Context for today

Today, we're going to learn about

1. DOM manipulation (slightly more advanced)

2. Create a mock Reconcilers

3. State and State management

Complex DOM manipulation

Creating a DOM element which has another DOM element inside

Lets write some code in which you have a button. When you click on a button, a slightly complex DOM element gets appended to the DOM.

```
<div>
    <h1>hi there<h1>
</div>
```

Approach #1

```html
<body>
    <button onclick="createComplexDomElement()">Add</button>
</body>
<script>
   function createComplexDomElement() {
      const div = document.createElement("div");
      div.innerHTML = "<h1> hi there </h1>";
      document.querySelector("body").appendChild(div);
   }
</script>
```

Finishing the TODO App - 2

Let's look at a slightly better approach of doing the same thing.

Creating a DOM element which has another DOM element inside

```html
<body>
    <button onclick="createComplexDomElement()">Add</button>
</body>
<script>
  function createComplexDomElement() {
    const div = document.createElement("div");
    const h1 = document.createElement("h1");
    h1.innerHTML = "random text";
    div.appendChild(h1);
    document.querySelector("body").appendChild(div);
  }
</script>
```

TODO app

Can you now create a TODO application with the slightly complex approach of appending DOM elements?

```html
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Todo List</title>
  <link href="style.css" rel="stylesheet" type="text/css" />
</head>

<body>
  <h1>Todo list</h1>
  <div id="todos">
    <div id="todo-1">
      <h4>1. Take class</h4>
      <button onclick="deleteTodo(1)">Delete</button>
    </div>
    <div id="todo-2">
      <h4>2. Go out to eat</h4>
      <button onclick="deleteTodo(2)">Delete</button>
    </div>
  </div>
  <div>
```

do</button>

```
</div>

<script>
let currentIndex = 3;

function addTodo() {
  const inputEl = document.getElementById("inp");
  const todoText = inputEl.value.trim();

  if (todoText === '') {
    alert('Please enter a todo item.');
    return;
  }

  const parentEl = document.getElementById("todos");

  // Create new todo div
  const newTodo = document.createElement('div');
  newTodo.setAttribute("id", 'todo-' + currentIndex);

  // Create new heading element
  const newHeading = document.createElement('h4');
  newHeading.textContent = currentIndex + '. ' + todoText;

  // Create new button element
  const newButton = document.createElement('button');
  newButton.textContent = 'Delete';
  newButton.setAttribute("onclick", "deleteTodo(" + currentIndex + ")");

  // Append elements to the new todo div
  newTodo.appendChild(newHeading);
  newTodo.appendChild(newButton);

  // Append new todo to the parent element
  parentEl.appendChild(newTodo);

  // Increment the index for the next todo item
  currentIndex++;
```

inputEl.value = '';

DOM Part 2 1 of 8

```
    function deleteTodo(index) {
      const element = document.getElementById("todo-" + index);
      if (element) {
        element.parentNode.removeChild(element);
      }
    }
  </script>
</body>

</html>
```

State derived frontends

To make frontends easier to code, the concept of state came into the picture. You will see this more when we reach react.

There are three jargon we need to understand

1. State - The variable parts of an app.

2. Components - How to render   state on screen.

3. Rendering - Taking the state and rendering it on the DOM based on the
   components

State

```javascript
const todos = [{
  id: 1,
  description: "Go to gym"
}, {
  id: 2,
  description: "Eat food"
}];
```

Component

```javascript
function todoComponent(todo) {
  const div = document.createElement("div");
  const h1 = document.createElement("h1");
  const button = document.createElement("button");
  button.innerHTML = "Delete";
  h1.innerHTML = todo.title;
  div.appendChild(h1);
  div.appendChild(button);
}
```
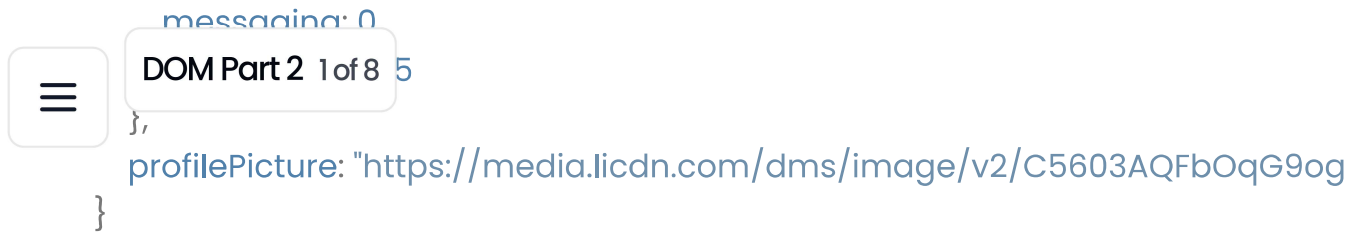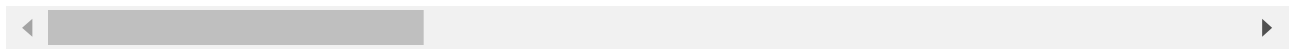
Linkedin Topbar

