Lab 4 - From Figma to React

Graphical User Interfaces*

In this lab, we will demonstrate a basic way to take a Figma project and turn it into a React web app. On completion of this tutorial you will have a greater understanding of:

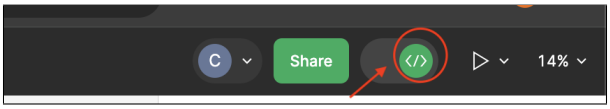
- How to identify and group Figma features to place into react project
- The pros and cons of taking from Figma and converting to a React project

We will be starting with a Figma project, similar to the one you created in the Figma tutorial 2 weeks ago. It can be downloaded at QMPLUS and uploaded to the Figma website. To understand how it works, try running the project in present mode.

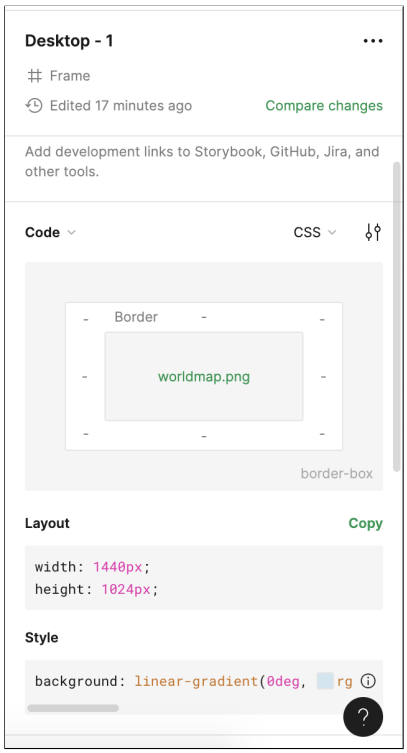
*Written by Corey Ford 2023/24 for Queen Mary University of London (c.j.ford@qmul.ac.uk)

1 Figma Developer Mode

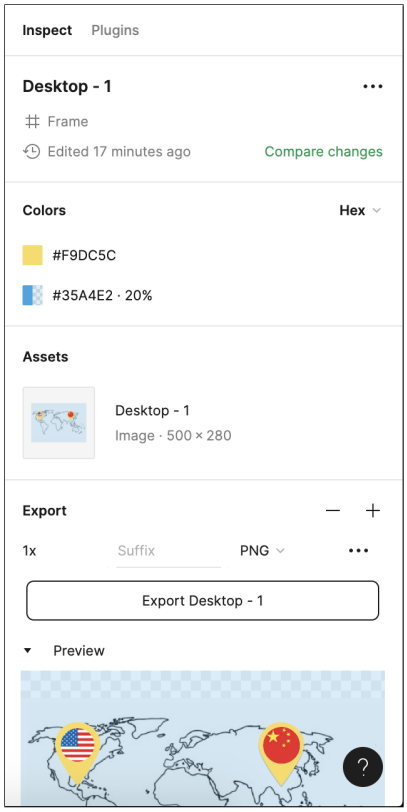
In Figma, we are able to inspect our designs and see each of our frame/object's CSS code. In the top right corner, click the switch to move from **design** mode to **developer** mode.



Now when you select a frame or object, the right hand panel will give you different types of information. For instance if we click "Desktop 1", we can scroll down and see the code pane. The code pane gives CSS layout code (the width and height) and style code (e.g. our tinted background image, using a linear-gradient).



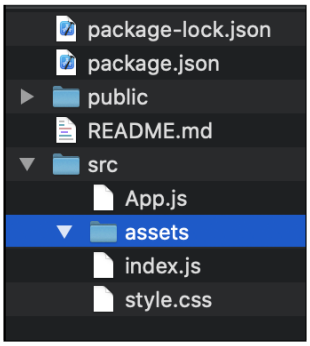
Scrolling further down, we can see the colours in hex-values used in the project. These can be helpful to capture in code as variables as it means that designers can keep experimenting in Figma, with you then as a developer being able to add these changes quickly (e.g. not having to re-download assets, but instead tweak just a few lines of code).



At the bottom of the pane on the right of figma is also the **export** panel (see above). Here you can export the selected frame/object as a PNG image. This can be useful where shapes may be more complex, such as the pins on our map, and best added as an image into our React application.

