Create a small service able to expose stats on a hypothetical massive COVID 19 testing, data comes from the attached `db.json` that can be read in an array like this:

```
const db = [
    {
        date: '2020-03-05T15:00:00.000Z',
        id: 0,
        locationId: 1,
        result: 'pending'
    },
    {
        date: '2020-03-06T15:00:00.000Z',
        id: 1,
        locationId: 5,
        result: 'pending'
    },
    {
        date: '2020-03-07T15:00:00.000Z',
        id: 2,
        locationId: 9,
        result: 'positive'
    },
    {
        date: '2020-03-08T15:00:00.000Z',
        id: 3,
        locationId: 6,
        result: 'positive'
    },
    ...
 ];
```

We won't use a full fledged database for the exercise, so it's enough to use the JSON db and put it in an array as an in memory database.

Candidates for the backend challenge must implement the following API endpoints:

```
 1. /examinations/ => return the full list
 2. /examinations/{locationId} => returns all exams of a specific IDs
 3. /examinations/stats/ => return data aggregated by of
positive/pending/negative
```

The stats endpoint is expected to return a payload like:

```
 [
 { locationId: 1, pending: 7, negative: 3, positive: 3},
 { locationId: 2, pending: 2, negative: 4, positive: 3},
 ...
 ]
```

Backend and Frontend

This challenge is intended both for Backend and Frontend devs. The former will implement the service as previously described, the other will implement a single page React frontend mocking the API responses. Frontend must show a table with results from API endpoints 1 and 2, allowing to select a location Id for the second one (it's fine to hardcode locationIds in frontend code). For the stats is up to the candidate to propose a UI (tabular, charts, other solutions).

Evaluation criteria

1. Working code: the solution provided must be working. If needed, provide a small `Readme` with information on how to install and run it
2. Project design (design patterns, component decoupling, ...).
3. Code readability and use of best practices
4. Error handling

Booster

The project will get extra points adding one or more of the following features:

1. Add pagination to `/examinations/` endpoint, in frontend add pagination handling simulating it in the mock API
2. Implement a service authentication (very simple one, without external dependencies or services). In frontend implement auth handling in mock API calls
3. Add date range filtering to stats endpoint, in frontend add control to filter data in the UI and handle them in the mock API calls