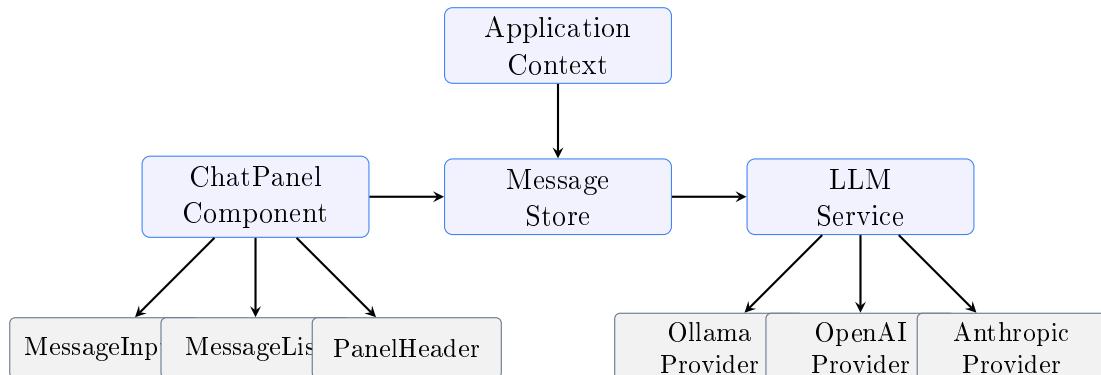


Figure 1: High-level architecture showing component relationships

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.
65                     statusText}`);
66             } catch (error) {
67                 lastError = error as Error;
68                 if (error instanceof Error && error.name === 'AbortError') {
69                     throw error;
70                 }
71                 if (i < retries) {
72                     await this.delay(Math.pow(2, i) * 1000);
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
55        const response = await this.fetchWithRetry(
56            `${this.config.baseUrl}/api/chat`,
57            {
58
```

```
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         signal: controller.signal,
111       });
112 }
```

```

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
```

```

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 }
```

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 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   <div>
75     <h1>Welcome to Chat Window</h1>
76     <p>This is a simple interface for interacting with a Large Language Model (LLM). You can type messages in the input field and send them to the LLM. The LLM will respond with its output, which you can view in the message list below.</p>
77     <p>To get started, click the "Ask anything..." button or type a message and press Enter. You can also click on a message in the list to view its details or delete it if needed.</p>
78   </div>
79
80   <div>
81     <h2>Recent Messages</h2>
82     <ul>
83       {messages.map((message) => (
84         <li>
85           <div>
86             <div>{message.text}</div>
87             <div>{message.createdAt}</div>
88           </div>
89           <button onClick={() => handleDeleteMessage(message.id)}>Delete</button>
90         </li>
91       ))}
92     </ul>
93   </div>
94
95   <div>
96     <h2>Ask anything...</h2>
97     <input type="text" value={input} onChange={handleInputChange} />
98     <button onClick={handleSend}>Send</button>
99   </div>
100
101 </div>
102
103 <script>
104   window.addEventListener('load', () => {
105     const inputField = document.querySelector('.message-input__field');
106     if (inputField) {
107       inputField.focus();
108     }
109   });
110 </script>
111
112 <style>
113   .message-list {
114     list-style-type: none;
115     padding-left: 0;
116   }
117   .message-list li {
118     margin-bottom: 10px;
119   }
120   .message-list li:last-child {
121     margin-bottom: 0;
122   }
123   .message-list li::before {
124     content: '';
125     width: 10px;
126     height: 10px;
127     border-radius: 50%;
128     background-color: #ccc;
129     display: inline-block;
130     margin-right: 10px;
131   }
132   .message-list li.message-user::before {
133     background-color: #f0f0f0;
134   }
135   .message-list li.message-user::after {
136     content: 'User';
137     position: absolute;
138     top: -10px;
139     left: 50px;
140     font-size: small;
141   }
142   .message-list li.message-llm::after {
143     content: 'LLM';
144     position: absolute;
145     top: -10px;
146     left: 50px;
147     font-size: small;
148   }
149   .message-list li.message-user::after {
150     content: 'User';
151     position: absolute;
152     top: -10px;
153     left: 50px;
154     font-size: small;
155   }
156   .message-list li.message-llm::after {
157     content: 'LLM';
158     position: absolute;
159     top: -10px;
160     left: 50px;
161     font-size: small;
162   }
163   .message-list li.message-user::after {
164     content: 'User';
165     position: absolute;
166     top: -10px;
167     left: 50px;
168     font-size: small;
169   }
170   .message-list li.message-llm::after {
171     content: 'LLM';
172     position: absolute;
173     top: -10px;
174     left: 50px;
175     font-size: small;
176   }
177   .message-list li.message-user::after {
178     content: 'User';
179     position: absolute;
180     top: -10px;
181     left: 50px;
182     font-size: small;
183   }
184   .message-list li.message-llm::after {
185     content: 'LLM';
186     position: absolute;
187     top: -10px;
188     left: 50px;
189     font-size: small;
190   }
191   .message-list li.message-user::after {
192     content: 'User';
193     position: absolute;
194     top: -10px;
195     left: 50px;
196     font-size: small;
197   }
198   .message-list li.message-llm::after {
199     content: 'LLM';
200     position: absolute;
201     top: -10px;
202     left: 50px;
203     font-size: small;
204   }
205   .message-list li.message-user::after {
206     content: 'User';
207     position: absolute;
208     top: -10px;
209     left: 50px;
210     font-size: small;
211   }
212   .message-list li.message-llm::after {
213     content: 'LLM';
214     position: absolute;
215     top: -10px;
216     left: 50px;
217     font-size: small;
218   }
219   .message-list li.message-user::after {
220     content: 'User';
221     position: absolute;
222     top: -10px;
223     left: 50px;
224     font-size: small;
225   }
226   .message-list li.message-llm::after {
227     content: 'LLM';
228     position: absolute;
229     top: -10px;
230     left: 50px;
231     font-size: small;
232   }
233   .message-list li.message-user::after {
234     content: 'User';
235     position: absolute;
236     top: -10px;
237     left: 50px;
238     font-size: small;
239   }
240   .message-list li.message-llm::after {
241     content: 'LLM';
242     position: absolute;
243     top: -10px;
244     left: 50px;
245     font-size: small;
246   }
247   .message-list li.message-user::after {
248     content: 'User';
249     position: absolute;
250     top: -10px;
251     left: 50px;
252     font-size: small;
253   }
254   .message-list li.message-llm::after {
255     content: 'LLM';
256     position: absolute;
257     top: -10px;
258     left: 50px;
259     font-size: small;
260   }
261   .message-list li.message-user::after {
262     content: 'User';
263     position: absolute;
264     top: -10px;
265     left: 50px;
266     font-size: small;
267   }
268   .message-list li.message-llm::after {
269     content: 'LLM';
270     position: absolute;
271     top: -10px;
272     left: 50px;
273     font-size: small;
274   }
275   .message-list li.message-user::after {
276     content: 'User';
277     position: absolute;
278     top: -10px;
279     left: 50px;
280     font-size: small;
281   }
282   .message-list li.message-llm::after {
283     content: 'LLM';
284     position: absolute;
285     top: -10px;
286     left: 50px;
287     font-size: small;
288   }
289   .message-list li.message-user::after {
290     content: 'User';
291     position: absolute;
292     top: -10px;
293     left: 50px;
294     font-size: small;
295   }
296   .message-list li.message-llm::after {
297     content: 'LLM';
298     position: absolute;
299     top: -10px;
300     left: 50px;
301     font-size: small;
302   }
303   .message-list li.message-user::after {
304     content: 'User';
305     position: absolute;
306     top: -10px;
307     left: 50px;
308     font-size: small;
309   }
310   .message-list li.message-llm::after {
311     content: 'LLM';
312     position: absolute;
313     top: -10px;
314     left: 50px;
315     font-size: small;
316   }
317   .message-list li.message-user::after {
318     content: 'User';
319     position: absolute;
320     top: -10px;
321     left: 50px;
