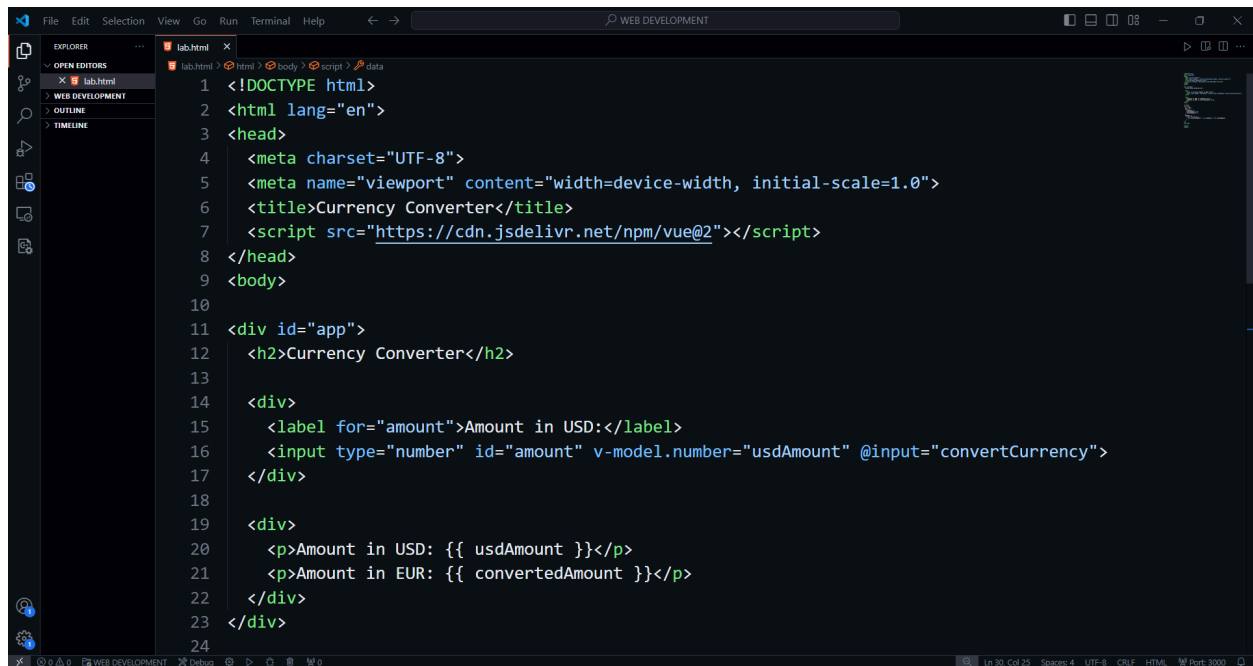


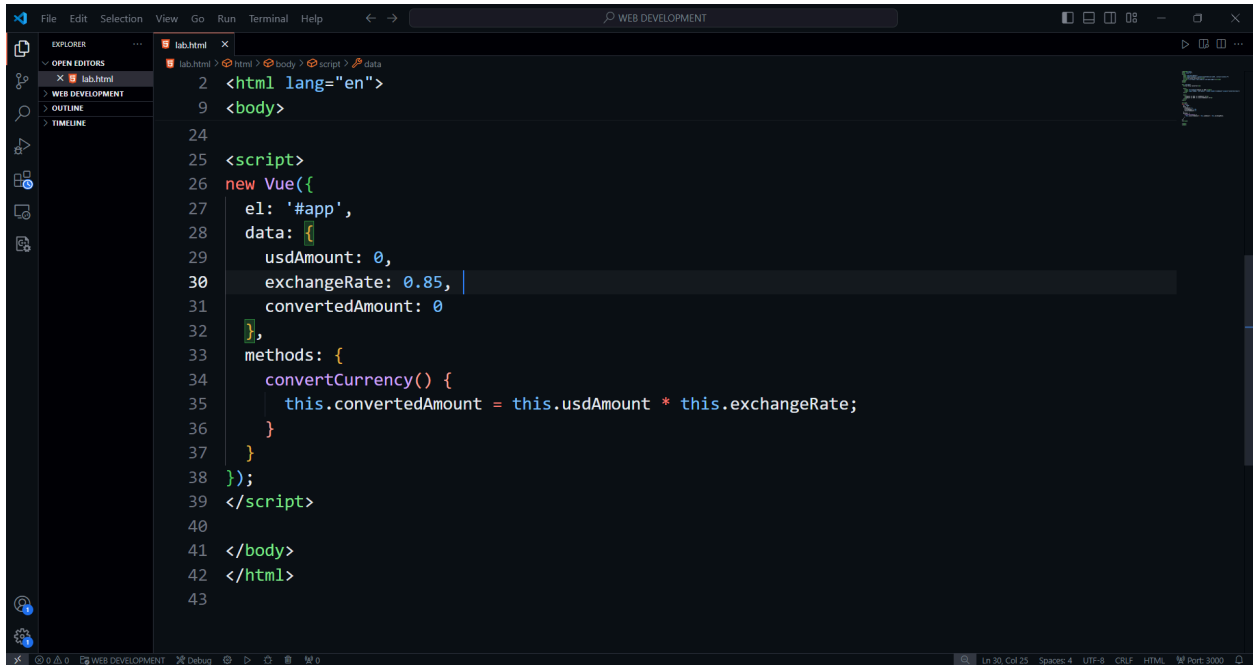
NAME:ARYAN KHAIWAL

ROLL NO.:22CS2029

T1. Develop a currency converter application that allows users to input an amount in one currency and convert it to another. For the sake of this challenge, you can use a hard-coded exchange rate. Take advantage of React state and event handlers to manage the input and conversion calculations.

