Problem Statement:

There are 80 seats in a coach of a train with only 7 seats in a row and the last row of only 3 seats. For simplicity, there is only one coach in this train. One person can reserve up to 7 seats at a time. If a person is reserving seats, the **priority will be to book them in one row**. 4. If seats are not available in one row then the booking should be done in such a way that the **nearby seats** are booked. Users can book as many tickets as s/he wants until the coach is full. You don't have to create login functionality for this application.

Input / Output:

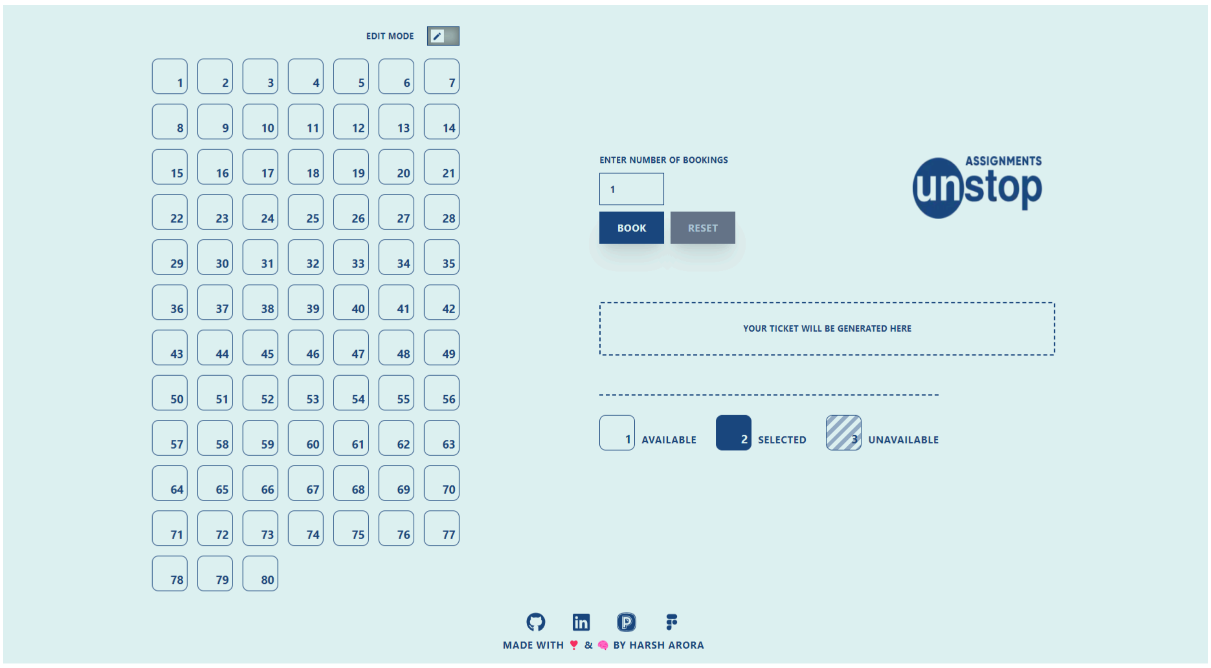
Input required will only be the required number of seats. Example: 2 or 4 or 6 or 1 etc. Output should be seat numbers that have been booked for the user along with the display of all the seats and their availability status through color or number or anything else that you may feel fit.

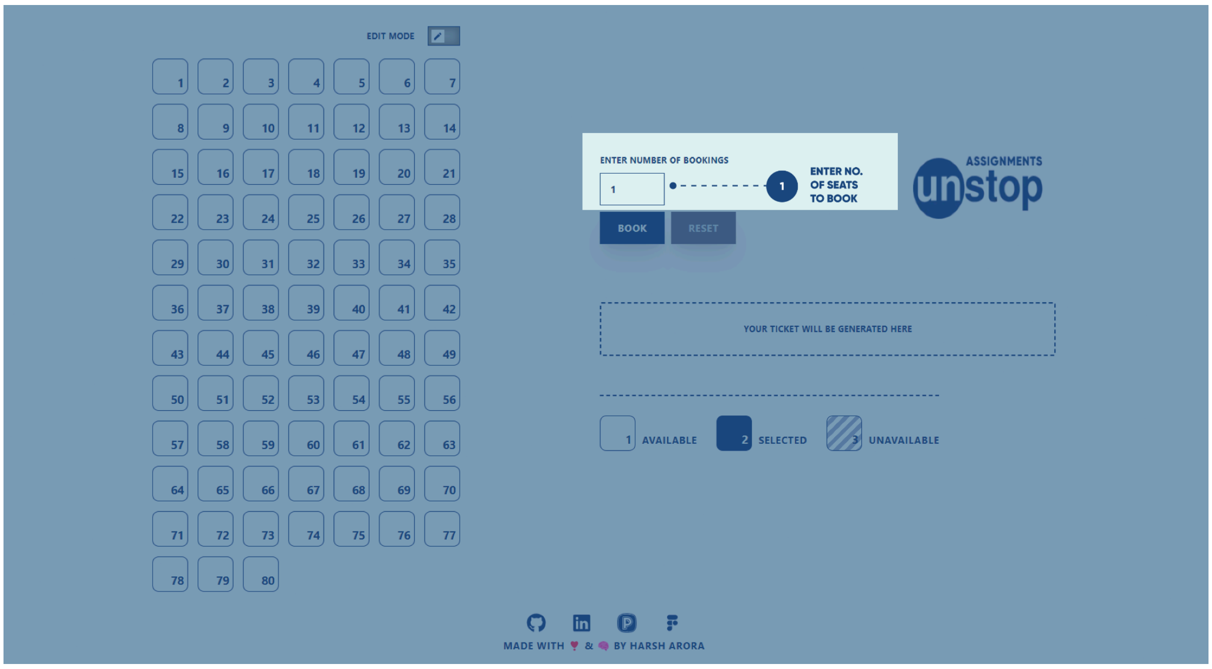
Solution 🔗:

To visualize the problem and its solution I built a Next.js application with minimal user interface that allows a tester to edit train configuration at any stage and book upto 7 seats in a session.
Simply put, a user can enter how many seats he wants to book (from 1 to 7), based on that my algo books the best possible seat allocation in an optimized way rather than trying all possible combinations. My solution is based on a greedy approach trying to minimize intra and inter row spreads.

