✅

📌Lecture 32 [Frontend 5] - React State and Its Lifecycle

Lecture 32 [Frontend 5] - React State and Its Lifecycle

%[https://youtu.be/F7zCmo7XM4Q]

Introduction

আজকে আমরা স্টেট এবং এর লাইফসাইকেল নিয়ে আলোচনা করবো। তার আগে আমরা একটু `jsx` নিয়ে আলোচনা করবো। কারণ আমরা মোটামুটি রিয়্যাক্টের অনেক কিছু শিখেছি এতদিনে। এখন যদি `jsx` ভালভাবে শিখতে পারি তাহলে আমরা অনেকখানি এগিয়ে যেতে পারবো। `jsx` এর অনেক কিছু শেখার আছে।

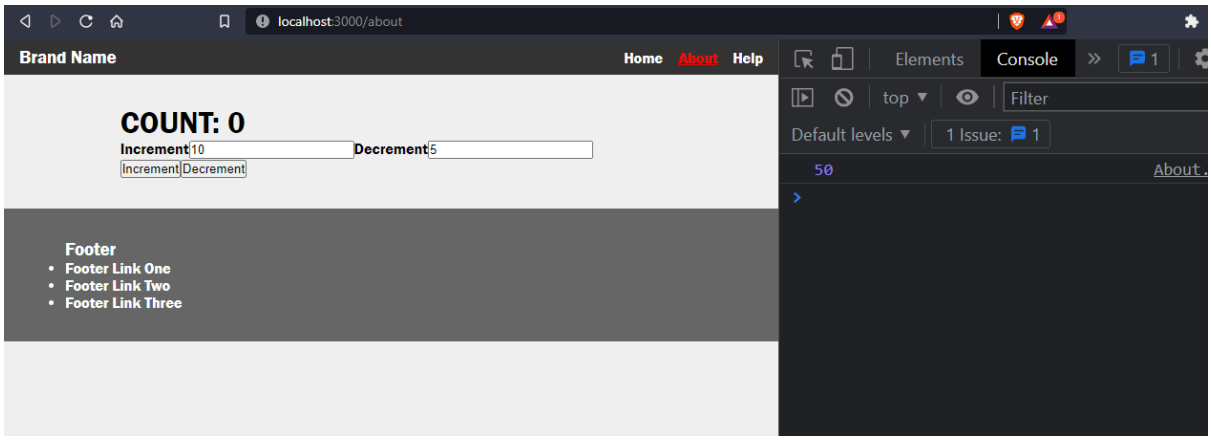
Short-circuit in JavaScript

জাভাস্ক্রিপ্টে শর্টসার্কিট বলে একটা টার্ম আছে। সেটা আমরা
জেনে হোক না জেনে হোক সবাই ব্যবহার করেছি। জাভাস্ক্রিপ্টের অর (||) এবং
অ্যান্ড (&&) অপারেটরগুলো ব্যবহার করে শর্টসার্কিট ইভ্যাল্যুয়েশন
করা হয়। নরমালি আমরা এসব অপারেটর ব্যবহার করি আমাদের কন্ডিশনে। যেমন `if(a || b)` বা `if(a && b)`
এভাবে। যখন আমরা কন্ডিশনের এগুলো ব্যবহার করবো তখন কিন্তু শর্ট সার্কিট
ইভ্যাল্যুয়েশন বলা যাবে না। তখন এগুলো জাস্ট লজিক্যাল অপারেটর। শর্ট
সার্কিট ইভ্যাল্যুয়েশন হবে অন্য জায়গায় যদি আমরা এগুলো ব্যবহার করি। যেমন
আমরা আমদের গত ক্লাসের করা কোডের pages/About.jsx ফাইলে যাই। এরপর নিচের কোডটা লিখবো।

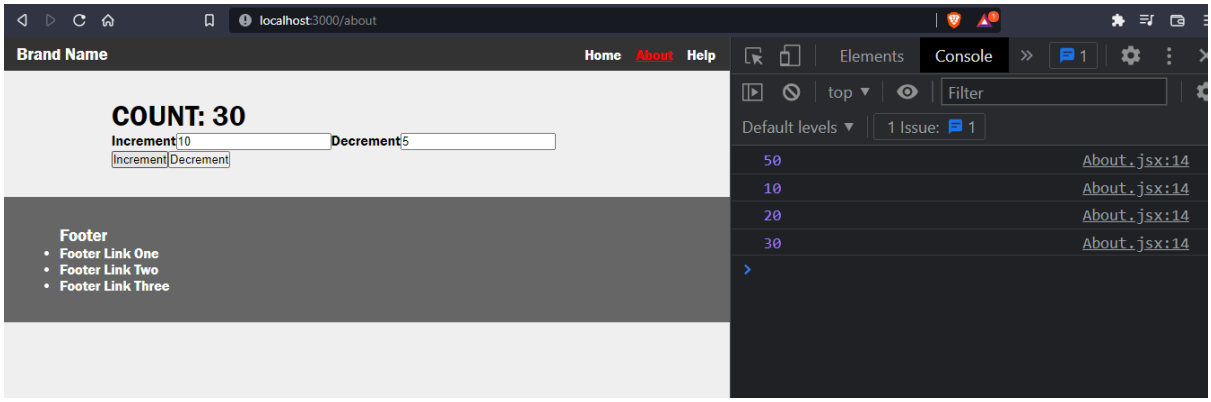
```
const [count, setCount] = useState(0);

const o = count || 50;
```

এখনে `const o = count || 50;` হলো শর্ট সার্কিট ইভ্যাল্যুয়েশন। এই জিনিস রিয়্যাক্টে আমরা প্রচুর ব্যবহার করবো। এটার মানে হচ্ছে যদি `count` এর ভ্যালু `falsy` বা `0` হয় তাহলে `o` এর

ভ্যালু হবে `50`, আর যদি `falsey` বা `0` না হয় তাহলে `o` এর ভ্যালু দাঁড়াবে `count` এর ভ্যালু। এখন যদি আমরা অ্যাপ্লিকেশন রান করে ব্রাউজারে গিয়ে কনসোল দেখি, দেখবো সেখানে 50 দেখাচ্ছে।



কারণ প্রাথমিক অবস্থায় `count` এর স্টেট হচ্ছে `0` যা `falsy` ভ্যালু, তাই `o` এর ভ্যালু `50` দেখাচ্ছে। এখন যদি আমরা `Increment` বাটনে ক্লিক করে স্টেট চেইঞ্জ করি তাহলে দেখতে পাবো, স্টেট চেইঞ্জের সাথে সাথে আমাদের `o` এর ভ্যালুও পরিবর্তিত হচ্ছে।

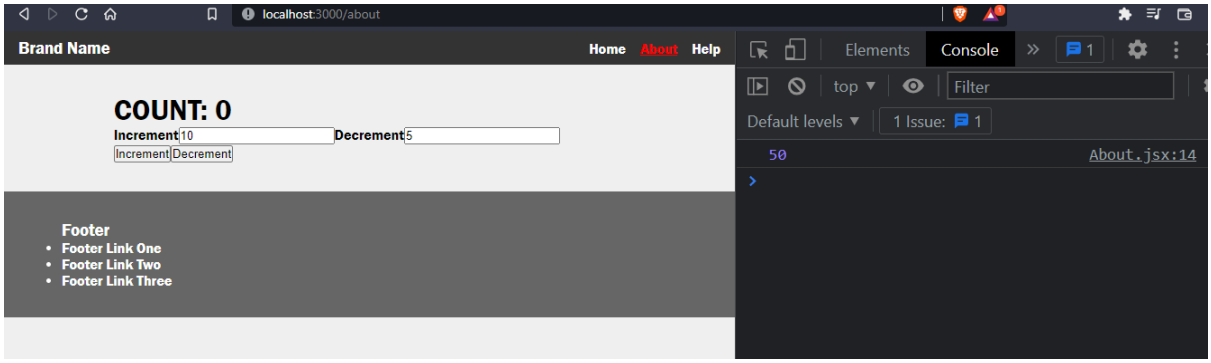


তার মানে হলো যদি বাম পাশের ভ্যালু কোনো `falsy` ভ্যালু হয় তাহলে ডান পাশের ভ্যালুটাই হবে আমাদের আসল ভ্যালু এবং তা আমাদের ভ্যারিয়েবলের মধ্যে বসে যাবে। আর যদি বাম পাশের ভ্যালু `truthy`
হয় তাহলে আমাদের আসল ভ্যালু হবে বাম পাশের ভ্যালুটাই এবং সেটাই আমাদের ভ্যারিয়েবলে স্টোর হবে। এটাই হচ্ছে শর্টসার্কিটের একমাত্র কনসেপ্ট। এটা ছাড়া আর কোনো কনসেপ্ট নাই।

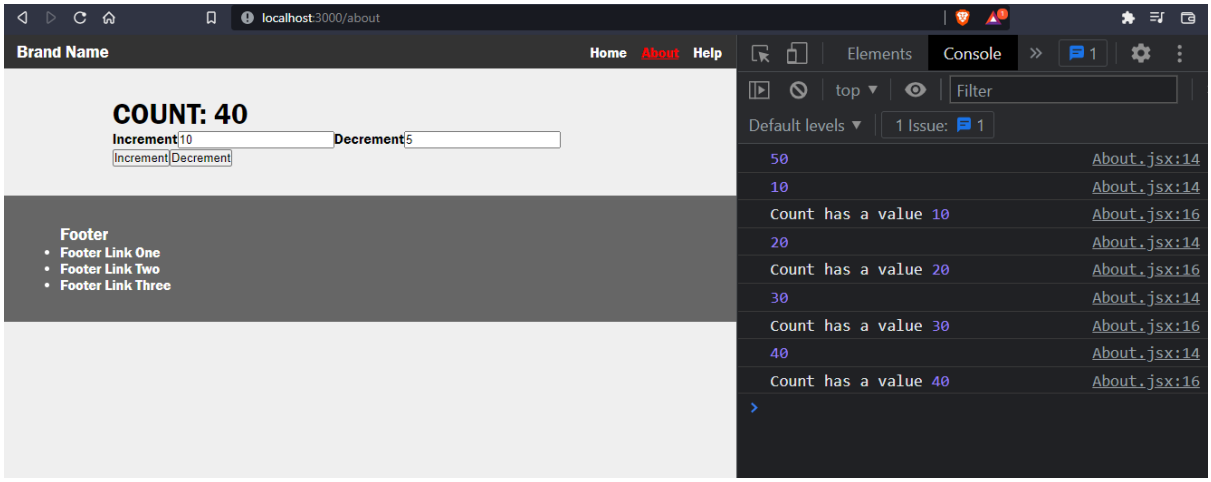
এবার আমরা `&&` অপারেটর ব্যবহার করে শর্টসার্কিটের একটা উদাহরণ দেখবো। `||` অপারেটর ব্যবহার হয় যখন আমরা কন্ডিশনালি কোনো ভ্যালু ভ্যারিয়েবলের মধ্যে অ্যাসাইন করতে চাই। কিন্তু যখন আমরা কোনো কোড এক্সিকিউট করতে চাই কিনা সেই ডিসিশন নিতে যাই, তখন আমরা ব্যবহার করবো `&&` অপারেটর। তখন আর `||` অপারেটর কাজ করবে না। যেমন -

```
count && console.log('Count has a value', count);
```

