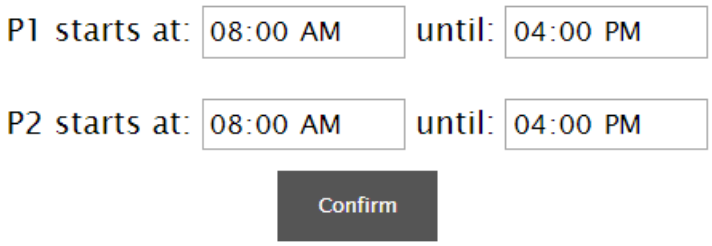
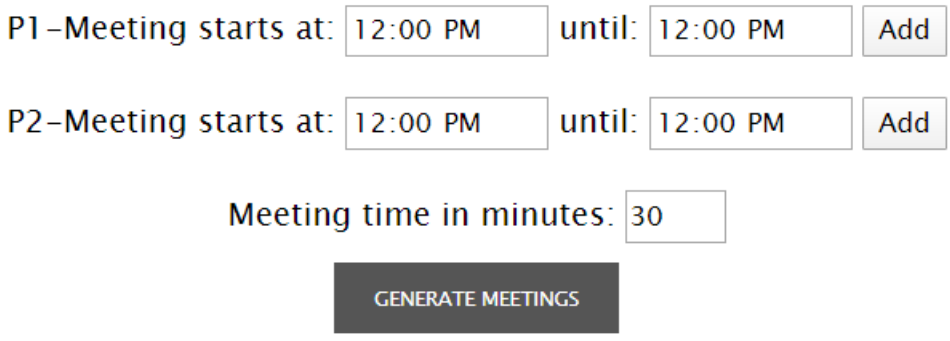
I have decided to give this problem a little bit of visual experience, and that's why I created a simple web app, check it out : https://tonicaia.github.io/calendar-problem/ . I know that the design is poor, but I tried to keep it as simple as possible, and focus more on functionality. JavaScript is not the programming language that I feel most comfortable with, but I wanted to give it a try, because I plan to have more projects in the web area.

1.DATA INPUTS



The user can set each of the calendar's limits (min and max range) from the form above. When he clicks on confirm this area gets disabled. I have decided to merge these 2 intervals into a smaller one (intersection). Example: P1: 8:30 -> 16:00 and P2: 8:00 -> 17:00 => 8:30 -> 16:00. They can't meet if they are not both available = > the new range [maxStart,minEnd];



From this form the user can add the booked (unavailable) time for each person, the time they already have something else going on. If he clicks on Add after some **validations**, the interval gets pushed into bookedTime[] as [[start0,end0], … , [startn,endn]] (n represents the number of valid inputs). Another mention here; I considered the following case : if the interval that has been submitted has one of the start or end values not in the [maxStart,minEnd] range they are still added but the interval gets shorter.

```
if(start < maxStart){
        bookedTime.push([maxStart,end]);
   }
if(end> minEnd){
    bookedTime.push([start,minEnd]);
}
```

The required time for the meeting that will have place can also be set from here, by default it is 30 minutes.

I used for time<->numerical conversion 2 functions. I read data as time, work with it in a numerical form (converted in minutes), and after that display it as time;

```
function toNum(hour,min){
    return parseInt(hour*60)+parseInt(min);
}
function toTime(val){
    return Math.floor(val/60)+":"+val%60;
}
```

If the input is not correct the user gets the message: ”Invalid times!” and if it is “ Booked !”. The input can be wrong in these cases:

```
function verify(start, end,p){
    if(end>start){
       if(p==1 && (start<p1Calendar[0] || end>p1Calendar[1]))
           return false;
        if(p==2 && (start<p2Calendar[0] || end>p2Calendar[1]))
           return false;
        return true;
    }


    return false;

}
```

, where pXCalendar[0] = starting hour for person x and pXCalendar[1] = ending hour for person x.

2. Working with the intervals

At this point I got all the intervals stored in bookedTime[]. I need to check if there are any overlaps,

for example(13:30->13:50 and 13:40->14:00 can be merged into 13:30->14:00).

First thing that I must do is to sort them depending on the start of each, in ascending order.

```
 sortedBookedTime = bookedTime.sort((a,b)=>{
    if (a[0] === b[0]) {
    return 0;
}
else {
    return (a[0] < b[0]) ? -1 : 1;
}
});
```

I need a new array for merged intervals that I initialize with the first one of the sorted ones. Then I go through each sorted meeting and check if I can merge it with the one before it ; otherwise I just add it to the merged array.

```
const mergedIntervals = [sortedBookedTime[0]];

for(var i=1;i<sortedBookedTime.length;i++){
    const lastMerged=mergedIntervals[mergedIntervals.length-1];
    if(sortedBookedTime[i][0] <= lastMerged[1]){
        lastMerged[1] = Math.max(lastMerged[1],sortedBookedTime[i][1]);
    }
    else{
        mergedIntervals.push(sortedBookedTime[i]);
    }
}
```

3. Displaying all the possible meetings

A special case here would be if there are 0 meetings booked => they can meet any time in [maxStart,minEnd] range.

A simple iteration through all the merged meetings is enough.

```
    let lastPos = maxStart;

for(var i =0 ;i <mergedMeetings.length;i++){
    if(mergedMeetings[i][0]-lastPos>=reqMeetTime){
     listForPrint.push(toTime(lastPos)+"->"+toTime(mergedMeetings[i][0]));
    }
    lastPos=mergedMeetings[i][1];
    if(i==mergedMeetings.length-1){
        if(minEnd-mergedMeetings[i][1]>=reqMeetTime){
         listForPrint.push(toTime(mergedMeetings[i][1])+"->"+toTime(minEnd));
        }
    }
}
```

I can check if there is enough time between to meetings for the required meeting time set before. I need to take in consideration also the ranges [maxStart->firstStart] and [lastEnd->minEnd] .