```

import {makeHttpRequest} from './helper.js';

export class Log {

    constructor(logType, startTime = new Date(), fileName, id, weather, notes) {

        // If starttime is a string try and create a date from it and then assign it.
        // Otherwise it's a date and it can be assigned directly.
        this.startTime = typeof startTime == "string" ? new Date(startTime) : startTime;

        this.logType = logType;
        this.data = [];
        this.currentCount = 0;
        this.fileName = fileName;
        this.id = id;
        this.weather = weather;
        this.notes = notes;
        this.redoCache = [];
        this.syncCache = [];
    }

    addData(count, time = new Date()) {
        count = parseInt(count);
        this.currentCount += count;

        const delta = time.getTime() - this.startTime.getTime();

        // Get the index where we should insert the new entry
        let insertIndex = this.data.findIndex(entry => entry.time >= delta)
        if (insertIndex < 0) insertIndex = this.data.length;

        // Insert the entry and add it to the sync cache
        let entry = {time: delta, count};
        this.data.splice(insertIndex,0,entry);
        this.syncCache.push(entry);

        this.redoCache = [];

        this.syncWithDb();
    }


    getDataByTimeInterval(startTime,duration) {

        const startDelta = startTime.getTime() - this.startTime.getTime();
        const endDelta = startDelta + duration;

        var startIndex = this.data.findIndex(data => data.time >= startDelta)

        var endIndex = this.data.findIndexFrom(data => data.time >= endDelta, startIndex);
        if (endIndex < 0) endIndex = this.data.length;

        return this.data.slice(startIndex,endIndex)
            .map(entry => {
                return {count: entry.count, time: new Date(this.startTime.getTime() + entry.time)}
            });

    }


```

```
62      incrementTime(increment) {
63          this.startTime = new Date(this.startTime.getTime() + increment);
64
65          let data = {
66              'date': this.startTime
67          }
68
69          makeHttpRequest(`api/logs/${this.id}`,'PATCH',JSON.stringify(data),'application/json')
70          .then(res => {
71              console.log(res);
72          })
73          .catch(error => {
74              console.log("error", error)
75          });
76
77      }
78
79
80      addLogToDb() {
81
82          this.addingToDb = true;
83
84          return new Promise((resolve, reject) => {
85
86              // Grab the data that hasn't been uploaded yet and set its upload status to pending
87              let data = this.syncCache;
88              data.forEach(entry => {entry.uploaded = "pending"})
89
90              // Build the object that will be sent to the API
91              let body = {
92                  'logType': this.logType,
93                  'date': this.startTime,
94                  'fileName': this.fileName,
95                  'weather': this.weather,
96                  'notes': this.notes,
97                  'data': this.syncCache
98              }
99
100             makeHttpRequest('api/logs','POST',JSON.stringify(body),'application/json')
101             .then(res => {
102                 res = JSON.parse(res);
103                 this.id = res.logId;
104                 this.addingToDb = false;
105
106                 data.forEach(entry => {entry.uploaded = true});
107
108                 // Remove successfully uploaded entries from the sync cache
109                 this.syncCache = this.syncCache.filter(entry => entry.uploaded != true);
110
111                 console.log("log id: "+this.id);
112                 console.log(this.syncCache);
113
114                 if (this.onLogAddedToDb) this.onLogAddedToDb(this);
115
116                 resolve(res);
117             })
118             .catch(error => {
119                 this.addingToDb = false;
120                 console.log("error in addlogtodb", error)
121                 reject(error);
122             });
123
```

```
124         });
125 
126     }
127 
128     syncWithDb() {
129         // If there isn't an ID and we're not already trying to get the ID add the log to the database
130         if (!this.id && !this.addingToDb) {
131             this.addLogToDb()
132                 .then(() => {
133                     this.syncWithDb();
134                 })
135                 .catch(error => {
136                     console.log("error in syncwith db", error);
137                 })
138         }
139         // Otherwise add the new entries
140         else if (this.id && !this.addingToDb) {
141             // Get the entries that haven't been uploaded yet
142             let data = this.syncCache.filter(entry => !entry.uploaded)
143 
144             if (data.length) {
145                 //  Add a pending state to each entry
146                 data.forEach(entry => {
147                     entry.uploaded = "pending";
148                 })
149 
150                 makeHttpRequest(`api/logs/${this.id}`,'POST',JSON.stringify(data),'application/json')
151                 .then(res => {
152                     res = JSON.parse(res);
153 
154                     // Update each entry with the uploaded status (true if the call was successful, false otherwise)
155                     data.forEach(entry => {
156                         entry.uploaded = res.success ? true : false;
157                     })
158 
159                     // Remove the entries from the sync cache that were uploaded
160                     this.syncCache = this.syncCache.filter(entry => entry.uploaded != true);
161 
162                     console.log(this.syncCache);
163 
164                 })
165                 .catch(error => {
166                     data.forEach(entry => {
167                         entry.uploaded = false;
168                     })
169 
170                     console.log("error", error)
171                 });
172             }
173         }
174     }
175 
176 
177     getCountByInterval(interval) {
178 
179         var intervalData = [];
180 
