```
let messages = [
  {
    "role": "system",
    "content": `You are a knowledgeable assistant that specializes in programming and systems design,
    helping develop a user interface for artificial intelligence systems based on LLM technology.`
  },
];

let systemMessages = [
  {
    "name": "LLM User Interface",
    "role": "system",
    "content": `You are a knowledgeable assistant that specializes in programming and systems design,
    helping develop a user interface for artificial intelligence systems based on LLM technology.`
  },
  {
    "name": "IT Assistant",
    "role": "system",
    "content": `You are an IT assistant that helps users with their computer problems.`
  },
  {
    "name": "IT Assistant2",
    "role": "system",
    "content": "System Message 3..."
  },
];

let model = "gpt-3.5-turbo-0613"; // Declare the model variable here
let activeConversationId = null; // This will keep track of the currently selected conversation
```

```
// This function updates the system message dropdown menu
let systemMessageDropdown = $('#system_message_dropdown');
systemMessages.forEach((message, index) => {
    systemMessageDropdown.append(
        $('')
        .addClass('dropdown-item')
        .attr('href', '#')
        .attr('data-index', index)
        .text(message.name)
    );
});



function displaySystemMessage(messageContent) {
    // Define the system message
    const systemMessage = {
      role: 'system',
      content: messageContent  // use passed argument
    };

    // Use marked to render the message content as HTML
    const renderedContent = renderOpenAI(systemMessage.content);

    // Display the system message with consistent styling
    $('#chat').append('<div class="chat-entry ' + systemMessage.role + ' system-message">System: ' +
renderedContent + '</div>');
}
```

```
function detectAndRenderMarkdown(content) {
    // Check for headers, lists, links, etc.
    const markdownPatterns = [
        /^# .+/gm,       // Headers
        /^\* .+/gm,      // Unordered lists
        /^\d+\. .+/gm,   // Ordered lists
        /\[.+\]\(.+\)/g // Links
    ];

    let containsMarkdown = false;
    for (let pattern of markdownPatterns) {
        if (pattern.test(content)) {
            containsMarkdown = true;
            break;
        }
    }

    // If potential markdown structures are detected, process the content with the Markdown parser
    if (containsMarkdown) {
        // Avoid processing content inside triple backticks
        const codeBlockRegex = /```[\s\S]*?```/g;
        const codeBlocks = [...content.matchAll(codeBlockRegex)];

        content = content.replace(codeBlockRegex, '%%%CODE_BLOCK%%%'); // Temporary placeholder

        content = marked.parse(content);

        // Restore code blocks
        let blockIndex = 0;
```

```
        content = content.replace(/%%%CODE_BLOCK%%%/g, () => {
            return codeBlocks[blockIndex++] && codeBlocks[blockIndex - 1][0];
        });
    }

    return content;
}

// This function renders the content inside triple backticks - the OpenAI Playground style.
function renderOpenAI(content) {
    console.log('renderOpenAI called with content:', content);

    // First, check for potential markdown structures and process them
    content = detectAndRenderMarkdown(content);

    // Insert a line break before the first numbered item
    let isFirstNumberedItem = true;
    content = content.replace(/\n\n(\d+\. )/g, (match, p1) => {
        if (isFirstNumberedItem) {
            isFirstNumberedItem = false;
            return '<br><br>\n\n' + p1;
        }
        return match;
    });

    // Insert an additional newline between numbered list items
    content = content.replace(/(\n)(\d+\. )/g, '\n\n$2');
```

```
  // Regular expression to match content inside triple backticks, capturing the optional language declaration
  const regex = /```(\w+)?([\s\S]*?)```/g;

  // Function to process matched content
  const replacer = (match, lang, p1) => {
    console.log('Matched content inside backticks with language:', lang, 'Content:', p1.trim());

    let renderedContent;
    if (!lang || lang === 'markdown') {
      // If no language is provided or the language is markdown, process it with the Markdown parser
      renderedContent = marked.parse(p1.trim());
    } else {
      // Otherwise, treat it as a code block
      renderedContent = '<pre><code class="' + (lang ? 'language-' + lang : '') + '">' + p1.trim() + '</code></pre>';
    }
    console.log('Rendered Content:', renderedContent);

    // Create a temporary element to hold the rendered content
    const tempElement = document.createElement('div');
    tempElement.innerHTML = renderedContent;

    // Apply Prism highlighting to the content inside the temporary element
    Prism.highlightAllUnder(tempElement);

    // Log Prism highlighting to the content inside the temporary element
    console.log('Highlighted content:', tempElement.innerHTML);
```

```
        // Return the highlighted content
        return tempElement.innerHTML || match; // Ensure we return the original match if nothing else
    };


    // Replace content inside triple backticks with processed content
    const processedContent = content.replace(regex, replacer);
    console.log('Final processed content:', processedContent);

    return processedContent;
}


function updateConversationList() {
    console.log('Starting to update conversation list...');
    fetch('/api/conversations')
        .then(response => {
            if (!response.ok) {
                throw new Error(`HTTP error! Status: ${response.status}`);
            }
            return response.json();
        })
        .then(data => {
            console.log(`Received ${data.length} conversations from server.`);

            // Prepare new HTML content for conversation list
            let newConversationListContent = '';
            // Add each conversation to the new content.