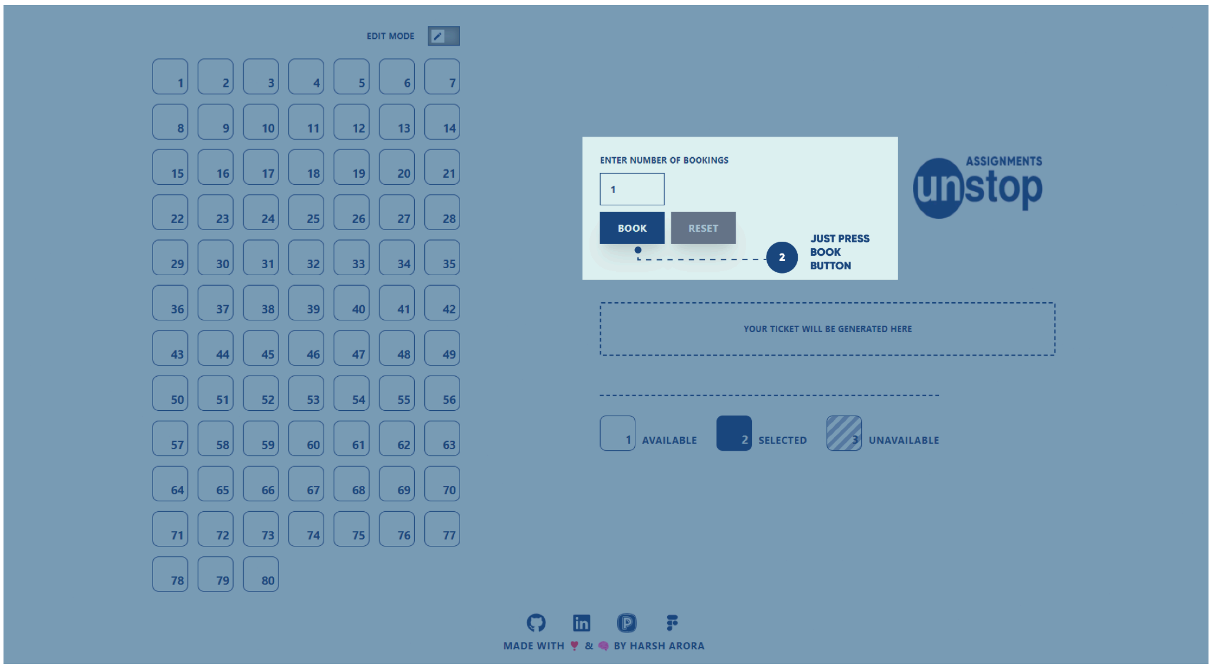
Application: link
Code: link
Design: link

Application Overview:

