```
import {leftPad, formatTimeForDisplay} from './helper.js';

export class TimeSlider {

    constructor(video,div,startTime,duration) {
        this.video = video;
        this.div = div;

        // Time variables
        this.startTime = startTime;
        this.duration = duration;
        this.currentTime = startTime;

        // Drag variables
        this.dragListener;
        this.dragStartPosMouse = 0;
        this.dragStartPosKmob = 0;

        this.paused = true;
        this.onEvents = {};

        this.initialize();
    }

    initialize() {
        // get the slider knob and duration bar
        this.sliderKnob = this.div.getElementsByClassName('sliderKnob')[0];
        this.sliderBack = this.div.getElementsByClassName('sliderBack')[0];
        this.durationBar = this.div.getElementsByClassName('durationBar')[0];
        this.playbackTime = this.div.getElementsByClassName('time')[0];
        this.playbackSpeedSelect = document.getElementById('playbackSpeed');
        this.skipAhead = document.getElementById('skipAhead');
        this.skipBack = document.getElementById('skipBack');
        this.playBut = document.getElementById('play');

        // Set the playback time to the startTime
        this.playbackTime.innerHTML = formatTimeForDisplay(this.startTime);

        // Get the current playback speed and skip distance
        const selectedPlaybackSpeed = this.playbackSpeedSelect.options[this.playbackSpeedSelect.selectedIndex];
        this.playbackSpeed = selectedPlaybackSpeed.value;
        this.skipDistance = Number(selectedPlaybackSpeed.dataset.skipdistance);

        // Add event listeners
        this.sliderKnob.addEventListener('mousedown',this.handleMouseDown.bind(this));
        this.sliderBack.addEventListener('mousedown', this.handleSeek.bind(this));
        this.playbackSpeedSelect.addEventListener('change', this.handlePlayBackSpeedChange.bind(this));
        this.skipAhead.addEventListener('click',this.handleSkip.bind(this));
        this.skipBack.addEventListener('click',this.handleSkip.bind(this));
        this.playBut.addEventListener('click', this.handlePlayPause.bind(this));
        this.video.addEventListener('click',this.handlePlayPause.bind(this));
        this.playbackTime.addEventListener('click',this.handleTimeUpdate.bind(this));

        // Keyboard Events
        addEventListener('keyup', this.handleKeyPress.bind(this))
    }

    handleSeek(e) {
        if (e.target != this.sliderKnob) {
            // This is where the user clicked on the playback bar
            const clickPos = e.offsetX;
```

```
62
63              // Move the slider knob and the duration bar to the point where the user clicked
64              this.sliderKnob.style.left = clickPos + 'px';
65              this.durationBar.style.width = clickPos + 'px';
66
67              // Update the time display with the current time
68              const time = this.getTimeFromKnob();
69              this.currentTime = time;
70              this.updateTimeDisplay(time)
71              this.updateVideo(time,true)
72
73              // Track if the user starts dragging the knob after clicking
74              this.handleMouseDown(e);
75          }
76      }
77
78      handleMouseDown(e) {
79          e.preventDefault();
80          e.stopPropagation();
81
82          // Pause the video
83          this.video.pause();
84
85          // Get the position of the mouse and the knob when the user clicks down
86          this.dragStartPosMouse = e.clientX;
87          this.dragStartPosKnob = this.sliderKnob.offsetLeft;
88
89          // Bind the handle mouse up and drag functions to this (we do this here instead of in addeventlistener so we
90  can remove them later)
91          this.handleMouseUpBound = this.handleMouseUp.bind(this);
92          this.handleDragBound = this.handleDrag.bind(this);
93
94          // Add the event listeners
95          document.addEventListener('mousemove',this.handleDragBound);
96          document.addEventListener('mouseup',this.handleMouseUpBound);
97      }
98
99      handleMouseUp(e) {
100         if (!this.paused) this.video.play();
101
102         // If the user lets go of hte mouse remove the mouse up and mouse move event listeners
103         document.removeEventListener('mouseup',this.handleMouseUpBound)
104         document.removeEventListener('mousemove',this.handleDragBound)
105     }
106
107     handleDrag(e) {
108         e.preventDefault();
109         e.stopPropagation();
110
111         // Determine how far the user has moved the mouse from the original position
112         const delta = e.clientX - this.dragStartPosMouse;
113
114         // Calculate where the knob should be positioned
115         const newPos = this.dragStartPosKnob + delta + this.sliderKnob.offsetWidth / 2;
116
117         // If the calcualted position isn't beyond the edges of the slider back move the knob and duration bar to the
118 new position
119         if (newPos >= 0 && newPos <= this.sliderBack.offsetWidth) {
120             this.sliderKnob.style.left = newPos + 'px';
121             this.durationBar.style.width = newPos + 'px';
122         }
123
```

```
124         // Update the time display with the new time
125         const time = this.getTimeFromKnob();
126         this.currentTime = time;
127         this.updateTimeDisplay(time)
128         this.updateVideo(time)
129 
130         return true;
131     }
132 
133     handlePlayBackSpeedChange() {
134         this.playbackSpeed = this.playbackSpeedSelect.value;
135         this.skipDistance =
136 Number(this.playbackSpeedSelect.options[this.playbackSpeedSelect.selectedIndex].dataset.skipdistance);
