Introduction

The exercise below is intentionally left open-ended. This is so you have room to show your skills, what you like, and how you like to work. Not all steps are required and many have bonus options. You can even add your own bonus options if you think it adds value.

Everybody gets the same exercise but of course we expect more from senior than from a junior applicant.

Things to note:

- Step 1,2,3 are required in the sense we expect some minimal implementations for those.
- We expect a medior/senior to make an attempt at step 4.
- Step 5 and 6 are totally optional and are there if you want to show off some Frontend and Devops skills.
- The code should be either in Golang or Python 3+
- You have 7 days to complete the task. We expect most people to take 1 or 2 evenings to complete the exercise.
- You are allowed to make use of frameworks in your code.
- After you complete the exercise you'll be invited to a session where you can present your solution and discuss the choices you made while completing the exercise.
- If anything is unclear don't hesitate to ask questions.
- **Make sure we can run your code!**

Theater Seating Algorithm / API

1. Create a data structure that defines a seating layout for a hall in a venue:

Requirements of layout.

- Different sections (e.g. main hall, 1st balcony, 2nd balcony).
- Different ranks defined across sections (1st rank, 2nd rank 3rd rank)
- So the 1st rank could be on both the main hall and 1st balcony, they're not restricted by section.
- Rows of seats, numbered by row, seat. Seats can be differently numbered: sometimes 1 … 6, sometimes 1,3,5,6,4,2

Bonus options:

- Store the layout in a database.
    - What database do you think is appropriate?
    - Do you use an ORM?

- Support additional properties on seats:
  - Aisle seat
  - Front row seat
  - High seat (e.g. on balcony)

- Support layout properties on sections
  - Is the section curved?
  - Add spatial layout between sections.

2. Create a seating algorithm.

Given a list of "groups of users" per rank (basically sizes, e.g. (1, 3, 4, 4, 5, 1, 2, 4) in a specific order,
try to place the users in their seats, e.g.

```
1 2 2 2 3 3 3 3
5 5 5 5 4 4 4 4
5 6 7 7 8 8 8 8
```

So the group of size 1 at index 1 gets the frontmost left seat. Then the group at index 2 of 3 people next to it, until the
row fills and wraps to the next row and fills in the other direction.

You can assume that sum(groups_of_users_for_rank) <= seats_in_that_rank

Bonus options:

- Take preferences into account based on seat properties. E.g. ("aisle", 2) would mean a group of 2 where one of the members wishes to be near the aisle
- Allow seats to be blocked (e.g. for technical purposes). This means a group should not be split across such a block
- Implement the seating algorithm in a non-blocking way, e.g. task, async, so it can be called using an API.

3. Design / Create a REST API to retrieve the layout of the allocations.

- Authentication / security is not a requirement (everything public)

Bonus options:

- Create a "ticket wallet" endpoint where a user can retrieve their seats.
- Create API endpoint to trigger the seating algorithm in a non-blocking way.

4. Improve seating algorithm.

Improve the algorithm in such a way that no individual people sit alone. In the above example this happened with the group at index 5 where a single individual sits on the 3rd row (even though the rest of the group is in front of him)

```
1 2 2 2 3 3 3 3
8 8 8 8 4 4 4 4
5 5 5 5 5 6 7 7
```

would be a better solution. Try to preserve the order as much as possible because the lowest numbers should get the "best" (frontmost) seats.

5. Create a simple consumer of this API

Suggestions:

- Render the seating layout/allocations to a visually understandable HTML layout.
- Create button to to trigger seating algorithm (in a non blocking way).
- Create a "ticket wallet" showing the user's seats.

6. Create a deployment of the application

Suggestions:

- Create Dockerfile/docker-compose file.
- Make it a 12-factor app

Extra: Seating examples

For inspiration, this is a theater hall we actually filled with such an algorithm: https://www.amphion.nl/site/assets/files/3162/plattegrond_rabobank_zaal.pdf

green/orange/yellow are the ranks. Most of it is first rank.

Another example (our algorithm filled that as well):

flint.nl
plattegrond-flint-plus-legenda.pdf

21.08 MB

It has more ranks and is quite a puzzle.

You don't have to base your actual solution on these floors, they just for illustration.