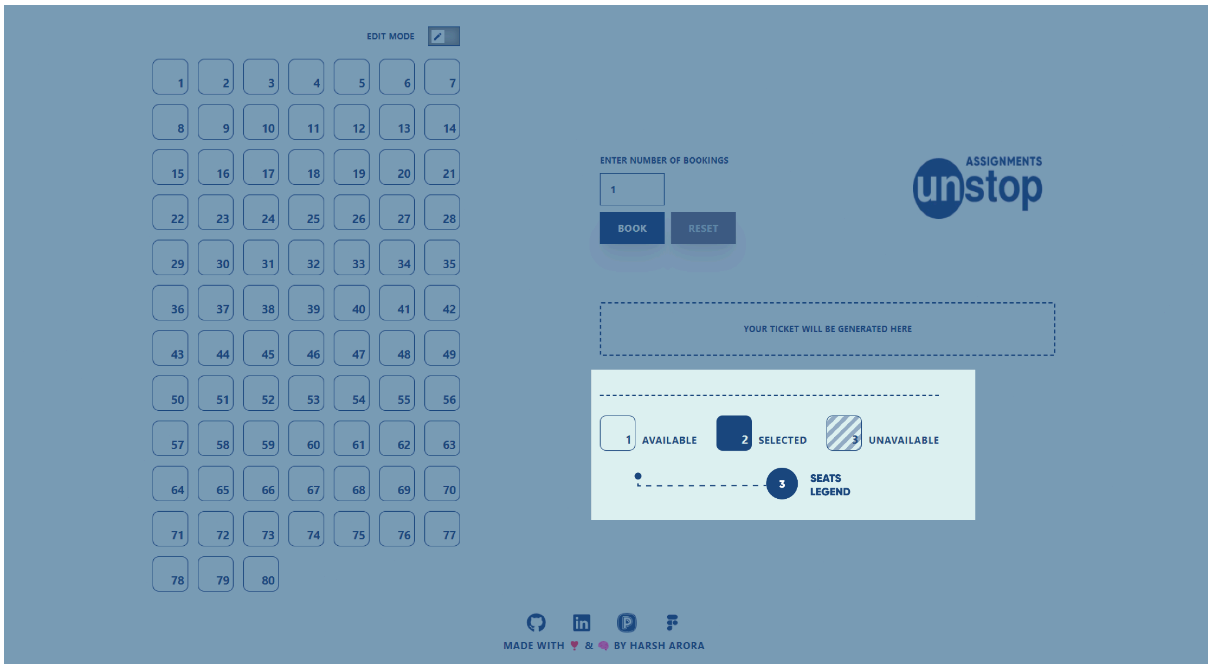




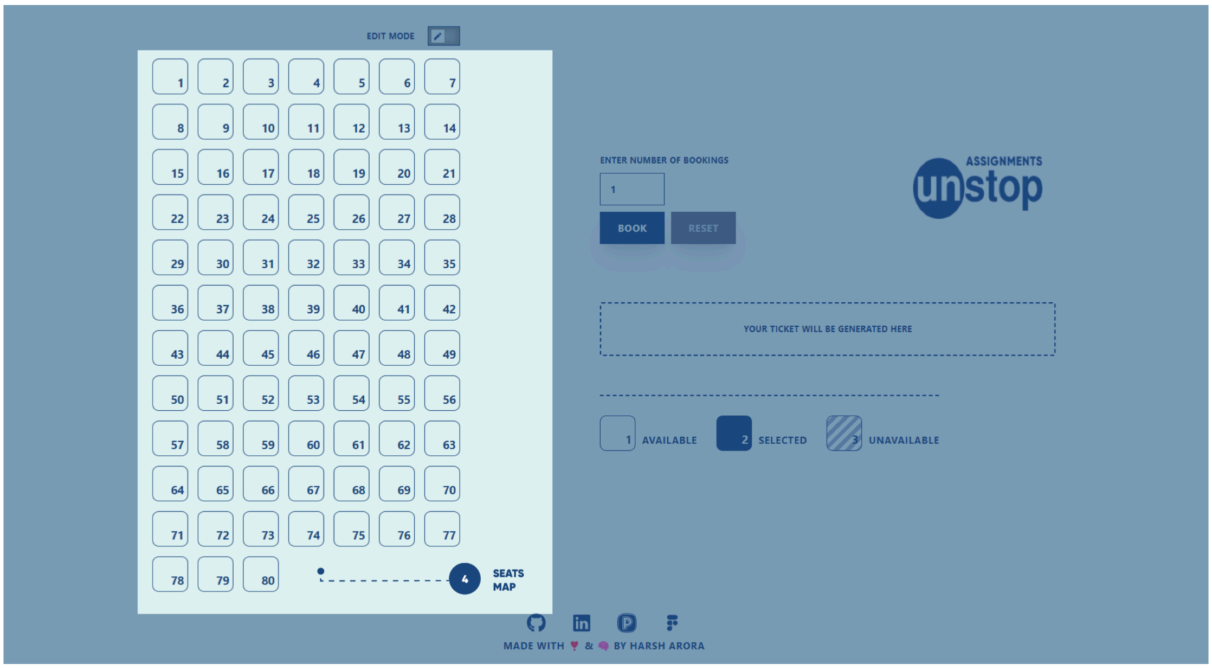
1. Enter the input.



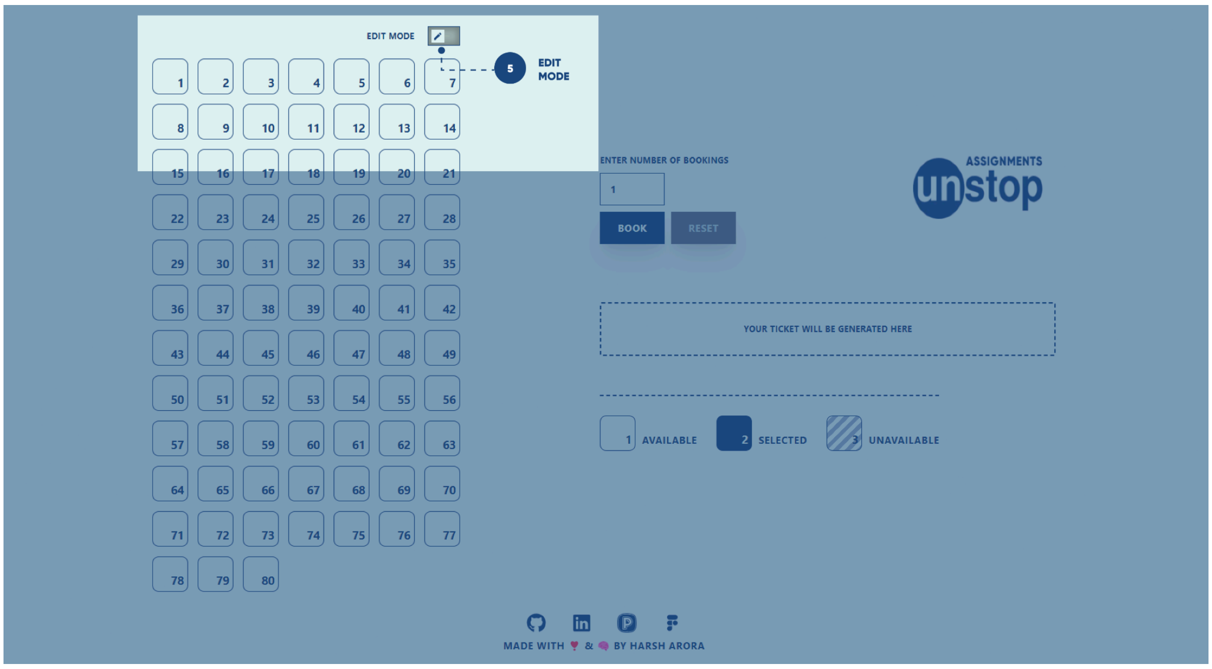
2. Book n seats. Simple as that.