            data.forEach((conversation, index) => {
```

```
            newConversationListContent += `
                <div class="conversation-item" data-id="${conversation.id}">
                <div class="conversation-title">${conversation.title}</div>
                <div class="conversation-meta">

                    <span class="model-name" title="AI Model used for this conversation">LLM: ${conversation.model_name}</span>
                </div>
            </div>
            `;
        });

        // Replace conversation list content with new content
        $('#conversation-list').html(newConversationListContent);
        console.log('Conversation list updated.');

        // Add click event handlers to the conversation elements.
        $('.conversation-item').click(function() {
            const conversationId = $(this).data('id');
            console.log(`Loading conversation with id: ${conversationId}`);

            // Update the URL to reflect the conversation being loaded
            window.history.pushState({}, '', `/c/${conversationId}`);

            // Load the conversation data
            loadConversation(conversationId);
        });
    })
    .catch(error => {
```

```
            console.error(`Error updating conversation list: ${error}`);
        });
}



$('#edit-title-btn').click(function() {
    const newTitle = prompt('Enter new conversation title:', $('#conversation-title').text());
    if (newTitle) {
        $.ajax({
            url: `/api/conversations/${activeConversationId}/update_title`,
            method: 'POST',
            contentType: 'application/json',
            data: JSON.stringify({ title: newTitle }),
            success: function(response) {
                $('#conversation-title').text(newTitle);

                // Update the title in the sidebar
                const targetConversationItem = $(`.conversation-item[data-id="${activeConversationId}"]
.conversation-title`);

                // Log for debugging purposes
                console.log('Attempting to update sidebar title for conversation ID:', activeConversationId);
                console.log('Targeted element:', targetConversationItem);

                targetConversationItem.text(newTitle);
            },
            error: function(error) {
                console.error("Error updating title:", error);
            }
```

```
        });
    }
});



$('#delete-conversation-btn').click(function() {
    const confirmation = confirm('Are you sure you want to delete this conversation? This action cannot be undone.');
    if (confirmation) {
        $.ajax({
            url: `/api/conversations/${activeConversationId}`,
            method: 'DELETE',
            success: function(response) {
                // Upon successful deletion, redirect to the main URL.
                window.location.href = 'http://127.0.0.1:5000/';
            },
            error: function(error) {
                console.error("Error deleting conversation:", error);
            }
        });
    }
});



// This function shows the conversation controls (title, rename and delete buttons)
function showConversationControls(title = "AI &infin; UI", tokens = {prompt: 0, completion: 0, total: 0}) {
    // Update the title
    console.log("Inside showConversationControls function. Title:", title);
    console.log("Inside showConversationControls. Tokens:", tokens);
```

```
    $("#conversation-title").html(title);
    $("#conversation-title, #edit-title-btn, #delete-conversation-btn").show();

    // Update token data
    $("#prompt-tokens").text(`Prompt Tokens: ${tokens.prompt}`);
    $("#completion-tokens").text(`Completion Tokens: ${tokens.completion}`);
    $("#total-tokens").text(`Total Tokens: ${tokens.total}`);
}


// This function fetchs and displays a conversation.
