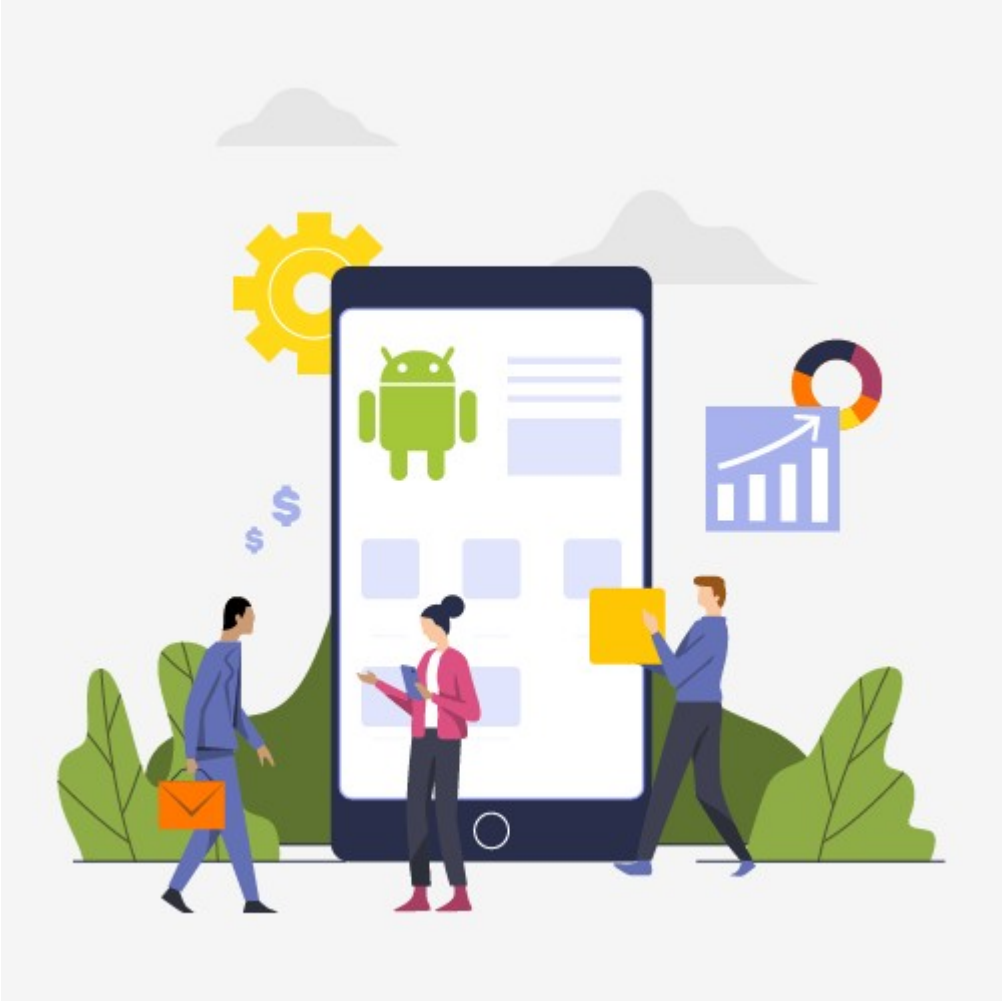
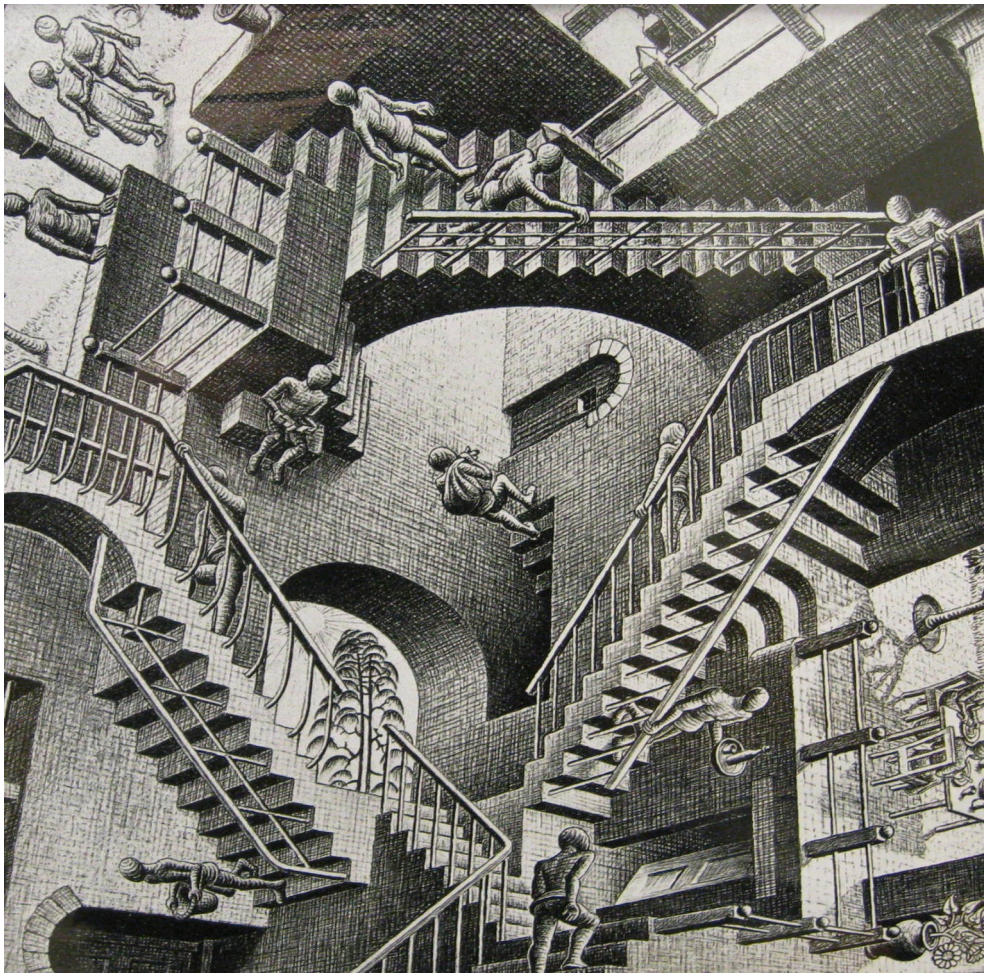
CS385 Mobile Application Development (Lecture 20)

A quick look at the map function again....



