Coding exercise

Goal: Build 3 microservices that sort large objects together.
Deliverable: Please submit this exercise in a private git repository online, and send an invitation when you're done.

Specs

You will be creating 3 separate meteor apps that each perform one small part of the puzzle:

1. **M1**: A backend that creates large randomly generated objects and stores them
2. **M2**: A second backend that receives objects and sorts their keys, returning the duration of the sort
3. **M3**: A backend with frontend to display the stats and allows the user to configure how it works

Requirements

- The 3 meteor apps have to communicate in a specific way, to show that you understand a few concepts in the meteor ecosystem
    - M3 will be orchestrating both backends M1 and M2, it will communicate via DDP (the default communication protocol for Meteor) with M1, but via a different API with M2 (REST/graphQL/etc.)
    - You will have to expose meteor methods on both M1 and M3, but an API via Meteor's WebApp.connectHandlers on M2
    - Everything that you display on M3's frontend will have to pass through M3's backend, whose methods will call M1 and M2 according to your needs
- There should only be a mongoDB in M1, it will store the generated objects, and it will store the sort timings. Your DB will have 2 collections, unsortedObjects, and sortStats
    - The unsortedObjects should have 5 keys:
        - _id: String, the mongo id
        - object: Object, the object you generated
        - keyCount: Number, the amount of keys at the root of the object
        - depth: Number, the maxDepth used to generate the object
        - size: Number, the size in bytes of your object, use the npm library object-sizeof to determine the size
        - generationTime: Number, the time it took to generate the object in milliseconds
    - The sortStats should have 3 keys:
        - _id: String, the mongo id
        - objectId: String, the _id of the object that was sorted

        - *sortDuration*: Number, the time it took to sort the object in ms
- You will have to write 2 separate JS algorithms: one that generates an object, and one that sorts them alphabetically by key (as deep as possible). Both algorithms should be unit tested properly.
    - The object generation algorithm receives 2 inputs: *rootKeyCount*, and *maxDepth*
        - For each random key in the object, it should randomly decide to make it a number, string, object, boolean, or array
            - If it is an object, an object will have half the amount of keys as its parent (floored value)
            - Numbers and strings can be random of any length, upto 20 characters
            - Arrays should simply be empty
        - Your objects should not have more than *maxDepth* amount of nesting. i.e. if *maxDepth* is 2, you can not have more than 2 levels of nested objects
        - When you reach maxDepth, any object will simply be an empty object
    - The object sort function will work recursively until all objects are sorted by key
- You should be able to start all apps with a single command, documented in your README
- M2 should expose a single endpoint: */sort* that takes an object as its body, and returns the duration it took to sort the object.
- The frontend should have the following features
    - I should be able to change the URL to M2, so that we can point it to a different backend. Simply enter your localhost URL by default (like localhost:4000)
    - I should be able to set the total amount of keys at the root of the generated object
    - I should be able to set the maximum amount of nested levels of hierarchy in the generated objects
    - I should be able to generate a new object by clicking on a button, each one of them should appear in a list with its stats (i.e. don't display the object, but just how many keys it has, its depth, its generation time, and how much memory it takes up (formatted properly B/KB/MB))
        - Each object should have a "Sort" button next to it, that sorts it, displays a loader, and then displays the time it took in milliseconds/seconds to sort it.
        - If an object was sorted multiple times, it should display all the sort durations for that object.
    - There should also be a button that lets me sort all objects at once, and a button that sorts all unsorted objects at once
- Make sure you handle the edge cases in all user inputs
- The UI should refresh itself automatically, you shouldn't have to request updated data, it does not necessarily need to use pub/sub however, up to you
- Sorting should time out after 200 milliseconds
- You should only test 3 things:
    - The object generation function
    - The object sorting function
- Pay attention to details, make it work well and clean up your UI

Here’s an example of an unsorted and sorted object

```
// rootKeyCount = 5
// maxDepth = 3
const unsortedObject = {
 khjsdf789345kflhs: {
  sfdhk453sdf: 'jkhdsjkfsdf',
  sdufzi: {
   kjhkwjdfs: {},
  },
 },
 DSFnk348975tdfg: 45,
 Asdflkjfsd984: {
  lkjhkfhgkjldfg: {
   dfgkljdf: 'sdfjkgsfd',
  },
  sdfhkjhsdf: true,
 },
 '00FSGsdflis': false,
 fdsskhj5379: [],
};

const sortedObject = {
 '00FSGsdflis': false,
 Asdflkjfsd984: {
  lkjhkfhgkjldfg: {
   dfgkljdf: 'sdfjkgsfd',
  },
  sdfhkjhsdf: true,
 },
 DSFnk348975tdfg: 45,
 fdsskhj5379: [],
 khjsdf789345kflhs: {
  sdufzi: {
   kjhkwjdfs: {},
  },
  sfdhk453sdf: 'jkhdsjkfsdf',
 },
};
```