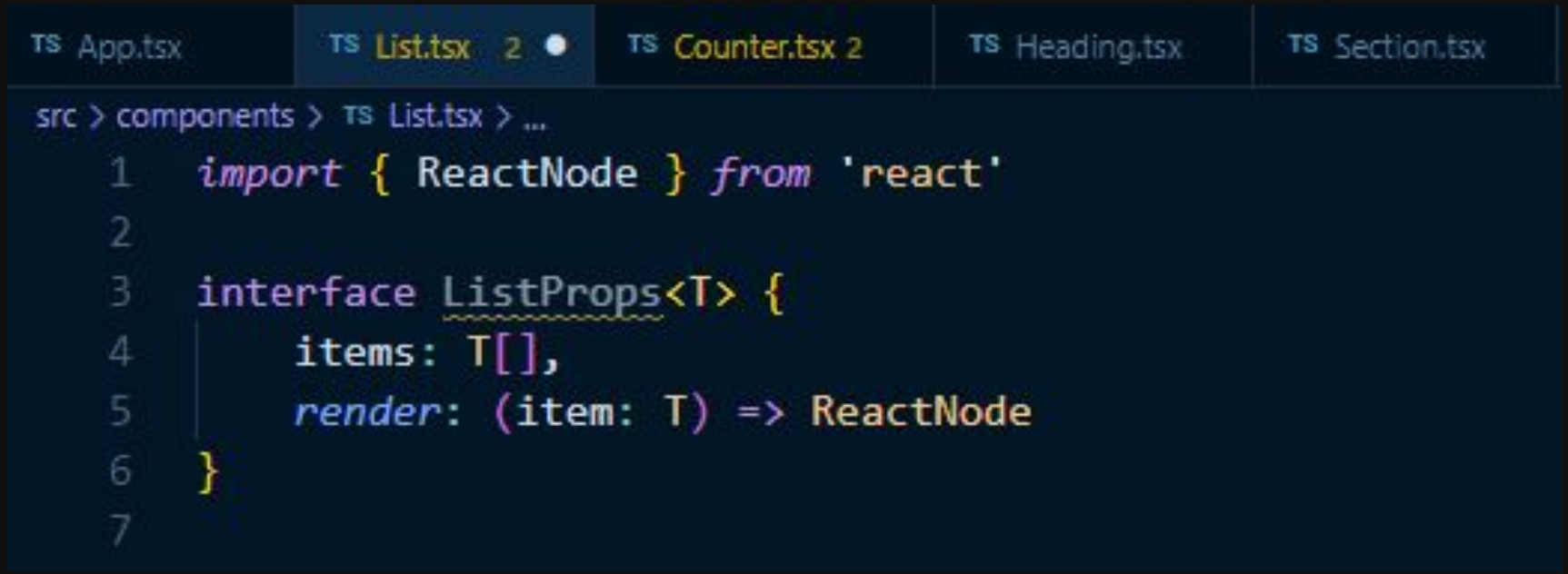
DAY 21 : REACT 101

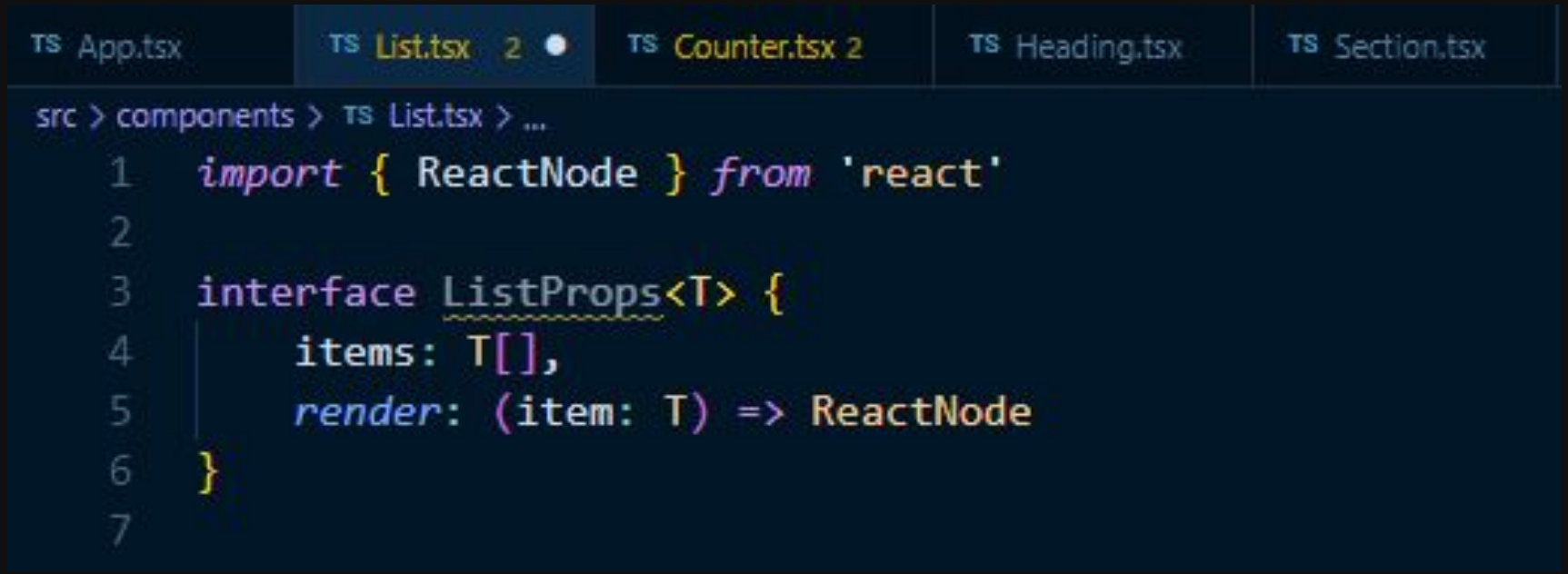
Road to Front-end Developer Bootcamp

Generic List

สร้างไฟล์ List.tsx



สร้าง Interface ขึ้นมา

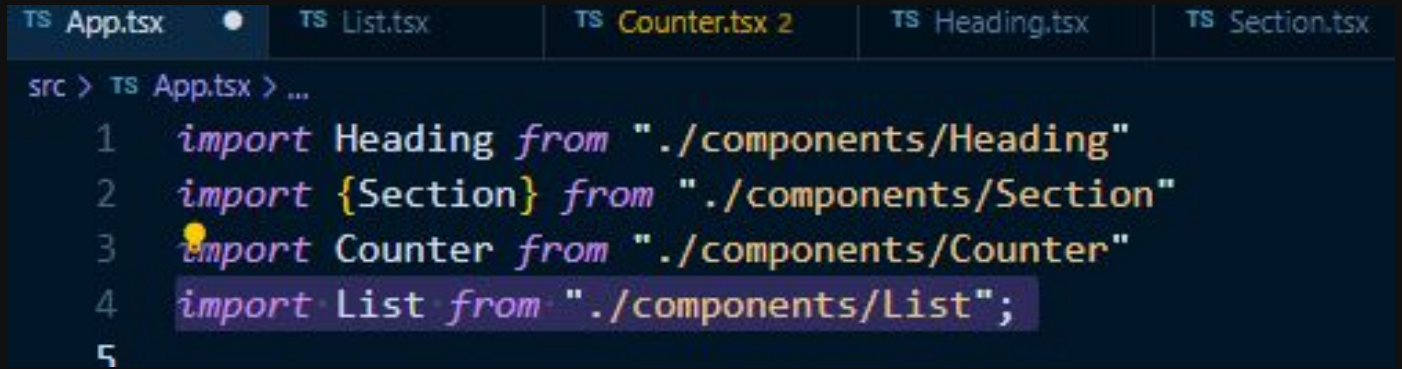


ต่อมาทำการสร้าง List ขึ้นมา

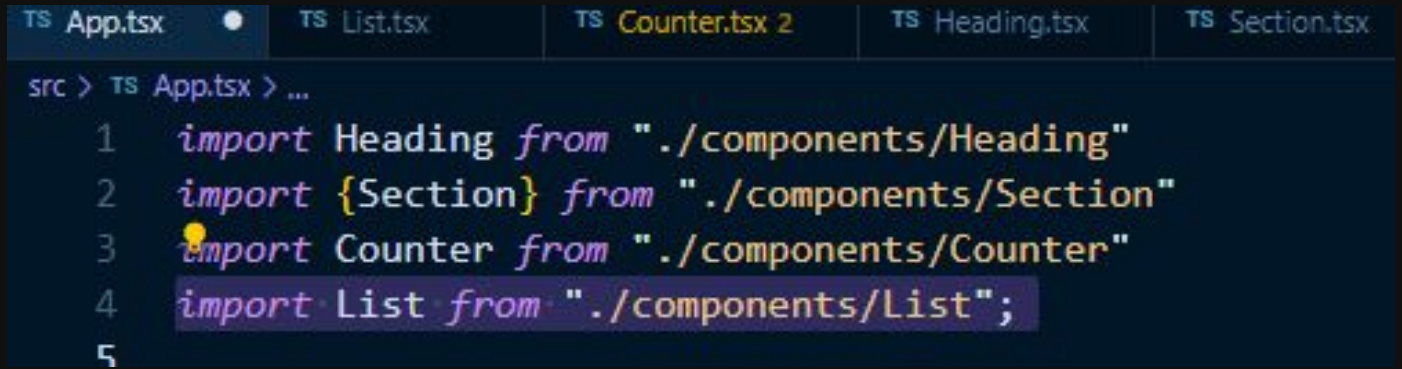
src > components > TS List.tsx > ...

```tsx
import { ReactNode } from 'react'

interface ListProps<T> {
    items: T[],
    render: (item: T) => ReactNode
}

const List = <T,>({ items, render }: ListProps<T>) => {
    return (
        <ul>
            {items.map((item, index) => (
                <li key={index}>
                    {render(item)}
                </li>
            ))}
        </ul>
    )
}

export default List
```