State

```javascript
const state = {
  notifications: {
    home: 0,
```

```
    messaging: 0
    },
    profilePicture: "https://media.licdn.com/dms/image/v2/C5603AQFbOqG9og
    }
```

Components

Started code

```html
<body>
 <input type="text"></input>
 <button onclick="addTodo()">Add todo!</button>
 <script>
  let todos = [];
  function addTodo() {
   todos.push({
    title: document.querySelector("input").value
   })
   render();
  }

  function deleteTodo() {

   render();
  }

  function render() {

  }
 </script>
```

State derived rendering

Given a state variable called todos , can you write a function called render that takes this as an input and renders the current list of todos

Todos look something like this -

```js
const todos = [{
    id: 1,
    title: "Go to gym"
}, {
    id: 2,
    title: "Clean the car"
}]
```

Boilerplate code

```html
<body>
  <div id="root"></div>
  <script>
    function render(todos) {
      // your code here
    }
  </script>
</body>
```

▼ Approach #1 - Clean the screen everytime we re-render

```
<script>

                  er(todos) {
    const todoList = document.getElementById('root');
    todoList.innerHTML = ''; // Clear the list

    todos.forEach(todo => {
      const div = document.createElement('div');
      const h1 = document.createElement('h4');
      h1.textContent = todo.title;
      div.appendChild(h1);
      div.setAttribute('data-id', todo.id);
      todoList.appendChild(div);
    });
  }
  render([{
    id: 1,
    title: "Go to gym"
  }, {
    id: 2,
    title: "Clean the car"
  }])
</script>
</body>
```

DOM Part 2 1 of 8

💡 There is a better approach —– You find the diff and only do deletes /
 updates / additions that are necessary. But that'll boggle most folks
heads so we're not going there. The general goal should be to
minimize the number of interactions in the DOM.
React does this by using something called the virtual DOM.

Add TODO functionality

Lets add the functionality to

1. Add more TODOs

2. Delete functionality

> 💡 You only need to update state and call the `render` function. You `DONT` need to do the actual DOM manipulations, the `render` function will do it for you.

▼ Solution

```html
<body>
  <div id="root"></div>
  <div>
    <input type="text"></input>
    <button onclick="addTodo()">Add Todo</button>
  </div>
  <script>
    let ctr = 2;
    let todos = [{
      id: 1,
      title: "Go to gym"
    }, {
      id: 2,
      title: "Clean the car"
    }]

    function addTodo() {
      todos.push({
        id: ctr,
        title: document.querySelector("input").value
      })
      render(todos);
```

```
function render(todos) {
    t = document.getElementById('root');
    todoList.innerHTML = ''; // Clear the list

    todos.forEach(todo => {
      const div = document.createElement('div');
      const h1 = document.createElement('h4');
      h1.textContent = todo.title;
      div.appendChild(h1);
      div.setAttribute('data-id', todo.id);
      todoList.appendChild(div);
    });
  }
  render(todos)
 </script>
</body>
```

DOM Part 2 1 of 8

Delete functionality

Can you add the delete functionality next?

💡 Again, we don't have to do any DOM manipulations here. It's all handled by our render function.

Started code

```html
<button onclick="addTodo()">Add todo!</button>

<button onclick="deleteLastTodo()">Delete last todo</button>

<button onclick="deleteFirstTodo()">Delete first todo</button>
<div id="todos"></div>
<script>
 let todos = [];
 function addTodo() {
   todos.push({
     title: document.querySelector("input").value
   })
   render()
 }

 function deleteLastTodo() {
   todos.splice(todos.length - 1, 1) // remove the last element from the arr
   render()
 }


 function deleteFirstTodo() {
   todos.splice(0, 1) // remove the last element from the arr
   render()
 }


 function createTodoComponent(todo) {
   const div = document.createElement("div");
   const h1 = document.createElement("h1");
   const button = document.createElement("button");
   button.innerHTML = "Delete"
   h1.innerHTML = todo.title;
   div.append(h1)
   div.append(button)
   return div
 }
```

```
      document.querySelector("#todos").innerHTML = "";
      DOM Part 2  1 of 8  odos.length; i++) {
      const element = createTodoComponent(todos[i]);
      document.querySelector("#todos").appendChild(element)
    }
  }
```

```
    </script>
  </body>
```

Code - https://github.com/Master-utsav/Render-Todo

$50