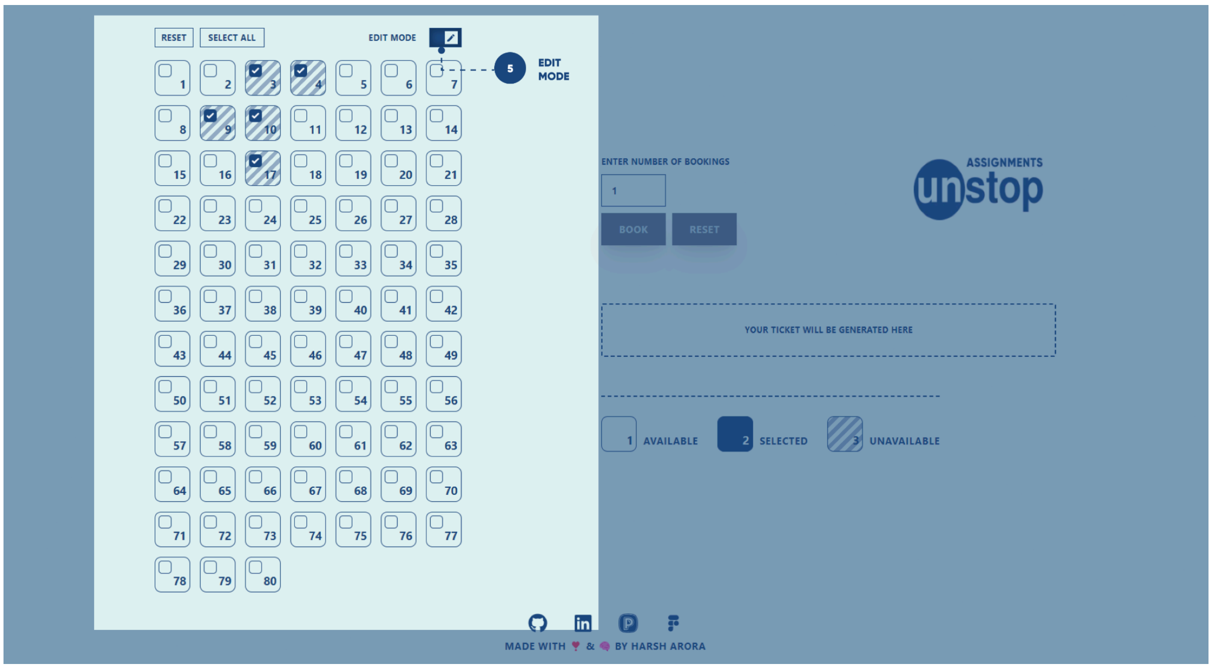


3. Seats Legend. Empty seat implies the seat is available for booking. Filled seat implies the seat is just booked. Striped seat indicates that seat has been booked in past.

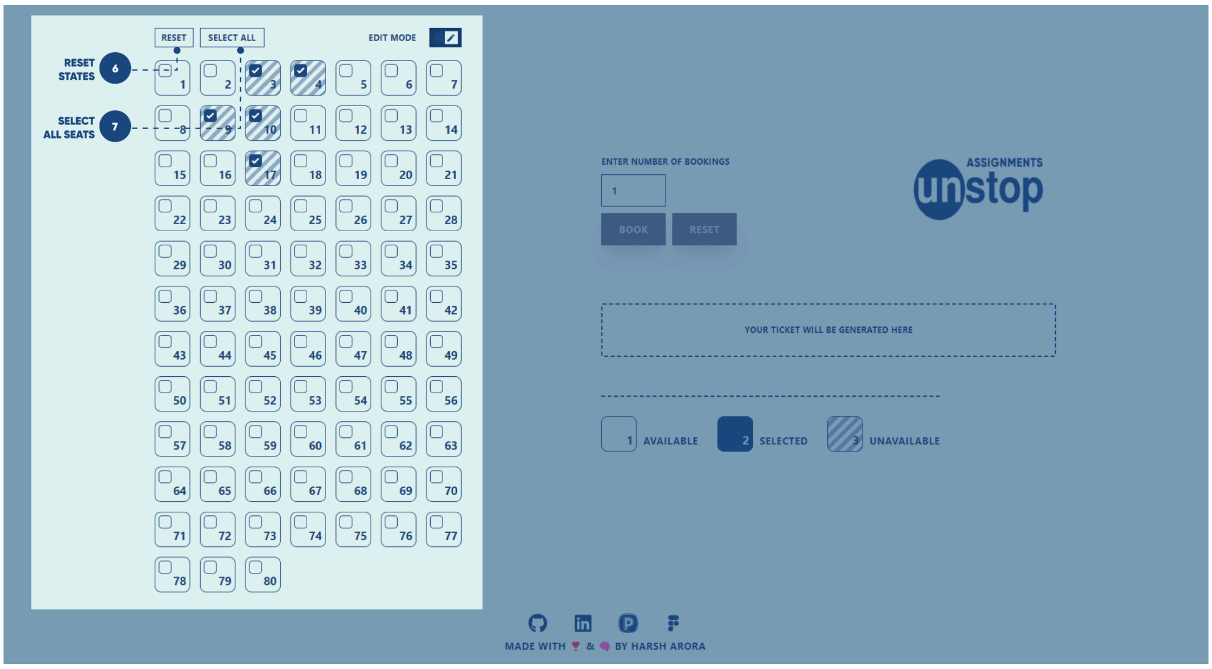

EDIT MODE
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
36 37 38 39 40 41 42
43 44 45 46 47 48 49
50 51 52 53 54 55 56
57 58 59 60 61 62 63
64 65 66 67 68 69 70
71 72 73 74 75 76 77
78 79 80
4
SEATS MAP
ENTER NUMBER OF BOOKINGS
1
BOOK
RESET
ASSIGNMENTS
unstop
YOUR TICKET WILL BE GENERATED HERE
1 AVAILABLE
2 SELECTED
3 UNAVAILABLE
MADE WITH ♥ & BY HARSH ARORA