এর মানে হলো যদি `count` এর
ভ্যালু থাকে অর্থাৎ এটা সত্য রিটার্ন করে তবেই ডান পাশের অংশ এক্সিকিউট হবে। নাহয় হবে না। এখানে ডান পাশে ফাংশন, কন্ডিশন, এক্সপ্রেশন ইত্যাদি অনেক কিছুই থাকতে পারে। সাধারণত আমরা এক্সপ্রেশন নিয়ে কাজ করে থাকি। এবার যদি আমরা আমাদের কনসোলে গিয়ে দেখি দেখবো এটা প্রিন্ট হয়নি। কারণ `count` এর ভ্যালু `truthy` না। সেজন্য সেটা দেখা যাবে না।



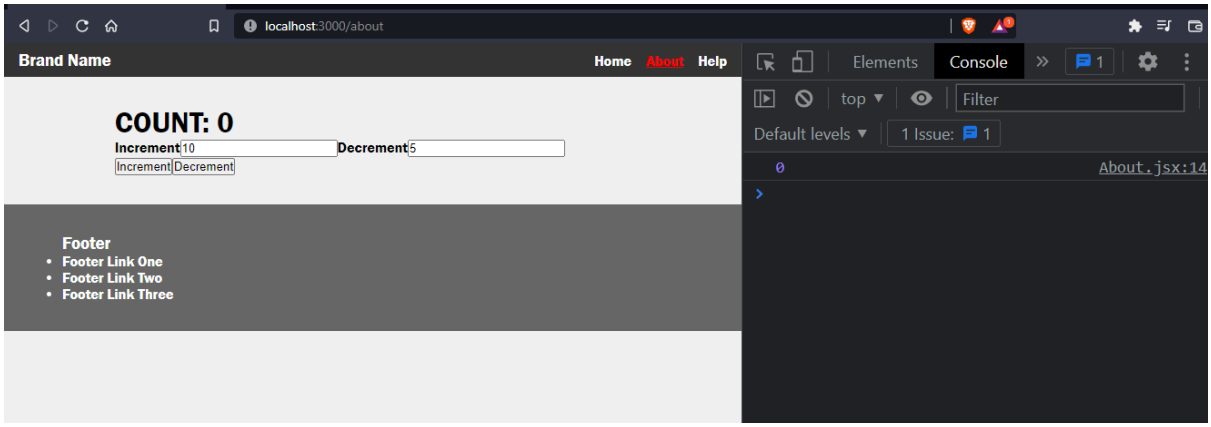
কিন্তু যদি আমরা ইনক্রিমেন্ট করি তাহলে দেখা যাবে এই লাইনটা প্রিন্ট হতে থাকবে।



এই দুইটা কনসেপ্টের সাথে আরো একটা কনসেপ্ট যুক্ত হয়েছে। সেটা হলো `nullish coalescing` অপারেটর বা `??`। এটার মানে হলো যদি বাম পাশের অংশ `null` বা `undefined` হয় তখন তা ডান পাশের অংশ রিটার্ন করে। যেমন আমাদের উদাহরণে আমি যদি `0` ভ্যালুটাও পেতে চাই সেটা `||` দিয়ে সম্ভব হয় না। কারণ সে `falsey` ভ্যালু দেখলেই সেটাকে স্কিপ করে যাবে। সেই সমস্যার সল্যুশন হলো এই `??` অপারেটর। এটা শুধুমাত্র বাম পাশের অংশ `null` বা `undefined` হলেই ডান পাশের অংশে যাবে। অন্য যেকোনো ভ্যালুর জন্য সে বাম পাশের ভ্যালু রিটার্ন করবে। এখন যদি আমরা আমাদের কোডকে এই অপারেটর দিয়ে লিখি, যেমন -

```
const o = count ?? 50;
```

তাহলে কনসোলে গেলে দেখবো `0` প্রিন্ট করেছে, যেটা আমরা `||` ব্যবহার করে পাইনি।



এগুলোর কাজ আমরা বিভিন্ন জায়গায় বিভিন্নভাবে দেখতে পাবো। বিশেষ করে `jsx` এর ভিতর এটা ব্যাপকভাবে ব্যবহৃত হয়।

মূলত `||` এবং `&&` কে বলা হয় **শর্টসার্কিট ইভ্যাল্যুয়েশন**।

JSX

JSX সম্পর্কে আমরা আগের ক্লাসগুলোতে জেনেছিলাম। এটা মূলত একটা ফাংশন। আমরা বিহাইন্ড দ্য সীন জাভাস্ক্রিপ্ট ফাংশনই কল করছি। কিন্তু যাতে আমাদের লিখতে সুবিধা কয়, পড়তে সুবিধা হয়, বুঝতে সুবিধা হয় সেজন্য আমরা HTML এর মতো করে JSX ব্যবহার করি। নিচে jsx সম্পর্কে পয়েন্ট আকারে বিস্তারিত আলোচনা করা হলো।

- আমরা যদি নিচের কোডটা লিখি -

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

তাহলে তা বিহাইন্ড দ্য সীন দেখতে হবে এরকম -

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

তার মানে React.createElement(component, props, …children) এই ফরম্যাটে যা লিখতাম তা আমরা jsx দিয়ে খুব সুন্দরভাবে লিখতে পারি।

- আমরা html এ আমাদের ইচ্ছামতো সেলফ ক্লোজিং ট্যাগ ব্যবহার করতে পারতাম না। কিন্তু jsx এ আমাদের যদি কোনো children না থাকে তাহলে আমরা সেলফ ক্লোজিং ট্যাগ ব্যবহার করতে পারবো। সেটা আমাদের নিজস্ব কম্পোনেন্টের জন্যও পারবো, আবার html এর ট্যাগের ক্ষেত্রেও পারবো।

```
<div className="sidebar" />
```

- কোনো এলিমেন্টের ক্ষেত্রে রিয়্যাক্ট প্রথমে চেক করে এটা রিয়্যাক্টের এলিমেন্ট কিনা। এরপর সে কাস্টম কম্পোনেন্ট কিনা তা দেখবে। আমরা যদি অন্য কোনো ফাইলে কম্পোনেন্ট বানাই তাহলে অবশ্যই তা আমাদের ওয়ার্কিং ফাইলে ইমপোর্ট করে নিতে হবে। অর্থাৎ স্কোপের মধ্যে থাকতে হবে।

```
import React from 'react';
import CustomButton from './CustomButton';

function WarningButton() {
  // return React.createElement(CustomButton, {color: 'red'}, null);
  return <CustomButton color="red" />;
}
```

- রিয়্যাক্ট কম্পোনেন্টকে আমরা জাভাস্ক্রিপ্টের ফার্স্ট ক্লাস সিটিজেন হিসেবে ধরতে পারি। এটাকে একটা ভ্যারিয়েবলের মধ্যে রাখতে পারি, ফাংশনের মধ্যে রাখতে পারি, অবজেক্টের মধ্যে রাখতে পারি, আবার ফাংশনের আর্গুমেন্ট হিসেবেও দিতে পারি।

```
import React from 'react';

const MyComponents = {
  DatePicker: function DatePicker(props) {
    return <div>Imagine a {props.color} datepicker here.</div>;
  }
}

function BlueDatePicker() {
  return <MyComponents.DatePicker color="blue" />;
}
```

এখানে দেখুন `MyComponents` একটা অবজেক্ট। এর মধ্যে `DatePicker নামে একটা ফাংশন আছে। আমরা ডট নোটেশন ব্যবহার করে jsx এ ব্যবহার করতে পারি উপরের কোডের মতো করে।

- আমরা যে কম্পোনেন্ট বানাবো তার নাম অবশ্যই ক্যাপিটেল লেটার দিয়ে শুরু করতে হবে।
- আমরা চাইলে একটা কম্পোনেন্টের ভিতরে প্রপ্স আকারে আরেকটা কম্পোনেন্ট পাস করতে পারি।
- আমরা রানটাইমে ডায়নামিক্যালি কম্পোনেন্ট সিলেক্ট করতে পারি।
  যেমন আমরা একটা কোর্স প্লেয়ার বানাতে চাই। সেখানে ভিডিও থাকবে, আর্টিকেল থাকবে, অ্যাসাইনমেন্ট থাকবে, কুইজ থাকবে ইত্যাদি। এখন কে কি দেখবে সেটার উপর ডিপেন্ড করে সেই কম্পোনেন্ট রেন্ডার হবে। সেটা আমরা রানটাইমে ডায়নামিক্যালি করতে পারি। যেমন -

```
import React from 'react';
import { PhotoStory, VideoStory } from './stories';

const components = {
  photo: PhotoStory,
  video: VideoStory
};

function Story(props) {
  const SpecificStory = components[props.storyType];
  return <SpecificStory story={props.story} />;
}
```

এখানে আমরা `const SpecificStory = components[props.storyType];` লাইনটা ব্যবহার না করে সরাসরি `return <components[props.storyType] story={props.story} />;` লিখলে সেটা ভুল হতো। কারণ কম্পোনেন্টের নাম কখনও ছোট হাতে অক্ষর দিয়ে শুরু হতে পারবে না। তাই আমরা আগে একটা ক্যাপিটাল লেটারযুক্ত ভ্যারিয়েবলে সেই কম্পোনেন্টকে নিয়ে নিলাম। এরপর সেই ভ্যারিয়েবলের নামটা ব্যবহার করলাম।

- আমরা প্রপ্স আকারে যেকোনো একটা এক্সপ্রেশন jsx এর মধ্যে পাস করতে পারি। সেটা ফাংশন হতে পারে, অ্যারে ম্যাপ হতে পারে, কোনো অপারেশন হতে পারে, ডেইট হতে পারে মোটকথা যেকোনো একটা এক্সপ্রেশন পাস করতে পারি।
  এক্সপ্রেশন এবং স্টেটমেন্টের মধ্যে পার্থক্য ভালভাবে বোঝার জন্য আপনারা Expression vs. Statement আর্টিকেলটি পড়ুন। সংক্ষেপে বলতে গেলে Expression এবং Statement এর মধ্যে
  বেসিক যে পার্থক্য সেটা হলো এক্সপ্রেশন দিন শেষে কিছু না কিছু রিটার্ন করে, ডাটা প্রোডিউস করে, এবং একে কোনো এক জায়গায় স্টোর করে রাখা যায়। সেই হিসেবে ফাংশন কল এক ধরণের এক্সপ্রেশন। আর স্টেটমেন্ট কোনো ডাটা প্রোডিউস করেনা, কোথাও স্টোর করে রাখা যায় না, কিছু রিটার্ন করে না। ফাংশন লেখা হচ্ছে স্টেটমেন্ট, আর ফাংশন কল হচ্ছে এক্সপ্রেশন। কারণ ফাংশন লিখলে তা কিছু রিটার্ন করে না যতক্ষণ পর্যন্ত কল করা না হচ্ছে। আবার যদি অ্যারো ফাংশন

লেখা হয় সেটা এক্সপ্রেশন কারণ সেটাকে একটা ভ্যারিয়েবলে স্টোর করে রাখা হচ্ছে।

- যদি আমরা প্রপ্সের কোনো ভ্যালু না দিই সেটা by default true নিয়ে নিবে। যেমন

```
<MyTextBox autocomplete />

<MyTextBox autocomplete={true} />
```

উপরের দুইটা লাইনই একই। যদি আমরা কোনো ভ্যালু না বসাই তাহলে সেটা অটোমেটিক্যালি true ধরে নিবে।

- আমাদের যদি একাধিক প্রপ্স থাকে সবগুলো প্রপ্স একটা অবজেক্টের মধ্যে রেখে আমরা জাভাস্ক্রিপ্ট স্প্রেড অপারেটর (...) এর মাধ্যমে সব একবারেই পাস করে দিতে পারবো। যেমন -

```
function App() {
  const props = {firstName: 'Ben', lastName: 'Hector'};
  return <Greeting {...props} />;
}
```