Some peoples feelings about nested map function calls



https://mcescher.com/

Double or nested map function calls

Map function – our classic approach to rendering arrays

```
function App() {
  // Here one of the properties has the value of an a
  // The array contains strings

  let trains = [
    { id: 54, direction: "Arrival", times: ["07:40", "10:20", "13:50"] },
    {
      id: 123,
      direction: "Departure",
      times: ["11:20", "15:30", "18:40", "20:05"],
    },
    { id: 223, direction: "Departure", times: ["09:35", "21:35"] },
  ];
  return (
    <>
      <h1>Trains</h1>
      {trains.map((x, index) => (
        <li key={index}>
          {x.id},Direction: {x.direction}, Times: {x.times}
        </li>
      ))}
    </>
  );
}
```

Trains

- 54,Direction: Arrival, Times: 07:4010:2013:50
- 123,Direction: Departure, Times: 11:2015:3018:4020:05
- 223,Direction: Departure, Times: 09:3521:35

Look at the `times` property of the objects. This is an array – of string values – but when we render this …. it renders just as a long string.

We need a better way of handling this situation

We use a NESTED map function to render the array contents

```
function App() {
  // Here one of the properties has the value of an array
  // The array contains strings
  // we use a nested map function to render the array from the times property
  let trains = [
    { id: 54, direction: "Arrival", times: ["07:40", "10:20", "13:50"] },
    {id: 123,direction: "Departure",times: ["11:20", "15:30", "18:40", "20:05"]},
    { id: 223, direction: "Departure", times: ["09:35", "21:35"] },
  ];
  return (
    <>
      <ul>
        {trains.map((x, index) => (
          <li key={index}>
            {x.id}, Direction: {x.direction}, <br />
            Times:
            <ul>
              {x.times.map((t, tindex) => (
                <li key={tindex}>{t}</li>
              ))}
            </ul>
          </li>
        ))}
      </ul>
```

Trains

- 54, Direction: Arrival, Times:
  - 07:40
  - 10:20
  - 13:50
- 123, Direction: Departure, Times:
  - 11:20
  - 15:30
  - 18:40
  - 20:05
- 223, Direction: Departure, Times:
  - 09:35
  - 21:35

We apply a map function (inside the original map function) to **`x.times`** (the property with the array values)

Map function – including the use of reduce within the map

```jsx
function App() {
  // our reduce function callback
  function calculateTotalCA(acc, x) {
    return acc + x;
  }
  // an array where the caMarks property is a primitive array
  let students = [
    { id: 54, module: "CS123", caMarks: [10, 8, 20, 30] },
    { id: 123, module: "MH765", caMarks: [12, 10, 16] },
    { id: 223, module: "PH165", caMarks: [5, 5, 10, 8, 5, 7] },
  ];
  return (
    <>
      <h1>Student CA Report</h1>
      <ul>
        {students.map((x, index) => (
          <li key={index}>
            id: {x.id}, Module: {x.module}, Total CA Marks =
            {x.caMarks.reduce(calculateTotalCA, 0)}
          </li>
        ))}
      </ul>
    </>
  );
}
```

Student CA Report

- id: 54, Module: CS123, Total CA Marks =68
- id: 123, Module: MH765, Total CA Marks =38
- id: 223, Module: PH165, Total CA Marks =40

Map function includes a reduce call for every object – the reduce sums the values in the `caMarks` array

A map function – dealing with a property with an array of objects

```
13      { id: 123, year: "2", subjects: [{ name: "cs385", credits: 5 }] },
14      {
15        id: 223,
16        year: "HDip",
17        subjects: [
18          { name: "cs385", credits: 5 },
19          { name: "cs130", credits: 5 },
20        ],
21      },
22    ];
23    return (
24      <>
25        <h1>Student Subjects</h1>
26        <ul>
27          {students.map((x, index) => (
28            <li key={index}>
29              id: {x.id}, Year: {x.year}, Subjects ={x.subjects}
30            </li>
31          ))}
32        </ul>
```

We cannot just render the array for the **`subjects`** property using the {} approach.

We will have to use an alternative approach.

We again use our nested map function approach

```
let students = [
  {id: 54, year: "3", subjects: [{name: "cs385", credits: 5}
  {name: "mh123", credits: 7}] },
  {id: 123, year: "2", subjects: [{name: "cs385", credits: 5}]},
  {id: 223, year: "HDip", subjects: [{name: "cs385", credits: 5},
  {name: "cs130", credits: 5}] }
];
return (
  <>
    <h1>Student Subjects</h1>
    <ul>
      {students.map((x, index) => (
        <li key={index}>
          id: {x.id}, Year: {x.year}, <br/>Subjects

          <ul>
          {x.subjects.map((m,mindex) => (
          <li key = {mindex}>{m.name} -- {m.credits}</li>
          ))}
          </ul>

        </li>
      ))}
    </ul>
  </>
```