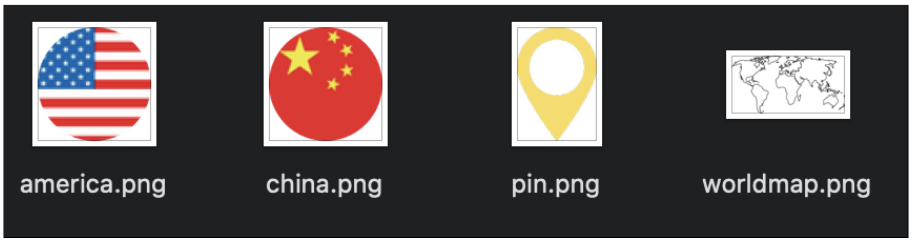
1.1 Exporting Components

Let's focus on the Desktop 1 frame in Figma. We will export the components that we need for our website. We will save this in a new folder called "assets" in our "src" folder, from a blank copy of our "gui-react-app" project that we used last week.



TASK: Export the following as PNG image:

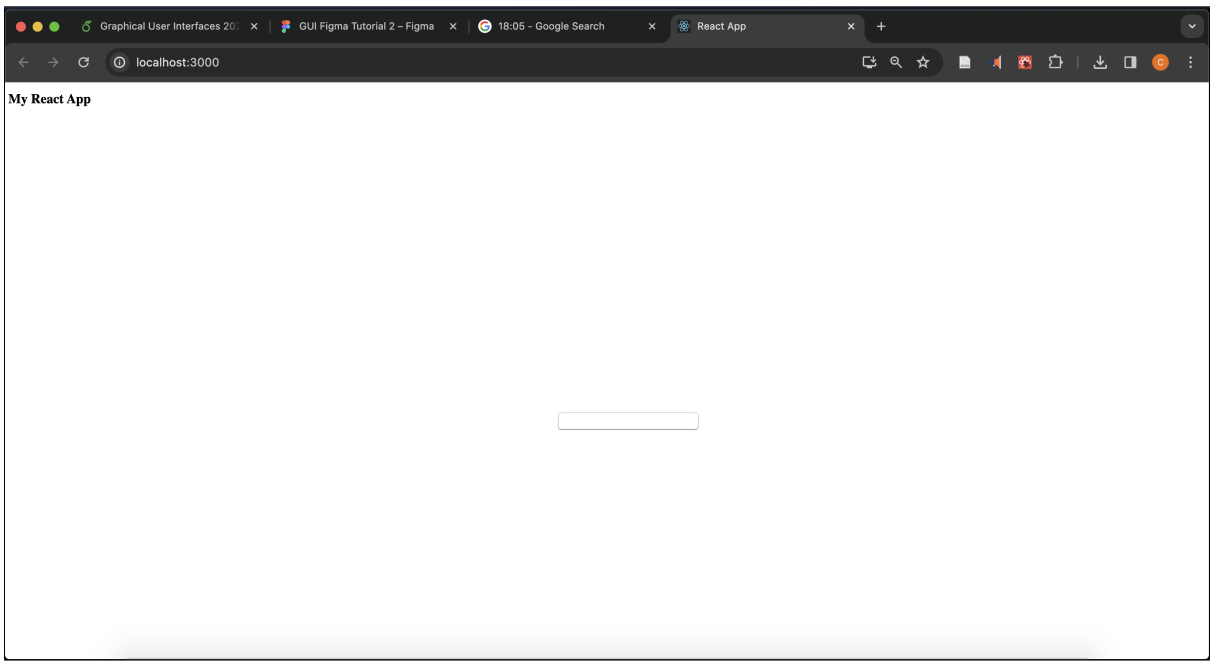
- **worldmap** – note that we export the white background version and not the coloured blue version. You can find this by clicking on *worldmap.png* in the code pane (within the rectange).
- **pin** – the yellow pin that we place ontop of the maps.
- **Both world maps** – in the shape of a circle. You need to select the group with both the map and the ellipse, where the masking was applied.