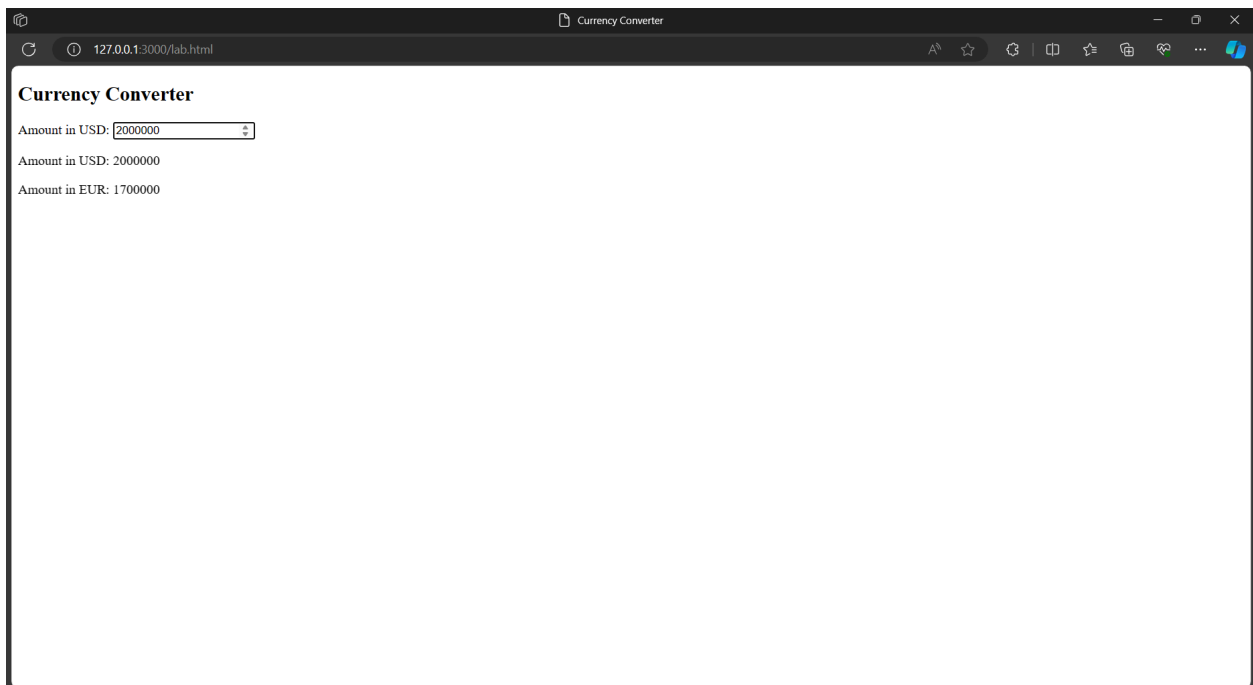
A screenshot of a Visual Studio Code editor window titled 'WEB DEVELOPMENT'. The Explorer sidebar on the left shows a file named 'lab.html' under the 'OPEN EDITORS' section. The main editor area displays the HTML code for 'lab.html'. The code includes a DOCTYPE declaration, a head section with meta tags for charset and viewport, and a title 'Currency Converter'. It also includes a script tag for Vue.js. The body contains a div with id 'app' which has a heading 'Currency Converter'. Inside this div, there is a label 'Amount in USD:', a number input field with id 'amount' and v-model 'usdAmount', and two paragraphs displaying the input and converted values using Vue.js interpolation. The status bar at the bottom indicates the cursor is at line 30, column 25, in a UTF-8 file with CRLF line endings.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Currency Converter</title>
7   <script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
8 </head>
9 <body>
10
11 <div id="app">
12   <h2>Currency Converter</h2>
13
14   <div>
15     <label for="amount">Amount in USD:</label>
16     <input type="number" id="amount" v-model.number="usdAmount" @input="convertCurrency">
17   </div>
18
19   <div>
20     <p>Amount in USD: {{ usdAmount }}</p>
21     <p>Amount in EUR: {{ convertedAmount }}</p>
22   </div>
23 </div>
24
```



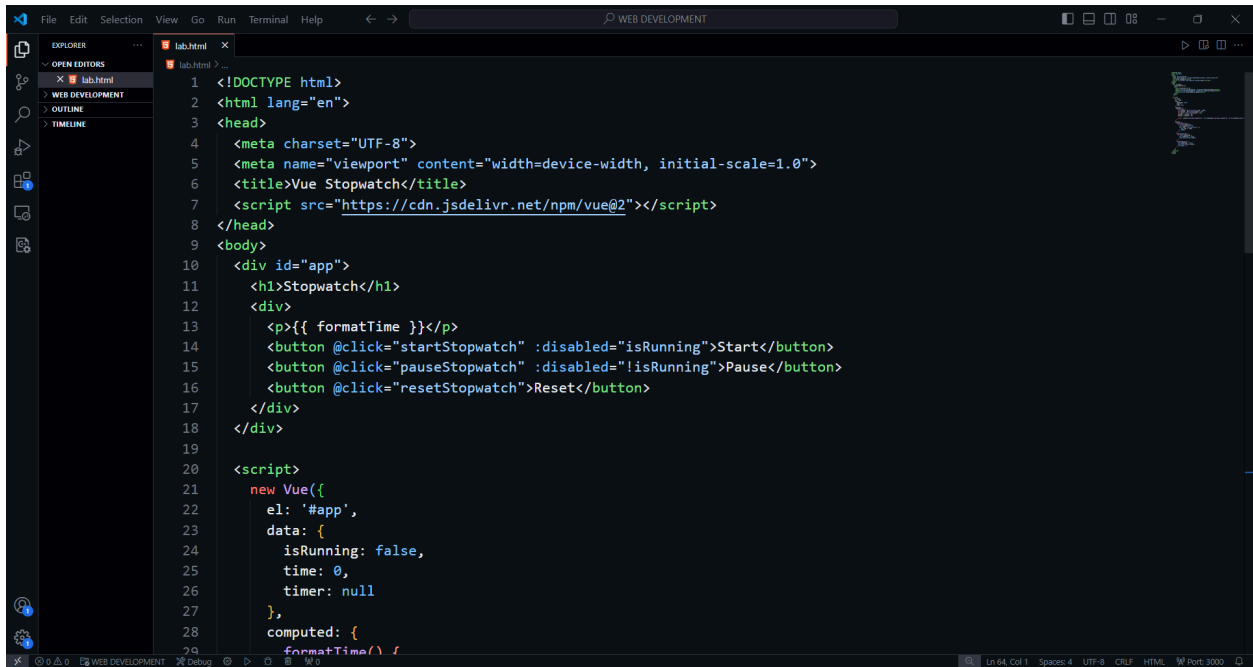
The screenshot shows the Visual Studio Code editor with a file named `lab.html` open. The code is a Vue.js application. The `data` object contains `usdAmount: 0`, `exchangeRate: 0.85`, and `convertedAmount: 0`. The `methods` object contains a `convertCurrency` function that calculates the converted amount based on the current `usdAmount` and `exchangeRate`.