Student Subjects

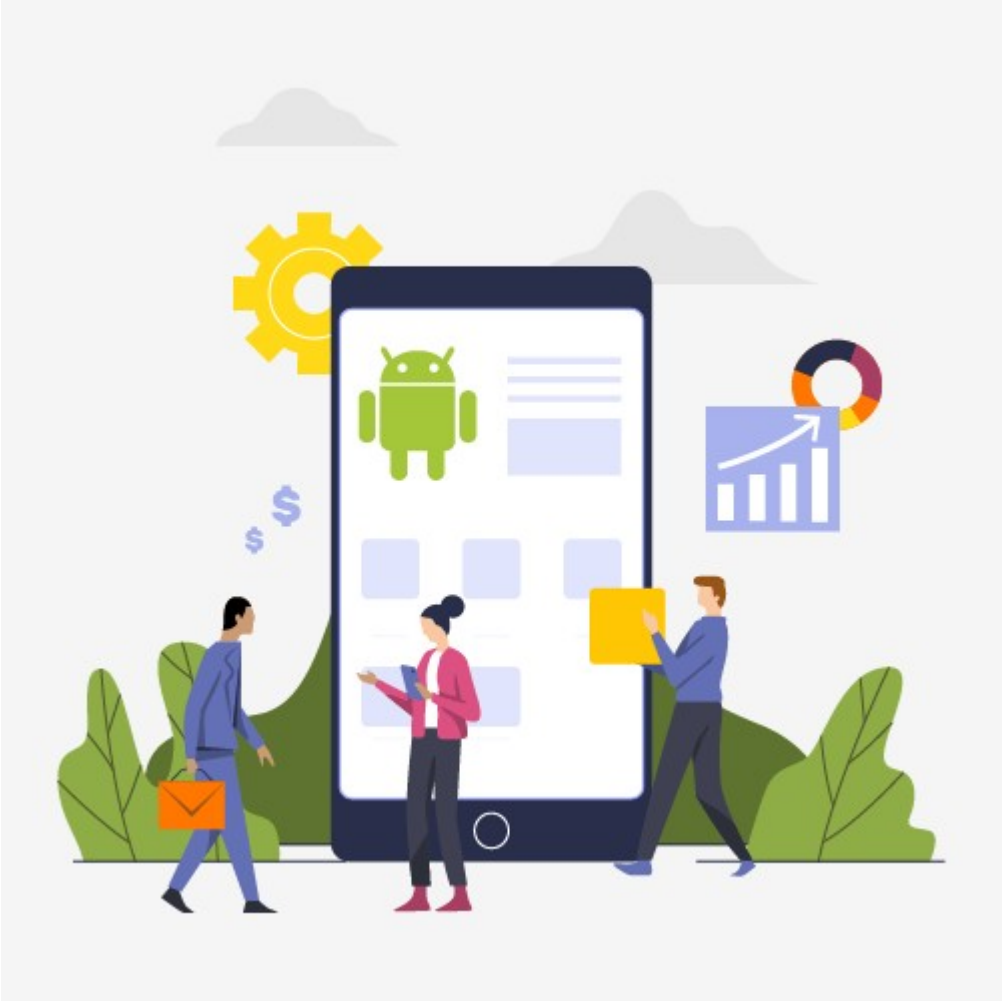
- id: 54, Year: 3, Subjects
  - cs385 -- 5
  - mh123 -- 7
- id: 123, Year: 2, Subjects
  - cs385 -- 5
- id: 223, Year: HDip, Subjects
  - cs385 -- 5
  - cs130 -- 5

As we have done before – we apply our map function to an array of objects (this time it is to the **`subject`** property of the **`students`** array)

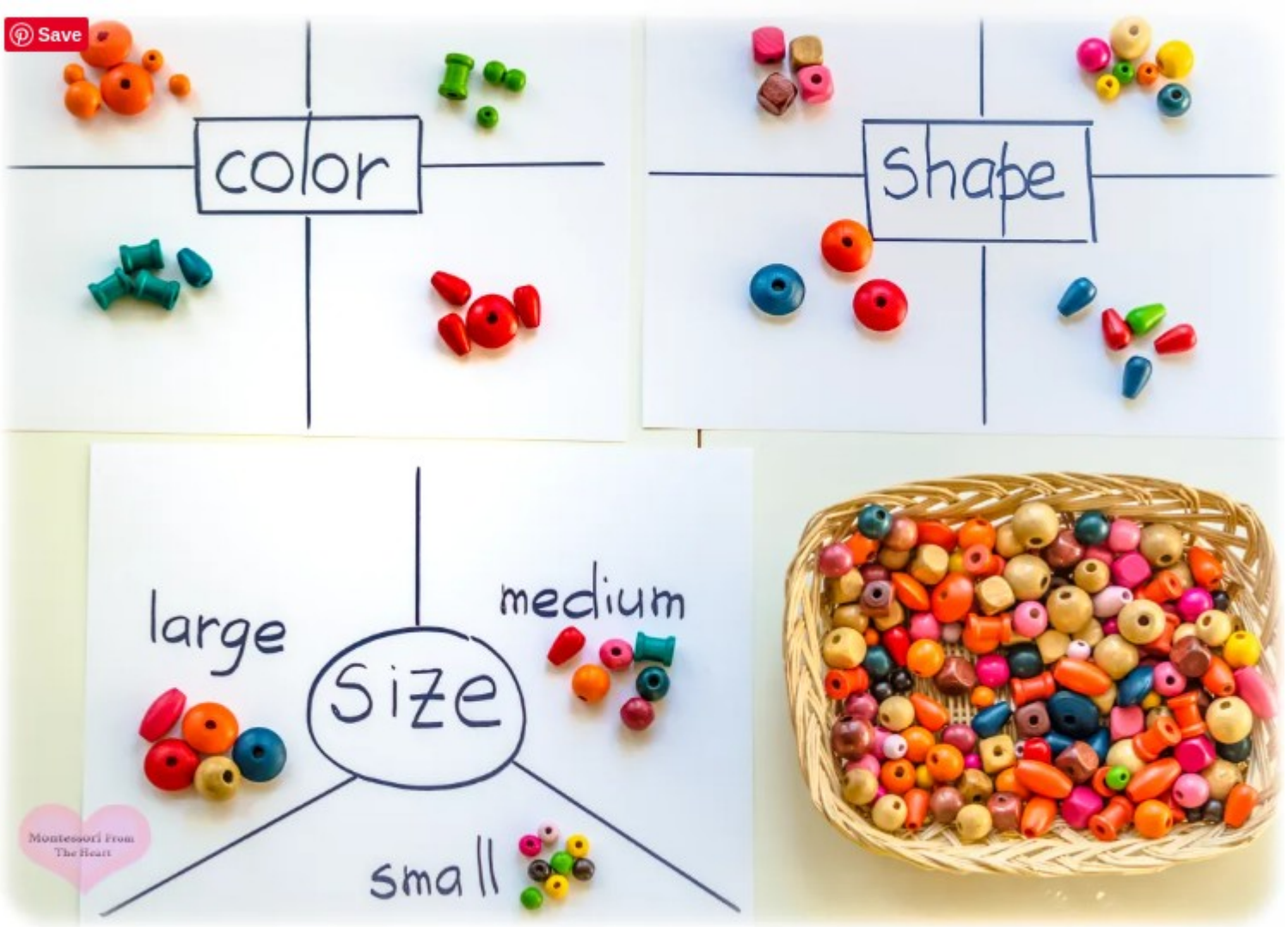
We'll see a few more examples of nested map functions later in lecture 21, 22