137 
138         this.skipAhead.innerHTML = "+" + this.skipDistance;
139         this.skipBack.innerHTML = "-" + this.skipDistance;
140 
141         this.video.playbackRate = this.playbackSpeed;
142     }
143 
144     handleSkip(e) {
145         const skipDistance = e.target.id == "skipAhead" ? this.skipDistance : this.skipDistance * -1;
146         this.setTime(new Date(this.currentTime.getTime() + skipDistance * 1000),true);
147     }
148 
149     handlePlayPause() {
150         console.log("handle play pause");
151 
152         if (this.video.paused) {
153             this.paused = false;
154             this.playBut.classList.remove("paused");
155             this.video.play();
156         }
157         else {
158             this.paused = true;
159             this.playBut.classList.add("paused");
160             this.video.pause();
161         }
162     }
163 
164     handleKeyPress(e) {
165         // Space bar
166         if (e.keyCode == 32) {
167             this.handlePlayPause();
168         }
169         // left arrow (37) or right arrow (39)
170         else if (e.keyCode == 39 || e.keyCode == 37) {
171             const direction = e.keyCode == 39 ? 1 : -1;
172             this.setTime(new Date(this.currentTime.getTime() + direction * this.skipDistance * 1000),true);
173         }
174         else if (e.keyCode == 188 || e.keyCode == 190) {
175             // Pause the video
176             this.paused = true;
177             this.video.pause();
178 
179             // Advance frame by frame
180             const direction = e.keyCode == 190 ? 1 : -1;
181             const newTime = new Date(this.currentTime.getTime() + (direction / 60 * 1000));
182             this.setTime(newTime,true)
183 
184         }
185         else if (e.keyCode == 187 || e.keyCode == 189) {
```

```
186                 const increment = e.keyCode == 187 ? -1 : 1;
187                 const currentIndex = this.playbackSpeedSelect.selectedIndex;
188                 const newIndex = currentIndex + increment;
189 
190                 if (newIndex >= 0 && newIndex < this.playbackSpeedSelect.length) {
191                     this.playbackSpeedSelect.value = this.playbackSpeedSelect[newIndex].value;
192                     this.handlePlayBackSpeedChange();
193                 }
194         }
195         else {
196             // console.log(e.keyCode);
197         }
198     }
199 
200     handleTimeUpdate(e) {
201 
202         const newTime = prompt("Enter time in format xx:xx:xx");
203 
204         let match = /(\d{1,2}):(\d{2}):(\d{2})/.exec(newTime);
205         if (match) {
206 
207             let [_,h,m,s] = match;
208 
209             // Create a new date with the same date but time set to the new time
210             let newTime = new Date(this.currentTime.getTime());
211             newTime.setHours(Number(h));
212             newTime.setMinutes(Number(m));
213             newTime.setSeconds(Number(s));
214 
215             // Calculate the difference in ms between the new time and the current time
216             let delta = newTime.getTime() - this.currentTime.getTime();
217 
218             // Update the start by incrementing it by the delta
219             this.startTime = new Date(this.startTime.getTime() + delta);
220 
221             // Update the current time with the new time and display it
222             this.currentTime = newTime;
223             this.updateTimeDisplay(newTime);
224 
225             // If there's an onNewStartTime listener then call it
226             if (this.onNewStartTime) this.onNewStartTime(this.startTime);
227         }
228 
229     }
230 
231     updateTimeDisplay(time) {
232         this.playbackTime.innerHTML = formatTimeForDisplay(time);
233     }
234 
235     getTimeFromKnob() {
236         const percent = (this.sliderKnob.offsetLeft + this.sliderKnob.offsetWidth / 2) / this.sliderBack.offsetWidth;
237         return new Date(this.startTime.getTime() + this.duration * percent);
238     }
239 
240     getTimeFromStart(time) {
241         return time.getTime() - this.startTime.getTime();
242     }
243 
244     setTime(newTime, updateVideo) {
245 
246         let timeSinceStart = 0;
247 
```

```
        if (newTime instanceof Date) {
            this.currentTime = newTime;
            timeSinceStart = this.getTimeFromStart(newTime);
        }
        else {
            timeSinceStart = newTime * 1000;
            this.currentTime = new Date(this.startTime.getTime() + timeSinceStart);
            newTime = this.currentTime;
        }

        // Don't let the time be before the movie starts or after it ends
        if (timeSinceStart < 0) timeSinceStart = 0;
        if (timeSinceStart > this.duration) timeSinceStart = this.duration;

        // Calculate the new position for the knob
        const percent = timeSinceStart / this.duration;
        const newPos = this.sliderBack.offsetWidth * percent;

        // Position the knob and duration slider
        this.sliderKnob.style.left = newPos + 'px';
        this.durationBar.style.width = newPos + 'px';

        // Update the time
        this.updateTimeDisplay(newTime);

        // If updateVideo is true then update the video to the new time
        if (updateVideo) {
            this.updateVideo(newTime);
        }
    }

    formatPlaybackTime(time) {
        return `${leftPad(time.getHours(),2,0)}:${leftPad(time.getMinutes(),2,0)}:${leftPad(time.getSeconds(),2,0)}`;
    }

    updateVideo(time) {
        const timeSinceStart = this.getTimeFromStart(time);
        this.video.currentTime = timeSinceStart/1000;

        if (this.onTimeUpdate) this.onTimeUpdate(time);
    }

}
```