```
2 <html lang="en">
9 <body>
24
25 <script>
26 new Vue({
27   el: '#app',
28   data: {
29     usdAmount: 0,
30     exchangeRate: 0.85,
31     convertedAmount: 0
32   },
33   methods: {
34     convertCurrency() {
35       this.convertedAmount = this.usdAmount * this.exchangeRate;
36     }
37   }
38 });
39 </script>
40
41 </body>
42 </html>
43
```

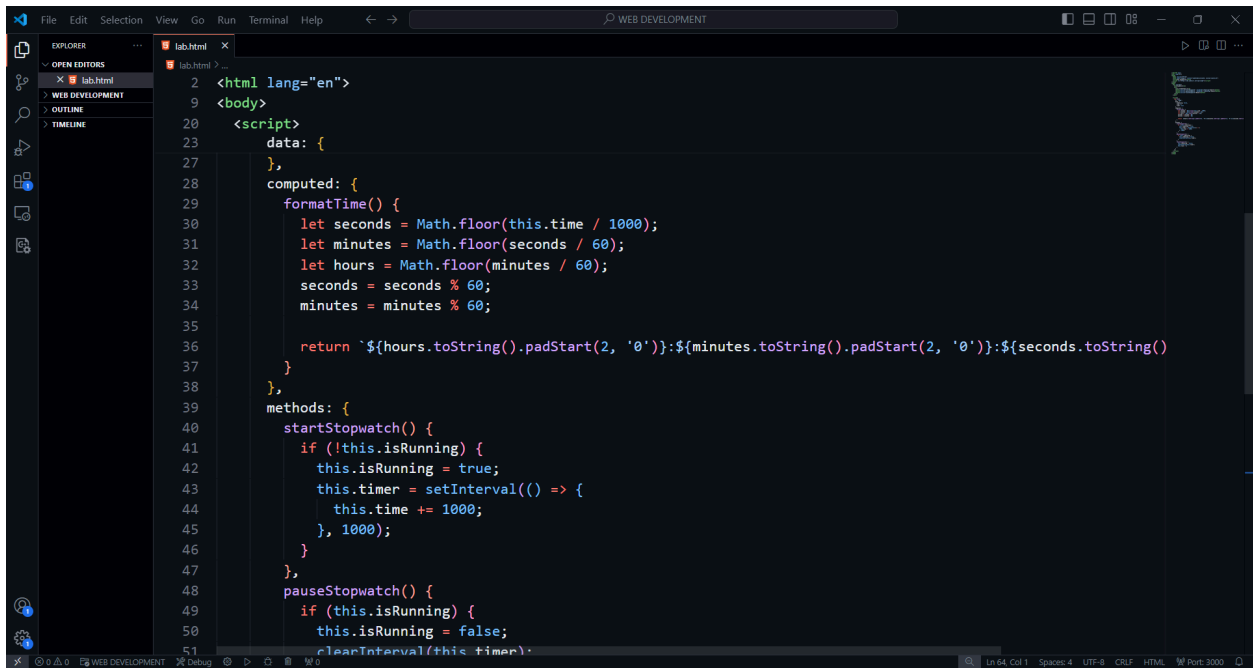


T2. Create a stopwatch application through which users can start, pause and reset the timer.  
Use React state, event handlers and the `setTimeout` or `setInterval` functions to manage the

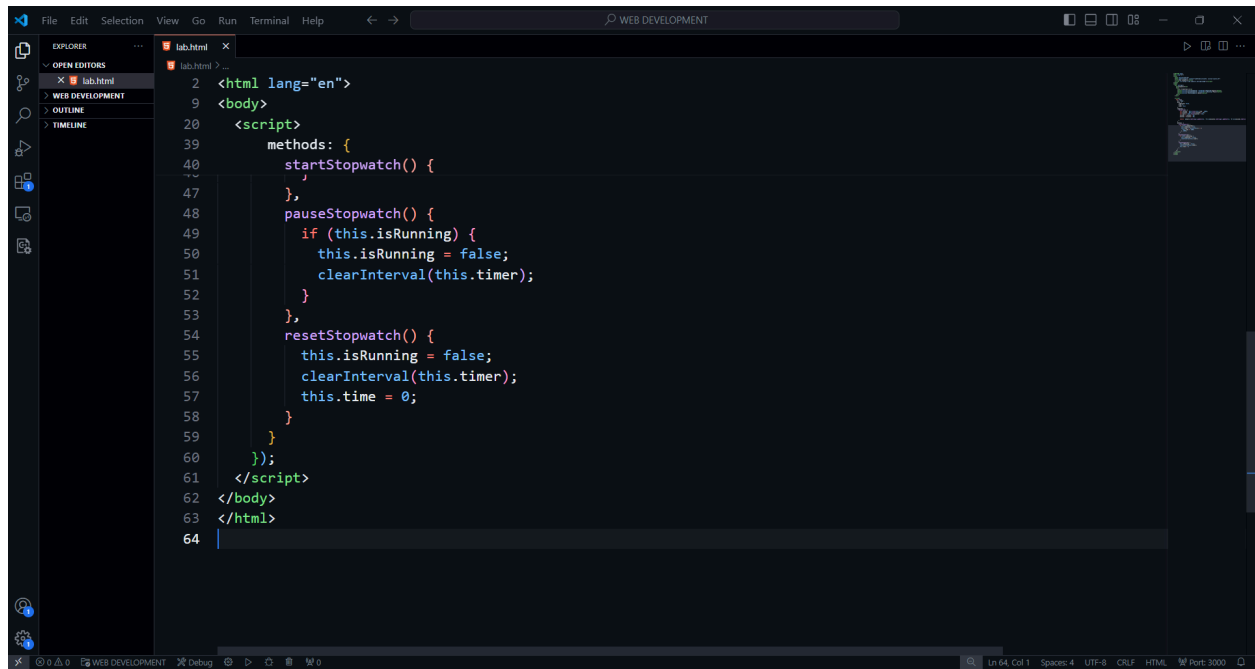
timer's state and actions.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Vue Stopwatch</title>
7   <script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
8 </head>
9 <body>
10   <div id="app">
11     <h1>Stopwatch</h1>
12     <div>
13       <p>{{ formatTime }}</p>
14       <button @click="startStopwatch" :disabled="isRunning">Start</button>
15       <button @click="pauseStopwatch" :disabled="!isRunning">Pause</button>
16       <button @click="resetStopwatch">Reset</button>
17     </div>
18   </div>
19
20   <script>
21     new Vue({
22       el: '#app',
23       data: {
24         isRunning: false,
25         time: 0,
26         timer: null
27       },
28       computed: {
29         formatTime() {
```