- কাস্টম কম্পোনেন্টের মাঝে আমরা প্রপ্স হিসেবে যা দিবো সেটাকে বলে **children props**।

JSX সম্পর্কে আরো জানতে রিয়্যাক্টের অফিসিয়াল সাইটের JSX in depth ভিজিট করতে পারেন।

এবার আমরা আমাদের গত ক্লাসে বানানো অ্যাপের help পেইজে যাবো। এবং আজকের সমস্ত কাজ আমরা এই পেইজে করে করে শিখবো।

Conditional Rendering

প্রথমে আমরা যে বিষয়টা শিখবো সেটা হলো jsx এর মধ্যে কিভাবে কন্ডিশনালি রেন্ডারিং করতে হয় সেটা। প্রথমে আমরা আমাদের Help.jsx এর মধ্যে একটা স্টেট নিয়ে নিই।

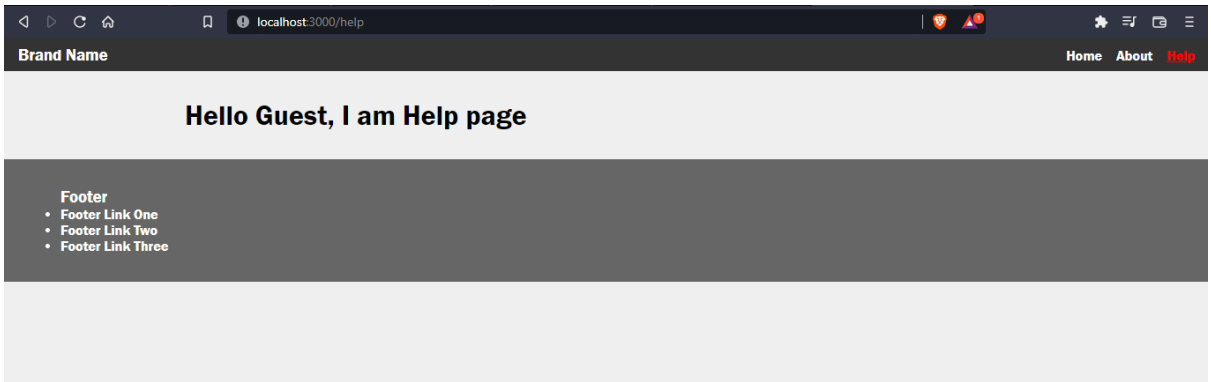
```
import { useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [name, setName] = useState('');

  return (
    <Layout>
      <h1>Hello, I am Help page</h1>
    </Layout>
```

```
  );
};

export default Help;
```

এবার আমরা কন্ডিশনালি রেন্ডার করবো। আমরা রেন্ডার করবো যদি নাম থাকে তাহলে `h1` ট্যাগে Hello এর পর সেই নামটা আসবে। আর নাম না থাকলে আসবে `Guest`। এখন আমরা আমাদের কন্ডিশনটা লিখি।

```
import { useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [name, setName] = useState('');

  return (
    <Layout>
      <h1>Hello {name ? name : 'Guest'}, I am Help page</h1>
    </Layout>
  );
};

export default Help;
```

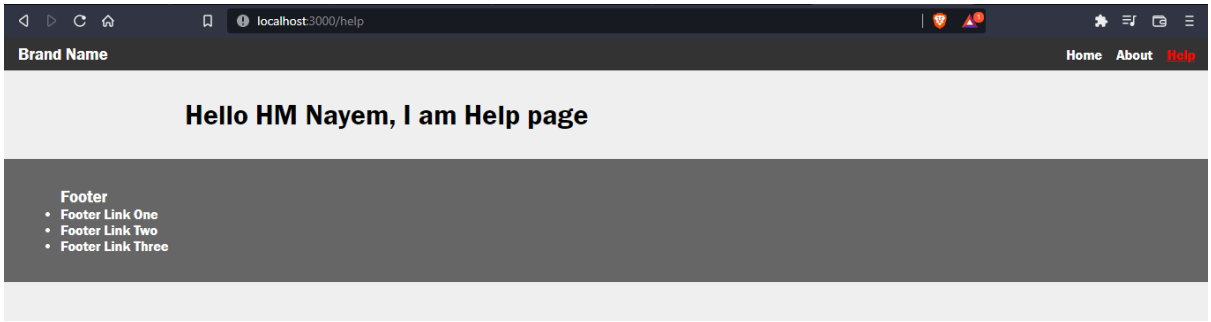
এখন যেহেতু কোনো নাম নেই তাই আমাদের আউটপুট আসবে Guest দিয়ে।



এখন যদি আমরা স্টেটের ইনিশিয়াল ভ্যালু `''` এর পরিবর্তে `'HM Nayem'` দিই তাহলে Guest এর জায়গায় সেই নামটা আসবে।

```
import { useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [name, setName] = useState('HM Nayem');

  return (
```

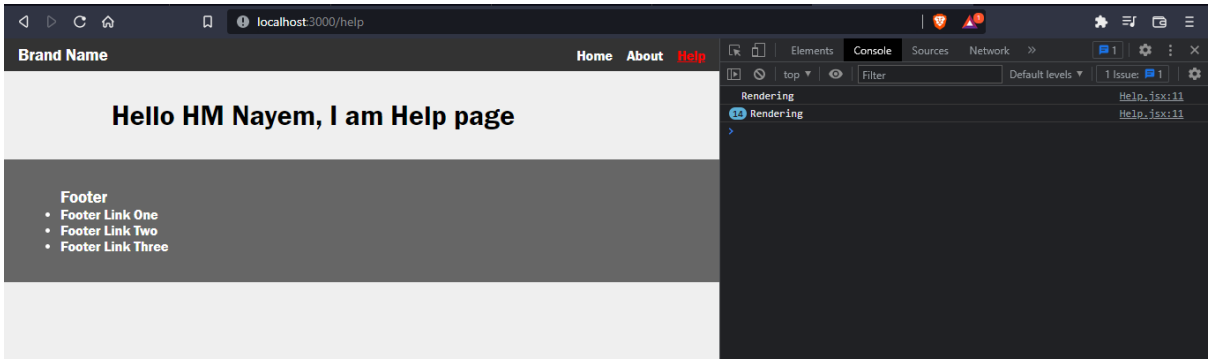
```
    <Layout>
      <h1>Hello {name ? name : 'Guest'}, I am Help page</h1>
    </Layout>
  );
};

export default Help;
```



এবার আমরা এমন একটা সিস্টেম বানাবো যেখানে প্রথমে কোনো নাম থাকবে না। কিছু সময় পর একটা নাম নিবে।

```
import { useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [state, setState] = useState({});

  setTimeout(() => {
    setState({ name: 'HM Nayem' });
  }, 1 * 1000);

  console.log('Rendering');

  return (
    <Layout>
      <h1>Hello {state.name ? state.name : 'Guest'}, I am Help page</h1>
    </Layout>
  );
};

export default Help;
```

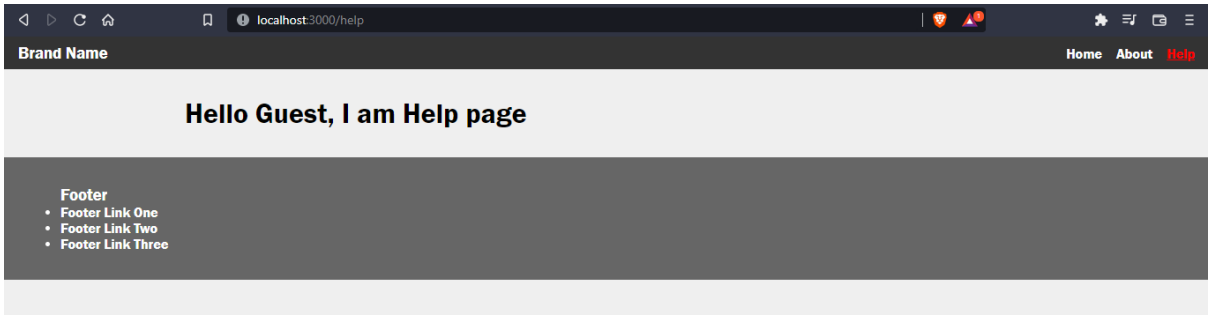
এবার যদি আমরা আমাদের ব্রাউজারে গিয়ে কনসোলে
দেখি দেখবো ১ সেকেন্ড পরপর এটা রেন্ডার হতেই থাকছে এবং Rendering লেখাটা বারবার প্রিন্ট হতে থাকছে।



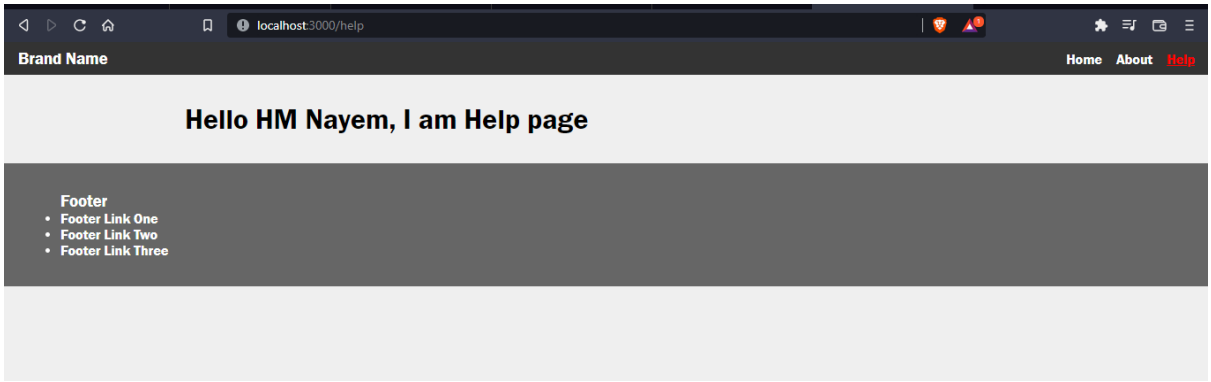
আমরা জাভাস্ক্রিপ্টের টারনারী অপারেশনের মাধ্যমে কন্ডিশনাল রেন্ডারিং দেখেছিলাম। এবার আমরা একটু শর্টসার্কিট ইভ্যাল্যুয়েশন এর মাধ্যমে সেটা দেখি।

```
import { useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [name, setName] = useState('');

  return (
    <Layout>
      {name && <h1>Hello {name}, I am Help page</h1>}
      {!name && <h1>Hello Guest, I am Help page</h1>}
    </Layout>
  );
};

export default Help;
```

আমরা চাইছি যদি নাম না থাকে তাহলে `<h1>Hello Guest, I am Help page</h1>` শো করবে, আর নাম থাকলে `<h1>Hello {name}, I am Help page</h1>` শো করবে। যেহেতু এখন এখানে কোনো নাম দেয়া নেই তাহলে শো করবে নিচের ছবি।



আর যদি নাম থাকে তাহলে শো করবে নিচের মতো।



jsx এর মধ্যে এরকম চেহারা আমরা প্রতিনিয়ত দেখতে পারবো।

এবার আমরা টারনারী অপারেটর ব্যবহার করে দেখি। আমরা শুরুতে টারনারী অপারেটর ব্যবহার করেছিলাম শুধু একটা অংশে। এবার আমরা পুরো `h1` ট্যাগের জন্য ব্যবহার করবো।

```
import { useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [name, setName] = useState('');

  return (
    <Layout>
      {name ? (
        <h1>Hello {name}, I am Help page</h1>
      ) : (
        <h1>Hello Guest, I am Help page</h1>
      )}

    </Layout>
  );
};

export default Help;
```