นำเข้า List มาใน App.tsx



นำเข้า List มาใน App.tsx



แก้ไขฟังก์ชัน App ในการ Return และ แสดงผล List ที่สร้างไว้

```tsx
function App() {
  const [count, setCount] = useState<number>(1);

  return (
    <>
      <Heading title="Hello It's me"/>
      <Section> My Section</Section>
      <Counter setCount={setCount}>Count = {count}</Counter>
      <List items={["⭐ Star","🍴 Fork","🔥 Fire"]} render={(item) => <b>{item}</b>}></List>
    </>
  )
}

export default App
```

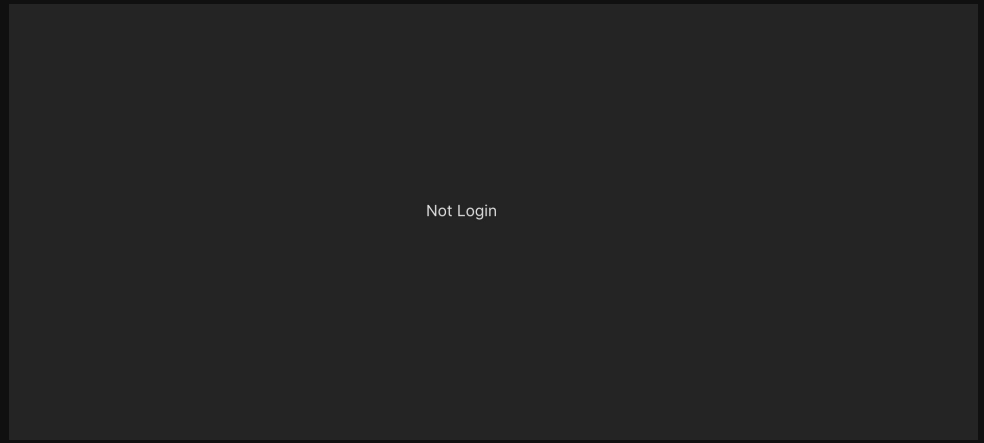
✅ Conditional Rendering

✅ Conditional Rendering
คือ เงื่อนไขที่เราสามารถกำหนดได้
ก่อนจะทำการ Render หน้าเว็บ

```tsx
import React from 'react'


let isLogin:boolean = false;

const Dashboard = () => {
  return (
    <div>Dashboard</div>
  )
}

export default Dashboard
```

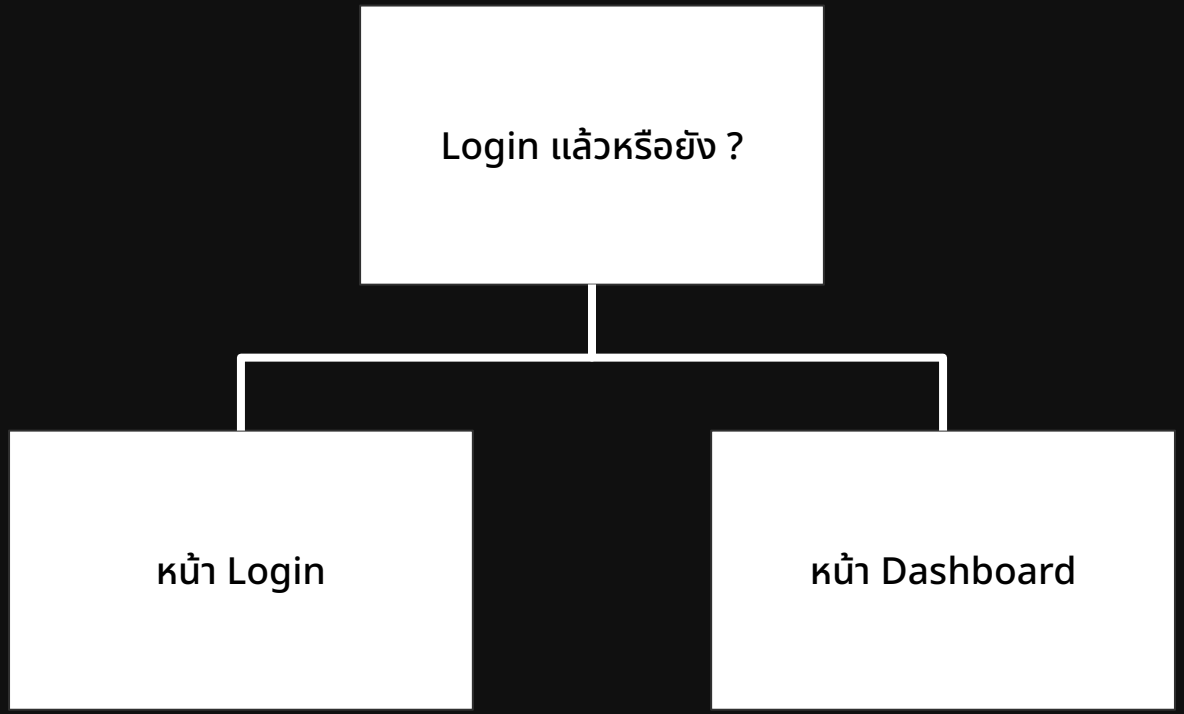
Quiz : ถ้า Login เป็น True ให้ขึ้น Dashboard ถ้า False ให้ขึ้น Not Login

```tsx
import React from 'react'


let isLogin:boolean = false;

const Dashboard = () => {
  return (
    <div>Dashboard</div>
  )
}

export default Dashboard
```