4. Seat Map

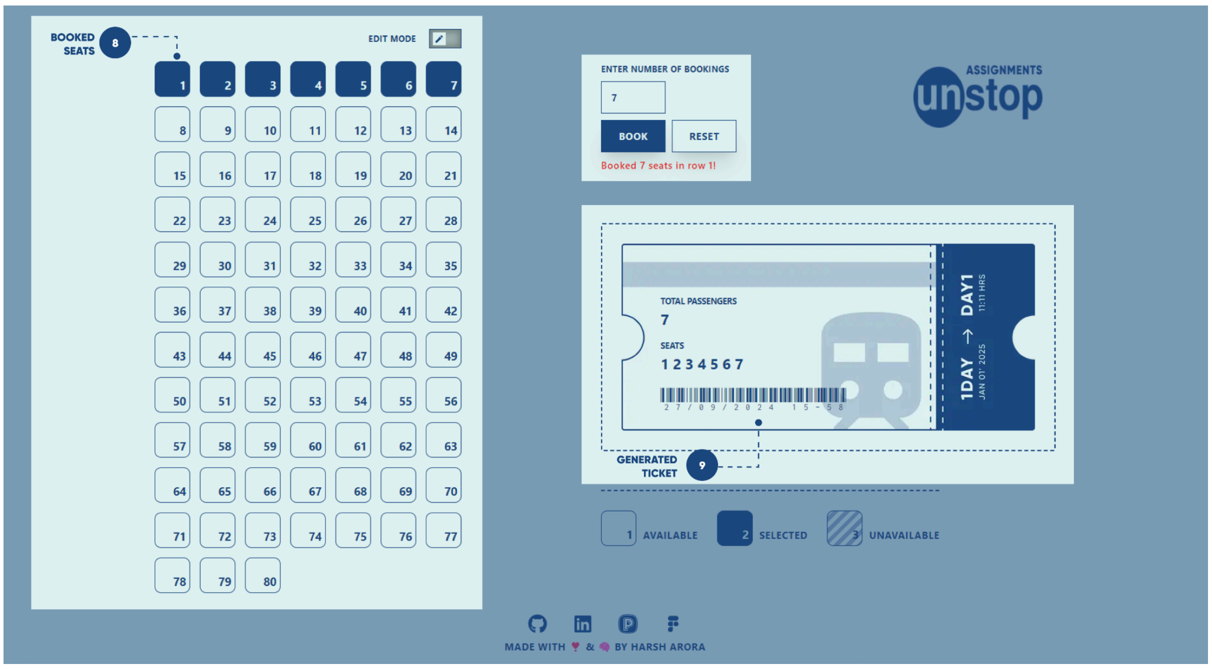

EDIT MODE
5
EDIT MODE
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
36 37 38 39 40 41 42
43 44 45 46 47 48 49
50 51 52 53 54 55 56
57 58 59 60 61 62 63
64 65 66 67 68 69 70
71 72 73 74 75 76 77
78 79 80
ENTER NUMBER OF BOOKINGS
1
BOOK
RESET
ASSIGNMENTS
unstop
YOUR TICKET WILL BE GENERATED HERE
1 AVAILABLE
2 SELECTED
3 UNAVAILABLE
MADE WITH ♥ & BY HARSH ARORA



5. Edit mode to allow tester to test different test cases, pre book some seats..



6. Reset and Select All seats buttons.



7

Booked 7 seats in row 1!