আপনারা আউটপুট দেখলে দেখবেন আগের মতোই আউটপুট দিচ্ছে।

আমরা এই কন্ডিশনাল রেন্ডারিং jsx এর যেকোনো জায়গাতেই করতে পারি। যেমন - ক্লাসনেইম সিলেক্ট করার ক্ষেত্রে করতে পারি, অ্যাট্রিবিউট সিলেক্ট করার ব্যাপারে করতে পারি, স্টাইলসের ক্ষেত্রে ব্যবহার করতে পারি, ভ্যালু প্রোভাইড করার ক্ষেত্রে করতে পারি, কোন কম্পোনেন্ট রেন্ডার হবে সেই ডিসিশন নেয়ার ক্ষেত্রে করতে পারি ইত্যাদি। ডায়নামিক ডাটা রেন্ডার করার জন্য এই কন্ডিশনাল রেন্ডারিং রিয়্যাক্টের অন্যতম একটা শক্তি।

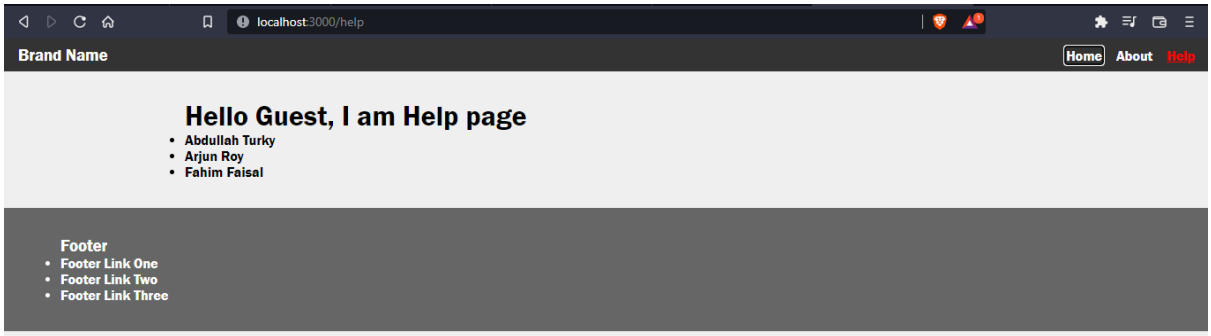
ধরি আমাদের কাছে তিনজন ইউজারের ডাটা আছে।

```
const data = [
  {
    name: 'Abdullah Turky',
    email: 'turky@test.com',
  },
  {
    name: 'Arjun Roy',
    email: 'arjun@test.com',
  },
  {
    name: 'Fahim Faisal',
    email: 'fahim@test.com',
  },
];
```

আমরা চাইছি এই তিনজনের নাম লিস্ট আকারে শো করাতে।

```
<ul>
  <li>{data[0].name}</li>
  <li>{data[1].name}</li>
  <li>{data[2].name}</li>
</ul>;
```

দেখবো আমাদের তিনজন ইউজারের নাম শো করছে।

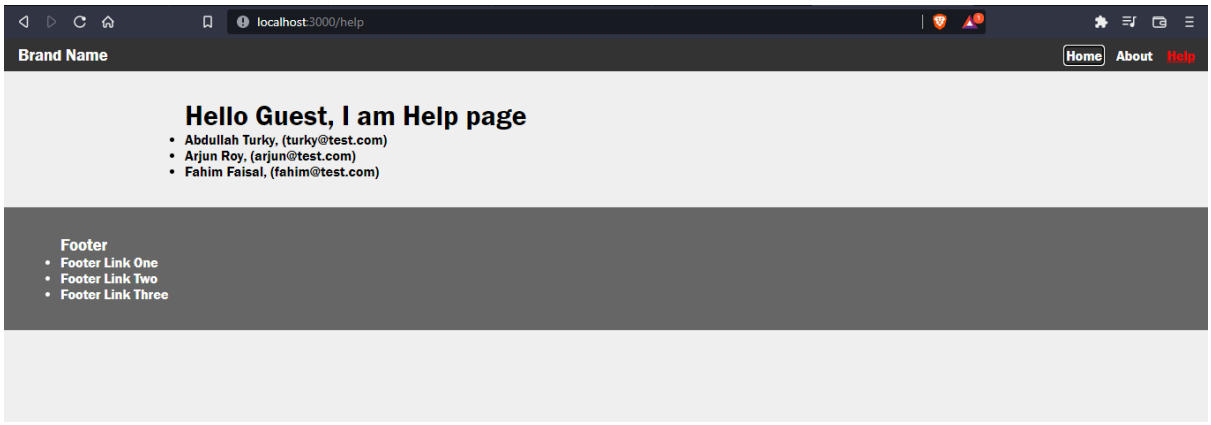


এখানে যেহেতু আমরা একই কাজ বারবার করছি তাহলে এতবার `li` ট্যাগ না নিয়ে আমরা জাভাস্ক্রিপ্টের অ্যারে ম্যাপ মেথড ব্যবহার করতে পারি।

```
<ul>
  {
  data.map((item) => (
    <li>
      {item.name}, ({item.email})
    </li>
  ));
```

```
}
</ul>;
```

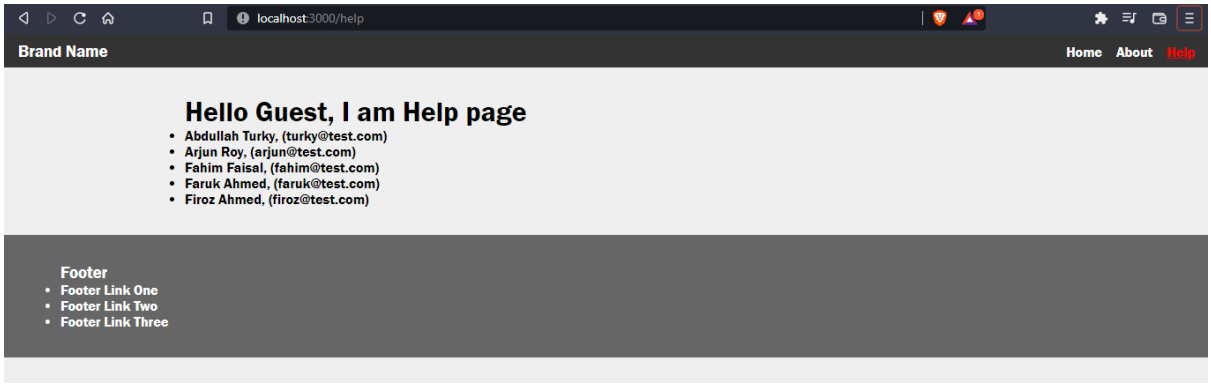
আমাদের আউটপুট আসবে এরকম-



এভাবে ডায়নামিক্যালি করলে যদি আমাদের ডাটা বৃদ্ধিও পায়
আমাদের কিছু করতে হবে না। তা ডায়নামিকভাবে রেন্ডার হয়ে যাবে। যেমন আমরা যদি আরো দুইটা ডাটা অ্যাড করি তাহলে দেখবো তা অটোমেটিক্যালি আমাদের UI এ চলে এসেছে।

```
const data = [
  {
    name: 'Abdullah Turky',
    email: 'turky@test.com',
  },
  {
    name: 'Arjun Roy',
    email: 'arjun@test.com',
  },
  {
    name: 'Fahim Faisal',
    email: 'fahim@test.com',
  },
  {
    name: 'Faruk Ahmed',
    email: 'faruk@test.com',
  },
  {
    name: 'Firoz Ahmed',
    email: 'firoz@test.com',
  },
];
```

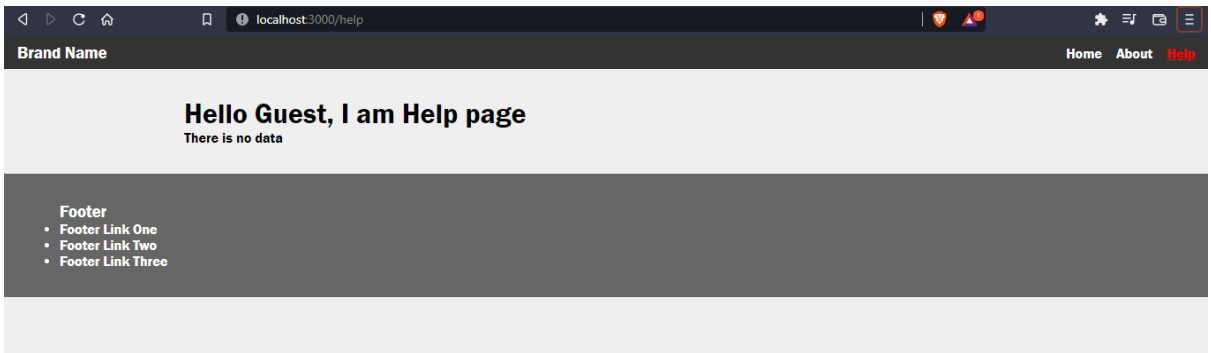
এবার যদি আমরা ব্রাউজারে যাই দেখবো এই ডাটাগুলো সেখানে চলে এসেছে।



এখন যদি ডাটা না থাকে তাহলে কি হবে? ডাটা না থাকলে কিছুই শো করবে না। কিন্তু তাহলে তো কিছুই বুঝা যাবে না। কারণ ডাটা না থাকলে অন্তত আমাদের এটা ইনফর্ম করতে হবে যে এখানে আমাদের এখন কোনো ডাটা নেই। তার মানে আমাদের যে `ul` ট্যাগ আছে সেটাকে কন্ডিশনালি রেন্ডার করতে পারি।
অর্থাৎ যদি ডাটা থাকে তাহলে সে ইউজারের লিস্ট শো করবে আর না থাকলে আমাদেরকে একটা ম্যাসেজ দিবে যে এখানে কোনো ডাটা নাই।

```
{
  data.length > 0 ? (
    <ul>
      {data.map((item) => (
        <li>
          {item.name}, ({item.email})
        </li>
      ))}
    </ul>
  ) : (
    <p>There is no data</p>
  );
}
```

যদি ডাটা না থাকে তাহলে নিচের মতো শো করবে।



আর যদি ডাটা থাকে তাহলে আগের মতোই ডাটা শো করবে।

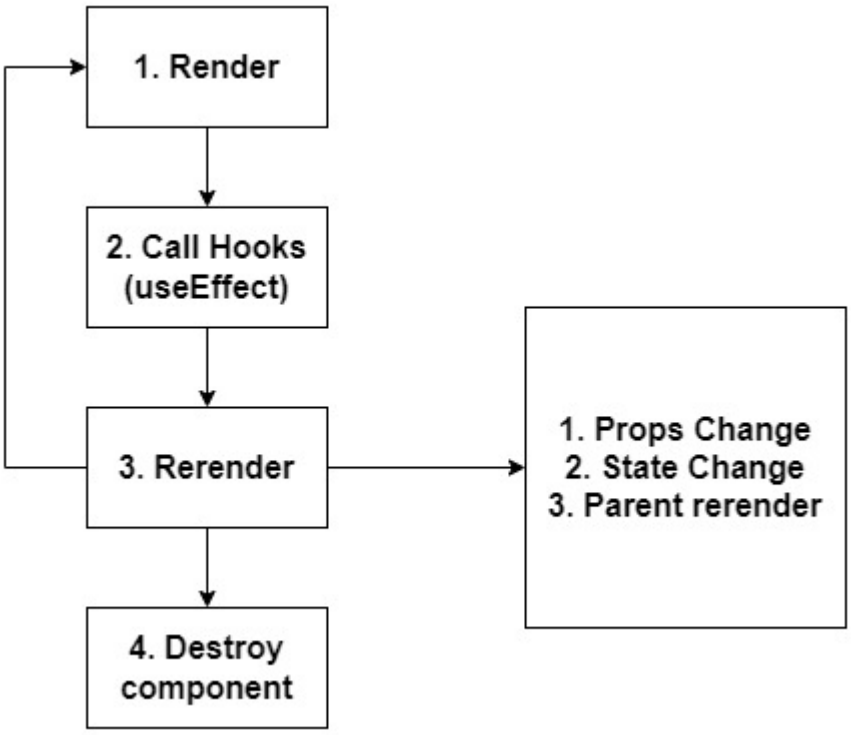
এটুকু কনসেপ্ট যদি বুঝে থাকি তাহলে আমরা প্রজেক্ট করার জন্য রেডি। আর কিছু কনসেপ্ট আছে যেমন ইভেন্ট হ্যান্ডেল করা। সেটা খুব কঠিন কিছু না। আমরা যারা ডম পারি তারা ইভেন্ট হ্যান্ডলিং নিয়ে অলরেডি কাজ করেছি।