322     font-size: small;
323   }
324   .message-list li.message-llm::after {
325     content: 'LLM';
326     position: absolute;
327     top: -10px;
328     left: 50px;
329     font-size: small;
330   }
331   .message-list li.message-user::after {
332     content: 'User';
333     position: absolute;
334     top: -10px;
335     left: 50px;
336     font-size: small;
337   }
338   .message-list li.message-llm::after {
339     content: 'LLM';
340     position: absolute;
341     top: -10px;
342     left: 50px;
343     font-size: small;
344   }
345   .message-list li.message-user::after {
346     content: 'User';
347     position: absolute;
348     top: -10px;
349     left: 50px;
350     font-size: small;
351   }
352   .message-list li.message-llm::after {
353     content: 'LLM';
354     position: absolute;
355     top: -10px;
356     left: 50px;
357     font-size: small;
358   }
359   .message-list li.message-user::after {
360     content: 'User';
361     position: absolute;
362     top: -10px;
363     left: 50px;
364     font-size: small;
365   }
366   .message-list li.message-llm::after {
367     content: 'LLM';
368     position: absolute;
369     top: -10px;
370     left: 50px;
371     font-size: small;
372   }
373   .message-list li.message-user::after {
374     content: 'User';
375     position: absolute;
376     top: -10px;
377     left: 50px;
378     font-size: small;
379   }
380   .message-list li.message-llm::after {
381     content: 'LLM';
382     position: absolute;
383     top: -10px;
384     left: 50px;
385     font-size: small;
386   }
387   .message-list li.message-user::after {
388     content: 'User';
389     position: absolute;
390     top: -10px;
391     left: 50px;
392     font-size: small;
393   }
394   .message-list li.message-llm::after {
395     content: 'LLM';
396     position: absolute;
397     top: -10px;
398     left: 50px;
399     font-size: small;
400   }
401   .message-list li.message-user::after {
402     content: 'User';
403     position: absolute;
404     top: -10px;
405     left: 50px;
406     font-size: small;
407   }
408   .message-list li.message-llm::after {
409     content: 'LLM';
410     position: absolute;
411     top: -10px;
412     left: 50px;
413     font-size: small;
414   }
415   .message-list li.message-user::after {
416     content: 'User';
417     position: absolute;
418     top: -10px;
419     left: 50px;
420     font-size: small;
421   }
422   .message-list li.message-llm::after {
423     content: 'LLM';
424     position: absolute;
425     top: -10px;
426     left: 50px;
427     font-size: small;
428   }
429   .message-list li.message-user::after {
430     content: 'User';
431     position: absolute;
432     top: -10px;
433     left: 50px;
434     font-size: small;
435   }
436   .message-list li.message-llm::after {
437     content: 'LLM';
438     position: absolute;
439     top: -10px;
440     left: 50px;
441     font-size: small;
442   }
443   .message-list li.message-user::after {
444     content: 'User';
445     position: absolute;
446     top: -10px;
447     left: 50px;
448     font-size: small;
449   }
450   .message-list li.message-llm::after {
451     content: 'LLM';
452     position: absolute;
453     top: -10px;
454     left: 50px;
455     font-size: small;
456   }
457   .message-list li.message-user::after {
458     content: 'User';
459     position: absolute;
460     top: -10px;
461     left: 50px;
462     font-size: small;
463   }
464   .message-list li.message-llm::after {
465     content: 'LLM';
466     position: absolute;
467     top: -10px;
468     left: 50px;
469     font-size: small;
470   }
471   .message-list li.message-user::after {
472     content: 'User';
473     position: absolute;
474     top: -10px;
475     left: 50px;
476     font-size: small;
477   }
478   .message-list li.message-llm::after {
479     content: 'LLM';
480     position: absolute;
481     top: -10px;
482     left: 50px;
483     font-size: small;
484   }
485   .message-list li.message-user::after {
486     content: 'User';
487     position: absolute;
488     top: -10px;
489     left: 50px;
490     font-size: small;
491   }
492   .message-list li.message-llm::after {
493     content: 'LLM';
494     position: absolute;
495     top: -10px;
496     left: 50px;
497     font-size: small;
498   }
499   .message-list li.message-user::after {
500     content: 'User';
501     position: absolute;
502     top: -10px;
503     left: 50px;
504     font-size: small;
505   }
506   .message-list li.message-llm::after {
507     content: 'LLM';
508     position: absolute;
509     top: -10px;
510     left: 50px;
511     font-size: small;
512   }
513   .message-list li.message-user::after {
514     content: 'User';
515     position: absolute;
516     top: -10px;
517     left: 50px;
518     font-size: small;
519   }
520   .message-list li.message-llm::after {
521     content: 'LLM';
522     position: absolute;
523     top: -10px;
524     left: 50px;
525     font-size: small;
526   }
527   .message-list li.message-user::after {
528     content: 'User';
529     position: absolute;
530     top: -10px;
531     left: 50px;
532     font-size: small;
533   }
534   .message-list li.message-llm::after {
535     content: 'LLM';
536     position: absolute;
537     top: -10px;
538     left: 50px;
539     font-size: small;
540   }
541   .message-list li.message-user::after {
542     content: 'User';
543     position: absolute;
544     top: -10px;
545     left: 50px;
546     font-size: small;
547   }
548   .message-list li.message-llm::after {
549     content: 'LLM';
550     position: absolute;
551     top: -10px;
552     left: 50px;
553     font-size: small;
554   }
555   .message-list li.message-user::after {
556     content: 'User';
557     position: absolute;
558     top: -10px;
559     left: 50px;
560     font-size: small;
561   }
562   .message-list li.message-llm::after {
563     content: 'LLM';
564     position: absolute;
565     top: -10px;
566     left: 50px;
567     font-size: small;
568   }
569   .message-list li.message-user::after {
570     content: 'User';
571     position: absolute;
572     top: -10px;
573     left: 50px;
574     font-size: small;
575   }
576   .message-list li.message-llm::after {
577     content: 'LLM';
578     position: absolute;
579     top: -10px;
580     left: 50px;
581     font-size: small;
582   }
583   .message-list li.message-user::after {
584     content: 'User';
585     position: absolute;
586     top: -10px;
587     left: 50px;
588     font-size: small;
589   }
590   .message-list li.message-llm::after {
591     content: 'LLM';
592     position: absolute;
593     top: -10px;
594     left: 50px;
595     font-size: small;
596   }
597   .message-list li.message-user::after {
598     content: 'User';
599     position: absolute;
600     top: -10px;
601     left: 50px;
602     font-size: small;
603   }
604   .message-list li.message-llm::after {
605     content: 'LLM';
606     position: absolute;
607     top: -10px;
608     left: 50px;
609     font-size: small;
610   }
611   .message-list li.message-user::after {
612     content: 'User';
613     position: absolute;
614     top: -10px;
615     left: 50px;
616     font-size: small;
617   }
618   .message-list li.message-llm::after {
619     content: 'LLM';
620     position: absolute;
621     top: -10px;
622     left: 50px;
623     font-size: small;
624   }
625   .message-list li.message-user::after {
626     content: 'User';
627     position: absolute;
628     top: -10px;