TOTAL PASSENGERS
7

SEATS
1 2 3 4 5 6 7

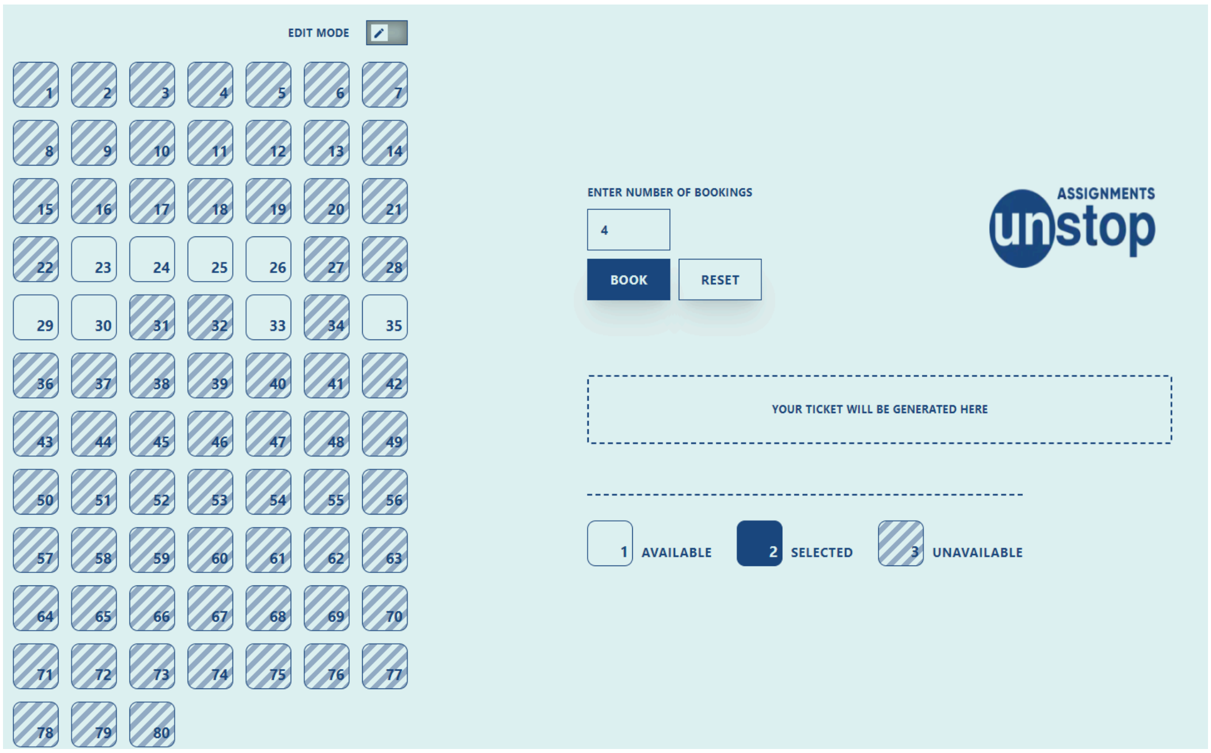
2 7 / 0 9 / 2 0 2 4   1 5 - 5 8

7. Ticket generated is displayed and booked seats are shown in the map.

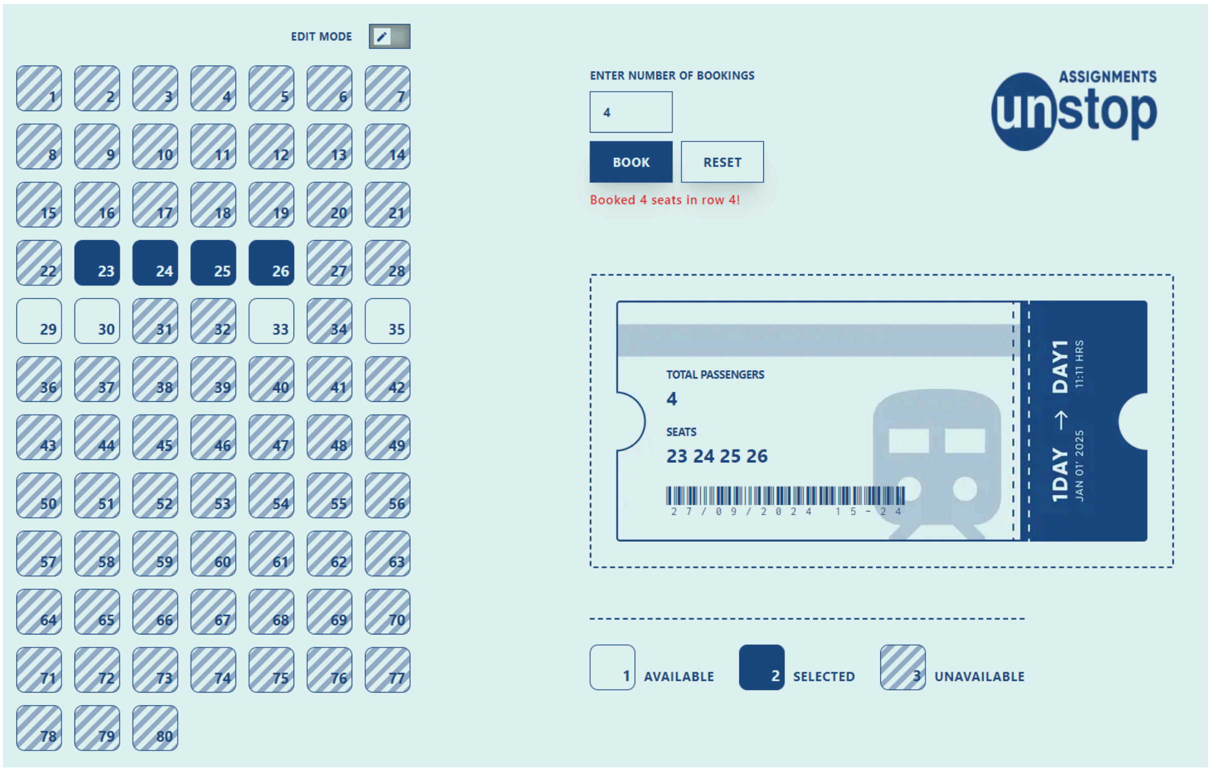
Assumptions and Test Cases:

To write an optimum solution I kept track of available seats in each row. My approach revolves around 2 main things:
First, try to book seats in a single row with **minimum intra-row seat spread**. For example two rows have 4 seats available and we wish to book 4 seats itself. In that case, the minimum total length the selected seats spread will give us optimal seat allocation.

Example:



Here, in the 4th row 4 seats spread across 23 to 26, total length is 4. However in the 5th row, 4 seats spread across 29 to 35, total length is 7. Hence in this case 4th row is chosen.



Booked 4 seats in row 4!