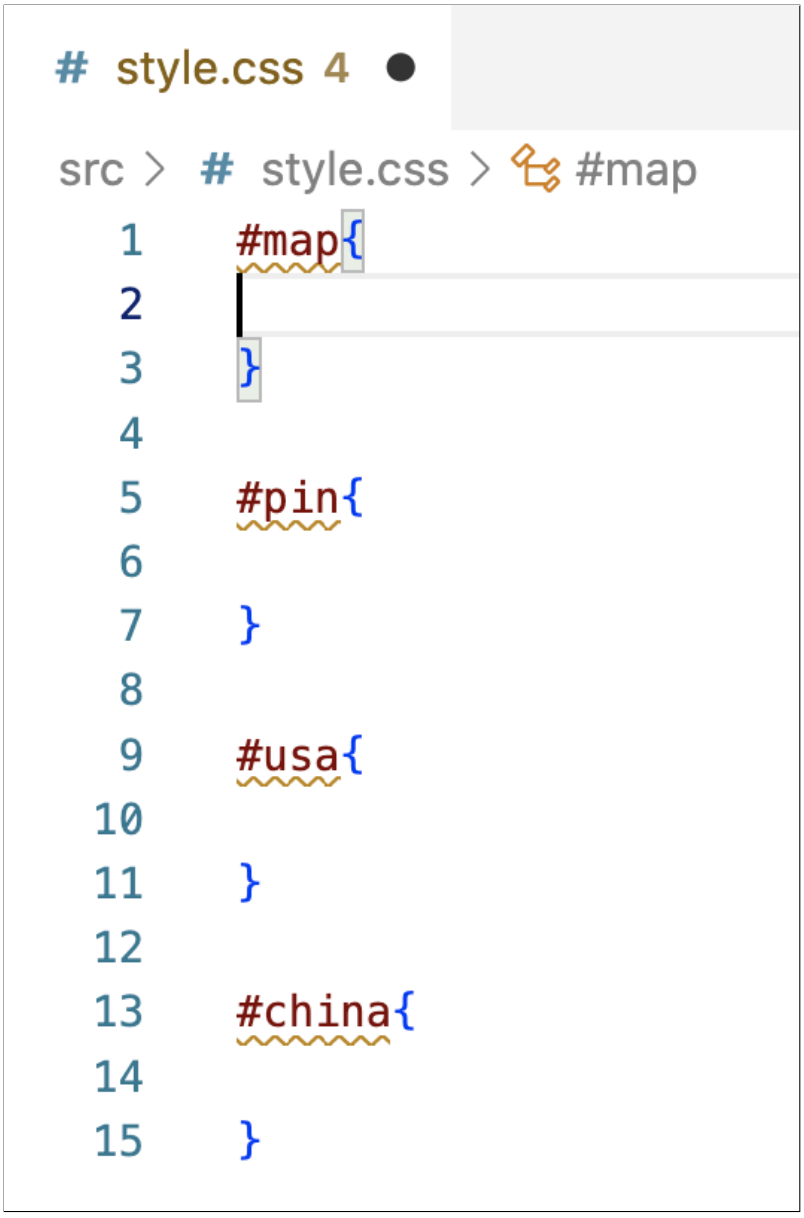
2 To React

Lets start running our new react project.

Remember from a clean copy of our template, you need to run **npm install** and then **npm start** from the terminal. Refer back to the previous tutorial if unsure. You should start with the the following in your browser:



TASK: Navigate to style.css. We will create a new style for each of our different assets, as shown below.



2.1 World Map

With this ready, let's start to create the map background.

We will create a new react component (or function) for the map itself. In **App.js**, we add the following:

```
1   function Map(){
2     return(
3       <div id="map"></div>
4     )
5   }
6
7   function App() {
8     return (
9       <>
10      <Map />
11      </>
12    );
13  }
```

- On line 1 we create a new react component Map(), which returns an empty div assigned the id **map** – matching the ID in our style.css file.
- On line 10, we change our <h1> to <Map />, so that our main visual component is the Map. At this stage our web app will be blank, as we have no CSS.

Moving back over to Figma, we inspect the code for the map in developer mode. Figma suggests both code for the layout and style.