State and its Lifecycle

আমরা স্টেট নিয়ে গত ক্লাসে আলোচনা করেছিলাম। এবার আমরা জানবো এর লাইফসাইকেল সম্পর্কে। ক্লাস বেইজ কম্পোনেন্ট এবং ফাংশনাল কম্পোনেন্টের লাইফসাইকেলের মধ্যে ভিন্নতা আছে একটু। ক্লাস বেইজড কম্পোনেন্টের লাইফসাইকেল নিয়ে রিয়্যাক্টের অফিসিয়াল ওয়েবসাইট এ বলে দেয়া আছে। যেহেতু আমরা বেশিরভাগ ক্ষেত্রে ফাংশনাল কম্পোনেন্ট ব্যবহার করবো, সেহেতু আমরা ফাংশনাল কম্পোনেন্টের মধ্যে স্টেটের লাইফসাইকেল কেমন হয় সে বিষয় আমরা স্টেপ বাই স্টেপ আলোচনা করবো।

- প্রথম স্টেপ হলো আমরা একটা কম্পোনেন্টকে রেন্ডার করবো। রেন্ডার করার পর ঐ কম্পোনেন্টের যা যা কোড আছে সব এক্সিকিউট হয়ে এর মধ্যে থাকা jsx কে স্ক্রিনে প্রিন্ট করার চেষ্টা করবে।
- দ্বিতীয় স্টেপ হচ্ছে রেন্ডার করার পর কোনো পরিবর্তন হচ্ছে কিনা সেটা দেখে এর Hooks কল করা। ফাংশনাল কম্পোনেন্টের সুবিধা হলো এখানে একটা মাত্র হুক থাকে, সেটা হলো `useEffect`।
- তৃতীয় আরেকটা স্টেপ হলো রিরেন্ডার হওয়া। রিরেন্ডার হওয়ার তিনটা কারণ আমরা গত ক্লাসে আলোচনা করেছিলা, সেগুলো হলো - প্রপ্স চেইঞ্জ হলে, স্টেট চেইঞ্জ হলে এবং প্যারেন্ট কম্পোনেন্ট রিরেন্ডার হলে। রিরেন্ডার হলে আবার প্রথম স্টেপে গিয়ে পুরো প্রসেসের পুনরাবৃত্তি হবে। তার মানে এখানে যদি একটা সিস্টেম করা যায় যে বারবার স্টেট আপডেট হচ্ছে তাহলে পুরো প্রসেস সারাজীবন ধরে চলতে থাকবে। যেটা আমরা কিছুক্ষণ আগে দেখেছিলাম `setTimeout` এর উদাহরণে।
- শেষ আরেকটা স্টেপ আছে, সেটা হলো কম্পোনেন্ট destroy হয়ে যাবে।

নিচের ডায়াগ্রামটা দেখলে ক্লিয়ার হবেন আপনারা।



এটাই হচ্ছে একটা ফাংশনাল কম্পোনেন্টের লাইফসাইকেল।

useEffect Hook:

এই লাইফসাইকেলটা মেইনটেইন করার জন্য `useEffect` hook খুবই গুরুত্বপূর্ণ। ক্লাস বেইজড কম্পোনেন্টের ডায়াগ্রাম

যদি আমরা দেখি, দেখবো সেখানে componentDidMount, componentWillUnmount, componentDidUpdate, shouldComponentUpdate এই চারটা হুক আমাদের আলাদা আলাদাভাবে মেইনটেইন করতে হতো। এই চারটার কাজ ফাংশনাল কম্পোনেন্টে একা useEffect হুক করে থাকে। তাহলে আমরা বলতে পারি, রিয়্যাক্টে সবচেয়ে পাওয়ারফুল টুল বা হুকই হচ্ছে এই useEffect hook. আমাদের বর্তমানে স্টেটের যে জ্ঞানটা রয়েছে তার সাথে যদি এই হুকের জ্ঞানটাও যুক্ত করতে পারি, তাহলে প্রায় যেকোনো প্রজেক্ট তৈরি করার সক্ষমতা অর্জন করে ফেলতে পারবো।

useEffect hook এর কাজ

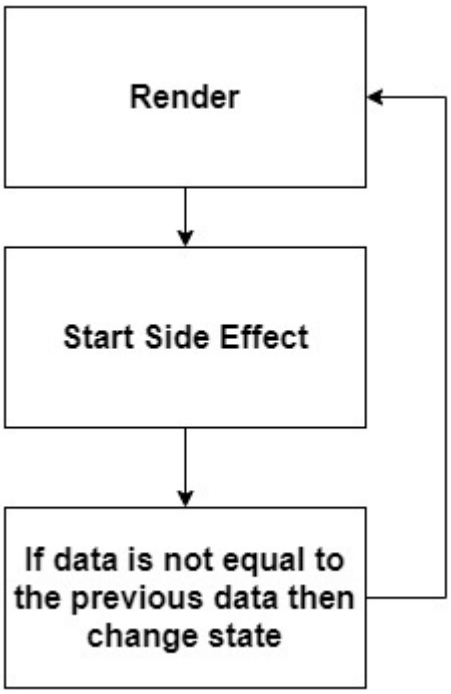
আমরা একদম শুরুতে যে কোড লিখেছিলাম সেটা একটু দেখি।

```
import { useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [state, setState] = useState({});

  setTimeout(() => {
```

```
    setState({ name: 'HM Nayem' });
  }, 1 * 1000);

  console.log('Rendering');

  return (
    <Layout>
      <h1>Hello {state.name ? state.name : 'Guest'}, I am Help page</h1>
    </Layout>
  );
};

export default Help;
```

এখানে setTimeout কম্পোনেন্টের মধ্যে বারবার
স্টেট আপডেট করছে। কিন্তু setTimeout একবার কাজ করবে, আর setInterval বারবার কাজ করবে। তাহলে তো এখানে একবার রেন্ডারিং কাজ শেষ হয়ে যাওয়ার কথা। কিন্তু সেটা হলো না। কারণ এটা স্টেটকে ট্রিগার করছে। যার ফলে এটা বারবার আপডেট হচ্ছে। আর আপডেট হলে সেটা বারবার Help ফাংশনকে কল করে সব কাজ বারবার করছে। এখানে আমরা যে ভুলটা করেছিলাম সেটা হলো আমরা ফাংশনের বডির মধ্যে এই setTimeout লিখেছিলাম। যার ফলে এটা একটা ইনফিনিটি লুপে পরিণত হয়েছে। এটা খুওই বাজে একটা সল্যুশন। কারণ ধরেন ১০০ জন ইউজার একটা রিকোয়েস্ট পাঠালে সেটা বারবার স্টেট আপডেট করবে। আর অ্যাপ্লিকেশন যদি সার্ভারের সাথে কানেক্টেড হয় তাহলে সার্ভারসহ ক্রাশ হয়ে যাবে। এই সমস্যার সল্যুশন হলো useEffect হুক।

আমাদের এই কোডের জন্য লজিকটা হওয়া উচিৎ ছিল নিচের ডায়াগ্রামের মতো।



আমরা মাঝখানের সাইড এফেক্ট অংশটা বাদ দিয়ে দিয়েছিলাম। যার কারণে এটা একটা ইনফিনিটি ফ্লোতে পরিণত হয়েছিল। এই ধরণের ফ্লো ব্রেক করার কাজই করে থাকে useEffect। কিভাবে আমরা useEffect নিয়ে কাজ করতে পারি একটু দেখা যাক।

```
import { useEffect, useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [name, setName] = useState('');
  // const [state, setState] = useState({});

  useEffect(() => {
    console.log('Use Effect Called');
  });

  // setTimeout(() => {
  //  setState({ name: 'HM Nayem' });
  // }, 1 * 1000);

  // console.log('Rendering');

  const data = [
    {
      name: 'Abdullah Turky',
      email: 'turky@test.com',
    },
    {
      name: 'Arjun Roy',
      email: 'arjun@test.com',
    },
    {
      name: 'Fahim Faisal',
      email: 'fahim@test.com',
    },
    {
      name: 'Faruk Ahmed',
      email: 'faruk@test.com',
    },
    {
      name: 'Firoz Ahmed',
      email: 'firoz@test.com',
    },
  ];

  // const data = [];

  return (
    <Layout>
      {/* {name && <h1>Hello {name}, I am Help page</h1>}
      {!name && <h1>Hello Guest, I am Help page</h1>} */}

      {name ? (
        <h1>Hello {name}, I am Help page</h1>
      ) : (
```

```
        <h1>Hello Guest, I am Help page</h1>
      )}

      {data.length > 0 ? (
        <ul>
          {data.map((item) => (
            <li>
              {item.name}, ({item.email})
            </li>
          ))}
        </ul>
      ) : (
        <p>There is no data</p>
      )}
    </Layout>
  );
};

export default Help;
```

আমরা যদি ব্রাউজারে গিয়ে কনসোলে দেখি দেখবো একবার কল হয়েছে এটা। useEffect কয়বার কল হবে সেটা নির্ভর করবে কি উদ্দেশ্যে আমরা সেটা ব্যবহার করছি এবং কিভাবে সেটা ব্যবহার করছি তার উপর। এখন যদি উপরের কোডের কমেন্ট করা অংশগুলো আমরা আনকমেন্ট করে দিই দেখবো Rendering এবং Use Effect called বারবার প্রিন্ট হয়ে যাচ্ছে। এখন Rendering এর ব্যপারটা নাহয় বুঝলাম সেখানে setTimeout এর কারণে হচ্ছে আগের মতো। কিন্তু Use Effect called কেন বারবার প্রিন্ট হচ্ছে? আমরা লাইফসাইকেলের ডায়াগ্রামে দেখেছিলাম যতবার রেন্ডার হবে ততবার এই হুক কল হবে। তাই এখানে বারবার কল হচ্ছে। এটাকে আমরা আটকে দিতে পারি। সেটা হচ্ছে আমরা যদি এটাকে স্বাধীন করে দিই অর্থাৎ যদি বলে দিই এর কোনো ডিপেন্ডেন্সি নেই তাহলে সেটা স্বাধীন হয়ে যাবে। আর সেটা আমরা করতে পারি সেকেন্ড আর্গুমেন্ট হিসেবে একটা ফাঁকা অ্যারে পাস করার মাধ্যমে।

```
useEffect(() => {
  console.log('Use Effect Called');
}, []);
```

