Simple Web Client Application

The goal is to build a single page web application that allows a user to view time series data. The user should be able to select or unselect at minimum one time series data to display. The more the better. The time series data should represent a system performance metrics such as CPU usage, disk capacity, memory usage, network traffic, etc from an imaginary computer system. We would prefer this data be displayed as a line graph chart.

The data can be generated randomly when the user views the chart but it must appear to be somewhat realistic. For example, a chart showing disk usage should probably look like a smooth line that mostly increases over time. New data can be generated each time a time series is plotted, but the data for the charts must be loaded dynamically into the UI from a back-end web server.

The user must be able to:

1. Toggle display of time series data on/off. An acceptable minimum requirement is to implement a single time series. However, if more than one is implemented, they should be displayed as separate charts on the screen versus separate lines on one chart. In the event that none of the time series data are selected to view, none of the time series charts will be displayed.
2. Choose to view either 1 hour or 1 week of time series data for the currently displayed time series. All time series displayed must show the same chosen time range.

Ground Rules:

1. The goal is to have functional code that can be the basis of a conversation around design trade-offs during the face-to-face interview process. This is your chance to highlight your development practices, thought process, and attention to user experience.
2. The server-side code that provides the chart data and the web application static files must be written in Go (https://golang.org/). We have written a simple http server for you. It's located in the included zip. It contains an example ping endpoint but you will need to write the data endpoints your front-end will use.
3. The front-end portion of the project must be implemented in React, but aside from that any compatible charting modules, build tools, or libraries are fair game. Please include instructions on how to setup the environment to build and run the code in a project README. We think of languages as tools and good engineers can adapt to the right tool for the job. We are impressed if you use a set of tools you haven't used recently and love hearing about your experience with it.
4. All code will be evaluated, but the primary area of critique will be around practical React component design, proper use of asynchronous code to fetch data from the backend, CSS styling, and React component testing.

5. This is not a timed assignment, but ask yourself the following questions when you write it:
    a. Does it work?
    b. Is it dead-simple to use? Keep in mind that our slogan is “Zero Touch Infrastructure.” The user is of utmost concern.
    c. Is it robust? The application should withstand some basic attempts to break it when interacted with.
    d. Is the code clear? We peer-review all work.
    e. Is the code maintainable? Keep your colleagues in mind as they’ll likely be building upon your code at some point in time.
6. Include all source code, test code, and a README describing how to build and run the solution.
7. There are no requirements for visual design, but all UI elements (including any fonts, colors, logos, header, footer, menus, etc) must look and act like they fit together to form a cohesive design.
8. The UI should indicate the state of the system -- if there are load times, for example, you should display some sort of indication that the system is loading (loading spinner, text that says “loading”, etc.)
9. When complete, please email us a .zip or .tar of your project folder. Please include a README with any relevant info.

Optional Bonuses:

1. **Only one time series data is required** for the project, but if time allows providing more will impress us.
2. **Use a flux implementation** such as Redux to manage state, or the newly standard Context React API.
3. **A continuous stream of live data** from the backend to make the charts periodically update would be an impressive improvement to having a static snapshot.
4. **Unit tests!** Bonus points for unit testing either your react components, server code, or both!

Suggested Resources:

Charting:

1. ChartJS React Component: https://github.com/jerairrest/react-chartjs-2
2. D3 React Component: https://github.com/codesuki/react-d3-components
3. C3 React Component: https://github.com/bcbcarl/react-c3js

Project Build/Bundler Tools:

1. Create React App: https://github.com/facebook/create-react-app
    a. The create-react-app CLI tool can be used to build an initial skeleton of the frontend project; however, this may be less configurable than other build tools such as Webpack or Parcel.
2. Webpack: https://webpack.github.io/
3. Parcel: https://github.com/parcel-bundler/parcel