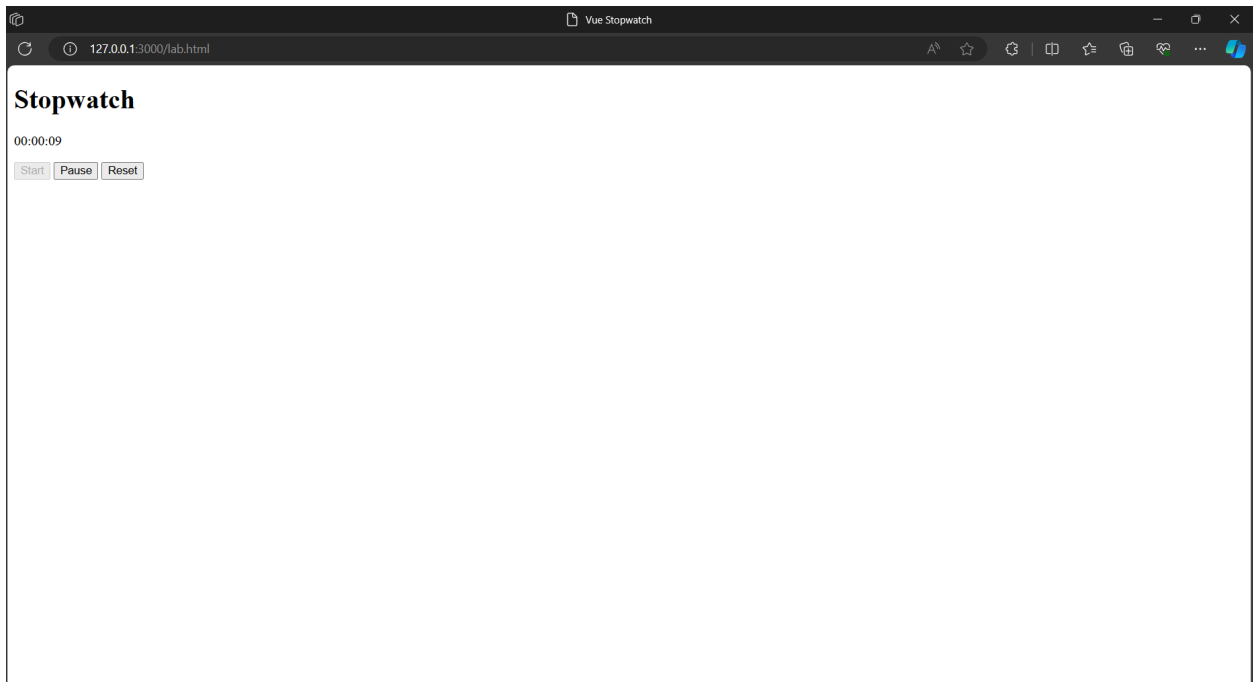
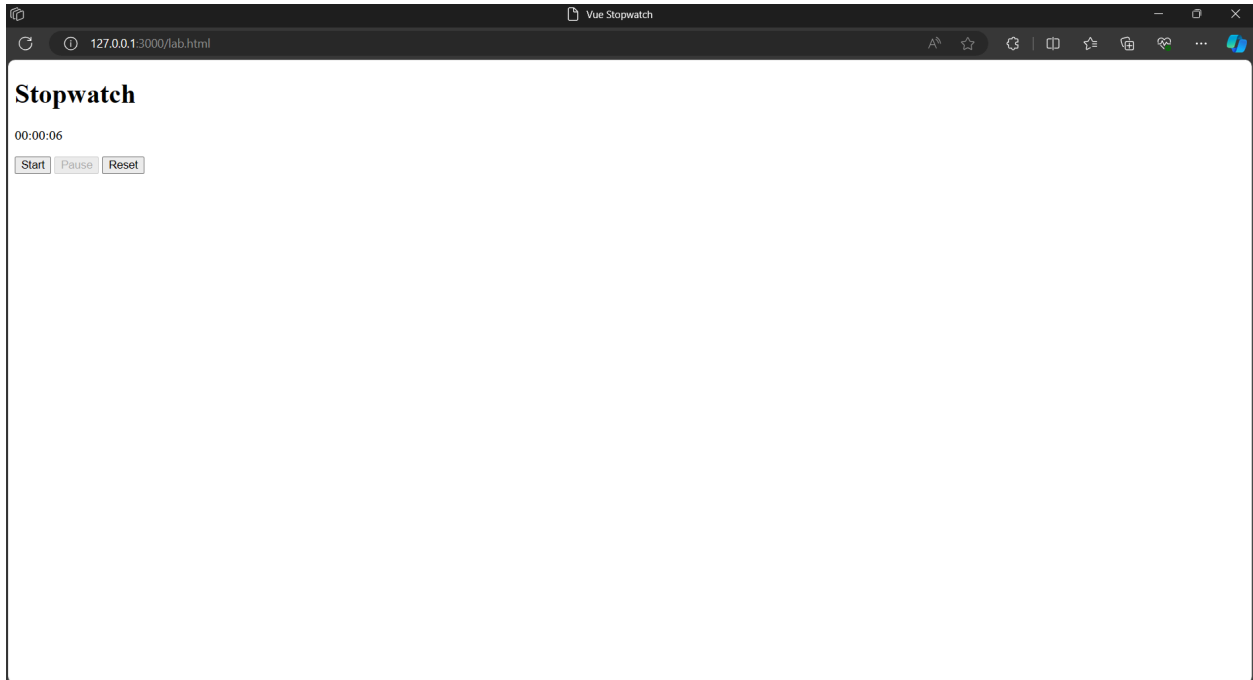
```
30         let seconds = Math.floor(this.time / 1000);
31         let minutes = Math.floor(seconds / 60);
32         let hours = Math.floor(minutes / 60);
33         seconds = seconds % 60;
34         minutes = minutes % 60;
35
36         return `${hours.toString().padStart(2, '0')}:${minutes.toString().padStart(2, '0')}:${seconds.toString()}
37       }
38     },
39     methods: {
40       startStopwatch() {
41         if (!this.isRunning) {
42           this.isRunning = true;
43           this.timer = setInterval(() => {
44             this.time += 1000;
45           }, 1000);
46         }
47       },
48       pauseStopwatch() {
49         if (this.isRunning) {
50           this.isRunning = false;
51           clearInterval(this.timer);
```