And Lab Exam #3

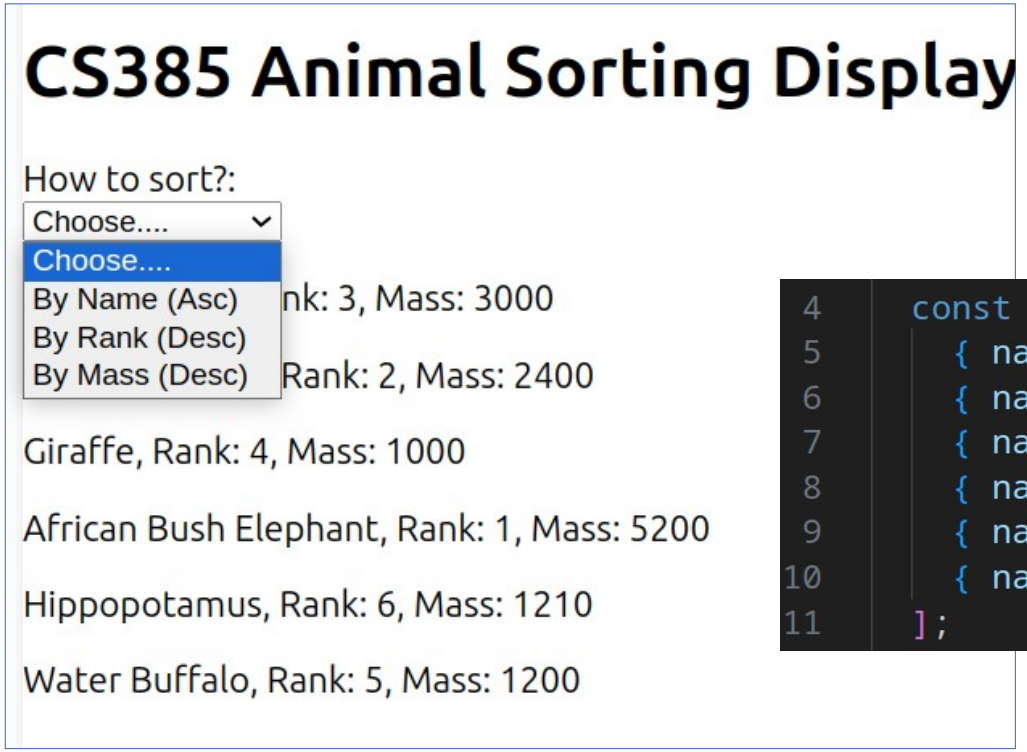
CS385 Mobile Application Development (Lecture 20)



Sorting (again)



Suppose in your application you want to provide different sorting options for the user




```
4   const animals = [
5     { name: "Indian Rhino", mass: 3000, rank: 3 },
6     { name: "Asian Elephant", mass: 2400, rank: 2 },
7     { name: "Giraffe", mass: 1000, rank: 4 },
8     { name: "African Bush Elephant", mass: 5200, rank: 1 },
9     { name: "Hippopotamus", mass: 1210, rank: 6 },
10    { name: "Water Buffalo", mass: 1200, rank: 5 },
11  ];
```

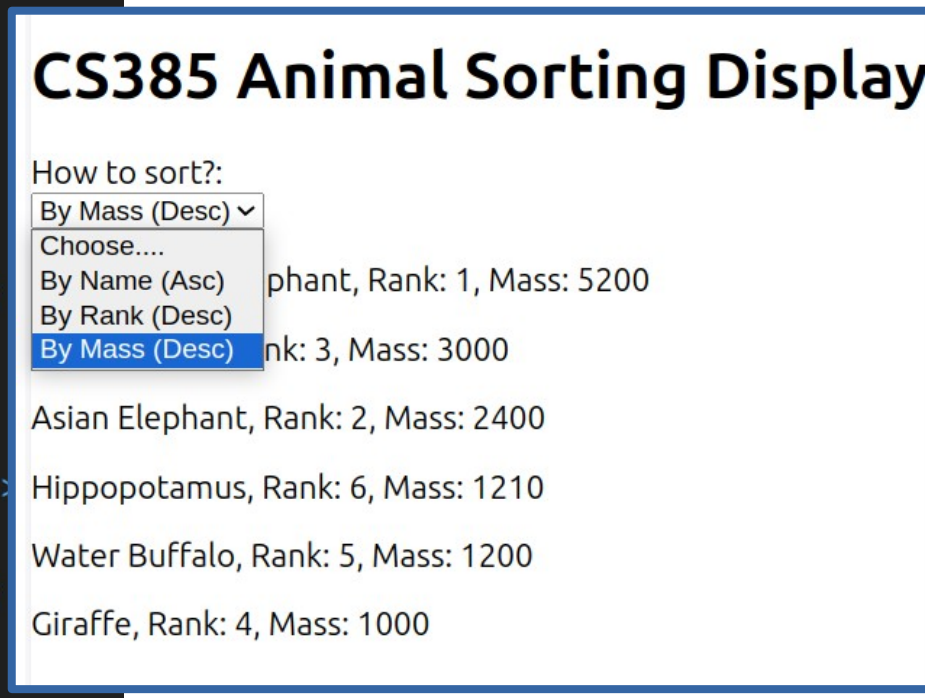
Without using a database/API (where a query could sort) – we must write sorting functions for each option