629     left: 50px;
630     font-size: small;
631   }
632   .message-list li.message-llm::after {
633     content: 'LLM';
634     position: absolute;
635     top: -10px;
636     left: 50px;
637     font-size: small;
638   }
639   .message-list li.message-user::after {
640     content: 'User';
641     position: absolute;
642     top: -10px;
643     left: 50px;
644     font-size: small;
645   }
646   .message-list li.message-llm::after {
647     content: 'LLM';
648     position: absolute;
649     top: -10px;
650     left: 50px;
651     font-size: small;
652   }
653   .message-list li.message-user::after {
654     content: 'User';
655     position: absolute;
656     top: -10px;
657     left: 50px;
658     font-size: small;
659   }
660   .message-list li.message-llm::after {
661     content: 'LLM';
662     position: absolute;
663     top: -10px;
664     left: 50px;
665     font-size: small;
666   }
667   .message-list li.message-user::after {
668     content: 'User';
669     position: absolute;
670     top: -10px;
671     left: 50px;
672     font-size: small;
673   }
674   .message-list li.message-llm::after {
675     content: 'LLM';
676     position: absolute;
677     top: -10px;
678     left: 50px;
679     font-size: small;
680   }
681   .message-list li.message-user::after {
682     content: 'User';
683     position: absolute;
684     top: -10px;
685     left: 50px;
686     font-size: small;
687   }
688   .message-list li.message-llm::after {
689     content: 'LLM';
690     position: absolute;
691     top: -10px;
692     left: 50px;
693     font-size: small;
694   }
695   .message-list li.message-user::after {
696     content: 'User';
697     position: absolute;
698     top: -10px;
699     left: 50px;
700     font-size: small;
701   }
65
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
156
161
166
171
176
181
186
191
196
201
206
211
216
221
226
231
236
241
246
251
256
261
266
271
276
281
286
291
296
301
306
311
316
321
326
331
336
341
346
351
356
361
366
371
376
381
386
391
396
401
406
411
416
421
426
431
436
441
446
451
456
461
466
471
476
481
486
491
496
501
506
511
516
521
526
531
536
541
546
551
556
561
566
571
576
581
586
591
596
601
606
611
616
621
626
631
636
641
646
651
656
661
666
671
676
681
686
691
696
701
706
711
716
721
726
731
736
741
746
751
756
761
766
771
776
781
786
791
796
801
806
811
816
821
826
831
836
841
846
851
856
861
866
871
876
881
886
891
896
901
906
911
916
921
926
931
936
941
946
951
956
961
966
971
976
981
986
991
996
1001
1006
1011
1016
1021
1026
1031
1036
1041
1046
1051
1056
1061
1066
1071
1076
1081
1086
1091
1096
1101
1106
1111
1116
1121
1126
1131
1136
1141
1146
1151
1156
1161
1166
1171
1176
1181
1186
1191
1196
1201
1206
1211
1216
1221
1226
1231
1236
1241
1246
1251
1256
1261
1266
1271
1276
1281
1286
1291
1296
1301
1306
1311
1316
1321
1326
1331
1336
1341
1346
1351
1356
1361
1366
1371
1376
1381
1386
1391
1396
1401
1406
1411
1416
1421
1426
1431
1436
1441
1446
1451
1456
1461
1466
1471
1476
1481
1486
1491
1496
1501
1506
1511
1516
1521
1526
1531
1536
1541
1546
1551
1556
1561
1566
1571
1576
1581
1586
1591
1596
1601
1606
1611
1616
1621
1626
1631
1636
1641
1646
1651
1656
1661
1666
1671
1676
1681
1686
1691
1696
1701
1706
1711
1716
1721
1726
1731
1736
1741
1746
1751
1756
1761
1766
1771
1776
1781
1786
1791
1796
1801
1806
1811
1816
1821
1826
1831
1836
1841
1846
1851
1856
1861
1866
1871
1876
1881
1886
1891
1896
1901
1906
1911
1916
1921
1926
1931
1936
1941
1946
1951
1956
1961
1966
1971
1976
1981
1986
1991
1996
2001
2006
2011
2016
2021
2026
2031
2036
2041
2046
2051
2056
2061
2066
2071
2076
2081
2086
2091
2096
2101
2106
2111
2116
2121
2126
2131
2136
2141
2146
2151
2156
2161
2166
2171
2176
2181
2186
2191
2196
2201
2206
2211
2216
2221
2226
2231
2236
2241
2246
2251
2256
2261
2266
2271
2276
2281
2286
2291
2296
2301
2306
2311
2316
2321
2326
2331
2336
2341
2346
2351
2356
2361
2366
2371
2376
2381
2386
2391
2396
2401
2406
2411
2416
2421
2426
2431
2436
2441
2446
2451
2456
2461
2466
2471
2476
2481
2486
2491
2496
2501
2506
2511
2516
2521
2526
2531
2536
2541
2546
2551
2556
2561
2566
2571
2576
2581
2586
2591
2596
2601
2606
2611
2616
2621
2626
2631
2636
2641
2646
2651
2656
2661
2666
2671
2676
2681
2686
2691
2696
2701
2706
2711
2716
2721
2726
2731
2736
2741
2746
2751
2756
2761
2766
2771
2776
2781
2786
2791
2796
2801
2806
2811
2816
2821
2826
2831
2836
2841
2846
2851
2856
2861
2866
2871
2876
2881
2886
2891
2896
2901
2906
2911
2916
2921
2926
2931
2936
2941
2946
2951
2956
2961
2966
2971
2976
2981
2986
2991
2996
3001
3006
3011
3016
3021
3026
3031
3036
3041
3046
3051
3056
3061
3066
3071
3076
3081
3086
3091
3096
3101
3106
3111
3116
3121
3126
3131
3136
3141
3146
3151
3156
3161
3166
3171
3176
3181
3186
3191
3196
3201
3206
3211
3216
3221
3226
3231
3236
3241
3246
3251
3256
3261
3266
3271
3276
3281
3286
3291
3296
3301
3306
3311
3316
3321
3326
3331
3336
3341
3346
3351
3356
3361
3366
3371
3376
3381
3386
3391
3396
3401
3406
3411
3416
3421
3426
3431
3436
3441
3446
3451
3456
3461
3466
3471
3476
3481
3486
3491
3496
3501
3506
3511
3516
3521
3526
3531
3536
3541
3546
3551
3556
3561
3566
3571
3576
3581
3586
3591
3596
3601
3606
3611
3616
3621
3626
3631
3636
3641
3646
3651
3656
3661
3666
3671
3676
3681
3686
3691
3696
3701
3706
3711
3716
3721
3726
3731
3736
3741
3746
3751
3756
3761
3766
3771
3776
3781
3786
3791
3796
3801
3806
3811
3816
3821
3826
3831
3836
3841
3846
3851
3856
3861
3866
3871
3876
3881
3886
3891
3896
3901
3906
3911
3916
3921
3926
3931
3936
3941
3946
3951
3956
3961
3966
3971
3976
3981
3986
3991
3996
4001
4006
4011
4016
4021
4026
4031
4036
4041
4046
4051
4056
4061
4066
4071
4076
4081
4086
4091
4096
4101
4106
4111
4116
4121
4126
4131
4136
4141
4146
4151
4156
4161
4166
4171
4176
4181
4186
4191
4196
4201
4206
4211
4216
4221
4226
4231
4236
4241
4246
4251
4256
4261
4266
4271
4276
4281
4286
4291
4296
4301
4306
4311
4316
4321
4326
4331
4336
4341
4346
4351
4356
4361
4366
4371
4376
4381
4386
4391
4396
4401
4406
4411
4416
4421
4426
4431
4436
4441
4446
4451
4456
4461
4466
4471
4476
4481
4486
4491
4496
4501
4506
4511
4516
4521
4526
4531
4536
4541
4546
4551
4556
4561
4566
4571
4576
4581
4586

```

```

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   const [language, setLanguage] = useState('text');
79
80   const handleChangeLanguage = (language: string) => {
81     setLanguage(language);
82     highlightedCode();
83   }
84
85   return (
86     <div>
87       <div>
88         <h3>Code Block</h3>
89         <div>
90           <input type="text" value={code} />
91           <button onClick={handleCopy}>Copy</button>
92         </div>
93         <div>
94           <label>Language:</label>
95           <select value={language} onChange={handleChangeLanguage}>
96             <option value="text">Text</option>
97             <option value="javascript">JavaScript</option>
98             <option value="python">Python</option>
99             <option value="java">Java</option>
100            <option value="csharp">C#</option>
101          </select>
102        </div>
103      </div>
104      <div>
105        <pre>{highlightedCode()}</pre>
106      </div>
107    </div>
108  );
109
110  <div>
111    <h3>Output</h3>
112    <div>
113      <pre>{output}</pre>
114    </div>
115  </div>
116
```

```

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                        );
153
154                    }
155                });
156            }
157        );
158    }
159
160    // Set active model
161    setActiveModel(modelId) {
162        activeModel = modelId;
163