The image shows a screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a file named 'lab.html' under the 'WEB DEVELOPMENT' folder. The main editor area displays the content of 'lab.html', which is an HTML document with a JavaScript script for a stopwatch. The code is as follows:

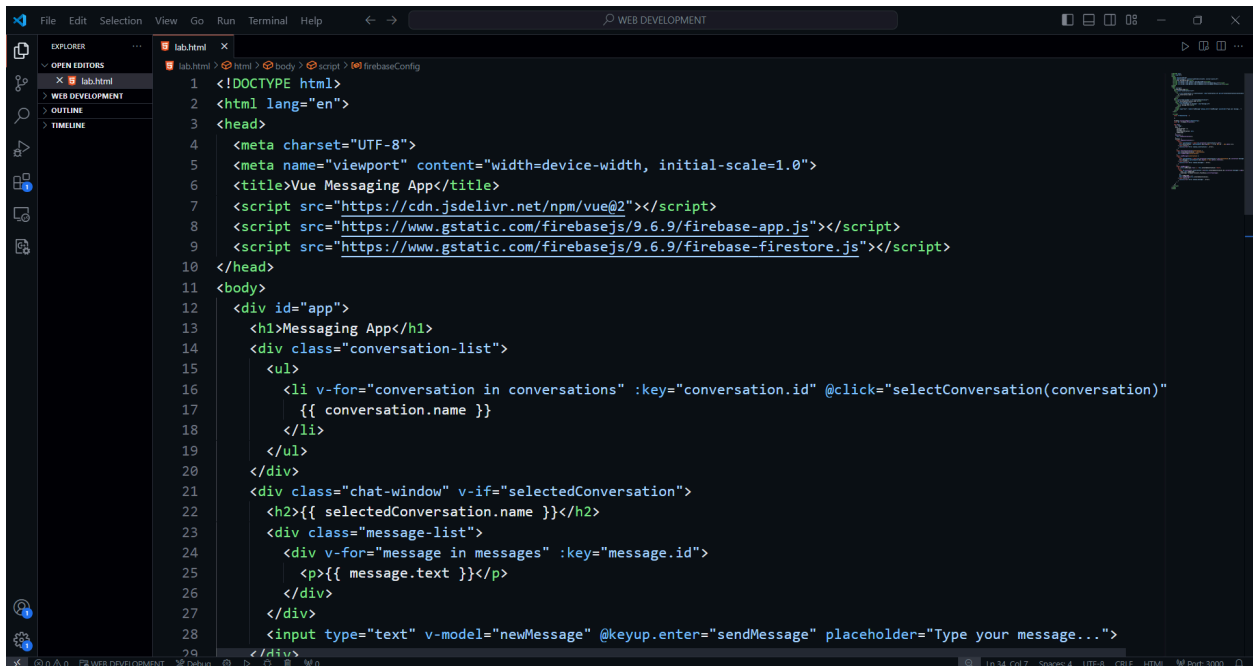
```
2 <html lang="en">
9 <body>
20 <script>
39     methods: {
40         startStopwatch() {
47         },
48         pauseStopwatch() {
49             if (this.isRunning) {
50                 this.isRunning = false;
51                 clearInterval(this.timer);
52             }
53         },
54         resetStopwatch() {
55             this.isRunning = false;
56             clearInterval(this.timer);
57             this.time = 0;
58         }
59     };
60 });
61 </script>
62 </body>
63 </html>
64
```

The status bar at the bottom indicates the file is 'lab.html', the encoding is 'UTF-8', and the line/character count is 'Ln 64, Col 1'. The 'Port 3000' indicator is also visible.