এখন যদি আমরা দেখি দেখবো Rendering বারবার কল হচ্ছে কিন্তু Use Effect called একবার কল হয়ে থেমে গেছে। এবার যদি আমরা ডিপেন্ডেন্সি আকারে state দিয়ে দিই তাহলে এটা বুঝাবে আমাদের হুক এই state এর উপর ডিপেন্ডেন্ট। অর্থাৎ এটা পরিবর্তন হলে এই হুক কল হতে থাকবে।

```
const [state, setState] = useState({});
useEffect(() => {
```

```
  console.log('Use Effect Called');
}, [state]);
```

এবার যদি ব্রাউজারে গিয়ে দেখি দেখবো আগের
মতোই Rendering এবং Use Effect called বারবার প্রিন্ট হচ্ছে। কারণ
ডিপেন্ডেন্সি দেয়ার অর্থই হচ্ছে ঐ স্টেটের পরিবর্তনের উপর আমাদের হুক কল
হবে। তাহলে সেই স্টেট যতবার পরিবর্তন হবে ততবারই হুক কল হবে। এবার আমাদের
setTimeout ফাংশনকে আমরা আমাদের হুকের মধ্যে লিখি এভাবে।

```
const Help = () => {
  const [name, setName] = useState(‘’);
  const [state, setState] = useState({name: ‘’});

  useEffect(() => {
    setTimeout(() => {
      setState({ name: 'HM Nayem' });
    }, 1 * 1000);
    console.log('Set timeout');
  }, [state]);

  console.log('Rendering');
}
```

এবার যদি ব্রাউজারে যাই দেখবো Set timeout
এবং Rendering বারবার কল হচ্ছে। তার মানে আমরা এখনও কোনো সুবিধা পেলাম না।
এখানে যদি আমরা ডিপেন্ডেন্সি আকারে ফাঁকা অ্যারে দিতাম তাহলে পুরো জিনিসটাই
চেইঞ্জ হয়ে যেতো। সেক্ষেত্রে Set timeout এবং Rendering একবারই কল হতো
শুধু। বারবার না। এবার যদি আমরা আমাদের কোডকে একটু নিচের মতো লিখি অর্থাৎ
jsx এ name এর জায়গায় state.name ব্যবহার করি এবং setTimeout এর সময় বাড়িয়ে
৩ সেকেন্ড করি দেখবো ৩ সেকেন্ড পর আমাদের কম্পোনেন্ট রেন্ডার হচ্ছে,
কিন্তু রিরেন্ডারিং আর হচ্ছে না।

```
import { useEffect, useState } from 'react';
import Layout from '../components/layout/Layout';

const Help = () => {
  const [state, setState] = useState({name: ''});

  useEffect(() => {
    setTimeout(() => {
      setState({ name: 'HM Nayem' });
    }, 3000);
    console.log('Set timeout');
  }, []);

  console.log('Rendering');
```

```
  return (
    <Layout>
      {/* {name && <h1>Hello {name}, I am Help page</h1>}
      {!name && <h1>Hello Guest, I am Help page</h1>} */}

      {state.name ? (
        <h1>Hello {state.name}, I am Help page</h1>
      ) : (
        <h1>Hello Guest, I am Help page</h1>
      )}
    </Layout>
  );
};

export default Help;
```

এটাই হচ্ছে useEffect এর পাওয়ার। useEffect এ চাইলে আমরা একটা ফাংশন রিটার্ন করতে পারি সেটা সম্পর্কে আমরা পরবর্তীতে জানবো।

Clock Application

এবার আমরা একটা অ্যাপ্লিকেশন বানাই। আমরা আমাদের pages ফোল্ডারে ClockPage.jsx নামে একটা ফাইল ক্রিয়েট করবো।

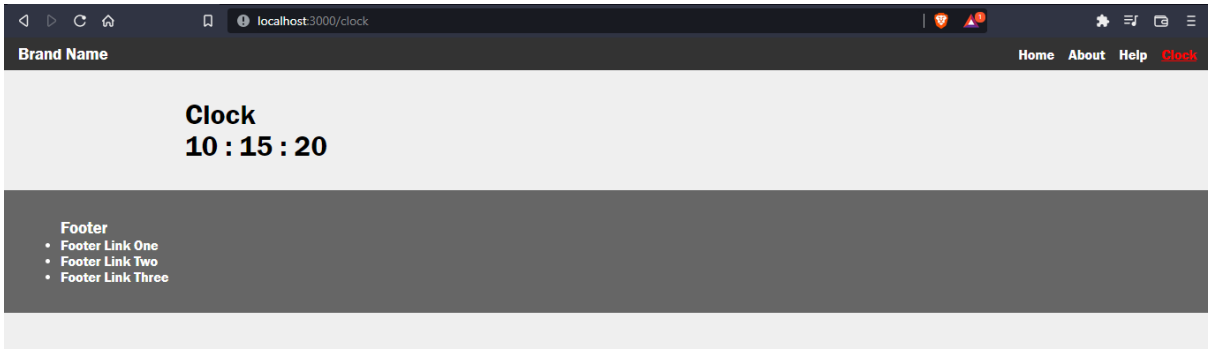
```
import Layout from '../components/layout/Layout';

const ClockPage = () => {
  return (
    <Layout>
      <h1>Clock</h1>
    </Layout>
  );
};

export default ClockPage;
```

এবার এই পেইজকে আমাদের App.jsx এবং Layout.jsx এ গিয়ে লিংক করতে হবে। এরপর আমাদের ClockPage এ jsx লিখে ফেলি।

```
import Layout from '../components/layout/Layout';

const ClockPage = () => {
  return (
    <Layout>
      <h1>Clock</h1>
      <h1>
        {10} : {15} : {20}
      </h1>
    </Layout>
```

```
  );
};

export default ClockPage;
```

তাহলে আমাদের পেইজের চেহারা দাঁড়াবে এরকম -



এবার আমাদের প্রতি সেকেন্ড পরপর স্টেট চেইঞ্জ করতে হবে। এই কাজটা করার জন্য আমরা date-fns প্যাকেজটি ইনস্টল করে নিবো।

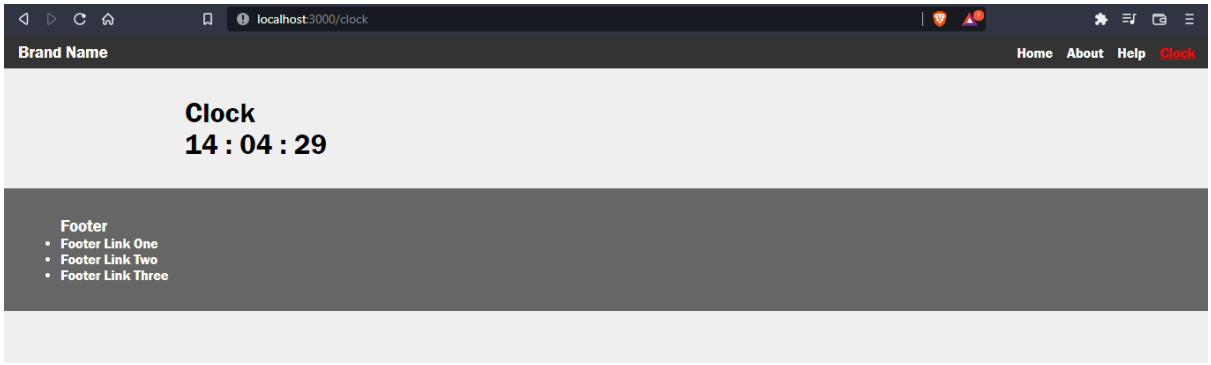
এবার যদি আমরা নিচের কোড লিখি তাহলে দেখবো আমাদের ব্রাউজার কনসোলে প্রতি সেকেন্ড পরপর টাইম আপডেট হচ্ছে।

```
import { useEffect, useState } from 'react';
import Layout from '../components/layout/Layout';

const ClockPage = () => {
  const [date, setDate] = useState(new Date());

  useEffect(() => {
    setTimeout(() => {
      setDate(new Date());
    }, 1000);
  }, [date]);

  console.log(date);

  return (
    <Layout>
      <h1>Clock</h1>
      <h1>
        {10} : {15} : {20}
      </h1>
    </Layout>
  );
};

export default ClockPage;
```

এবার আমরা আমাদের date থেকে ডাটা পাওয়ার
জন্য কম্পোনেন্ট ফাংশনের বাইরে একটা ফাংশন বানাবো। টাইম ফরমেটের জন্য একটা ফাংশন বানাবো এবং সেগুলো আমরা আমাদের jsx এ ব্যবহার করবো।

```
import * as DateFns from 'date-fns';
import { useEffect, useState } from 'react';
import Layout from '../components/layout/Layout';

const getTimes = (date) => {
  return {
    hours: DateFns.getHours(date),
    minutes: DateFns.getMinutes(date),
    seconds: DateFns.getSeconds(date),
  };
};

const formatTime = (time) => {
  return time < 10 ? `0${time}` : time;
};

const ClockPage = () => {
  const [date, setDate] = useState(new Date());

  useEffect(() => {
    setTimeout(() => {
      setDate(new Date());
    }, 1000);
  }, [date]);

  const time = getTimes(date);

  return (
    <Layout>
      <h1>Clock</h1>
      <h1>
        {formatTime(time.hours)} : {formatTime(time.minutes)} :{' '}
        {formatTime(time.seconds)}
      </h1>
    </Layout>
  );
};

export default ClockPage;
```