164        // Update conversation
165        set(state => {
166            state.conversations.set(conversationId, {
167                ...state.conversations.get(conversationId),
168                activeModel,
169            });
170        });
171
172        // Stream response
173        try {
174            const llmService = getLLMService();
175            const conv = get().conversations.get(conversationId);
176
177            await llmService.stream(
178            {
179                messages: conv?.messages ?? [],
180                model: activeModel,
181                temperature: 0.7,
182                maxTokens: 4096,
183            },
184            // On chunk
185            (chunk) => {
186                set(state => {
187                    const conv = state.conversations.get(conversationId
188                        !);
189                    if (conv) {
190                        const msgIndex = conv.messages.findIndex(
191                            m => m.id === assistantMessage.id
192                        );
193
194                    }
195                });
196            }
197        );
198    }
199
200    // Get conversation
201    getConversation(id) {
202        return state.conversations.get(id);
203    }
204
205    // Set conversation
206    setConversation(id, conv) {
207        state.conversations.set(id, conv);
208
209        // Stream response
210        try {
211            const llmService = getLLMService();
212            const conv = get().conversations.get(id);
213
214            await llmService.stream(
215            {
216                messages: conv?.messages ?? [],
217                model: activeModel,
218                temperature: 0.7,
219                maxTokens: 4096,
220            },
221            // On chunk
222            (chunk) => {
223                set(state => {
224                    const conv = state.conversations.get(id
225                        !);
226                    if (conv) {
227                        const msgIndex = conv.messages.findIndex(
228                            m => m.id === assistantMessage.id
229                        );
230
231                    }
232                });
233            }
234        );
235    }
236
237    // Get conversations
238    getConversations() {
239        return state.conversations;
240    }
241
242    // Set conversations
243    setConversations(conv) {
244        state.conversations = conv;
245
246        // Stream response
247        try {
248            const llmService = getLLMService();
249            const conv = get().conversations.get(id);
250
251            await llmService.stream(
252            {
253                messages: conv?.messages ?? [],
254                model: activeModel,
255                temperature: 0.7,
256                maxTokens: 4096,
257            },
258            // On chunk
259            (chunk) => {
260                set(state => {
261                    const conv = state.conversations.get(id
262                        !);
263                    if (conv) {
264                        const msgIndex = conv.messages.findIndex(
265                            m => m.id === assistantMessage.id
266                        );
267
268                    }
269                });
270            }
271        );
272    }
273
274    // Get active model
275    getActiveModel() {
276        return activeModel;
277    }
278
279    // Set active model
280    setActiveModel(modelId) {
281        activeModel = modelId;
282
283        // Stream response
284        try {
285            const llmService = getLLMService();
286            const conv = get().conversations.get(id);
287
288            await llmService.stream(
289            {
290                messages: conv?.messages ?? [],
291                model: activeModel,
292                temperature: 0.7,
293                maxTokens: 4096,
294            },
295            // On chunk
296            (chunk) => {
297                set(state => {
298                    const conv = state.conversations.get(id
299                        !);
300                    if (conv) {
301                        const msgIndex = conv.messages.findIndex(
302                            m => m.id === assistantMessage.id
303                        );
304
305                    }
306                });
307            }
308        );
309    }
310
311    // Get error
312    getError() {
313        return state.error;
314    }
315
316    // Set error
317    setError(error) {
318        state.error = error;
319
320        // Stream response
321        try {
322            const llmService = getLLMService();
323            const conv = get().conversations.get(id);
324
325            await llmService.stream(
326            {
327                messages: conv?.messages ?? [],
328                model: activeModel,
329                temperature: 0.7,
330                maxTokens: 4096,
331            },
332            // On chunk
333            (chunk) => {
334                set(state => {
335                    const conv = state.conversations.get(id
336                        !);
337                    if (conv) {
338                        const msgIndex = conv.messages.findIndex(
339                            m => m.id === assistantMessage.id
340                        );
341
342                    }
343                });
344            }
345        );
346    }
347
348    // Get loading
349    isLoading() {
350        return state.isLoading;
351    }
352
353    // Set loading
354    setIsLoading(isLoading) {
355        state.isLoading = isLoading;
356
357        // Stream response
358        try {
359            const llmService = getLLMService();
360            const conv = get().conversations.get(id);
361
362            await llmService.stream(
363            {
364                messages: conv?.messages ?? [],
365                model: activeModel,
366                temperature: 0.7,
367                maxTokens: 4096,
368            },
369            // On chunk
370            (chunk) => {
371                set(state => {
372                    const conv = state.conversations.get(id
373                        !);
374                    if (conv) {
375                        const msgIndex = conv.messages.findIndex(
376                            m => m.id === assistantMessage.id
377                        );
378
379                    }
380                });
381            }
382        );
383    }
384
385    // Get error
386    getError() {
387        return state.error;
388    }
389
390    // Set error
391    setError(error) {
392        state.error = error;
393
394        // Stream response
395        try {
396            const llmService = getLLMService();
397            const conv = get().conversations.get(id);
398
399            await llmService.stream(
400            {
401                messages: conv?.messages ?? [],
402                model: activeModel,
403                temperature: 0.7,
404                maxTokens: 4096,
405            },
406            // On chunk
407            (chunk) => {
408                set(state => {
409                    const conv = state.conversations.get(id
410                        !);
411                    if (conv) {
412                        const msgIndex = conv.messages.findIndex(
413                            m => m.id === assistantMessage.id
414                        );
415
416                    }
417                });