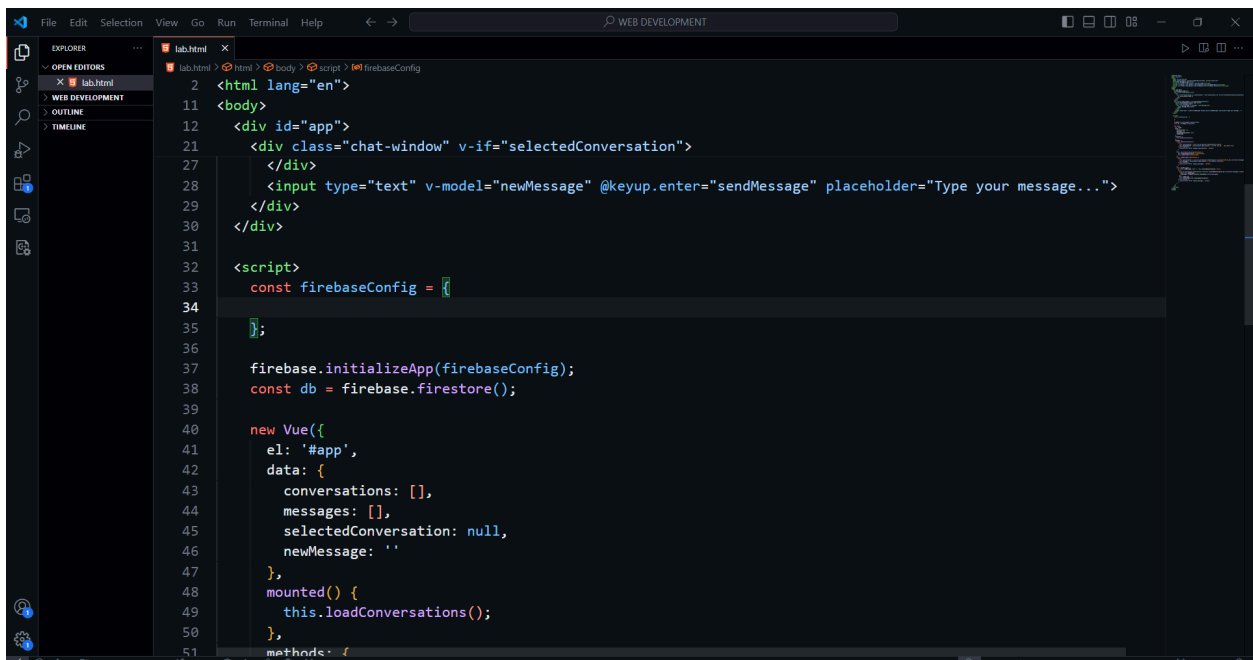


T3. Develop a messaging application that allows users to send and receive messages in real time. The application should display a list of conversations and allow the user to select a

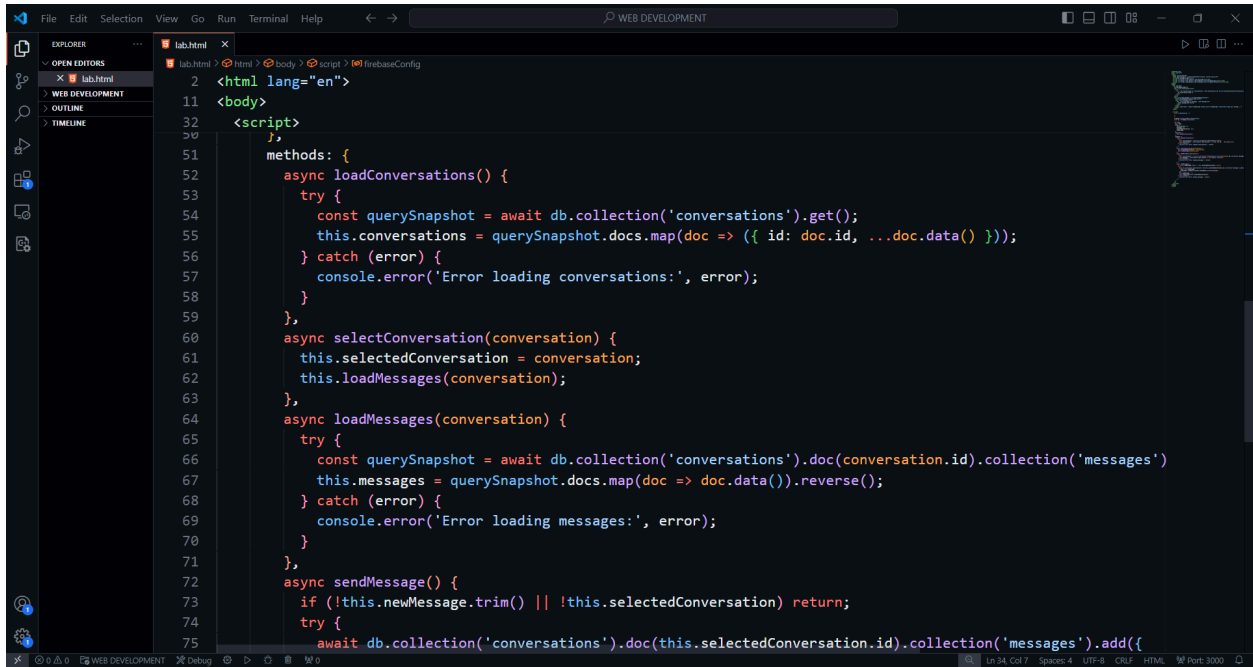
specific conversation to view its messages. The messages should be displayed in a chat interface with the most recent message at the top. Users should be able to send new messages and receive push notifications.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Vue Messaging App</title>
7   <script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
8   <script src="https://www.gstatic.com/firebasejs/9.6.9/firebase-app.js"></script>
9   <script src="https://www.gstatic.com/firebasejs/9.6.9/firebase-firestore.js"></script>
10 </head>