আমাদের ক্লক রেডি।

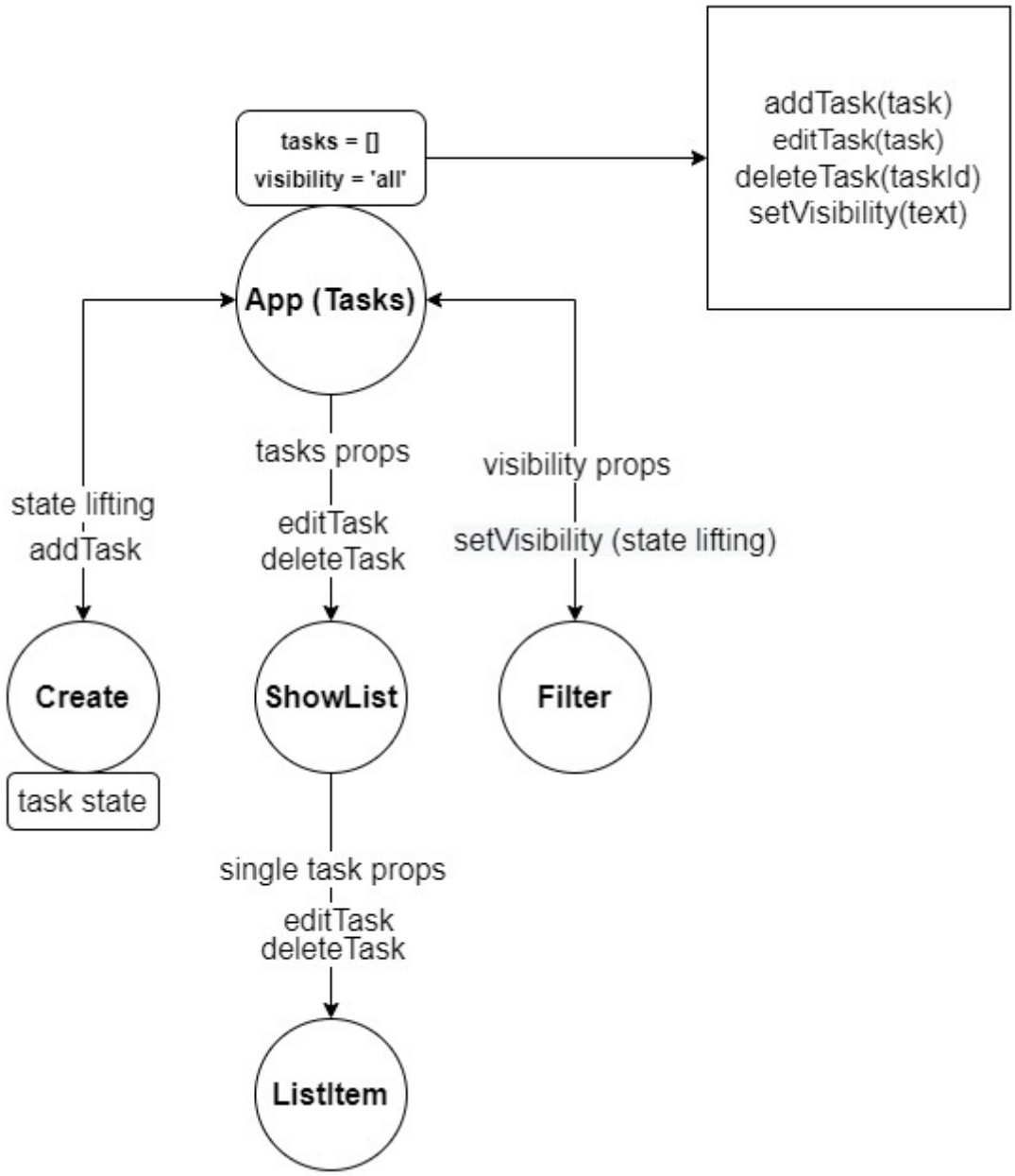


Tasks Application

আমরা আরেকটা ছোট প্রজেক্ট করবো। আমরা প্রথমে pages ফোল্ডারে Tasks.jsx নামে একটা ফাইল ক্রিয়েট করে App.jsx এবং Layout.jsx এ লিংক করে দিবো। এই অ্যাপ্লিকেশনের কাজগুলো নিচে দেয়া হলো -

- Create new task
- Display all tasks
- Filter tasks by complete, incomplete and all
- Delete tasks
- Edit tasks

এবার আমাদের অ্যাপের আর্কিটেকচারটা যদি একটু ডিজাইন করি তাহলে আমাদের জন্য কাজ করা সহজ হয়ে যাবে। চলুন একটু চেষ্টা করি।



আমরা স্টেপ বাই স্টেপ বর্ণনা করি এটা।

- আমাদের সমস্ত কিছু থাকবে আমাদের App বা Tasks এর কাছে। সেখানে দুইটা স্টেট থাকবে tasks এবং visibility। tasks ইনিশিয়ালি একটা ফাঁকা অ্যারে হিসেবে থাকবে এবং visibility বলতে আমরা বুঝাচ্ছি কি হিসেবে ফিল্টারিং হবে - complete, incomplete or all। ইনিশিয়ালি আমরা all ধরে নিই।
- এই অ্যাপের **তিনটা কম্পোনেন্ট** থাকবে। Create, Filter এবং ShowList।
- অ্যাপ থেকে সকল টাস্ক প্রপ্স আকারে ShowList এর কাছে আসবে। ShowList একটা করে সিঙ্গেল টাস্ক প্রপ্স আকারে ListItem কম্পোনেন্টের মধ্যে পাস করবে।

- Filter এর মধ্যে অ্যাপ থেকে visibility আসবে প্রপ্স আকারে।
- Create এর মধ্যেই থাকবে টাস্ক স্টেট। কারণ যেখানে ক্রিয়েট হচ্ছে সেখানেই তো টাস্ক স্টেট থাকবে। এখন এই Create কম্পোনেন্ট থেকে অ্যাপের কাছে টাস্ক স্টেট পাঠানো যাবে state lifting এর মাধ্যমে। কিন্তু আমরা জানি রিয়্যাক্টে সবসময় কম্পোনেন্ট ট্রী এর উপর থেকে নিচে ডাটা পাস হয়। নিচ থেকে উপরে পাঠানো যায় না। নিচ থেকে উপরেও পাঠানো যায় এবং সেটা state lifting এর মাধ্যমে।

এবার আমাদের অ্যাপের কি কি ফাংশন বানাতে হবে দেখি।

- addTask(task) - এটা ক্রিয়েট কম্পোনেন্টের মধ্যে যখনই Create Task বাটনে প্রেস করবো তখন এই ফাংশনটা কল হবে। মজার ব্যাপার হলো এই ফাংশন এক্সিকিউট হবে আমাদের অ্যাপে, কিন্তু ইনভোক হবে Create কম্পোনেন্ট থেকে। Create কম্পোনেন্ট অ্যাপের স্টেটে কি হচ্ছে না হচ্ছে কিছুই জানবে না। সে শুধু পাঠাবে addTask ফাংশনটা।
- editTask(task) - টাস্ক এডিট করার জন্য। এটা থাকবে ShowList এর কাছে এবং সে সেটা ফরওয়ার্ড করে দিবে ListItem এর কাছে।
- deleteTask(taskId) - টাস্ক ডিলিট করার জন্য। এটাও থাকবে ShowList এর কাছে এবং ফরওয়ার্ড করে দিবে ListItem এর কাছে।
- setVisibility(text) - ফিল্টারিং এর জন্য। এটা থাকবে ফিল্টারের কাছে। এখান থেকে ইনভোক হবে এবং অ্যাপে এক্সিকিউট হবে।

আমাদের আর্কিটেকচার শেষ। এবার আমরা এটা তৈরি করার চেষ্টা করি।

```
// Tasks.jsx

import { useState } from 'react';
import Layout from '../components/layout/Layout';
import CreateTask from '../components/tasks/CreateTask';

const Tasks = () => {
  const [tasks, setTasks] = useState([]);
  const [visibility, setVisibility] = useState('all');

  return (
    <Layout>
      <h1>TODO List</h1>
      <CreateTask />
    </Layout>
  );
};

export default Tasks;
```

```
// components/tasks/CreateTsks.jsx

import { useState } from 'react';

const CreateTask = () => {
  const [text, setText] = useState('sample task');

  return (
    <div>
      <input
        type="text"
        placeholder="type your task"
        value={text}
        onChange={(event) => setText(event.target.value)}
      />
      <button
        onClick={() => {
          alert(text);
        }}
      >
        Create Task
      </button>
    </div>
  );
};

export default CreateTask;
```

যেখানেই কোনো ইনপুট থাকবে সেখানেই আমরা চাইবো
ডাটার কন্ট্রোল আমাদের হাতে রাখতে। এখানেও একই। তাই আমরা আমাদের ইনপুটের কন্ট্রোল রাখবো আমাদের হাতে। এখানে আমরা স্টেট এর ডাটাকে ভ্যালু হিসেবে ইনপুট ফিল্ডে পাস করেছি, সেটাকে বলে সিঙ্গেল ওয়ে বাইন্ডিং। কারণ স্টেট থেকে ডাটা ইনপুট ফিল্ডে আসছে। কিন্তু ইনপুট থেকে ডাটা আপডেট হচ্ছে না। তাই এটাকে বলে সিঙ্গেল ওয়ে বাইন্ডিং। কিন্তু আমাদেরকে টু ওয়ে বাইন্ডিং করতে হবে। তার জন্য onChange দিয়ে আমাদের ইভেন্ট হ্যান্ডলিং করতে হবে। এখন আমাদের বাটনে ক্লিক করলে Task অ্যাড হবে আমাদের অ্যাপে অর্থাৎ Tasks.jsx এ। সেটার জন্য আমাদের একটা ফাংশন বানাতে হবে Tasks.jsx এ।

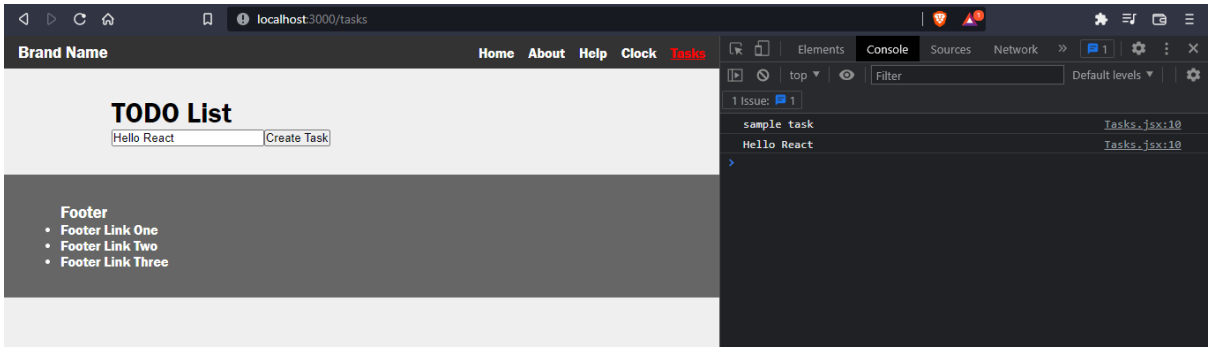
```
const addNewTask = (text) => {
console.log(text)
}
```

এখন এই ফাংশনকে আমরা আমাদের কম্পোনেন্ট থেকে কিভাবে কল করবো? অবশ্যই প্রপ্স আকারে পাস করার মাধ্যমে।

```
// Tasks.jsx

<Layout>
  <h1>TODO List</h1>
  <CreateTask addNewTask={addNewTask} />
</Layout>;
```

```
// CreateTask.jsx

import { useState } from 'react';

const CreateTask = ({ addNewTask }) => {
  const [text, setText] = useState('sample task');

  return (
    <div>
      <input
        type="text"
        placeholder="type your task"
        value={text}
        onChange={(event) => setText(event.target.value)}
      />
      <button
        onClick={() => {
          addNewTask(text);
          setText('');
        }}
      >
        Create Task
      </button>
    </div>
  );
};

export default CreateTask;
```

এবার যদি আমরা ব্রাউজারে গিয়ে Create Task বাটনে ক্লিক করি সেটা Tasks.jsx থেকে addNewTask ফাংশনকে কল করবে এবং কনসোলে লগ করবে।

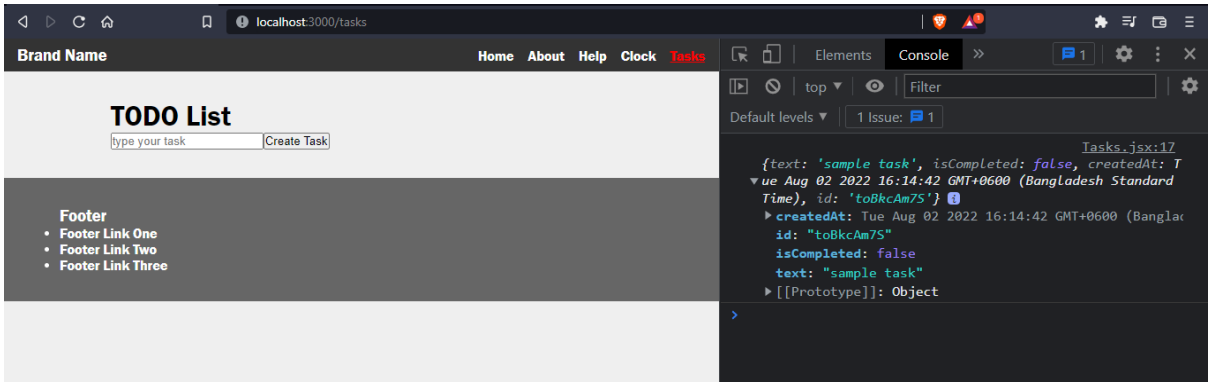


এটাকে বলে স্টেট লিফটিং। অর্থাৎ আমরা ফাংশন তৈরি করলাম আমাদের অ্যাপে, কিন্তু কল করলাম যেখানে স্টেট আছে সেখানে, এবং সেটাকে আর্গুমেন্ট আকারে অ্যাপের কাছে পাঠিয়ে দিলাম। অর্থাৎ আমরা স্টেটকে উপরে তুলে নিয়ে আসলাম।