418            }
419        );
420    }
421
422    // Get error
423    getError() {
424        return state.error;
425    }
426
427    // Set error
428    setError(error) {
429        state.error = error;
430
431        // Stream response
432        try {
433            const llmService = getLLMService();
434            const conv = get().conversations.get(id);
435
436            await llmService.stream(
437            {
438                messages: conv?.messages ?? [],
439                model: activeModel,
440                temperature: 0.7,
441                maxTokens: 4096,
442            },
443            // On chunk
444            (chunk) => {
445                set(state => {
446                    const conv = state.conversations.get(id
447                        !);
448                    if (conv) {
449                        const msgIndex = conv.messages.findIndex(
450                            m => m.id === assistantMessage.id
451                        );
452
453                    }
454                });
455            }
456        );
457    }
458
459    // Get error
460    getError() {
461        return state.error;
462    }
463
464    // Set error
465    setError(error) {
466        state.error = error;
467
468        // Stream response
469        try {
470            const llmService = getLLMService();
471            const conv = get().conversations.get(id);
472
473            await llmService.stream(
474            {
475                messages: conv?.messages ?? [],
476                model: activeModel,
477                temperature: 0.7,
478                maxTokens: 4096,
479            },
480            // On chunk
481            (chunk) => {
482                set(state => {
483                    const conv = state.conversations.get(id
484                        !);
485                    if (conv) {
486                        const msgIndex = conv.messages.findIndex(
487                            m => m.id === assistantMessage.id
488                        );
489
490                    }
491                });
492            }
493        );
494    }
495
496    // Get error
497    getError() {
498        return state.error;
499    }
500
501    // Set error
502    setError(error) {
503        state.error = error;
504
505        // Stream response
506        try {
507            const llmService = getLLMService();
508            const conv = get().conversations.get(id);
509
510            await llmService.stream(
511            {
512                messages: conv?.messages ?? [],
513                model: activeModel,
514                temperature: 0.7,
515                maxTokens: 4096,
516            },
517            // On chunk
518            (chunk) => {
519                set(state => {
520                    const conv = state.conversations.get(id
521                        !);
522                    if (conv) {
523                        const msgIndex = conv.messages.findIndex(
524                            m => m.id === assistantMessage.id
525                        );
526
527                    }
528                });
529            }
530        );
531    }
532
533    // Get error
534    getError() {
535        return state.error;
536    }
537
538    // Set error
539    setError(error) {
540        state.error = error;
541
542        // Stream response
543        try {
544            const llmService = getLLMService();
545            const conv = get().conversations.get(id);
546
547            await llmService.stream(
548            {
549                messages: conv?.messages ?? [],
550                model: activeModel,
551                temperature: 0.7,
552                maxTokens: 4096,
553            },
554            // On chunk
555            (chunk) => {
556                set(state => {
557                    const conv = state.conversations.get(id
558                        !);
559                    if (conv) {
560                        const msgIndex = conv.messages.findIndex(
561                            m => m.id === assistantMessage.id
562                        );
563
564                    }
565                });
566            }
567        );
568    }
569
570    // Get error
571    getError() {
572        return state.error;
573    }
574
575    // Set error
576    setError(error) {
577        state.error = error;
578
579        // Stream response
580        try {
581            const llmService = getLLMService();
582            const conv = get().conversations.get(id);
583
584            await llmService.stream(
585            {
586                messages: conv?.messages ?? [],
587                model: activeModel,
588                temperature: 0.7,
589                maxTokens: 4096,
590            },
591            // On chunk
592            (chunk) => {
593                set(state => {
594                    const conv = state.conversations.get(id
595                        !);
596                    if (conv) {
597                        const msgIndex = conv.messages.findIndex(
598                            m => m.id === assistantMessage.id
599                        );
600
601                    }
602                });
603            }
604        );
605    }
606
607    // Get error
608    getError() {
609        return state.error;
610    }
611
612    // Set error
613    setError(error) {
614        state.error = error;
615
616        // Stream response
617        try {
618            const llmService = getLLMService();
619            const conv = get().conversations.get(id);
620
621            await llmService.stream(
622            {
623                messages: conv?.messages ?? [],
624                model: activeModel,
625                temperature: 0.7,
626                maxTokens: 4096,
627            },
628            // On chunk
629            (chunk) => {
630                set(state => {
631                    const conv = state.conversations.get(id
632                        !);
633                    if (conv) {
634                        const msgIndex = conv.messages.findIndex(
635                            m => m.id === assistantMessage.id
636                        );
637
638                    }
639                });
640            }
641        );
642    }
643
644    // Get error
645    getError() {
646        return state.error;
647    }
648
649    // Set error
650    setError(error) {
651        state.error = error;
652
653        // Stream response
654        try {
655            const llmService = getLLMService();
656            const conv = get().conversations.get(id);
657
658            await llmService.stream(
659            {
660                messages: conv?.messages ?? [],
661                model: activeModel,
662                temperature: 0.7,
663                maxTokens: 4096,
664            },
665            // On chunk
666            (chunk) => {
667                set(state => {
668                    const conv = state.conversations.get(id
669                        !);
670                    if (conv) {
671                        const msgIndex = conv.messages.findIndex(
672                            m => m.id === assistantMessage.id
673                        );
674
675                    }
676                });
677            }
678        );
679    }
680
681    // Get error
682    getError() {
683        return state.error;
684    }
685
686    // Set error
687    setError(error) {
688        state.error = error;
689