11 <body>
12   <div id="app">
13     <h1>Messaging App</h1>
14     <div class="conversation-list">
15       <ul>
16         <li v-for="conversation in conversations" :key="conversation.id" @click="selectConversation(conversation)">
17           {{ conversation.name }}
18         </li>
19       </ul>
20     </div>
21     <div class="chat-window" v-if="selectedConversation">
22       <h2>{{ selectedConversation.name }}</h2>
23       <div class="message-list">
24         <div v-for="message in messages" :key="message.id">
25           <p>{{ message.text }}</p>
26         </div>
27       </div>
28       <input type="text" v-model="newMessage" @keyup.enter="sendMessage" placeholder="Type your message...">
29     </div>
30   </div>
31 </body>
32 </html>
```

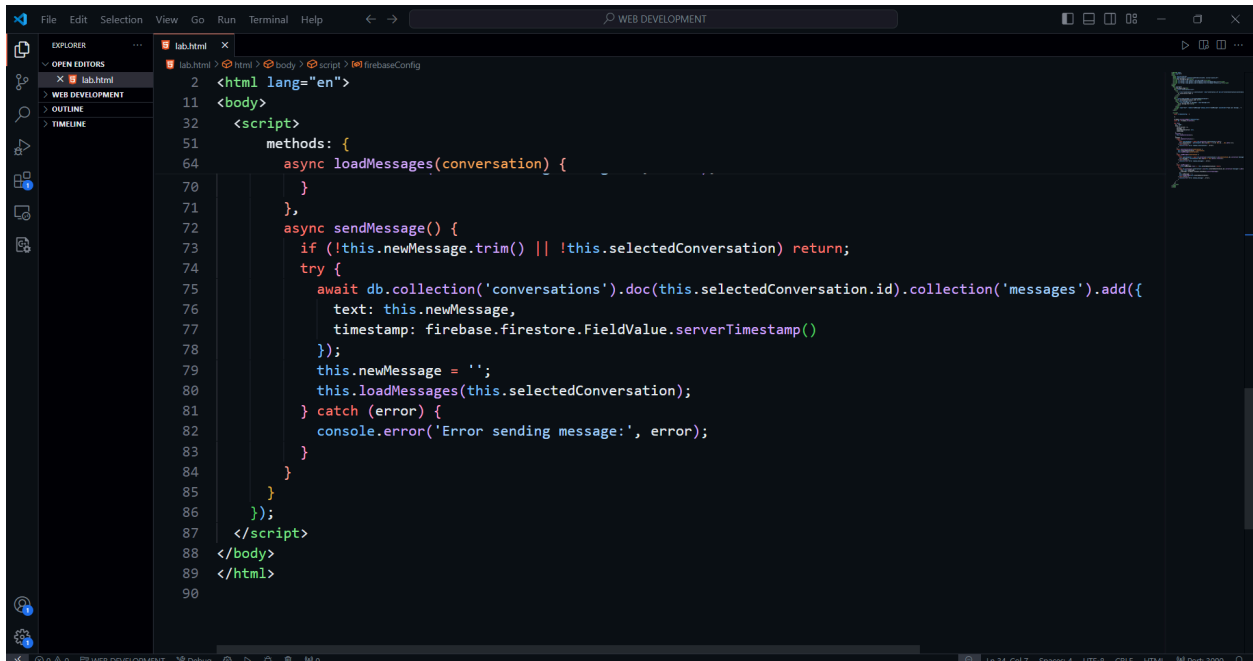