ให้เวลาประมาณ 3 นาที

✅ เราสามารถกำหนดเงื่อนไขในลักษณะนี้
กับ Component อื่น ๆ ก็ได้เช่นกัน



Quiz : ลองสร้าง หน้า Login แยกออกมา และ ตรวจสอบเงื่อนไขว่า Login อยู่หรือไม่ ถ้าใช่ให้ Render Dashbaord ถ้าไม่ใช่ ให้เป็นหน้า Login

ให้เวลาประมาณ 5 นาที

กรณีที่เราต้องการ Interactive Form ณ. เวลานั้น ๆ เลย ..

ทำการสร้าง State ไว้รอรับ User / Pwd

```
1   import React from 'react'
2   import { useState } from 'react'
3
4   const Login = () => {
5       const [Username, setUsername] = useState<string>();
6       const [Password, setPassword] = useState<string>();
7
8     return (
9       <div>Login</div>
10    )
11  }
12
13  export default Login
```

สร้างฟังก์ชันสำหรับรองรับเมื่อผู้ใช้งาน ได้เปลี่ยน Username หรือ รหัสผ่าน

```
const onChangeUsername = (event: React.ChangeEvent<HTMLInputElement>) => {
    setUsername(event.target.value)
    console.log(Username)
  }

  const onChangePassword = (event: React.ChangeEvent<HTMLInputElement>) => {
    setPassword(event.target.value)
    console.log(Password)
  }
```

ฟังก์ชันที่จะทำงานเมื่อมีการ Submit

```
const onSubmit = (event: React.FormEvent<HTMLFormElement>) => {
  event.preventDefault()
  console.log("Submitted")
}
```

สิ่งที่เราจะทำการสร้างในฟอร์ม

```
    return (
      <>
      <div>
        <form onSubmit={onSubmit}>
          <label htmlFor="name">Name</label>
          <input type="text" id="username" name="name" onChange={onChangeUsername}/>
          <input type="password" id="password" name="name" onChange={onChangePassword}/>
        </form>
      </div>
      </>
    )
  }

export default Login
```

Tips : กรณีแปลง Str > number เราสามารถใส่ + ด้านหน้าได้เลย

```tsx
const [Username, setUsername] = useState<number>();
const [Password, setPassword] = useState<number>();
const onChangeUsername = (event: React.ChangeEvent<HTMLInputElement>) => {
    setUsername(+event.target.value)
    console.log(Username)
  }

  const onChangePassword = (event: React.ChangeEvent<HTMLInputElement>) => {
    setPassword(+event.target.value)
    console.log(Password)
  }
```

2250

Exactly like in JavaScript, you can use the `parseInt` or `parseFloat` functions, or simply use the unary `+` operator:

```
var x = "32";
var y: number = +x;
```

All of the mentioned techniques will have correct typing and will correctly parse simple decimal integer strings like `"123"`, but will behave differently for various other, possibly expected, cases (like `"123.45"`) and corner cases (like `null`).