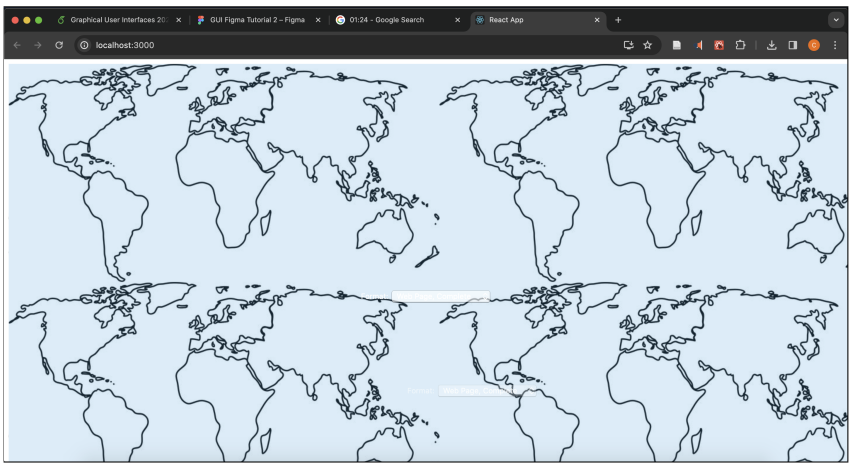
We copy and paste this into the CSS file as follows.

```
1   #map{
2       width: 1440px;
3       height: 1024px;
4       background: linear-gradient(0deg, rgba(53, 164, 226, 0.20) 0%, rgba(53, 164, 226, 0.20) 100%),
5                                         url(<path-to-image>), lightgray 50% / contain no-repeat;
6   }
```

Line 2 sets the width of the element to 1440 pixels; Line 3 sets the height of the element to 1024 pixels. Line 4 defines a background for the element using a linear gradient, which moves from the same colour to another colour also the same (largely the code is redudant, but we will stick with the Figma suggestion for now). Line 5 adds an additional background layer using an image specified by the *<path-to-image>*. Also on line 5, **lightgray 50% / contain no-repeat** sets the background of the element to *lightgray* color at 50% of the container's width and height. The */ contain* part ensures that the background image is contained within the element, and *no-repeat* prevents the image from repeating.

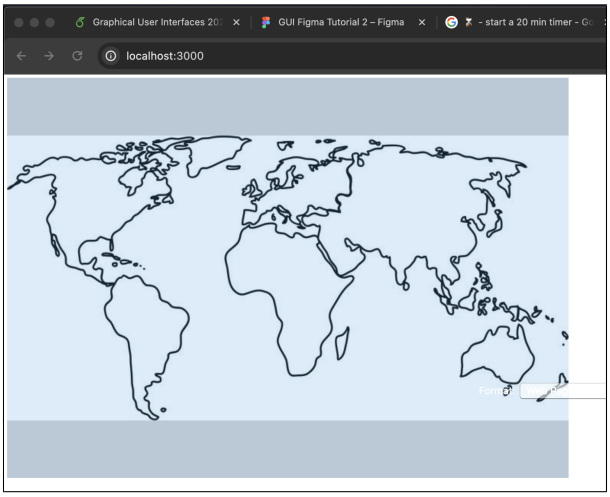
We replace `<path-to-image>` with the actual file path or URL to the image: **url(assets/worldmap.png)**.

However, our App looks like this....



This often happens when porting Figma to CSS, which is why a solid foundation of CSS is still vitally important. Indeed, here, we need to remove the comma after our URL.

```
4       background: linear-gradient(0deg, rgba(53, 164, 226, 0.20) 0%, rgba(53, 164, 226, 0.20) 100%),
5                                         url(assets/worldmap.png) lightgray 50% / contain no-repeat;
```



Identifying these challenges can be frustrating at first, but it becomes easier with practice, as you develop expertise. There are many other ways to simplify the code from Figma too, which is why having an understanding of both coding and design principles is crucial to GUI success!

Below we explain how to add the pin images.... **BUT BEFORE THIS**... try to recreate the app yourself. All the key principles have been taught to you already in the previous tutorials – you will learn lots by trying to apply these yourself instead of simply following the steps below.

2.2 TASK: Try to add the pins, China and USA images to the React project. The steps will be similar, but not identical, to the above. Some hints are:

In Figma, clicking the drop down arrow next to Code (where the CSS is shown) will show *Properties*, including values for both Top and Left. This can help to position the items, setting the CSS to use absolute values (see Week 1's lab).