function loadConversation(conversationId) {
    console.log(`Fetching conversation with id: ${conversationId}...`);
    fetch(`/conversations/${conversationId}`)
        .then(response => {
            console.log('Response received for conversation fetch', response);
            if (!response.ok) {
                throw new Error(`HTTP error! Status: ${response.status}`);
            }
            return response.json();
        })
        .then(data => {
            // Update the conversation title in the UI
            $('#conversation-title').text(data.title);

            // Update the messages array with the conversation history

            console.log('Parsed JSON data from conversation:', data);
```

```
        messages = data.history;

        console.log(`Received conversation data for id: ${conversationId}`);

        console.log('Token Count:', data.token_count);

        console.log("Retrieved model name:", data.model_name);  // Log the retrieved model name

        // Update the dropdown display based on the model name from the conversation
        const modelName = data.model_name;
        $('.current-model-btn').text(modelNameMapping(modelName));
        // Also update the global model variable
        model = modelName;

        // Update the token data in the UI
        const tokenCount = data.token_count || 0;
        showConversationControls(data.title || "AI &infin; UI", {prompt: 0, completion: 0, total:
tokenCount});

        // Save this conversation id as the active conversation
        activeConversationId = conversationId;

        // Check if data.history is already an array, if not try parsing it
        let history;
        if (Array.isArray(data.history)) {
            history = data.history;
        } else {
            try {
```

```
        history = JSON.parse(data.history);
    } catch (e) {
        console.error(`Error parsing history for conversation ${conversationId}: ${e}`);
        return;
    }
}

// Clear the chat
$('#chat').empty();

let lastRole = null;  // added to keep track of the last role

// Add each message to the chat. Style the messages based on their role.
history.forEach(message => {
        let prefix = '';
        let messageClass = '';
        if (message.role === 'system') {
            prefix = 'System: ';
            messageClass = 'system-message';
        } else if (message.role === 'user') {
            prefix = '<i class="far fa-user"></i> ';
            messageClass = 'user-message';
        } else if (message.role === 'assistant') {
            prefix = '<i class="fas fa-robot"></i> ';
            messageClass = 'bot-message';
        }
        // Use marked to render the message content as HTML
        const renderedContent = renderOpenAI(message.content);
```

```
            $('#chat').append('<div class="chat-entry ' + message.role + ' ' + messageClass + '">' + prefix + renderedContent + '</div>');
            lastRole = message.role;  // update the last role
        }
    );

    Prism.highlightAll(); // Highlight code blocks

    // Append blank user message if the last message was from the assistant
    if (lastRole === 'assistant') {
    $('#chat').append('<div class="chat-entry user user-message"><div class="buffer-message"></div></div>');
    }

    // Important! Update the 'messages' array with the loaded conversation history
    messages = history;

    console.log(`Chat updated with messages from conversation id: ${conversationId}`);

    // Scroll to the bottom after populating the chat
    const chatContainer = document.getElementById('chat');
    chatContainer.scrollTop = chatContainer.scrollHeight;
  })
  .catch(error => {
    console.error(`Error fetching conversation with id: ${conversationId}. Error: ${error}`);
  });
}

// Helper function to map model names to their display values
```

```
function modelNameMapping(modelName) {
    switch(modelName) {
        case "gpt-3.5-turbo-0613": return "GPT-3.5";
        case "gpt-4-0613": return "GPT-4";
        default: return "Unknown Model"; // Handle any unexpected values
    }
}

//Record the default height
var defaultHeight = $('#user_input').css('height');

// This function is called when the user submits the form.
$('#chat-form').on('submit', function (e) {
    console.log('Chat form submitted with user input:', $('#user_input').val());
    e.preventDefault();
    var userInput = $('#user_input').val();

    var userInputDiv = $('<div class="chat-entry user user-message">')
    .append('<i class="far fa-user"></i>')
    .append($('<span>').text(userInput));

    $('#chat').append(userInputDiv);
    $('#chat').scrollTop($('#chat')[0].scrollHeight);

    messages.push({"role": "user", "content": userInput});

    var userInputTextarea = $('#user_input');
    userInputTextarea.val('');
    userInputTextarea.css('height', defaultHeight);
```

```
    document.getElementById('loading').style.display = 'block';
    messages[0].content = systemMessages[0].content; // replace the system message in messages with the first message in systemMessages
    let requestPayload = {messages: messages, model: model};
    if (activeConversationId !== null) {
      requestPayload.conversation_id = activeConversationId;
    }
    console.log('Sending request payload:', JSON.stringify(requestPayload));

    fetch('/chat', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(requestPayload)
    })
    .then(response => {
        console.log('Received response from /chat endpoint:', response);

        document.getElementById('loading').style.display = 'none';

        if (!response.ok) { // Check if response status code is OK
            return response.text().then(text => {
                // Use response.text() instead of response.json() if you suspect the server might be returning HTML error pages.
                throw new Error(text);  // Throw an error with the server's response text.