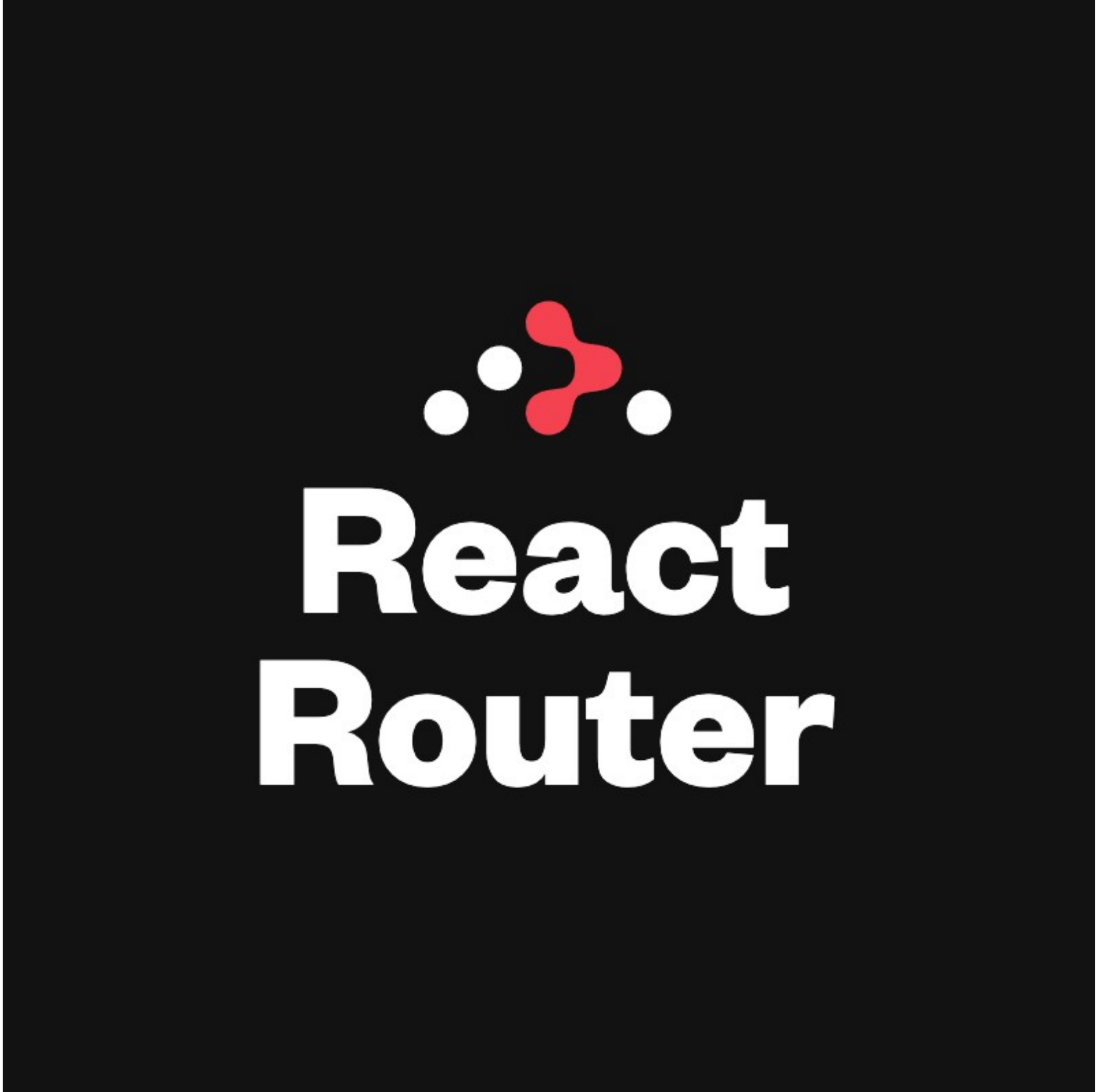
```
function sortAnimalsName(dx, dy) {
  let DX = dx.name.toUpperCase();
  let DY = dy.name.toUpperCase();
  if (DX > DY) return 1;
  else if (DX < DY) return -1;
  else return 0;
}
// for rank - we want smallest to largest values.
function sortAnimalsRank(dx, dy) {
  if (dx.rank > dy.rank) return 1;
  else if (dx.rank < dy.rank) return -1;
  else return 0;
}

// for mass we want largest to smallest values
function sortAnimalsMass(dx, dy) {
  if (dx.mass < dy.mass) return 1;
  else if (dx.mass > dy.mass) return -1;
  else return 0;
}
```

Then use conditional rendering to render the sorted data (based on the user's choice)

```
      <select onChange={handleListChange}>
        <option>Choose....</option>
        <option value="a">By Name (Asc)</option>
        <option value="b">By Rank (Desc)</option>
        <option value="c">By Mass (Desc)</option>
      </select>
    </form>
    {sortChoice === null &&
      animals.map((a, key) => (
        <p key={key}>
          {a.name}, Rank: {a.rank}, Mass: {a.mass}
        </p>
      ))}
    {sortChoice === "a" &&
      animals.sort(sortAnimalsName).map((a, key) =>
        <p key={key}>
          {a.name}, Rank: {a.rank}, Mass: {a.mass}
        </p>
      ))}
    {sortChoice === "b" &&
      animals.sort(sortAnimalsRank).map((a, key) => (
        <p key={key}>
          {a.name}, Rank: {a.rank}, Mass: {a.mass}
        </p>
      ))}
```




React
Router

What is React Router?

- **React Router is a standard library for routing in React.** It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.
- More simply put – it makes your React Application "look and feel" more like a traditional web-site (with links and navigation) than an SPA.
- **REFER – Lectures for Conditional Rendering using Components**

React Router – info

- **The Router in React JS is a pure JavaScript package that allows you to use React to create complicated client-side apps.** Initially launched in 2013, it has become one of the most prominent routing libraries in today's online applications.
- **React Router makes it simple to manage the URL and state of your application. You specify all of the potential URL patterns in your app and which UI component should be displayed for each one using React Router.** This Router decreases the amount of code an app requires to maintain its state and makes adding new features more accessible.
- **It enables the creation of single-page web or mobile apps that allow navigating without refreshing the page.** It also allows us to use browser history features while preserving the right application view.
- **A Router in React JS routes using a component-based architecture.** It offers various routing components as required by the application.