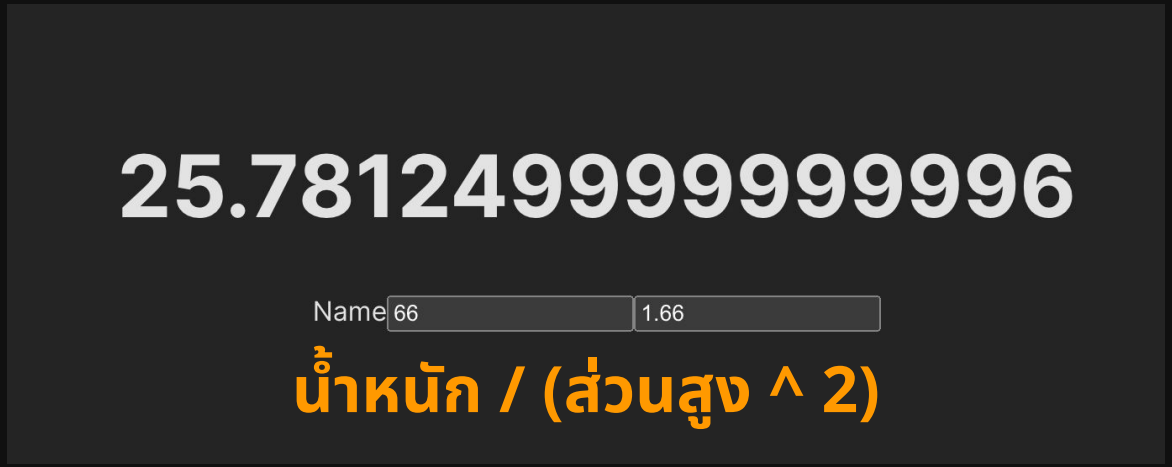
x	parseInt(x)	parseFloat(x)	Number(x)	+x	~~x	x>>>0	isNaN(x)
"123"	123	123	123	123	123	123	false
"+123"	123	123	123	123	123	123	false
"-123"	-123	-123	-123	-123	-123	4294967173	false
"123.45"	123	123.45	123.45	123.45	123	123	false
"-123.45"	-123	-123.45	-123.45	-123.45	-123	4294967173	false
"12e5"	12	1200000	1200000	1200000	1200000	1200000	false
"12e-5"	12	0.00012	0.00012	0.00012	0	0	false
"0123"	123	123	123	123	123	123	false
"0000123"	123	123	123	123	123	123	false
"0b111"	0	0	7	7	7	7	false
"0o10"	0	0	8	8	8	8	false
"0xBABE"	47806	0	47806	47806	47806	47806	false
"4294967295"	4294967295	4294967295	4294967295	4294967295	-1	4294967295	false
"123456789012345678"	123456789012345680	123456789012345680	123456789012345680	123456789012345680	-1506741424	2788225872	false
"12e999"	12	Infinity	Infinity	Infinity	0	0	false
""	NaN	NaN	0	0	0	0	false
"123foo"	123	123	NaN	NaN	0	0	true
"123.45foo"	123	123.45	NaN	NaN	0	0	true
" 123 "	123	123	123	123	123	123	false
"foo"	NaN	NaN	NaN	NaN	0	0	true
"12e"	12	12	NaN	NaN	0	0	true
"0b567"	0	0	NaN	NaN	0	0	true
"0o999"	0	0	NaN	NaN	0	0	true
"0xFUZZ"	15	0	NaN	NaN	0	0	true
"+0"	0	0	0	0	0	0	false
"-0"	0	0	0	0	0	0	false
"Infinity"	NaN	Infinity	Infinity	Infinity	0	0	false
"+Infinity"	NaN	Infinity	Infinity	Infinity	0	0	false
"-Infinity"	NaN	-Infinity	-Infinity	-Infinity	0	0	false
null	NaN	NaN	0	0	0	0	false
undefined	NaN	NaN	NaN	NaN	0	0	true
true	NaN	NaN	1	1	1	1	false
false	NaN	NaN	0	0	0	0	false
Infinity	NaN	Infinity	Infinity	Infinity	0	0	false
NaN	NaN	NaN	NaN	NaN	0	0	true
{}	NaN	NaN	NaN	NaN	0	0	true
{valueOf: function(){return 42}}	NaN	NaN	42	42	42	42	false
{toString: function(){return "56"}}	56	56	56	56	56	56	false

Quiz : จากตัวอย่างก่อนหน้า ลองรับข้อมูลตัวเลข 2 ตัว มาคำนวณหา BMI และ แสดงในรูปแบบ Console

```tsx
import React from 'react'
import { useState } from 'react'

const Login = () => {
    const [Username, setUsername] = useState<string>();
    const [Password, setPassword] = useState<string>();
    const onChangeUsername = (event: React.ChangeEvent<HTMLInputElement>) => {
        setUsername(event.target.value)
        console.log(Username)
      }

      const onChangePassword = (event: React.ChangeEvent<HTMLInputElement>) => {
        setPassword(event.target.value)
        console.log(Password)
      }

      const onSubmit = (event: React.FormEvent<HTMLFormElement>) => {
        event.preventDefault()
        console.log("Submitted")
      }

      return (
        <>
        <div>
          <form onSubmit={onSubmit}>
            <label htmlFor="name">Name</label>
            <input type="text" id="username" name="name" onChange={onChangeUsername
            <input type="password" id="password" name="pwd" onChange={onChangePassv
          </form>
        </div>
        </>
      )
}

export default Login
```