690        // Stream response
691        try {
692            const llmService = getLLMService();
693            const conv = get().conversations.get(id);
694
695            await llmService.stream(
696            {
697                messages: conv?.messages ?? [],
698                model: activeModel,
699                temperature: 0.7,
700                maxTokens: 4096,
701            },
702            // On chunk
703            (chunk) => {
704                set(state => {
705                    const conv = state.conversations.get(id
706                        !);
707                    if (conv) {
708                        const msgIndex = conv.messages.findIndex(
709                            m => m.id === assistantMessage.id
710                        );
711
712                    }
713                });
714            }
715        );
716    }
717
718    // Get error
719    getError() {
720        return state.error;
721    }
722
723    // Set error
724    setError(error) {
725        state.error = error;
726
727        // Stream response
728        try {
729            const llmService = getLLMService();
730            const conv = get().conversations.get(id);
731
732            await llmService.stream(
733            {
734                messages: conv?.messages ?? [],
735                model: activeModel,
736                temperature: 0.7,
737                maxTokens: 4096,
738            },
739            // On chunk
740            (chunk) => {
741                set(state => {
742                    const conv = state.conversations.get(id
743                        !);
744                    if (conv) {
745                        const msgIndex = conv.messages.findIndex(
746                            m => m.id === assistantMessage.id
747                        );
748
749                    }
750                });
751            }
752        );
753    }
754
755    // Get error
756    getError() {
757        return state.error;
758    }
759
760    // Set error
761    setError(error) {
762        state.error = error;
763
764        // Stream response
765        try {
766            const llmService = getLLMService();
767            const conv = get().conversations.get(id);
768
769            await llmService.stream(
770            {
771                messages: conv?.messages ?? [],
772                model: activeModel,
773                temperature: 0.7,
774                maxTokens: 4096,
775            },
776            // On chunk
777            (chunk) => {
778                set(state => {
779                    const conv = state.conversations.get(id
780                        !);
781                    if (conv) {
782                        const msgIndex = conv.messages.findIndex(
783                            m => m.id === assistantMessage.id
784                        );
785
786                    }
787                });
788            }
789        );
790    }
791
792    // Get error
793    getError() {
794        return state.error;
795    }
796
797    // Set error
798    setError(error) {
799        state.error = error;
800
801        // Stream response
802        try {
803            const llmService = getLLMService();
804            const conv = get().conversations.get(id);
805
806            await llmService.stream(
807            {
808                messages: conv?.messages ?? [],
809                model: activeModel,
810                temperature: 0.7,
811                maxTokens: 4096,
812            },
813            // On chunk
814            (chunk) => {
815                set(state => {
816                    const conv = state.conversations.get(id
817                        !);
818                    if (conv) {
819                        const msgIndex = conv.messages.findIndex(
820                            m => m.id === assistantMessage.id
821                        );
822
823                    }
824                });
825            }
826        );
827    }
828
829    // Get error
830    getError() {
831        return state.error;
832    }
833
834    // Set error
835    setError(error) {
836        state.error = error;
837
838        // Stream response
839        try {
840            const llmService = getLLMService();
841            const conv = get().conversations.get(id);
842
843            await llmService.stream(
844            {
845                messages: conv?.messages ?? [],
846                model: activeModel,
847                temperature: 0.7,
848                maxTokens: 4096,
849            },
850            // On chunk
851            (chunk) => {
852                set(state => {
853                    const conv = state.conversations.get(id
854                        !);
855                    if (conv) {
856                        const msgIndex = conv.messages.findIndex(
857                            m => m.id === assistantMessage.id
858                        );
859
860                    }
861                });
862            }
863        );
864    }
865
866    // Get error
867    getError() {
868        return state.error;
869    }
870
871    // Set error
872    setError(error) {
873        state.error = error;
874
875        // Stream response
876        try {
877            const llmService = getLLMService();
878            const conv = get().conversations.get(id);
879
880            await llmService.stream(
881            {
882                messages: conv?.messages ?? [],
883                model: activeModel,
884                temperature: 0.7,
885                maxTokens: 4096,
886            },
887            // On chunk
888            (chunk) => {
889                set(state => {
890                    const conv = state.conversations.get(id
891                        !);
892                    if (conv) {
893                        const msgIndex = conv.messages.findIndex(
894                            m => m.id === assistantMessage.id
895                        );
896
897                    }
898                });
899            }
900        );
901    }
902
903    // Get error
904    getError() {
905        return state.error;
906    }
907
908    // Set error
909    setError(error) {
910        state.error = error;
911
912        // Stream response
913        try {
914            const llmService = getLLMService();
915            const conv = get().conversations.get(id);
916
917            await llmService.stream(
918            {
919                messages: conv?.messages ?? [],
920                model: activeModel,
921                temperature: 0.7,
922                maxTokens: 4096,
923            },
924            // On chunk
925            (chunk) => {
926                set(state => {
927                    const conv = state.conversations.get(id
928                        !);
929                    if (conv) {
930                        const msgIndex = conv.messages.findIndex(
931                            m => m.id === assistantMessage.id
932                        );
933
934                    }
935                });
936            }
937        );
938    }
939
940    // Get error
941    getError() {
942        return state.error;
943    }
944
945    // Set error
946    setError(error) {
947        state.error = error;
948
949        // Stream response
950        try {
951            const llmService = getLLMService();
952            const conv = get().conversations.get(id);
953
954            await llmService.stream(
955            {