Expect to use trial and error – sometimes the mappings between Figma and CSS are not precise, and require some tinkering.

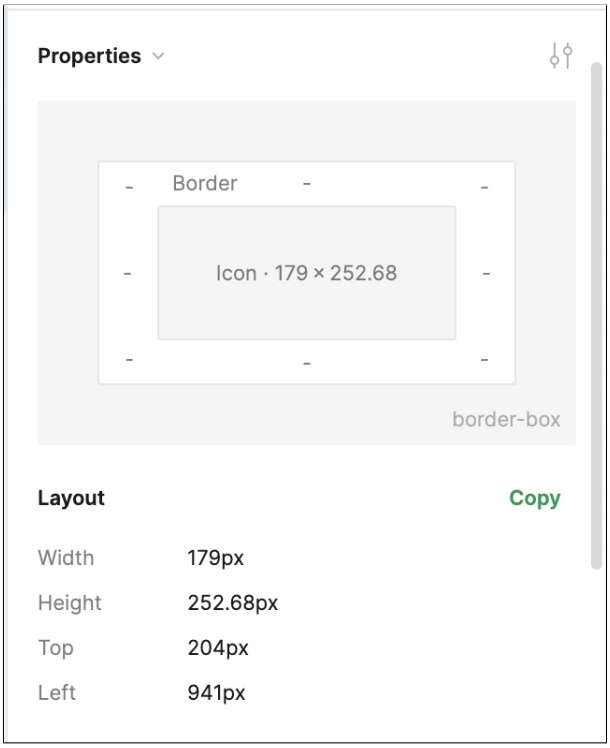
There are two pins which means the position will need to be set differently for each pin.

The china map gives some strange values from Figma. This is on purpose, to give you practice in using assets from other people.

The following subsection describes how to add the pins. You might have acheived this already – but if not, the next section will explain this in more detail. Skip to Section 3 if confident.

2.3 Pins

The CSS for the pins is taken from the Figma, as follows.




```
8   .pin{
9       width: 179px;
10      height: 252.685px;
11      flex-shrink: 0;
12      background-image: url(assets/pin.png);
13      filter: drop-shadow(0px 4px 4px rgba(0, 0, 0, 0.25));
14      position: absolute;
15  }
```

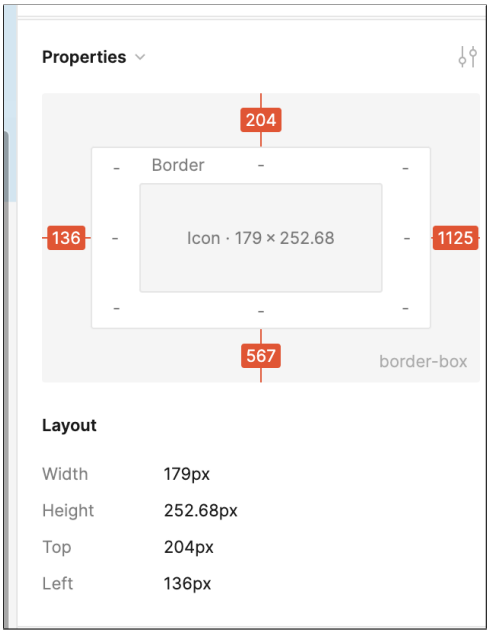
Note on line 14 we use absolute positioning. We do not set top or left however, as we will have two pins with different positions, so will add this to our React code. We also change pin from an ID (e.g. using the # on line 8) to a class (e.g. using the . on line 8).

```
7   function Pin({pinTop, pinLeft}){
8     return(
9       <div className="pin" style={{top: pinTop, left: pinLeft}}></div>
10      )
11  }
```

In App.js, we make a react component for Pin which takes two arguments as props (line 7 above). This is pinTop and pinLeft – where we want the pin positioned. We return a div with the class applied, and additional CSS styling, where pinTop and pinLeft are arguments. These additional styling will position the pin.