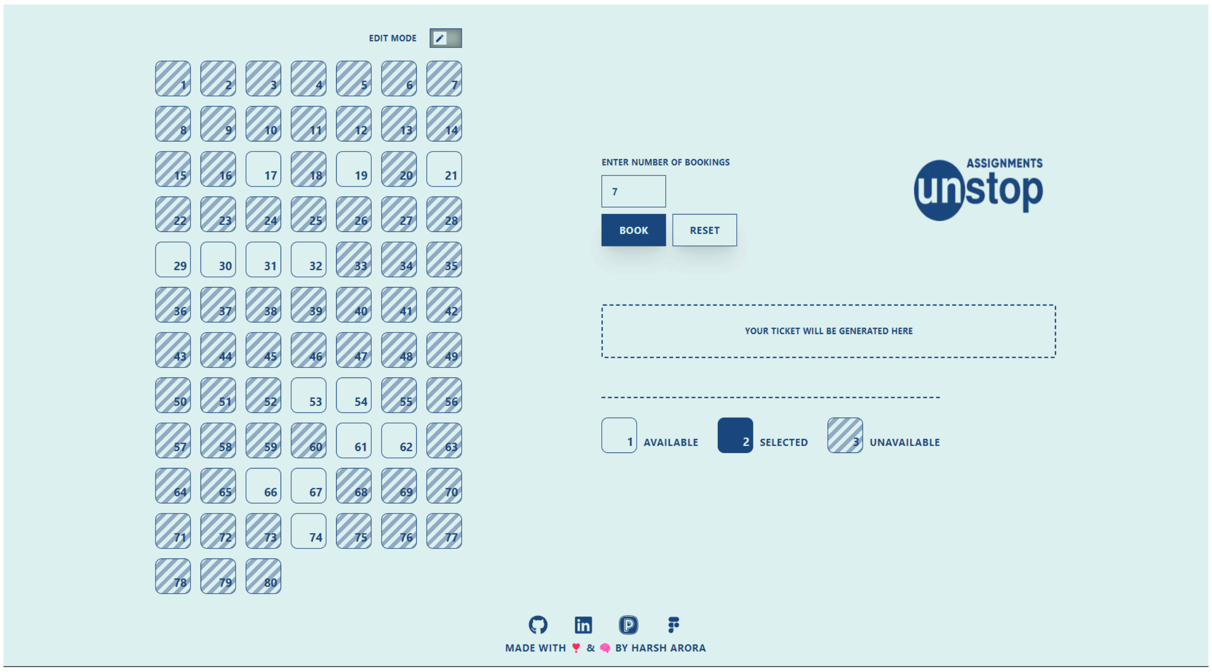
TOTAL PASSENGERS
4

SEATS
23 24 25 26

2 7 / 0 9 / 2 0 2 4 1 5 - 2 4

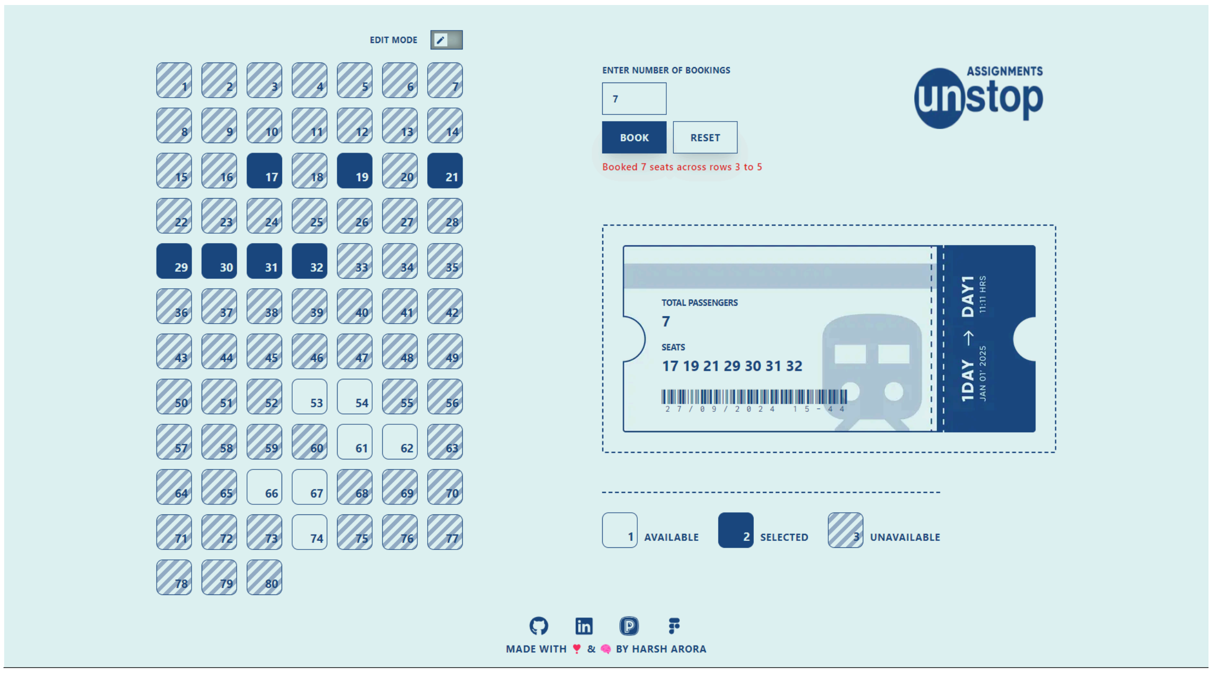
DAY1 11:11 HRS → 1DAY JAN 01 2025

Second, If seats are not available in a single row then we select the rows with **minimum inter-row spread**. We try to find the best possible window.
Example:



ENTER NUMBER OF BOOKINGS
7

In this case if a user wants to book 7 seats he can be allocated 2 windows. One window spans across 3rd to 5th row while other window spans across 8th to 11th row. Now the width of window 1 is 3 while width of window 2 is 4. Thus the first window is allocated.



ENTER NUMBER OF BOOKINGS
7

Booked 7 seats across rows 3 to 5

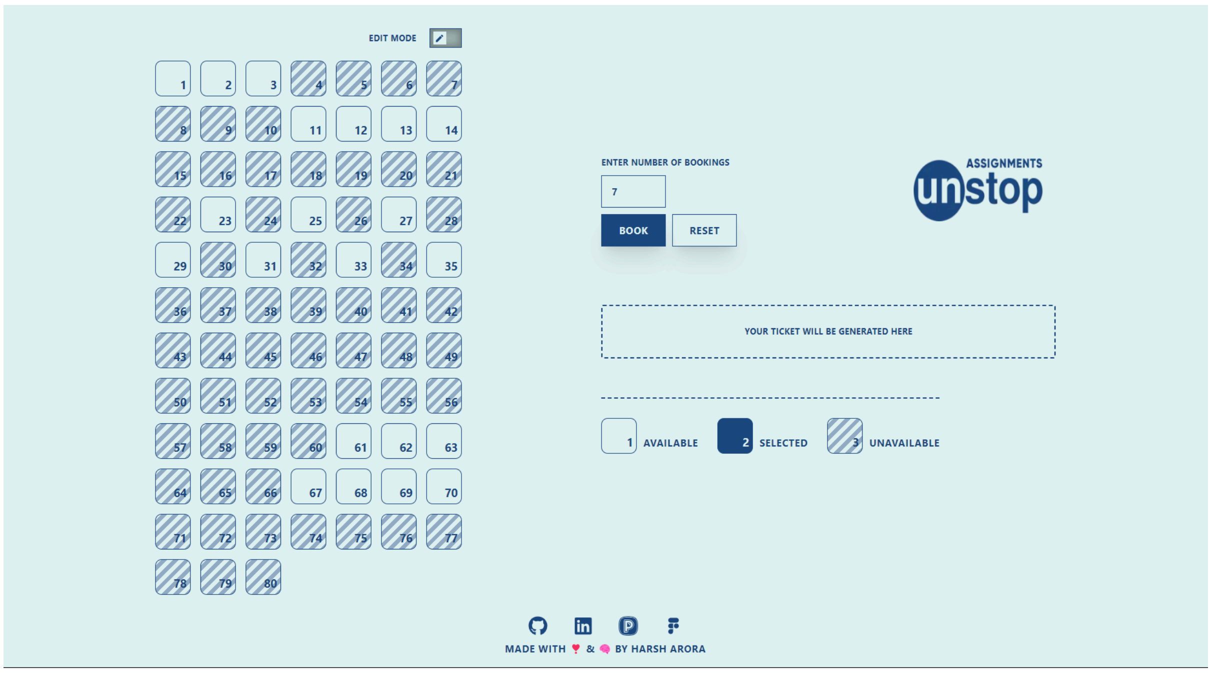
TOTAL PASSENGERS
7

SEATS
17 19 21 29 30 31 32

Now, there can be many edge cases where certain assumptions are made. In those cases there can be multiple solutions and pinpointing one solution would make it very complex to solve.