            });
        }
```

```
        return response.json();
    })
    .then(data => {
        console.log("Complete server reponse:", data);
        const renderedBotOutput = marked.parse(data.chat_output);
        console.log("Rendered bot output:", renderedBotOutput); // Log this to debug
        $('#chat').append('<div class="chat-entry bot bot-message"><i class="fas fa-robot"></i> ' +
renderedBotOutput + '</div>');

        console.log("Updated chat with server's response.");

        $('#chat').scrollTop($('#chat')[0].scrollHeight);  // Scroll to the bottom of the chat
        Prism.highlightAll();
        messages.push({"role": "assistant", "content": data.chat_output});
        updateConversationList();

        // Update the URL with the received conversation_id
        window.history.pushState({}, '', `/c/${data.conversation_id}`);

        if (data.conversation_title) {
            // Update the title in the UI
            console.log("Received conversation_title from server:", data.conversation_title);
            showConversationControls(data.conversation_title);

            // Updating the token data in the UI, assuming your server response includes the necessary token
data
            if (data.usage) {
            $('#prompt-tokens').text(`Prompt Tokens: ${data.usage.prompt_tokens}`);
```

```
            $('#completion-tokens').text(`Completion Tokens: ${data.usage.completion_tokens}`);
            $('#total-tokens').text(`Total Tokens: ${data.usage.total_tokens}`);
        }
        } else {
            // If there's no title provided, show default
            console.log("No conversation_title from server. Showing default.");
            showConversationControls();
        }
        console.log('End of chat-form submit function');

    })
});

// This function is called when the user clicks the "New chat" button.
document.getElementById("new-chat-btn").addEventListener("click", function() {
    fetch('/clear-session', {
        method: 'POST',
    })
    .then(response => response.json())
    .then(data => {
        console.log(data);

        activeConversationId = null; // Reset the active conversation

        window.location.href = 'http://127.0.0.1:5000/';

    })
    .catch((error) => {
        console.error('Error:', error);
```

```
    });
});



$(window).on('load', function () {  // This function is called when the page loads.
    updateConversationList(); // Load the list of conversations when the page loads.
    $('.dropdown-item').on('click', function(event){
        event.preventDefault();  // Prevent the # appearing in the URL
        $('#dropdownMenuButton').text($(this).text());
        model = $(this).attr('data-model'); // Update the model variable here
        console.log("Dropdown item clicked. Model is now: " + model);
    });

});

// Check if the chat is empty when the page loads
document.addEventListener("DOMContentLoaded", function() {
    if (!activeConversationId && $('#chat').children().length === 0) {
        // displaySystemMessage(messages[0]);
        displaySystemMessage(systemMessages[0].content); // display default system message, using "content" property.
    }
});



// This function checks if there's an active conversation in the session.
function checkActiveConversation() {
    fetch('/get_active_conversation')
        .then(response => {
```

```
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
    })
    .then(data => {
        const conversationId = data.conversationId;
        if (conversationId) {
            // If there's an active conversation ID in the session, load it
            loadConversation(conversationId);
            // Show the title, rename and delete buttons
            $('#conversation-title, #edit-title-btn, #delete-conversation-btn').show();
        } else {
            // Hide the title, rename and delete buttons
            $('#conversation-title, #edit-title-btn, #delete-conversation-btn').hide();
        }
    })
    .catch(error => {
        console.error('Error checking for active conversation:', error);
        // Optionally hide the elements in case of an error. Depends on your desired behavior.
        $('#conversation-title, #edit-title-btn, #delete-conversation-btn').hide();
    });
}


$(document).ready(function() {
    console.log("Document ready."); // Debug

    // Initialize autosize for the textarea
```

```
    autosize($('#user_input'));


    // Set default title
    $("#conversation-title").html("AI &infin; UI");


    // Get the current path from the URL
    var currentPath = window.location.pathname;


    // If there's a conversation ID passed from the backend (you'll render this in a <script> tag in your chat.html)
    if (typeof conversation_id !== 'undefined' && conversation_id) {
        loadConversation(conversation_id);  // Load this conversation
        $('#conversation-title, #edit-title-btn, #delete-conversation-btn').show();
    }
    // If the path is not the base URL and there's no conversation_id
    else if (currentPath !== '/') {
        console.log("Checking active conversation."); // Debugging: Log which branch we're going into
        checkActiveConversation();
        // Depending on the outcome of `checkActiveConversation`, you might need to .show() or .hide() the elements.
    }
    else {
        console.log("No conversation_id or active conversation found."); // Debugging: Log which branch we're going into


        // If there's no conversation_id and no active conversation, hide these elements
        $('#edit-title-btn, #delete-conversation-btn').hide(); // Hide only the edit and delete buttons
    }


    // New code for the Enter and Shift+Enter behavior
```

```
    $('#user_input').on('keydown', function(e) {
        if (e.key == 'Enter') {
            if (!e.shiftKey) {
                e.preventDefault();
                $('#chat-form').submit();  // Submit the form
            }
        }
    });

});

    // ... other initialization code that should run when the page is fully loaded ...
```