We then add these to the App() component using JSX, and passing values suggested by Figma for the pins.

```
13  function App() {
14    return (
15      <>
16      <Map />
17      <Pin pinTop={'204px'} pinLeft={'941px'} />
18      <Pin pinTop={'204px'} pinLeft={'136px'} />
19      </>
20    );
21  }
```



2.4 TASK: Add all the assets (map, pins, china and usa) to the file, giving each its own react component.

3 React State from Figma

In our Figma project, we have two separate information boxes for the USA and for China. Whilst we could export these as images, we make the decision here to add them as raw CSS and JSX – this makes them easier to edit as a developer in case a designer comes along with changes at the last moment! Which objects to export, in what grouping, and which to leave as html is one of the challenges of porting Figma to React code – skilled developers will develop an eye for doing this over time.

First, we need a way to track the project state, to know if China or USA has been clicked. We will do this in a react styled way – this is very similar to the exercise from the previous lab.

We add **import {useState} from 'react';** to line 1 of App.js, to be able to track state in our project.

We then add onClick functions parsed as props for our China and USA react components.

```
15  function China({onChinaClick}){
16    return(
17      <div id="china" onClick={onChinaClick}></div>
18    )
19  }
20
21  function USA({onUSAClick}){
22    return(
23      <div id="usa" onClick={onUSAClick}></div>
24    )
25  }
```

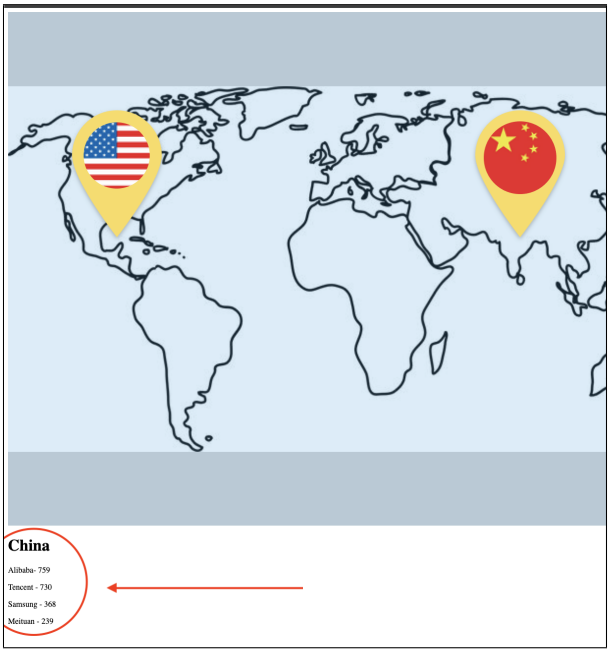
Next, we add an InfoBox component, using JSX to write standard CSS styles with the information from Figma. We pass an argument **isUSA** as props, and use an if-statement to return either the information for USA or China, based on if this value is true.

```
27  function InfoBox({isUSA}){
28    if(isUSA === true){
29      return(
30        <div>
31          <h1>USA</h1>
32          <p>Apple - 1,973</p>
33          <p>Microsoft - 1,617</p>
34          <p>Amazon - 1,598</p>
35          <p>Alphabet - 1,196</p>
36          <p>Facebook - 785</p>
37        </div>
38      )
39    }else{
40      return(
41        <div>
42          <h1>China</h1>
43          <p>Alibaba- 759</p>
44          <p>Tencent - 730</p>
45          <p>Samsung - 368</p>
46          <p>Meituan - 239 </p>
47        </div>
48      )
49    }
50  }
```

We then update our APP component with a state **isUSA**, and function **setIsUSA** to set the state, on line 54. We then parse these to our China and USA components on lines 61 and 62, using the () => syntax to avoid infinite rendering. Our info box is added on line 63.