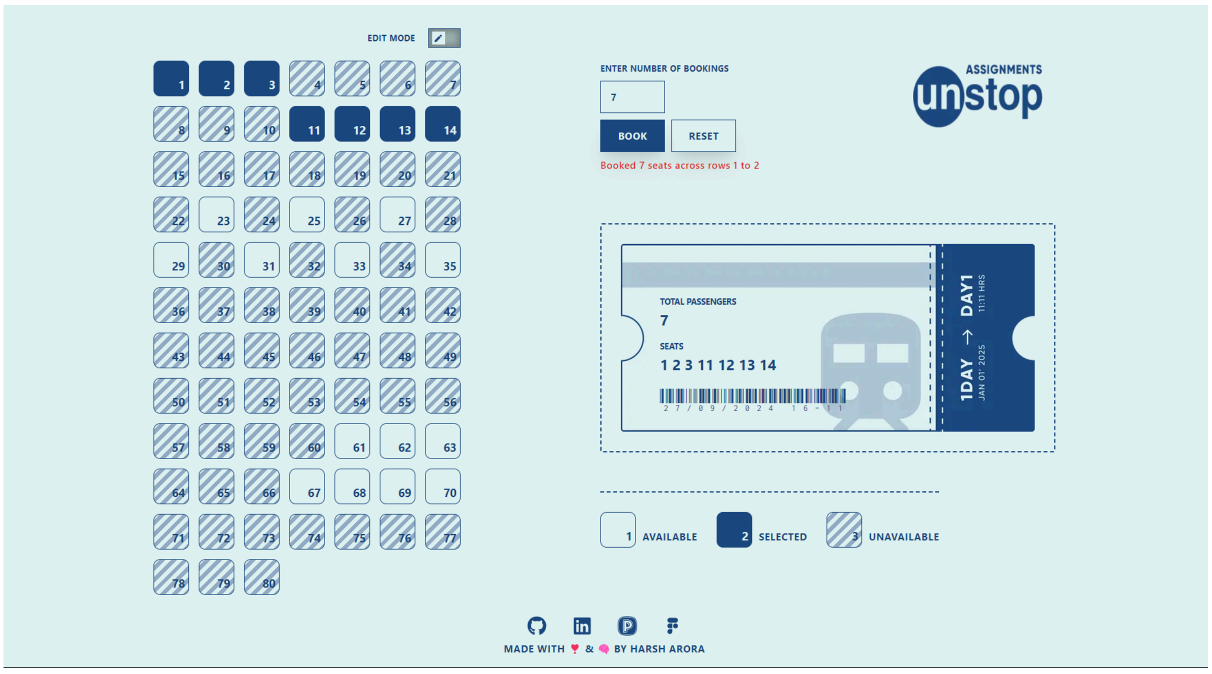
Example::

Booking 7 seats in this case.



YOUR TICKET WILL BE GENERATED HERE

In this case best window is third (row 9th - 10th) as all the seats are adjacent and in minimum rows possible. However, my approach assumes all windows as equal and would by default book first window. This is because, all three windows have same width and same availability.



Booked 7 seats across rows 1 to 2

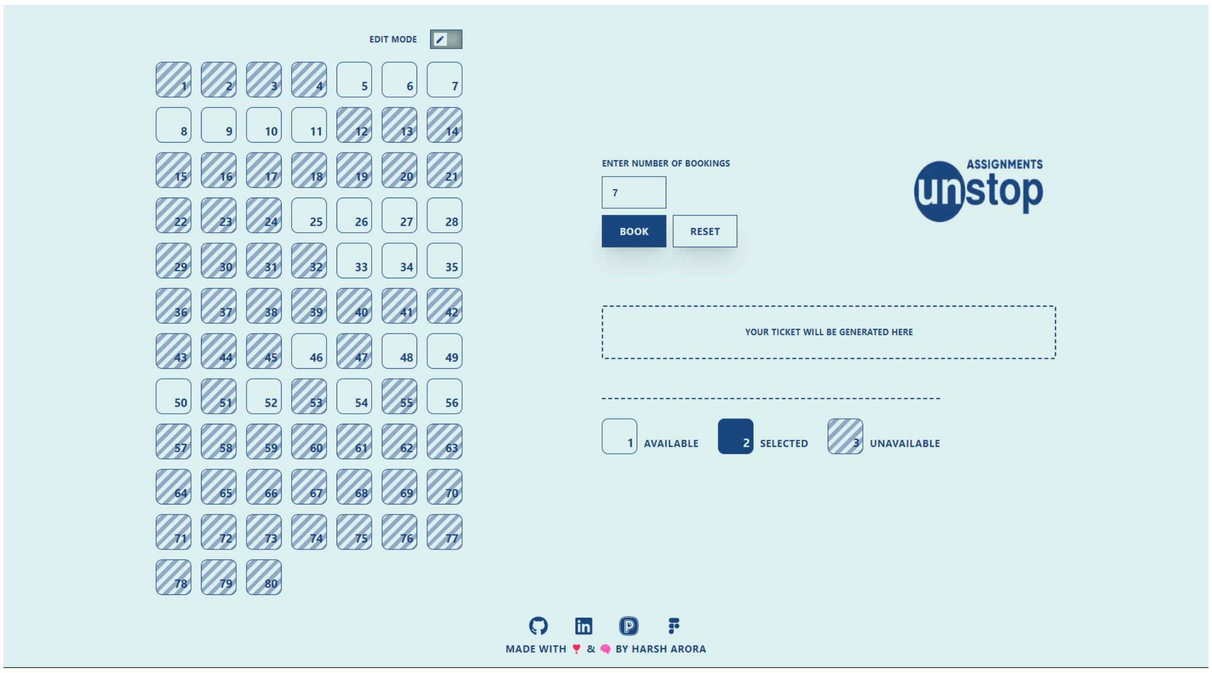
TOTAL PASSENGERS
7

SEATS
1 2 3 11 12 13 14

2 7 / 0 9 / 2 0 2 4 1 6 - 1 1

To solve this case we can find the best window among these windows based on the gaps between seats, and then choose the window with maximum no. of minimum gaps.

But will that approach be perfect? It might be if we allow minimizing gaps between two seats on cost of other required seats. Further minimizing gap may not guarantee best window.
For example in the same case:



If we minimize the gap then we will get the first window however logically the second window is best.
Further, the second and third window depicts the fact that the second window is preferred over the third window if we assume that minimum gaps allow best allocation even on the cost of other seats gap.

To actually tackle this we can include another logic to find the best window which is to find the **best window in which no. of connected components are minimum**.Connected components imply that all the seats are considered as nodes and one component consists of all the nodes which are connected to each other via some path (horizontally or vertically). Thus connected components are 4-adjacently connected nodes that form a single cluster.

This can be implemented using **DFS/BFS graph traversal techniques** on a crude level but will increase the overall complexity of the code.

Thus to keep it simple, **my code assumes that all the windows with the same available frequency, minimum row spread are treated as equal**.