```
2 <html lang="en">
3 <body>
4   <div id="app">
5     <div class="chat-window" v-if="selectedConversation">
6       <input type="text" v-model="newMessage" @keyup.enter="sendMessage" placeholder="Type your message...">
7     </div>
8   </div>
9 </body>
10 </html>
11
12 <script>
13   const firebaseConfig = {
14     // Firebase configuration object
15   };
16
17   firebase.initializeApp(firebaseConfig);
18   const db = firebase.firestore();
19
20   new Vue({
21     el: '#app',
22     data: {
23       conversations: [],
24       messages: [],
25       selectedConversation: null,
26       newMessage: ''
27     },
28     mounted() {
29       this.loadConversations();
30     },
31     methods: {
32       // Methods for the app
33     }
34   });
```



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left shows the project structure with files like `lab.html` and `firebaseConfig`. The main editor area displays the content of `lab.html`, which includes an HTML document with a `<script>` tag containing JavaScript code. The code uses the Firebase SDK to interact with a database, implementing methods for loading conversations, selecting a conversation, loading messages, and sending a new message. The code is as follows:

```
2 <html lang="en">
11 <body>
32 <script>
51 },
52 methods: {
53   async loadConversations() {
54     try {
55       const querySnapshot = await db.collection('conversations').get();
56       this.conversations = querySnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
57     } catch (error) {
58       console.error('Error loading conversations:', error);
59     }
60   },
61   async selectConversation(conversation) {
62     this.selectedConversation = conversation;
63     this.loadMessages(conversation);
64   },
65   async loadMessages(conversation) {
66     try {
67       const querySnapshot = await db.collection('conversations').doc(conversation.id).collection('messages').get();
68       this.messages = querySnapshot.docs.map(doc => doc.data()).reverse();
69     } catch (error) {
70       console.error('Error loading messages:', error);
71     }
72   },
73   async sendMessage() {
74     if (!this.newMessage.trim() || !this.selectedConversation) return;
75     try {
76       await db.collection('conversations').doc(this.selectedConversation.id).collection('messages').add({
```



```
lab.html x
2 <html lang="en">
11 <body>
32 <script>
51   methods: {
64     async loadMessages(conversation) {
70     }
71   },
72   async sendMessage() {
73     if (!this.newMessage.trim() || !this.selectedConversation) return;
74     try {
75       await db.collection('conversations').doc(this.selectedConversation.id).collection('messages').add({
76         text: this.newMessage,
77         timestamp: firebase.firestore.FieldValue.serverTimestamp()
78       });
79       this.newMessage = '';
80       this.loadMessages(this.selectedConversation);
81     } catch (error) {
82       console.error('Error sending message:', error);
83     }
84   }
85 }
86 });
87 </script>
88 </body>
89 </html>
90
```