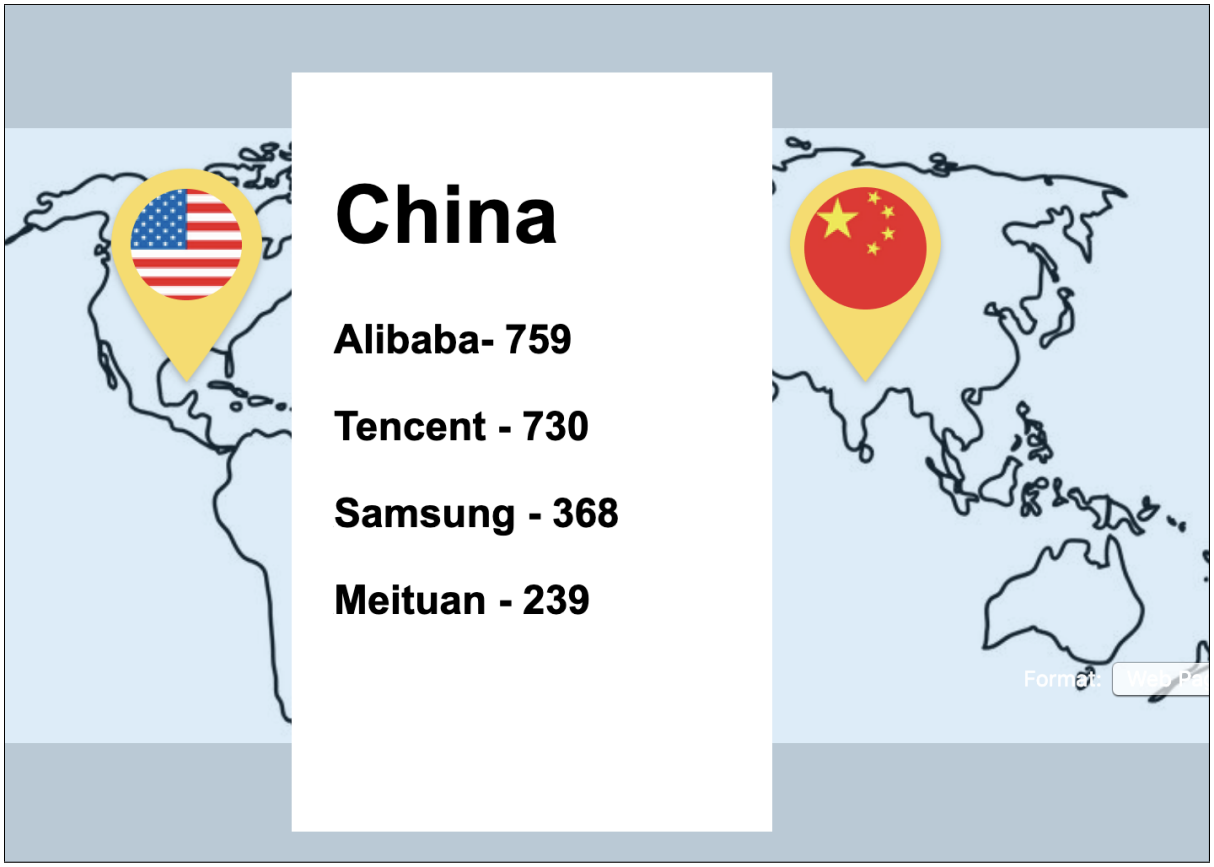
```
52  function App() {
53
54    const [isUSA, setIsUSA] = useState(true);
55
56    return (
57      <>
58      <Map />
59      <Pin pinTop={'204px'} pinLeft={'941px'} />
60      <Pin pinTop={'204px'} pinLeft={'136px'} />
61      <China onChinaClick={() => setIsUSA(false)}/>
62      <USA onUSAClick={() => setIsUSA(true)}/>
63      <InfoBox isUSA={isUSA}/>
64      </>
65    );
66  }
67
```

We now see our information change depending on the buttons we pressed underneath.



3.1 TASK: Implement the functionality above, and write some CSS code to style the our InfoBox component.

Remember that you can find CSS styles in the Figma file, and that you will likely have to do some experimenting.



4 Summary

Now that you've completed the tutorial, let's summarize what you've covered. In this tutorial, you explored the process of transforming a Figma project into a React web app. The key steps involved structuring the React project, exporting components from Figma, and implementing them in the React app. You delved into Figma's developer mode to extract CSS code, encountered challenges in translating Figma's CSS to React, and overcame issues with trial and error. Additionally, you learned how to manage state in React, utilizing the useState hook to toggle between information boxes for the USA and China based on user interaction. The tutorial emphasized the importance of a solid foundation in both coding and design principles for successful GUI development.