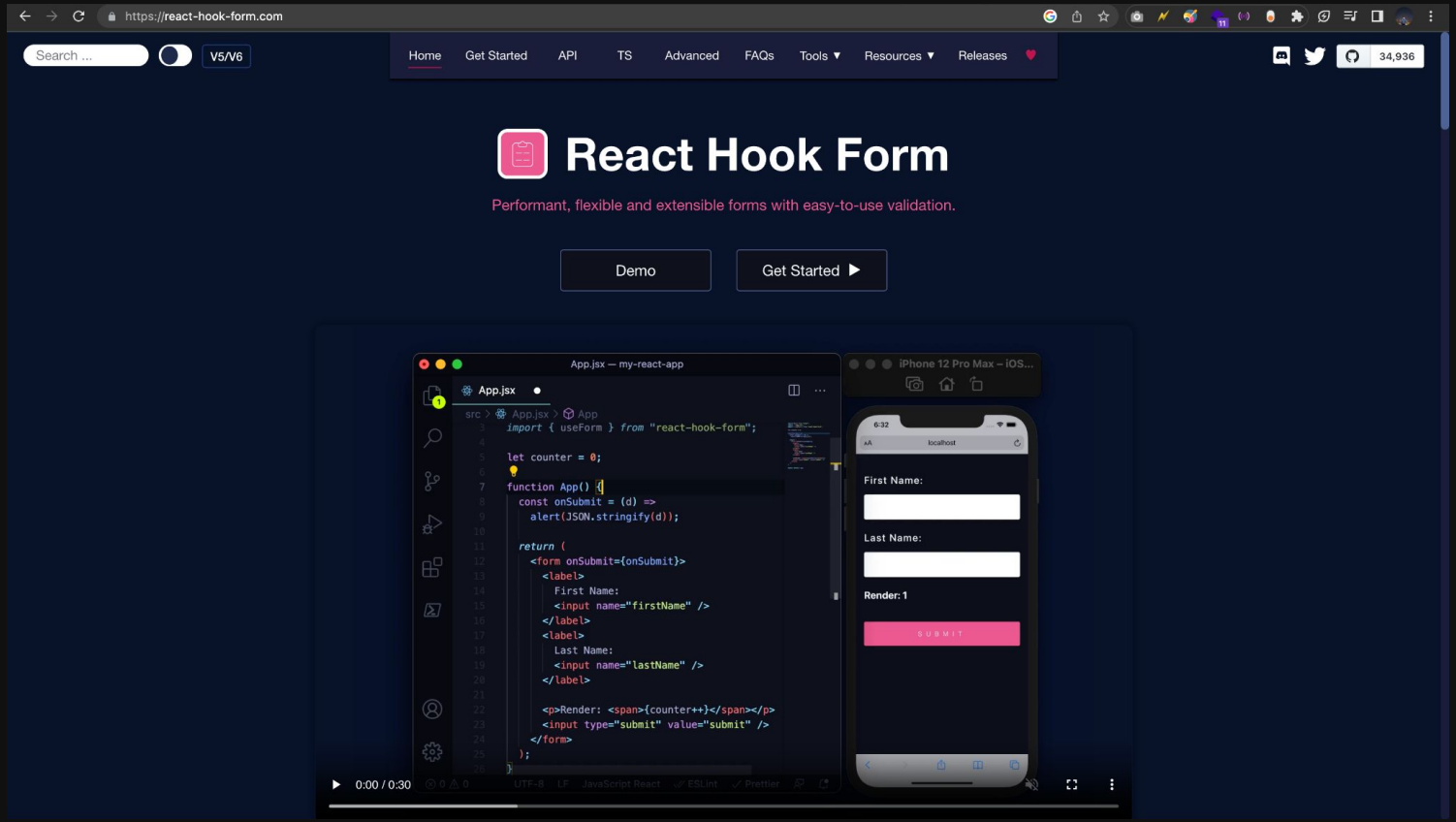
ให้เวลาประมาณ 5 นาที

Quiz 2 : จากตัวอย่างก่อนหน้า ลองรับข้อมูลตัวเลข 2 ตัว มาคำนวณหา BMI และ แสดงในรูปแบบ Text จาก State

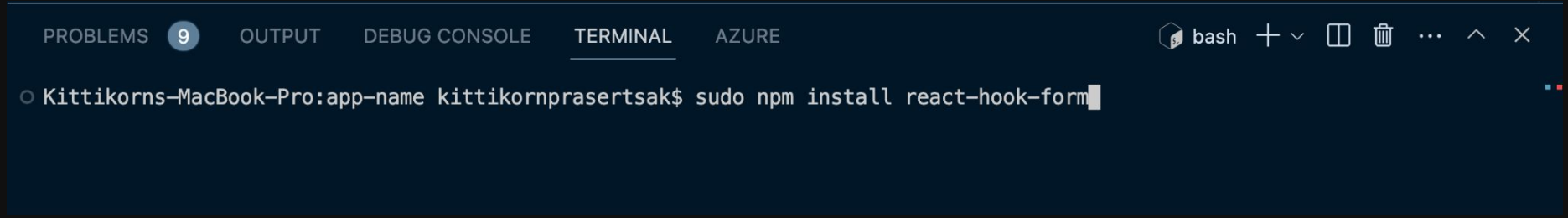


ให้เวลาประมาณ 10 นาที

⌨️ การทำ React ควบคู่กับ Form ทั่วไป



ถ้าใน Windows ไม่ต้อง sudo



เริ่มจาก Import ตัว react-hook-form

```
2  import { useForm } from 'react-hook-form'
3
4  type Inputs = {
5      email: string,
6      password: string
7  }
```

ต่อมาเขียนเนื้อด้านใน

```
9   const MyForm = () => {
10      const { register, handleSubmit } = useForm<Inputs>()
11
12      const onSubmit = (data:any) => console.log(data)
13
14      return (
15          <form onSubmit={handleSubmit(onSubmit)}>
16              <input {...register('email')} />
17              <input {...register('password')} />
18              <input type='submit' />
19          </form>
20      )
21  }
22
23  export default MyForm
```

Dashboard

dasd | 112233 | Submit

ต่อมาเขียนเนื้อด้านใน

```
9   const MyForm = () => {
10      const { register, handleSubmit } = useForm<Inputs>()
11
12      const onSubmit = (data:any) => console.log(data)
13
14      return (
15          <form onSubmit={handleSubmit(onSubmit)}>
16              <input {...register('email')} />
17              <input {...register('password')} />
18              <input type='submit' />
19          </form>
20      )
21  }
22
23  export default MyForm
```