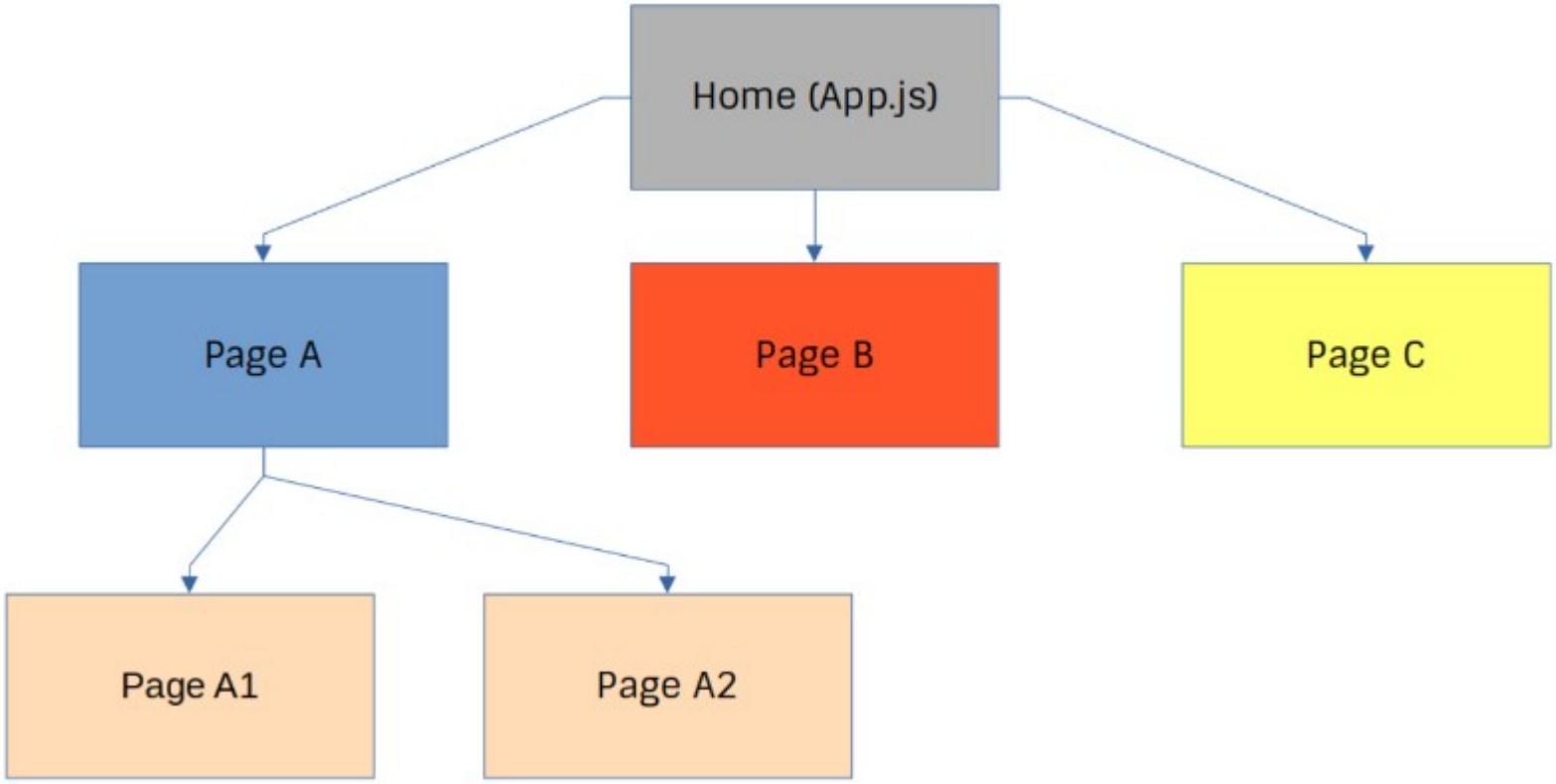
React Router – before you start

- **DEPENDENCY** – you'll need to add "**react-router-dom**" to your application.
- EXAMPLE – Let's create a few components (see above) which mimic a website/web app structure
- You may want to add Bootstrap or other CSS framework to style the pages

Before you code… make a plan

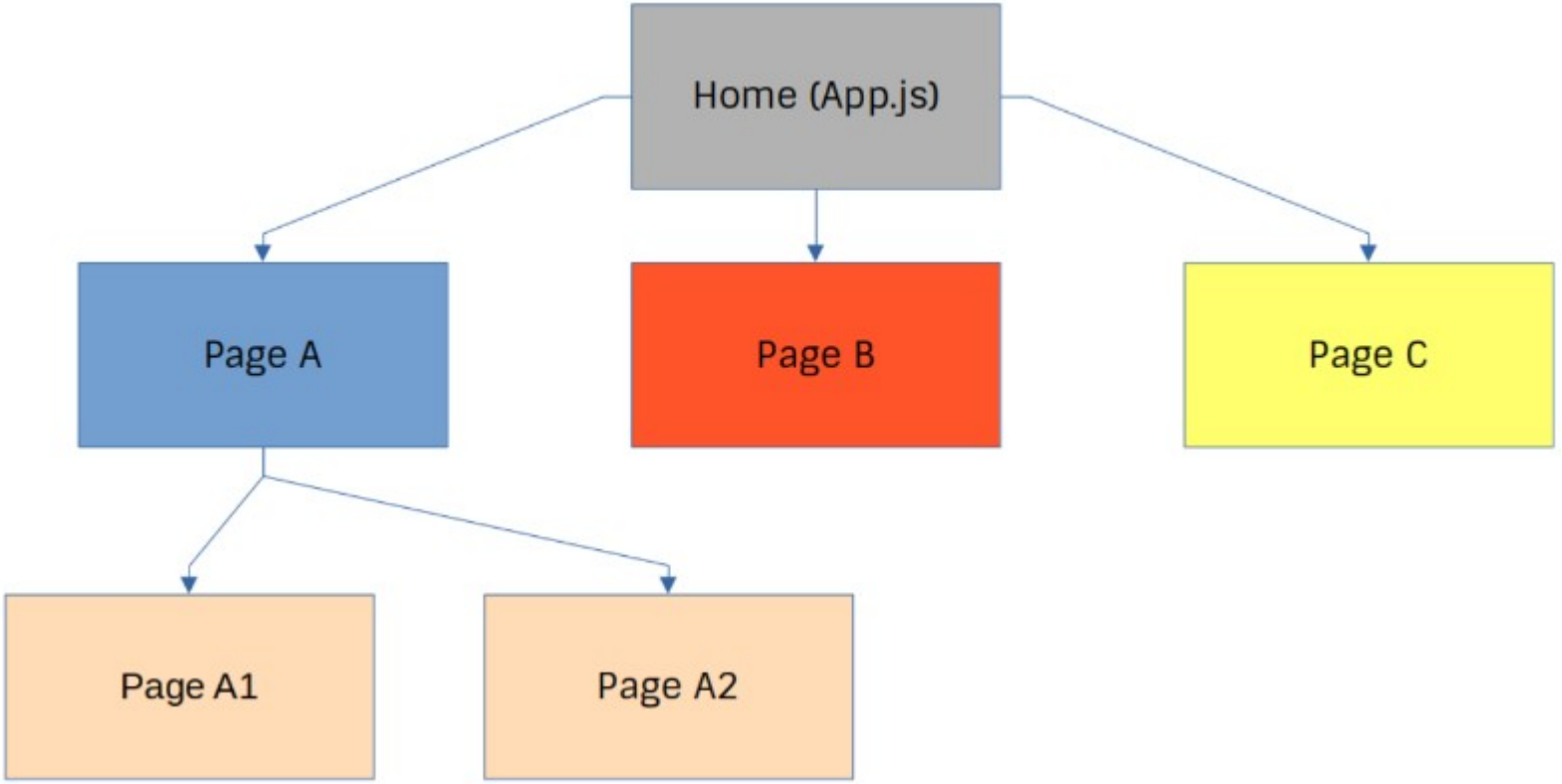
- Here is an overall schematic of what I want React Router to provide for me in my App

