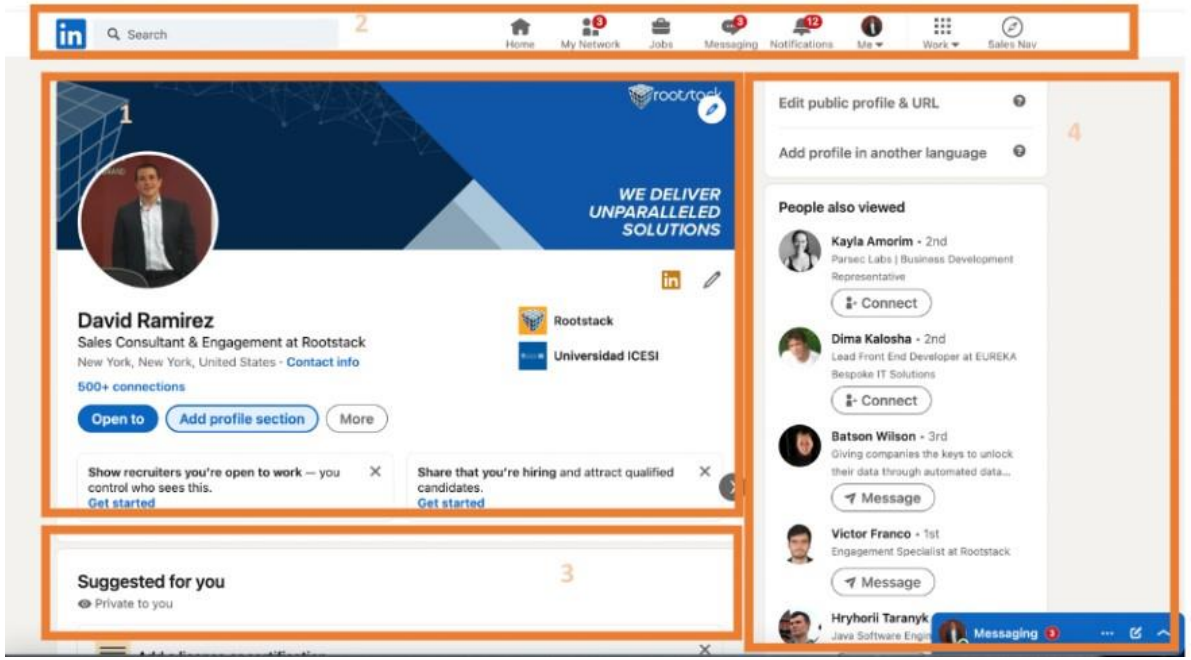
Component Driven Development

RUJUAN

How React.js components work

React is “component-based”, which means the user builds encapsulated components that manage their own state and then composes them to make complex UIs, each of these components can have any amount of “sub-components” which, together with the components, are all re-usable in other components or even in other applications.

To illustrate these “components” let´s look at our LinkedIn page:



The process of CDD

1. Component Decomposition: Splitting Interfaces, Extracting Components
2. Implement Static Components: Using Components to Achieve Static Page
3. Implement dynamic Components
    1) Display Initial Data Dynamically
        1) Data type
        2) Data name
        3) Which component to save data?
    2) Interact with user (start from listening events)