และ สุดท้ายให้เราทำการใส่ไว้ใน App.tsx

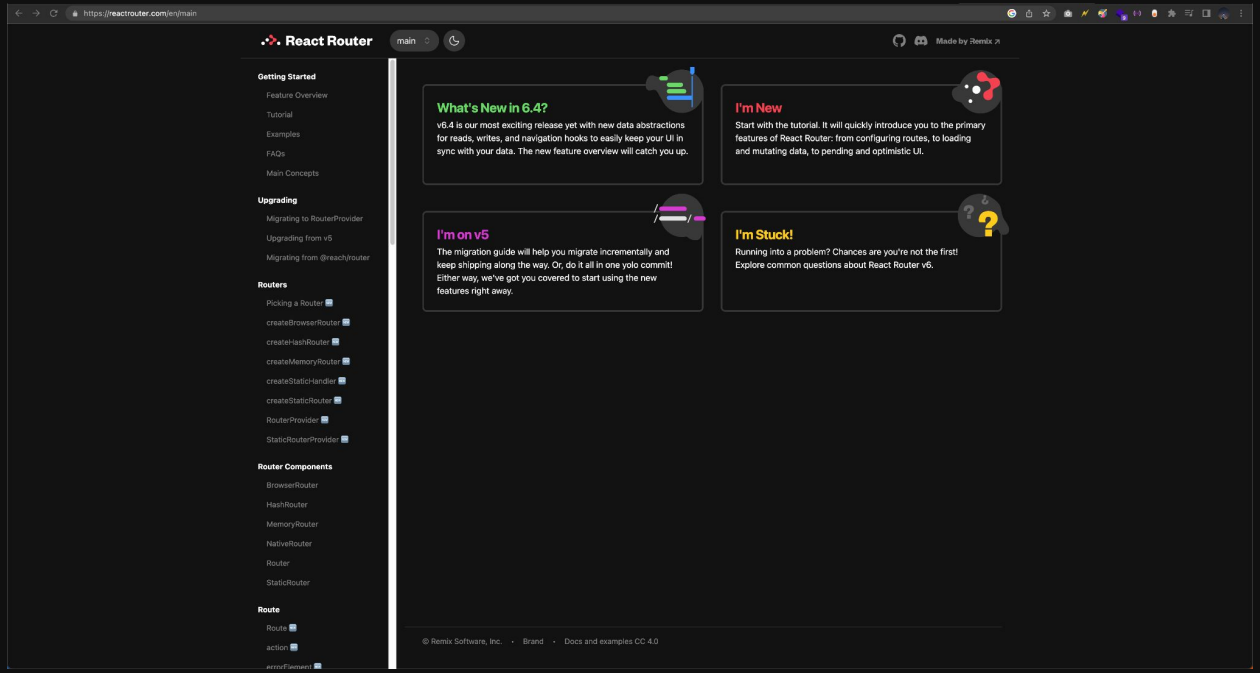
```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Dashboard from './components/Dashboard'
import Login from './components/Login'
import MyForm from './components/MyForm'

function App() {

  let isLogin:boolean = true
  if(isLogin)
  {
    return (<><Dashboard/> <MyForm/></>)
  }
  else{
    return (<Login/>)
  }
}

export default App
```

และ สุดท้ายให้เราทำการใส่ไว้ใน App.tsx

```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Dashboard from './components/Dashboard'
import Login from './components/Login'
import MyForm from './components/MyForm'

function App() {

  let isLogin:boolean = true
  if(isLogin)
  {
    return (<><Dashboard/> <MyForm/></>)
  }
  else{
    return (<Login/>)
  }
}

export default App
```

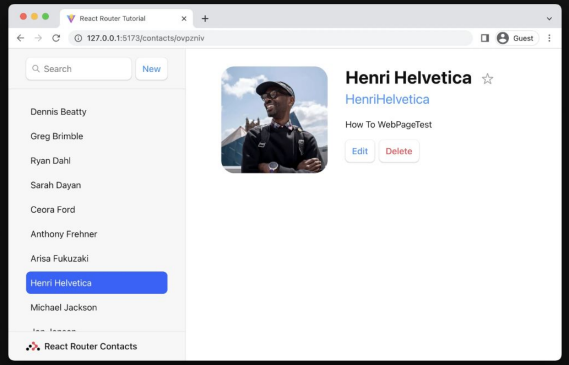
⭐ การลิงก์ไปยังหน้าต่าง ๆ ของ React

React Router



Tutorial

Welcome to the tutorial! We'll be building a small, but feature-rich app that lets you keep track of your contacts. We expect it to take between 30-60m if you're following along.



👉 **Every time you see this it means you need to do something in the app!**

The rest is just there for your information and deeper understanding. Let's get to it.

Setup

> ⓘ NOTE
> If you're not going to follow along in your own app, you can skip this section

We'll be using Vite for our bundler and dev server for this tutorial. You'll need Node.js installed for the `npm` command line tool.