Thanks to Zhoukan Wang (Frenkie) for this code example



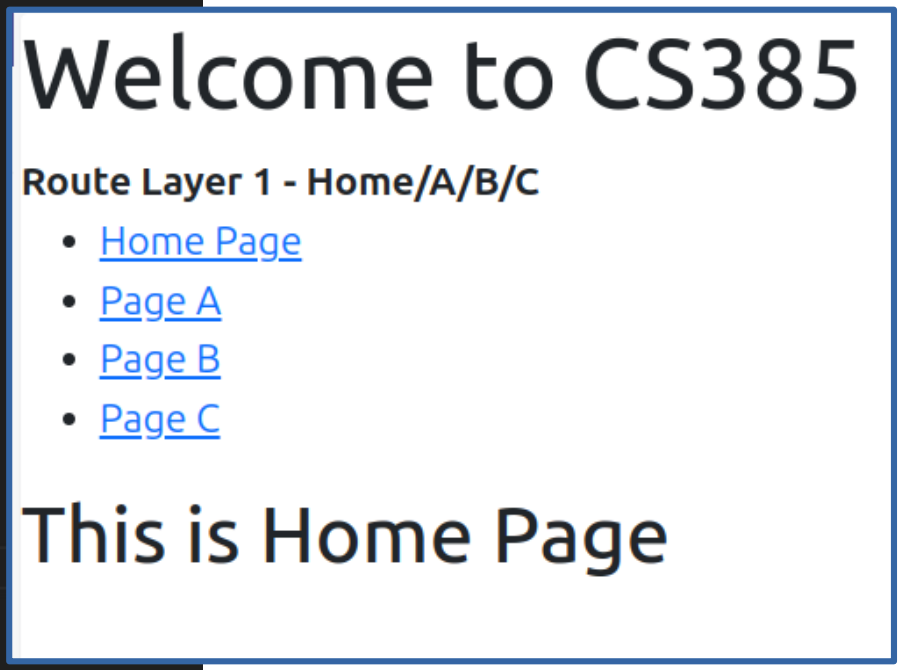
We have two route layers here

- Route Layer 1 – Page A, B, C from App.js
- Route Layer 2 – Page A1, A2 from Page A

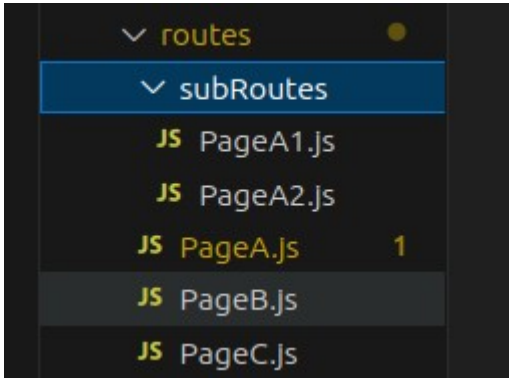


Setting up the Router and Routes within App.js (Home page)

```
10      <Router>
11        <div>
12          <h1>Welcome to CS385 React Router!</h1>
13          <b>Route Layer 1 - Home/A/B/C</b>
14          <ul>
15            <li>
16              <Link to="/">Home Page</Link>
17            </li>
18            <li>
19              <Link to="/a">Page A</Link>
20            </li>
21            <li>
22              <Link to="/b">Page B</Link>
23            </li>
24            <li>
25              <Link to="/c">Page C</Link>
26            </li>
27          </ul>
28          <Routes>
29            <Route path="/" element={<h2>This is Home Page</h2>} />
30            {/* Catch all route that begin with '/a' */}
31            <Route path="/a/*" element={<PageA />} />
32            <Route path="/b" element={<PageB />} />
33            <Route path="/c" element={<PageC />} />
34          </Routes>
35        </div>
36      </Router>
37    );
```



Setup of Page B and Page C is easily completed



src > routes > PageC.js > PageC

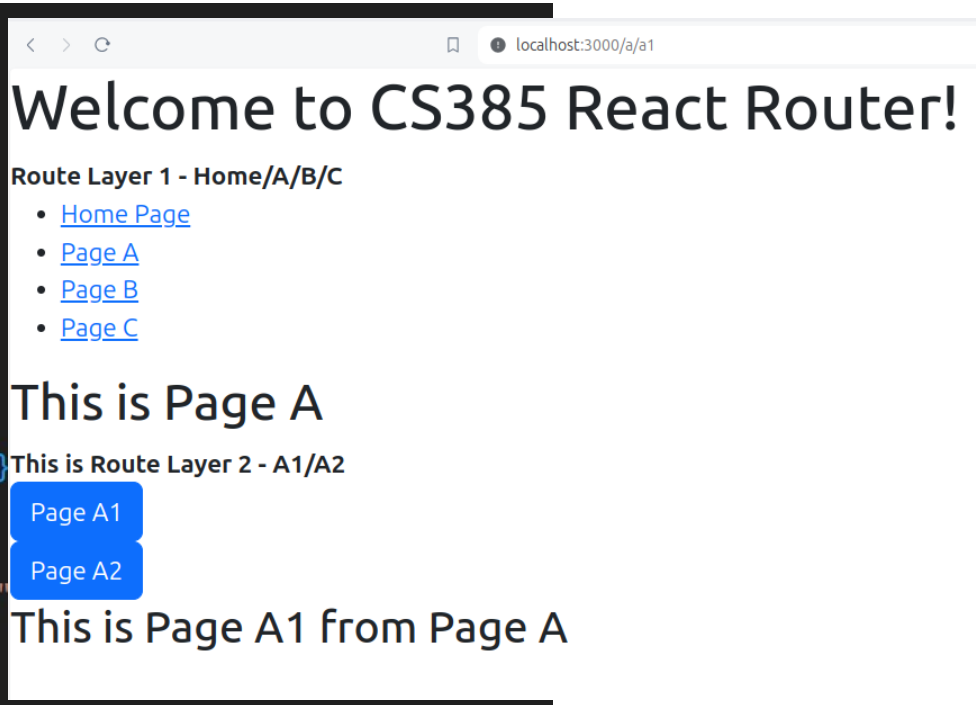
```
1  function PageC(){
2      return(
3          <h2>This is Page C</h2>
4      );
5  }
6
7  export default PageC;
```

src > routes > PageB.js > ...

```
1  function PageB() {
2    return <h2>This is Page B</h2>;
3  }
4
5  export default PageB;
6
```

There is more work setting up Page A (as it has sub routes)

```
3  import PageA1 from "./subRoutes/PageA1";
4  import PageA2 from "./subRoutes/PageA2";
5
6  function PageA() {
7    return (
8      <>
9        <h2>This is Page A</h2>
10       <b>This is Route Layer 2 - A1/A2</b>
11       {/* The current route of Page A is '/a'
12           The route of Page A1 would be '/a/a1'
13           The route of Page A2 would be '/a/a2' */}
14       <br />
15       <Link to="/a/a1">
16         <button type="button" class="btn btn-primary"
17       </Link>
18       <br />
19       <Link to="/a/a2">
20         <button type="button" class="btn btn-primary">Page A2</button>
21       </Link>
22
23       <Routes>
24         <Route path="a1" element={<PageA1 />} />
25         <Route path="a2" element={<PageA2 />} />
26       </Routes>
27     </>
28   );
29 }
30 export default PageA;
```



Look at the URL in the browser ... we see **a/a1** for this Page A1

Using state in React Router

- Working with ‘state’ in React Router is no different to any other React application.
- We simply use **props** – and remember that each ‘page’ in React Router is just a component.
- As before you need to plan the “**parent-child**” communication in order for the right state data to be routed to the correct pages.

