Instructions for the Test

- Solve the problems on the following pages (Part 1 to 5).
- It is recommended that you work through the parts sequentially, and communicate your progress to us periodically so that we can see how you work.
- We have made recommendations for the frameworks / tools that we would like you to use. If you feel particularly strongly about any of them, discuss this with us! We are happy to be flexible with these.
- **We will evaluate this test with the following parameters:**
    - Quality of the code: you should be able to follow good practices.
    - Communication and thought process! Write a few paragraphs explaining your thought process at the end of the project and include it in your README.
    - Simplicity of the solution: don't add features that have not been requested and don't over-engineer the solution.
    - Modernity of the code: we would like to see usage of modern paradigms, like React Hooks, f-strings, etc.
- **We will not evaluate:**
    - The design: you may use cat GIFs anywhere in the interface.
    - Time taken to complete the test

We want to implement a simple full stack application using React, Python and PostgresQL

Part 1: Front End

- Load in the frontend a static JSON file containing the following: [{ "type": "bank-draft", "title": "Bank Draft", "position": 0
}, { "type": "bill-of-lading", "title": "Bill of Lading", "position": 1 }, {"type": "invoice", "title": "Invoice", "position": 2}, {"type": "bank-draft-2", "title": "Bank Draft 2", "position": 3}, {"type": "bill-of-lading-2", "title":
"Bill of Lading 2", "position": 4}]

- Display the content as 5 cards, 3 in the first row and 2 in the second row.
Assign a different thumbnail of your choice to each document type.

- Display a placeholder spinner for each image that is loading.

- Make the application so the cards can be reordered via drag and drop.

- Make so clicking on a card displays the image as an overlay in the
middle of the webpage. Make so pressing ESC closes the image. Add a **README** file to explain how to run it.
Here's an example of how a row in the grid may look like:

Part 2: Making the call *(If you are more Front-end focused, you can skip the backend part and mention the same in your readme file, so our reviewers will only review the Front-end code)*

If you’re a Backend Developer:

- Create a PostgreSQL / SQLite table that can hold the data that was in the static json file from part 1 in a sensible way.

- Build a REST API that can fetch the data from this table and add data to this table. Preferably, use Python >= 3.6 along with starlette. You may find this helpful.

If you’re a Frontend Developer:

- Create a local service that mocks a server to send responses to. We suggest using: https://github.com/mswjs/msw

- Store the data in the browser to have data permanence across reloads. By default it should hold the data that was in the static json file from part 1 in a sensible way. Through this service, build a REST API that can fetch data from the browser storage and add data to it.

Part 3: Tying it up!

- Call the API from your front end application to display the same grid.

○ Also feel free to allow any domains and ports for CORS. (Do not waste your time on this)

- Have the frontend call the REST API for saving every five seconds (not every action). Display a loading spinner whenever it is saving, and how long has passed since the last save. Avoid saving if no changes have been made.

Part 4: Deployment

- Create a docker-compose file to start all the components as micro-services

- Write some simple documentation that makes it easy for us to understand and use it. Also write a little about how you approached the architectural / API design for the problem.

Part 5: General questions

- Design the hypothetical API for this project if you had to allow for adding, removing and updating the elements. Consider long-term maintenance as well.