956                messages: conv?.messages ?? [],
957                model: activeModel,
958                temperature: 0.7,
959                maxTokens: 4096,
960            },
961            // On chunk
962            (chunk) => {
963                set(state => {
964                    const conv = state.conversations.get(id
965                        !);
966                    if (conv) {
967                        const msgIndex = conv.messages.findIndex(
968                            m => m.id === assistantMessage.id
969                        );
970
971                    }
972                });
973            }
974        );
975    }
976
977    // Get error
978    getError() {
979        return state.error;
980    }
981
982    // Set error
983    setError(error) {
984        state.error = error;
985
986        // Stream response
987        try {
988            const llmService = getLLMService();
989            const conv = get().conversations.get(id);
990
991            await llmService.stream(
992            {
993                messages: conv?.messages ?? [],
994                model: activeModel,
995                temperature: 0.7,
996                maxTokens: 4096,
997            },
998            // On chunk
999            (chunk) => {
1000                set(state => {
1001                    const conv = state.conversations.get(id
1002                        !);
1003                    if (conv) {
1004                        const msgIndex = conv.messages.findIndex(
1005                            m => m.id === assistantMessage.id
1006                        );
1007
1008                    }
1009                });
1010            }
1011        );
1012    }
1013
1014    // Get error
1015    getError() {
1016        return state.error;
1017    }
1018
1019    // Set error
1020    setError(error) {
1021        state.error = error;
1022
1023        // Stream response
1024        try {
1025            const llmService = getLLMService();
1026            const conv = get().conversations.get(id);
1027
1028            await llmService.stream(
1029            {
1030                messages: conv?.messages ?? [],
1031                model: activeModel,
1032                temperature: 0.7,
1033                maxTokens: 4096,
1034            },
1035            // On chunk
1036            (chunk) => {
1037                set(state => {
1038                    const conv = state.conversations.get(id
1039                        !);
1040                    if (conv) {
1041                        const msgIndex = conv.messages.findIndex(
1042                            m => m.id === assistantMessage.id
1043                        );
1044
1045                    }
1046                });
1047            }
1048        );
1049    }
1050
1051    // Get error
1052    getError() {
1053        return state.error;
1054    }
1055
1056    // Set error
1057    setError(error) {
1058        state.error = error;
1059
1060        // Stream response
1061        try {
1062            const llmService = getLLMService();
1063            const conv = get().conversations.get(id);
1064
1065            await llmService.stream(
1066            {
1067                messages: conv?.messages ?? [],
1068                model: activeModel,
1069                temperature: 0.7,
1070                maxTokens: 4096,
1071            },
1072            // On chunk
1073            (chunk) => {
1074                set(state => {
1075                    const conv = state.conversations.get(id
1076                        !);
1077                    if (conv) {
1078                        const msgIndex = conv.messages.findIndex(
1079                            m => m.id === assistantMessage.id
1080                        );
1081
1082                    }
1083                });
1084            }
1085        );
1086    }
1087
1088    // Get error
1089    getError() {
1090        return state.error;
1091    }
1092
1093    // Set error
1094    setError(error) {
1095        state.error = error;
1096
1097        // Stream response
1098        try {
1099            const llmService = getLLMService();
1100            const conv = get().conversations.get(id);
1101
1102            await llmService.stream(
1103            {
1104                messages: conv?.messages ?? [],
1105                model: activeModel,
1106                temperature: 0.7,
1107                maxTokens: 4096,
1108            },
1109            // On chunk
1110            (chunk) => {
1111                set(state => {
1112                    const conv = state.conversations.get(id
1113                        !);
1114                    if (conv) {
1115                        const msgIndex = conv.messages.findIndex(
1116                            m => m.id === assistantMessage.id
1117                        );
1118
1119                    }
1120                });
1121            }
1122        );
1123    }
1124
1125    // Get error
1126    getError() {
1127        return state.error;
1128    }
1129
1130    // Set error
1131    setError(error) {
1132        state.error = error;
1133
1134        // Stream response
1135        try {
1136            const llmService = getLLMService();
1137            const conv = get().conversations.get(id);
1138
1139            await llmService.stream(
1140            {
1141                messages: conv?.messages ?? [],
1142                model: activeModel,
1143                temperature: 0.7,
1144                maxTokens: 4096,
1145            },
1146            // On chunk
1147            (chunk) => {
1148                set(state => {
1149                    const conv = state.conversations.get(id
1150                        !);
1151                    if (conv) {
1152                        const msgIndex = conv.messages.findIndex(
1153                            m => m.id === assistantMessage.id
1154                        );
1155
1156                    }
1157                });
1158            }
1159        );
1160    }
1161
1162    // Get error
1163    getError() {
1164        return state.error;
1165    }
1166
1167    // Set error
1168    setError(error) {
1169        state.error = error;
1170
1171        // Stream response
1172        try {
1173            const llmService = getLLMService();
1174            const conv = get().conversations.get(id);
1175
1176            await llmService.stream(
1177            {
1178                messages: conv?.messages ?? [],
1179                model: activeModel,
1180                temperature: 0.7,
1181                maxTokens: 4096,
1182            },
1183            // On chunk
1184            (chunk) => {
1185                set(state => {
1186                    const conv = state.conversations.get(id
1187                        !);
1188                    if (conv) {
1189                        const msgIndex = conv.messages.findIndex(
1190                            m => m.id === assistantMessage.id
1191                        );
1192
1193                    }
1194                });
1195            }
1196        );
1197    }
1198
1199    // Get error
1200    getError() {
1201        return state.error;
1202    }
1203
1204    // Set error
1205    setError(error) {
1206        state.error = error;
1207
1208        // Stream response
1209        try {
1210            const llmService = getLLMService();
1211            const conv = get().conversations.get(id);
1212
1213            await llmService.stream(
1214            {