Welcome to CS385 React Router!

Variable has the value 4

Increment Variable

Route Layer 1 - Home/A/B/C

- Home Page
- Page A
- Page B
- Page C

```
function App() {
  const [variable, setVariable] = useState(0);

  function changeVar() {
    setVariable(variable + 1);
  }
  return (
    <Router>
      <div>
        <h1>Welcome to CS385 React Router!</h1>
        <h2>Variable has the value {variable}</h2>
        <button onClick={changeVar}>Increment Variable</button>
```

This is Page A

This is Route Layer 2 - A1/A2

Page A1

Page A2

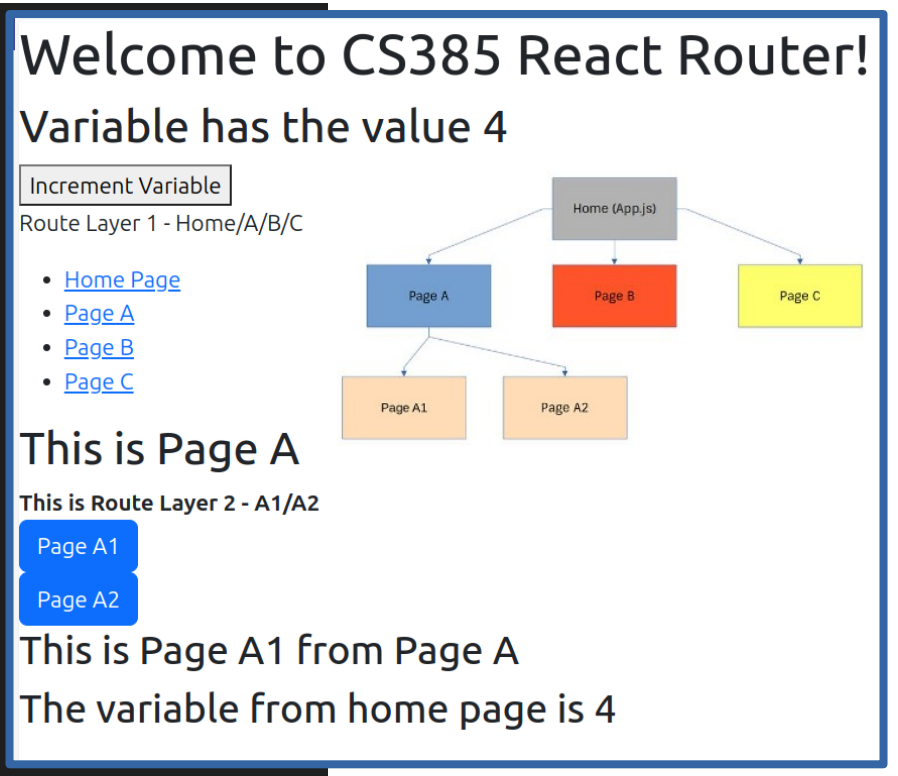
```
<Routes>
  <Route path="/" element={<h2>This is Home Page</h2>} />
  {/* Catch all route that begin with '/a' */}
  <Route path="/a/*" element={<PageA theVar={variable} />} />
  <Route path="/b" element={<PageB theVar={variable} />} />
  <Route path="/c" element={<PageC theVar={variable} />} />
</Routes>
```

This is Page A1 from Page A

The variable from home page is 4

```jsx
function PageA(props) {
  return (
    <>
      <h2>This is Page A</h2>
      <b>This is Route Layer 2 - A1/A2</b>
      {/* The current route of Page A is '/a'
          The route of Page A1 would be '/a/a1'
          The route of Page A2 would be '/a/a2' */}
      <br />
      <Link to="/a/a1">
        <button type="button" class="btn btn-primary">
          Page A1{" "}
        </button>
      </Link>
      <br />
      <Link to="/a/a2">
        <button type="button" class="btn btn-primary">
          Page A2
        </button>
      </Link>

      <Routes>
        <Route path="a1" element={<PageA1 theVar2={props.theVar} />} />
        <Route path="a2" element={<PageA2 />} />
      </Routes>
    </>
```



src > routes > subRoutes > JS PageA1.js > PageA1

```jsx
function PageA1(props) {
  return (
    <>
      <h3>This is Page A1 from Page A</h3>
      <h3>The variable from home page is {props.theVar2}</h3>
    </>
  );
}
```

Considerations on using React Router in your application

- **React router can be an overkill for certain projects** where all you need is basic navigation and routing functionalities.
- That said, **React Router is rich with navigational components that compose declaratively with your application**, which can be very useful for larger and more complex navigational requirements in React applications.
- Component-based 'routing' (by using state variables) can be a good alternatively.
- React Router is appealing to those with web development backgrounds

React Router – Advantages

- **Declarative Routing:** You can describe the navigation structure in a straightforward and readable manner using JSX.
- **Nested Routing:** React Router supports nested routing, allowing you to organize your application into modular components and routes. This makes it easy to manage complex UI structures and layouts.
- **Dynamic Routing:** You can handle dynamic routes and URL parameters easily with React Router.
- **Integration with React Ecosystem:** React Router integrates seamlessly with React, making it a natural choice for handling navigation in React applications.
- **Community and Documentation:** It is well-documented with extensive guides and examples, making it easy for developers to get started and find solutions to common problems.

React Router – Disadvantages

- **Learning Curve:** For beginners, there might be a learning curve in understanding how to set up and configure React Router.
- **Overkill for Simple Apps:** For very simple applications or static websites, using React Router might be overkill.
- **Routing Configuration Complexity:** As the application grows, managing and organizing a complex routing configuration might become challenging. It's important to design the routing structure carefully to avoid confusion

Remember …. React Router is OPTIONAL for all CS385.

There may be ONE question in Lab Exam 3 about React Router

Live – in class annoucements

Project update
Lab Exam 3
Lectures 21-22