এবার আমাদের একটা টাস্ক অবজেক্ট তৈরি করতে হবে। সেই অবজেক্টকে আমরা addNewTask ফাংশনের মধ্যে রাখবো। তার আগে আমাদের আইডি জেনারেট করার জন্য `shortid` নামক একটা প্যাকেজ ইনস্টল করবো এবং তা Tasks.jsx এ ইমপোর্ট করবো।

```
const addNewTask = (text) => {
  const task = {
    text,
    isCompleted: false,
    createdAt: new Date(),
    id: shortid.generate(),
  };
  console.log(task);
};
```

এবার যদি ব্রাউজারে গিয়ে আমাদের টাস্কের চেহারাটা দেখি তাহলে নিচের ছবির মতো চেহারা পাবো।

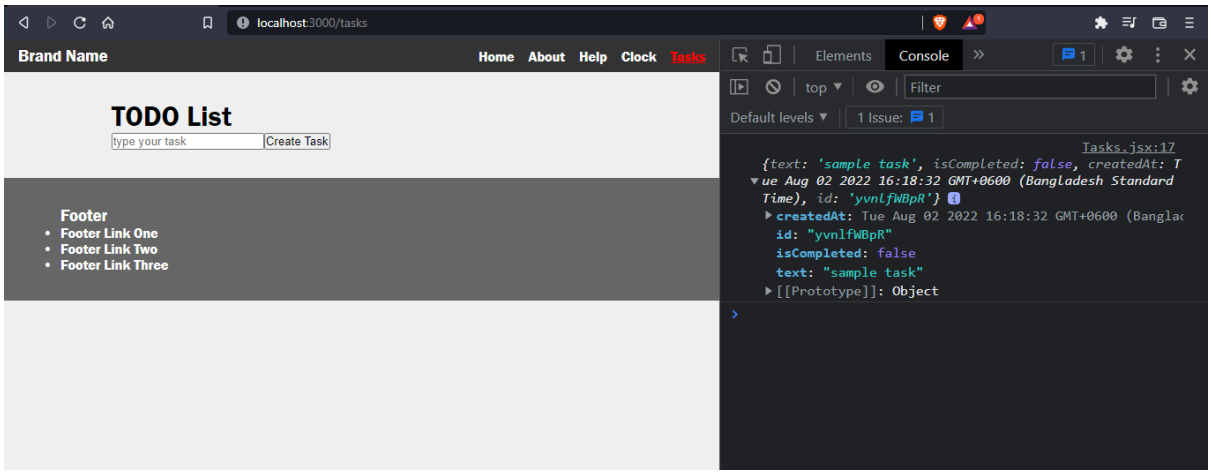


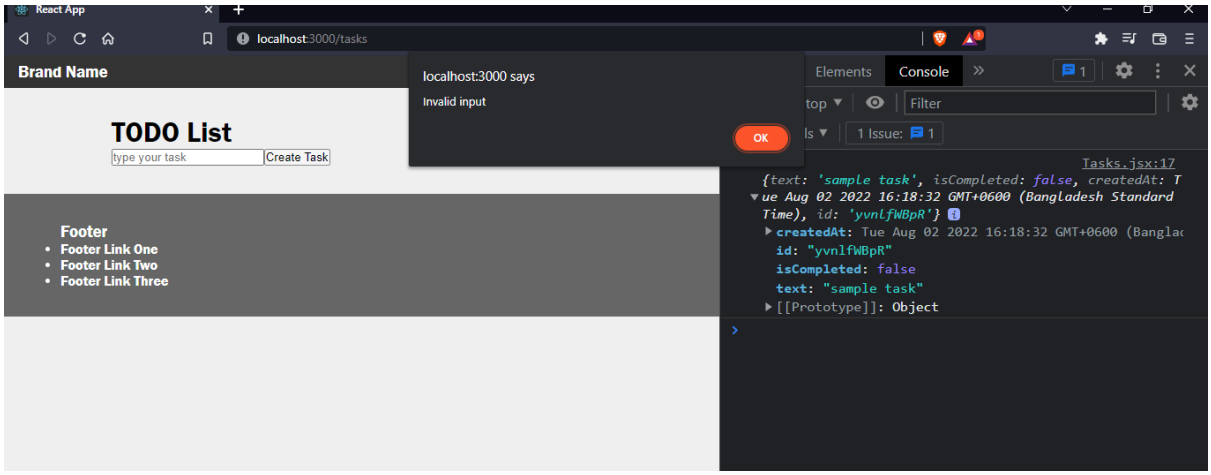
কিন্তু এখানে সমস্যা হচ্ছে আমরা যদি ইনপুট ফাঁকা রেখে বাটনে ক্লিক করি সেটা অ্যাড হয়ে যাবে ফাঁকা টেক্সট হিসেবে। কিন্তু সেটা তো হতে দেয়া যাবে না। সেটা আমরা ভ্যালিডেট করবো আমাদের ইনপুট ফিল্ড যেখানে আছে সেখানে, অর্থাৎ আমাদের কম্পোনেন্টে।

```
<button
  onClick={() => {
    if (text) {
      addNewTask(text);
      setText('');
    } else {
```

```
      alert('Invalid input');
    }
  }}
>
  Create Task
</button>;
```

তাহলে আমাদের যদি টেক্সট থাকে আর যদি না থাকে চেহারা হবে নিচের ছবিগুলোর মতো।





এবার এই টাস্ক অবজেক্ট থাকবে স্টেটের মধ্যে। সেখানে আমরা কিভাবে রাখতে পারি চলুন দেখি।

```
const addNewTask = (text) => {
  const task = {
    text,
    isCompleted: false,
    createdAt: new Date(),
    id: shortid.generate(),
  };
  setTasks([task, ...tasks]);
};
```

এখন আমাদের এই টাস্কগুলো শো করাতে হবে। তার জন্য ShowTasks.jsx নামের একটা কম্পোনেন্ট নিতে হবে।

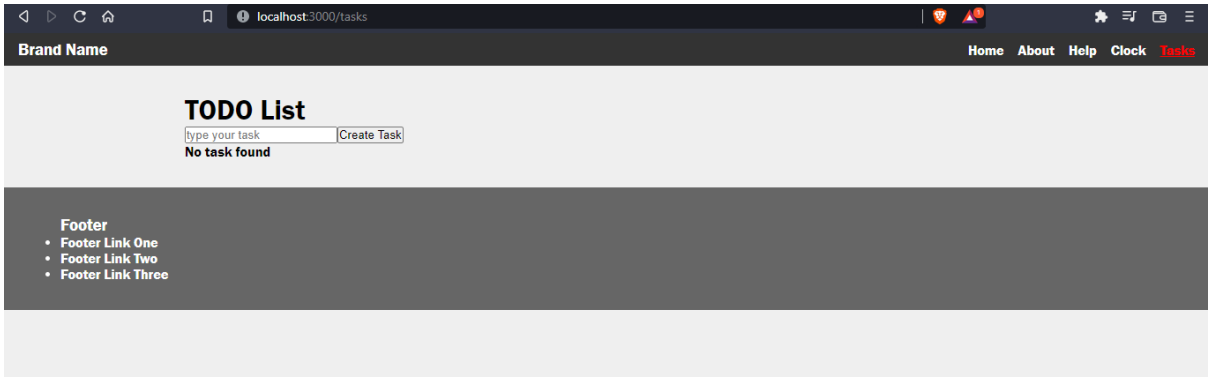
```
const ShowTasks = ({ tasks }) => {
  return (
    <div>
      {tasks.length > 0 ? (
        <ul>
          {tasks.map((task) => (
            <li key={task.id}>{task.text}</li>
          ))}
        </ul>
      ) : (
        <p>No task found</p>
      )}
    </div>
  );
};

export default ShowTasks;
```

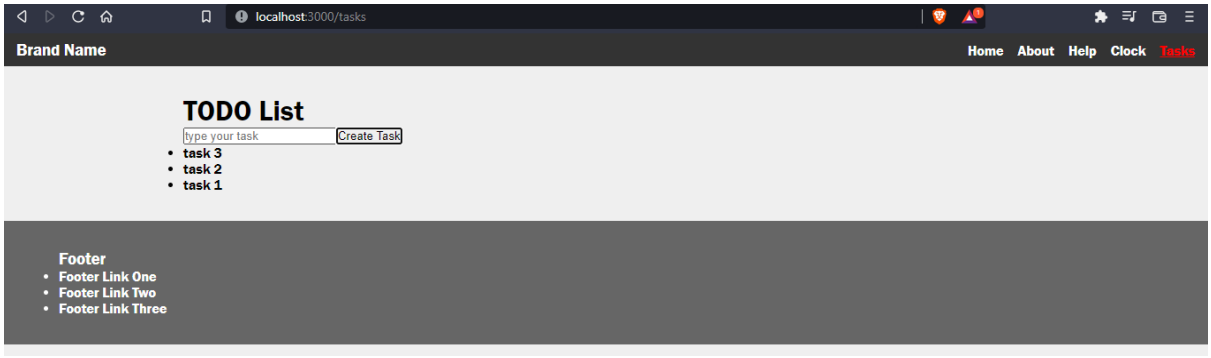
যখন আমরা ম্যাপ করি তখন রিয়্যাক্ট বুঝতে
পারে না কি করতে হবে। সে সব আইটেমকে একই ধরে নেয় তাই আমাদের প্রতিটা আইটেমের সাথে একটা ইউনিক `key` দিয়ে দিতে হয়। এবার আমরা এটাকে আমাদের Tasks.jsx এ ইমপোর্ট করে ব্যবহার করবো।

```
import { useState } from 'react';
import shortid from 'shortid';
import Layout from '../components/layout/Layout';
import CreateTask from '../components/tasks/CreateTask';
import ShowTasks from '../components/tasks/ShowTasks';

const Tasks = () => {
  const [tasks, setTasks] = useState([]);
  const [visibility, setVisibility] = useState('all');

  const addNewTask = (text) => {
    const task = {
      text,
      isCompleted: false,
      createdAt: new Date(),
      id: shortid.generate(),
    };
    setTasks([task, ...tasks]);
  };

  return (
    <Layout>
      <h1>TODO List</h1>
      <CreateTask addNewTask={addNewTask} />
```

```
      <ShowTasks tasks={tasks} />
    </Layout>
  );
};

export default Tasks;
```

প্রথম অবস্থায় আমাদের অ্যাপের চেহারা দেখাবে এরকম যেহেতু আমাদের কোনো টাস্ক নেই টাস্ক অ্যাড করলে দাঁড়াবে এরকম।





> আমরা আমাদের যে আর্কিটেকচার বানিয়েছিলাম অর্থাৎ যে ডায়াগ্রামটা বানিয়েছিলাম সেটার বাকি কাজগুলো হচ্ছে আপনাদের টাস্ক। আপনারা
> চেষ্টা করবেন এই অ্যাপ্লিকেশনটা কমপ্লিট করতে। এটা নিজেদের কাছে একটা
> চ্যালেঞ্জ হিসেবে নিবেন।

সোর্স কোডঃ

এই লেকচারে সকল সোর্স কোড এই লিংক এ পাবেন।