# Frontend Engineer Assignment: Weather Agent Chat Interface

### **Overview**

Build a responsive chat interface that connects to our weather agent API. This assignment will evaluate your frontend development skills, API integration capabilities, and attention to user experience.

### **Assignment Details**

### **Objective**

Create a functional chat window that allows users to interact with a weather agent through a streaming API endpoint.

### **Technology Stack**

- Frontend Framework: React or Next.js (required)
- Styling: CSS3, Tailwind CSS, styled-components, or CSS Modules
- Build Tools: Vite, Create React App, or Next.js built-in tooling

### **API Specification**

### **Endpoint**

```
POST https://brief-thousands-sunset-9fcb1c78-485f-4967-ac04-2759a8fa1462.mastra.cloud/api/agents/weatherAgent/stream
```

#### **Headers**

```
javascript
{
   'Accept': '*/*',
   'Accept-Language': 'en-GB,en-US;q=0.9,en;q=0.8,fr;q=0.7',
   'Connection': 'keep-alive',
   'Content-Type': 'application/json',
   'x-mastra-dev-playground': 'true'
}
```

## **Request Body**

**Important**: Replace YOUR\_COLLEGE\_ROLL\_NUMBER with your actual college roll number for the (threadId) field.

## Requirements

# **Core Functionality**

### 1. Chat Interface

- Message input field with send button
- Display conversation history
- Show user messages on the right
- Show agent responses on the left
- Auto-scroll to latest message

#### 2. API Integration

- Send user messages to the weather agent API
- Handle streaming responses appropriately
- Display loading states during API calls
- Implement proper error handling

### 3. Message Management

- Maintain conversation history
- Handle multiple message threads
- Clear chat functionality

# UI/UX Requirements 🎨

### 1. Responsive Design

- Mobile-first approach
- Works on desktop, tablet, and mobile
- Minimum width: 320px

#### 2. Visual Design

- Clean, modern interface
- Proper typography and spacing
- Loading indicators
- Message timestamps
- Distinct styling for user vs agent messages

### 3. User Experience

- Smooth animations/transitions
- Keyboard shortcuts (Enter to send)
- Disabled state for input during API calls
- Error messages for failed requests

# **Technical Requirements**

#### 1. Code Quality

- Clean, readable code
- Proper component structure
- Meaningful variable names
- Comments where necessary

#### 2. Performance

- Efficient re-rendering
- Proper state management
- Optimized API calls

#### 3. Error Handling

- Network failures
- API errors
- Invalid responses
- · User feedback for all error states



#### Advanced Features

- Message search functionality
- Export chat history
- Dark/light theme toggle
- Message reactions or feedback
- Typing indicators

#### **Technical Excellence**

- TypeScript implementation
- · Custom React hooks
- Unit tests (Jest/React Testing Library)
- Accessibility features (ARIA labels, keyboard navigation)
- Progressive Web App features
- Real-time streaming response display
- Next.js features (SSR, API routes, etc.)

#### Polish

- · Smooth animations
- · Custom weather-themed icons
- Sound notifications
- Message delivery status indicators

### Deliverables **P**



### Required

### 1. Source Code

- Complete, runnable project
- README.md with setup instructions
- Package.json with dependencies

### 2. Documentation

- Brief explanation of your approach
- Any assumptions made
- Known limitations or areas for improvement

### **Optional**

### 1. Live Demo

Deployed version (Netlify, Vercel, etc.)

• Include the URL in your README

#### 2. Video Walkthrough

- 2-3 minute demo of your implementation
- Highlight key features and design decisions

## Evaluation Criteria

### **Technical Implementation (40%)**

- React/Next.js implementation quality
- · Component architecture and reusability
- State management (useState, useReducer, Context, etc.)
- API integration with proper hooks usage
- Error handling implementation

### **User Experience (30%)**

- · Interface usability and intuitiveness
- Responsive design quality
- · Visual appeal and consistency
- Loading states and feedback

### Code Quality (20%)

- · Readability and maintainability
- Performance considerations
- Best practices adherence
- Documentation quality

## **Innovation & Polish (10%)**

- Creative problem-solving
- Attention to detail
- Bonus features implementation
- Overall professional finish

# **Submission Guidelines**

#### **Format**

- GitHub Repository: Create a public repository with your solution
- Email Subject: "Frontend Assignment Submission [Your Name]"

• Include: Repository URL, live demo URL (if applicable), any additional notes

# Sample Test Cases

Test your implementation with these scenarios:

#### 1. Basic Interaction

- Send message: "What's the weather in London?"
- Verify agent response displays correctly

### 2. Error Handling

- Send message with network disconnected
- Verify error message appears

### 3. Multiple Messages

- Send several messages in sequence
- Verify conversation flow is maintained

#### 4. Edge Cases

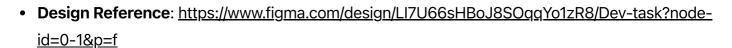
- Very long messages
- Empty messages
- Special characters in messages

# Tips for Success 7



- 1. Start Simple: Get basic chat functionality working first
- 2. Read the API: Understand the request/response format thoroughly
- 3. **Test Early**: Test with the actual API as soon as possible
- 4. Focus on UX: Prioritize user experience over complex features
- 5. **Document Decisions**: Explain your technical choices in the README
- 6. **Handle Errors**: Robust error handling will set you apart
- 7. **Mobile First**: Ensure it works well on mobile devices

## Resources 👺



#### Example Weather Queries:

- "What's the weather in [city]?"
- "Will it rain tomorrow in [city]?"
- "Weather forecast for next week"

Good luck! We're excited to see your implementation. Take your time to deliver a quality solution - we value thorough, well-executed work over rushed submissions.