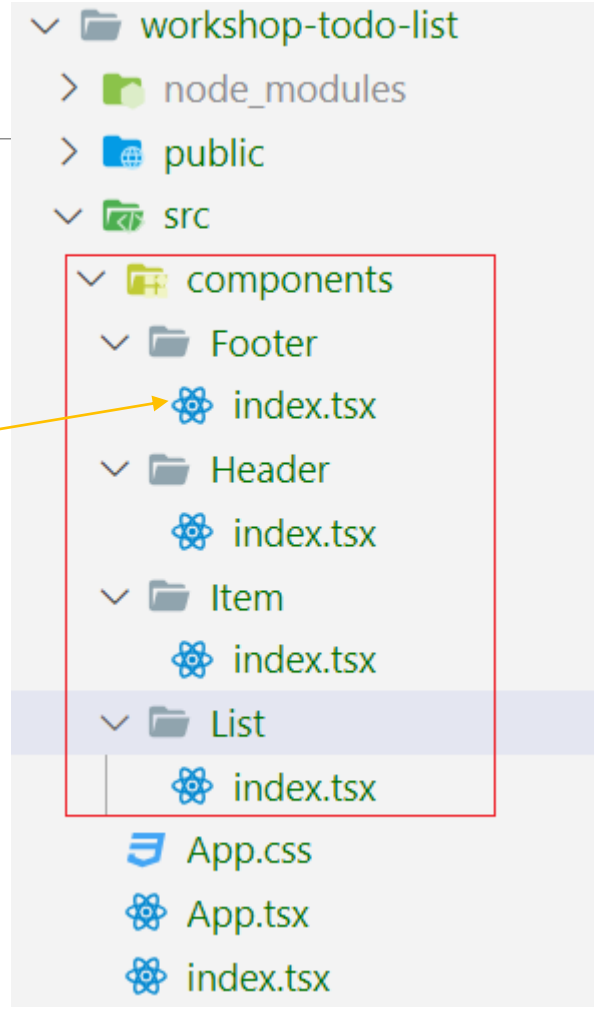
Workshop – Todo List

1. Component Decomposition

1. Component Decomposition

```
import React from 'react'

export default function Footer() {
  return (
    <div>Footer</div>
  )
}
```



2. Implement Static Components

First, we put everything inside App.tsx to ensure the static pages function properly

- `class` vs `className` – need to change `class` to `className` for external style
- update inline style – `style={{display:'none'}}`
- import external css
  - Create `App.css`, put all css rules here
  - Import into `App.tsx` – `import './App.css';`

2. Implement Static Components (cont.)

Second, Breaking down the content within App.tsx, then place the content in different components

```
function App() {
  return (
    <div className="todo-container">
      <div className="todo-wrap">
        <Header />
        <List />
        <Footer />
      </div>
    </div>
  );
}
```

```
export default function Header() {
  return (
    <div className="todo-header">
      <input type="text" placeholder="Enter task name" />
    </div>
  )
}
```

```
export default function List() {
  return (
    <ul className="todo-main">
      <Item />
    </ul>
  )
}
```

```
export default function Item() {
  return (
    <li>
      <label>
        <input type="checkbox" />
        <span>xxxxx</span>
      </label>
      <button className="btn btn-danger" style={{ display:
'none' }}>Delete</button>
    </li>
  )
}
```

2. Implement Static Componer

Third, Breaking down the content within App.css, then place the content in different components