181         if (this.data.length > 0 && interval) {
182 
183             var interval = interval * 1000;
184             var data = this.data.slice(0);
185 
```

```
            data = data.map(entry => {
                return {count: entry.count, time: new Date(this.startTime.getTime() + Number(entry.time))}
            });

            // Get the starting time by rounding the first time down to the nearest inteval
            var currentInterval = new Date(Math.floor(data[0].time / interval) * interval);
            var currentCount = 0;

            // While there are still items in the data array
            while (data.length > 0) {

                // If the entry is in the current interval add it to the count
                if (data[0].time < new Date(currentInterval.getTime() + interval)) {
                    var count = data.shift().count;
                    currentCount += count;
                }

                // If the entry isn't in the current interval
                else {
                    // Push the previous time / count to the intervalData array
                    intervalData.push({time: currentInterval, count: currentCount})

                    // Increment the current interval by the interval time and reset the count to 0
                    currentInterval = new Date(currentInterval.getTime() + interval);
                    currentCount = 0;
                }

            }

            // Add the last time to the intervalData array
            intervalData.push({time: currentInterval, count: currentCount})

        }

        return intervalData;

    }

    getTotalByInterval(interval) {

        var runningTotal = 0;

        // Get the interval counts
        var intervals = this.getCountByInterval(interval);

        // Iterate through the intervals
        intervals.forEach(function(interval) {
            // Add the interval's count to the running total
            runningTotal += interval.count;
            // Add the running total to each interval
            interval.runningTotal = runningTotal;
        })

        return intervals;


    }
```

```
248
249     static fromId(id) {
250
251         return new Promise((resolve, reject) => {
252
253             makeHttpRequest(`api/logs/${id}`,'GET')
254             .then(res => {
255                 res = JSON.parse(res);
256
257                 console.log(res);
258
259                 let log = new Log(new Date(res.start_time), res.file_name, res.id);
260                 log.data = res.entries.map(entry => {
261                     return {count: Number(entry.count), time: Number(entry.time)}
262                 });
263
264                 resolve(log);
265
266             })
267
268         });
269
270     }
271
272
273     static async fromId2(id) {
274
275             var res = await makeHttpRequest(`api/logs/${id}`,'GET');
276
277             res = JSON.parse(res);
278
279             let log = new Log(res.log_type, new Date(res.start_time), res.file_name, res.log_id, res.weather,
280 res.notes);
281             log.data = res.entries.map(entry => {
282                 return {count: Number(entry.count), time: Number(entry.time), uploaded: true}
283             });
284
285             log.currentCount = log.data.reduce((total, entry) => {return total += entry.count}, 0);
286
287             return log;
288
289     }
290
291
292
293
294 }
295
296 Array.prototype.findLastIndex = function(test) {
297
298     for (var i = this.length-1; i >= 0; i--) {
299         if (test(this[i]))
300             return i;
301     }
302
303     return 0;
304
305 }
306
307
308 Array.prototype.findIndexFrom = function(test, startIndex = 0) {
309
```

```
310     if (startIndex < 0) startIndex = 0;
311
312     for (var i = startIndex; i < this.length; i++) {
313         if (test(this[i]))
314             return i;
315     }
316
317     return -1;
318 }
319
320
321 export class OfflineLog extends Log {
322
323     addData(count, time = new Date()) {
324         count = parseInt(count);
325         this.currentCount += count;
326         const delta = time.getTime() - this.startTime.getTime();
327
328         // Get the index where we should insert the new entry
329         let insertIndex = this.data.findIndex(entry => entry.time >= delta)
330         if (insertIndex < 0) insertIndex = this.data.length;
331
332         // Insert the data
333         this.data.splice(insertIndex,0,{time: delta, count})
334
335         this.redoCache = [];
336         this.saveInProgress();
337     }
338
339     generateLocalStorageObject() {
340         let obj = {};
341         obj.startTime = this.startTime;
342         obj.data = this.data;
343         obj.logType = this.logType;
344         return obj;
345     }
346
347     generateDBInsertObject() {
348         // Build the object that will be sent to the API
349         let body = {
350             'logType': this.logType,
351             'date': this.startTime,
352             'fileName': this.fileName,
353             'weather': this.weather,
354             'notes': this.notes,
355             'data': this.data
356         }
357         return body;
358     }
359
360     saveInProgress() {
361         localStorage.setItem('currentLog', JSON.stringify(this.generateLocalStorageObject()));
362     }
363
364     undo() {
365         let entry = this.data.pop();
366
367         if (entry) {
368             this.currentCount -= entry.count;
369             this.redoCache.push(entry);
370         }
371
```

```
372         this.saveInProgress();
373     }
374 
375     redo() {
376         let entry = this.redoCache.pop();
377 
378         if (entry) {
379             this.currentCount += entry.count;
380             this.data.push(entry);
381         }
382 
383         this.saveInProgress();
384     }
385 
386     static fromData(data) {
387         let log = new OfflineLog(data.logType, data.startTime, undefined, undefined, data.weather, data.notes);
388         log.data = data.data;
389         log.currentCount = data.data.reduce((total, entry) => {return total += entry.count}, 0);
390 
391 
392         return log;
393     }
394 
}
```