1215                messages: conv?.messages ?? [],
1216                model: activeModel,
1217                temperature: 0.7,
1218                maxTokens: 4096,
1219            },
1220            // On chunk
1221            (chunk) => {
1222                set(state => {
1223                    const conv = state.conversations.get(id
1224                        !);
1225                    if (conv) {
1226                        const msgIndex = conv.messages.findIndex(
1227                            m => m.id === assistantMessage.id
1228                        );
1229
1230                    }
1231                });
1232            }
1233        );
1234    }
1235
1236    // Get error
1237    getError() {
1238        return state.error;
1239    }
1240
1241    // Set error
1242    setError(error) {
1243        state.error = error;
1244
1245        // Stream response
1246        try {
1247            const llmService = getLLMService();
1248            const conv = get().conversations.get(id);
1249
1250            await llmService.stream(
1251            {
1252                messages: conv?.messages ?? [],
1253                model: activeModel,
1254                temperature: 0.7,
1255                maxTokens: 4096,
1256            },
1257            // On chunk
1258            (chunk) => {
1259                set(state => {
1260                    const conv = state.conversations.get(id
1261                        !);
1262                    if (conv) {
1263                        const msgIndex = conv.messages.findIndex(
1264                            m => m.id === assistantMessage.id
1265                        );
1266
1267                    }
1268                });
1269            }
1270        );
1271    }
1272
1273    // Get error
1274    getError() {
1275        return state.error;
1276    }
1277
1278    // Set error
1279    setError(error) {
1280        state.error = error;
1281
1282        // Stream response
1283        try {
1284            const llmService = getLLMService();
1285            const conv = get().conversations.get(id);
1286
1287            await llmService.stream(
1288            {
1289                messages: conv?.messages ?? [],
1290                model: activeModel,
1291                temperature: 0.7,
1292                maxTokens: 4096,
1293            },
1294            // On chunk
1295            (chunk) => {
1296                set(state => {
1297                    const conv = state.conversations.get(id
1298                        !);
1299                    if (conv) {
1300                        const msgIndex = conv.messages.findIndex(
1301                            m => m.id === assistantMessage.id
1302                        );
1303
1304                    }
1305                });
1306            }
1307        );
1308    }
1309
1310    // Get error
1311    getError() {
1312        return state.error;
1313    }
1314
1315    // Set error
1316    setError(error) {
1317        state.error = error;
1318
1319        // Stream response
1320        try {
1321            const llmService = getLLMService();
1322            const conv = get().conversations.get(id);
1323
1324            await llmService.stream(
1325            {
1326                messages: conv?.messages ?? [],
1327                model: activeModel,
1328                temperature: 0.7,
1329                maxTokens: 4096,
1330            },
1331            // On chunk
1332            (chunk) => {
1333                set(state => {
1334                    const conv = state.conversations.get(id
1335                        !);
1336                    if (conv) {
1337                        const msgIndex = conv.messages.findIndex(
1338                            m => m.id === assistantMessage.id
1339                        );
1340
1341                    }
1342                });
1343            }
1344        );
1345    }
1346
1347    // Get error
1348    getError() {
1349        return state.error;
1350    }
1351
1352    // Set error
1353    setError(error) {
1354        state.error = error;
1355
1356        // Stream response
1357        try {
1358            const llmService = getLLMService();
1359            const conv = get().conversations.get(id);
1360
1361            await llmService.stream(
1362            {
1363                messages: conv?.messages ?? [],
1364                model: activeModel,
1365                temperature: 0.7,
1366                maxTokens: 4096,
1367            },
1368            // On chunk
1369            (chunk) => {
1370                set(state => {
1371                    const conv = state.conversations.get(id
1372                        !);
1373                    if (conv) {
1374                        const msgIndex = conv.messages.findIndex(
1375                            m => m.id === assistantMessage.id
1376                        );
1377
1378                    }
1379                });
1380            }
1381        );
1382    }
1383
1384    // Get error
1385    getError() {
1386        return state.error;
1387    }
1388
1389    // Set error
1390    setError(error) {
1391        state.error = error;
1392
1393        // Stream response
1394        try {
1395            const llmService = getLLMService();
1396            const conv = get().conversations.get(id);
1397
1398            await llmService.stream(
1399            {
1400                messages: conv?.messages ?? [],
1401                model: activeModel,
1402                temperature: 0.7,
1403                maxTokens: 4096,
1404            },
1405            // On chunk
1406            (chunk) => {
1407                set(state => {
1408                    const conv = state.conversations.get(id
1409                        !);
1410                    if (conv) {
1411                        const msgIndex = conv.messages.findIndex(
1412                            m => m.id === assistantMessage.id
1413                        );
1414
1415                    }
1416                });
1417            }
1418        );
1419    }
1420
1421    // Get error
1422    getError() {
1423        return state.error;
1424    }
1425
1426    // Set error
1427    setError(error) {
1428        state.error = error;
1429
14
```

```

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 );
182 } finally {
183     set(state => {
184         state.isLoading = false;
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 private formatSelection(selection: SelectionContext): string {
75   return `## Current Selection
76 Type: ${selection.type}
77 ${selection.elementIds ? 'Elements: ${selection.elementIds.join(', )}' : ''}
78 \`\`\`
79 ${selection.content}
80 \`\`\`;
81 }
82
83 private formatFile(file: FileContext): string {
84   const lang = file.language ?? this.detectLanguage(file.name);
85   let content = file.content;
86
87   // Highlight selection if present
88   if (file.selection) {
89     content = this.highlightSelection(content, file.selection);
90   }
91
92   return `## Active File: ${file.name}
93 Path: ${file.path}
94
95 \`\`\`${lang}
96 ${content}
97 \`\`\`;
98 }
99
100 private formatProject(project: ProjectInfo): string {
101   return `## Project Information
102 Name: ${project.name}
103 Type: ${project.type}
104 Path: ${project.rootPath}`;
105 }
106
107 private detectLanguage(filename: string): string {
108

```

```

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, // 10 MB
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