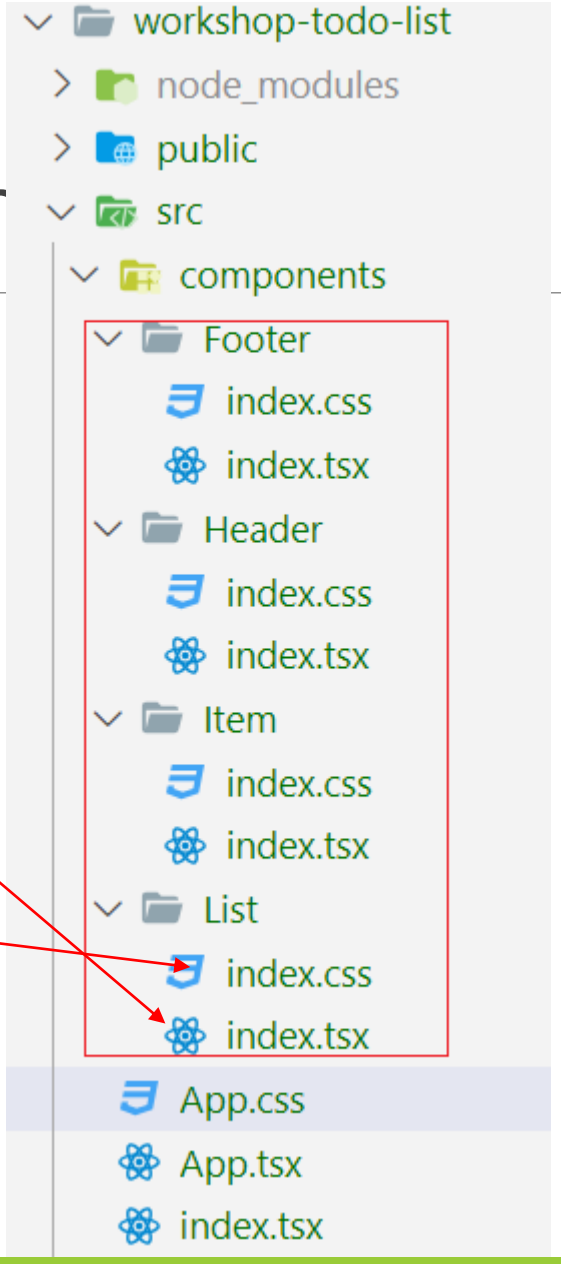
```css
/*main*/
.todo-main {
    margin-left: 0px;
    border: 1px solid #ddd;
    border-radius: 2px;
    padding: 0px;
  }

  .todo-empty {
    height: 40px;
    line-height: 40px;
    border: 1px solid #ddd;
    border-radius: 2px;
    padding-left: 5px;
    margin-top: 10px;
  }
```

```jsx
import Item from '../Item';

import './index.css';


export default function List() {
  return (
    <ul className="todo-main">
      <Item />
    </ul>
  )
}
```



3. Implement dynamic Components - Display Initial Data Dynamically

1. use json-server to host default todo list
2. fetch data from server
3. Display on the page

```
<ul className="todo-main">
      {todos?.map(todo => <Item key={todo.id} {...todo} /> )}
</ul>
```

```
function App() {
  const [todos, setTodos] = useState<Todo[] | null>(null);

  useEffect(() => {

    async function getList(){
      const res = await fetch('http://localhost:3004/todos');
      const data = await res.json();
      setTodos(data);
    }
    getList();
  }, []);

  return (
    <div className="todo-container">
      <div className="todo-wrap">
        <Header />
        <List todos={todos}/>
        <Footer />
      </div>
    </div>
  );
}
```

4. Add a new Todo

The todo list resides in the App component, while the input is located in the Header component. We require a method to transmit values from the Header component to the App component.

Ideas:

1. The relationship between App component and Header Component is parent to child

2. Think about how to communicate between parent and child.

5. Upate state when check/uncheck Todos

When user checks or unchecks a Todo, the state should be changed.

Ideas:

- Which component should we place the code for update the state of todo list?
  - General guideline: the same place where the todo list located.
- The checkbox is in Item component, how could we pass the updateTodo function to Item component.
  - Two ways: props or context

Code

```
const updateTodo = (id: string, done: boolean) => {
    const newTodos = todos!.map(todo => {
      if (todo.id === id) return ({ ...todo, done });
      else return todo;
    });
    setTodos(newTodos);
  }
```

```
<TodoUpdateContext.Provider value={updateTodo}>
          <List todos={todos} />
</TodoUpdateContext.Provider>
```

```
  const updateTodo = useContext(TodoUpdateContext);

  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    updateTodo!(id, e.target.checked);
  }
```

6. Delete hovered todo

When hovering over each todo item, a "delete" button is displayed. Upon clicking this button, a confirmation dialog appears asking, "Are you sure you want to delete this item?" If the user clicks the "OK" button in the dialog, the todo item will be deleted.

Ideas:

1. the deleteTodo function should be in App component, how could we pass the function to Item component?

2. attach event handler on ‘delete’ button for delete feature.

7. Footer Features

1. Finished: count the length of todo list where done:true

2. Total: count the length of todo list

3. Checkbox: When the user checks it, all above to-do items should also be checked. Similarly, when the user unchecks it, all above to-do items should also be unchecked. As the user checks off each above to-do item individually, this checkbox should be automatically marked as checked.

4. Delete Finished Tasks: when click this button, all tasks marked with done: true should be removed.