👉 **Open up your terminal and bootstrap a new React app with Vite:**

```
npm create vite@latest name-of-your-project -- --template react
# follow prompts
```

ทำการติดตั้ง React Router
ถ้าใน Windows ไม่ต้องใส่ sudo

```
sudo npm install react-route
```

การทำ React Router
เราจะต้องทำการ export “wrapped”
App conponent

ทำการ import HashRouter

```tsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import { HashRouter } from 'react-router-dom'
import App from './App.tsx'
import './index.css'
```

ครอบส่วน App ด้วย HashRouter

```
7   ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
8     <React.StrictMode>
9       <HashRouter>
10        <App />
11      </HashRouter>
12    </React.StrictMode>,
13  )
```

หรือในกรณีนี้เราสามารถนำไปใส่ใน App.tsx ได้

src > App.tsx > App

```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import { HashRouter } from 'react-router-dom'

```

และหากเราใส่ใน App.tsx เราก็สามารถสร้างฟังก์ชันขึ้นมาเพื่อ Wrap ตัว App ได้เช่นกัน

```
function WrappedApp(){
  return(
    <HashRouter>
      <App />
    </HashRouter>
  )
}

export default WrappedApp
```

Code หลักของเราจะได้ลักษณะประมาณนี้

App.tsx

src > App.tsx > ...

```tsx
import './App.css'
import { HashRouter } from 'react-router-dom'

function App() {

  return (
    <div>
      <h1>Hello Vite + React!</h1>
    </div>
  )
}

function WrappedApp(){
  return(
    <HashRouter>
      <App />
    </HashRouter>
  )
}

export default WrappedApp

```

ขั้นตอนต่อมา เราจะมาทำการสร้าง Routes หรือ ตัวที่ไว้ระบุ Path ต่าง ๆ ในหน้าเว็บเรา

แต่อย่าลืม ! import ของที่ต้องใช้มาด้วยจ้า

src > App.tsx > WrappedApp

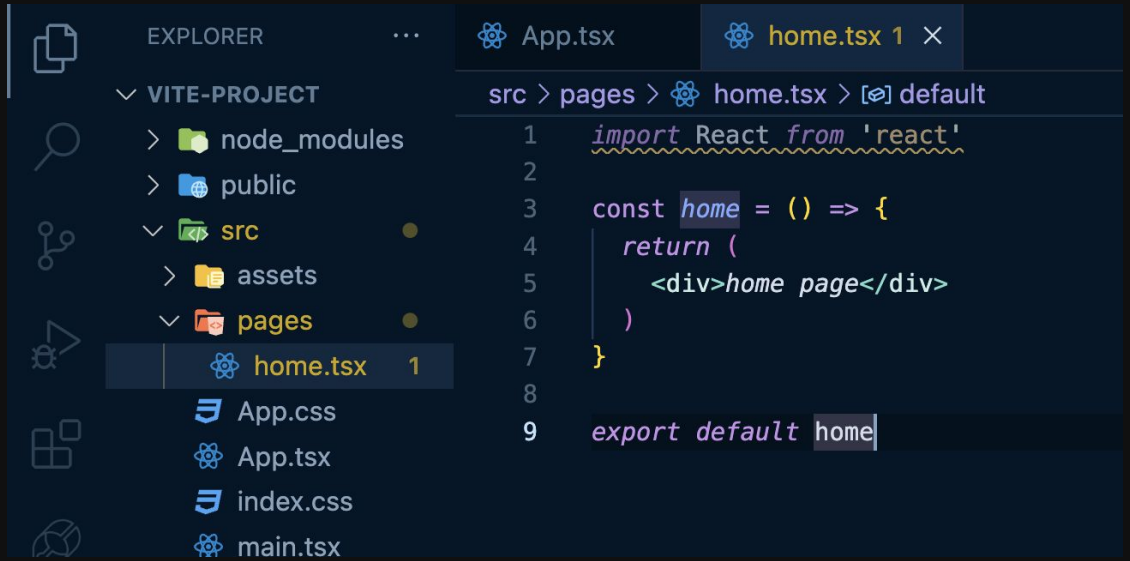
```tsx
import './App.css'
import { HashRouter, Route, Routes } from 'react-router-dom'

function App() {

  return (
    <div>
      <h1>Hello Vite + React!</h1>
      <Routes>
        <Route path="/"/>
      </Routes>
    </div>
  )
}
```

สำหรับ Path =”/” จะเป็น root หรือ หน้าหลักนั่นเอง

src > App.tsx > WrappedApp

```tsx
import './App.css'
import { HashRouter, Route, Routes } from 'react-router-dom'

function App() {

  return (
    <div>
      <h1>Hello Vite + React!</h1>
      <Routes>
        <Route path="/"/>
      </Routes>
    </div>
  )
}
```

ก่อนที่เราจะสร้าง Route เพื่อลิงก์กัน เราไปสร้างหน้าใหม่กันก่อน ให้เราสร้างใน โฟลเดอร์ pages/home.tsx กรณีไม่มีสร้างใหม่ได้เลย

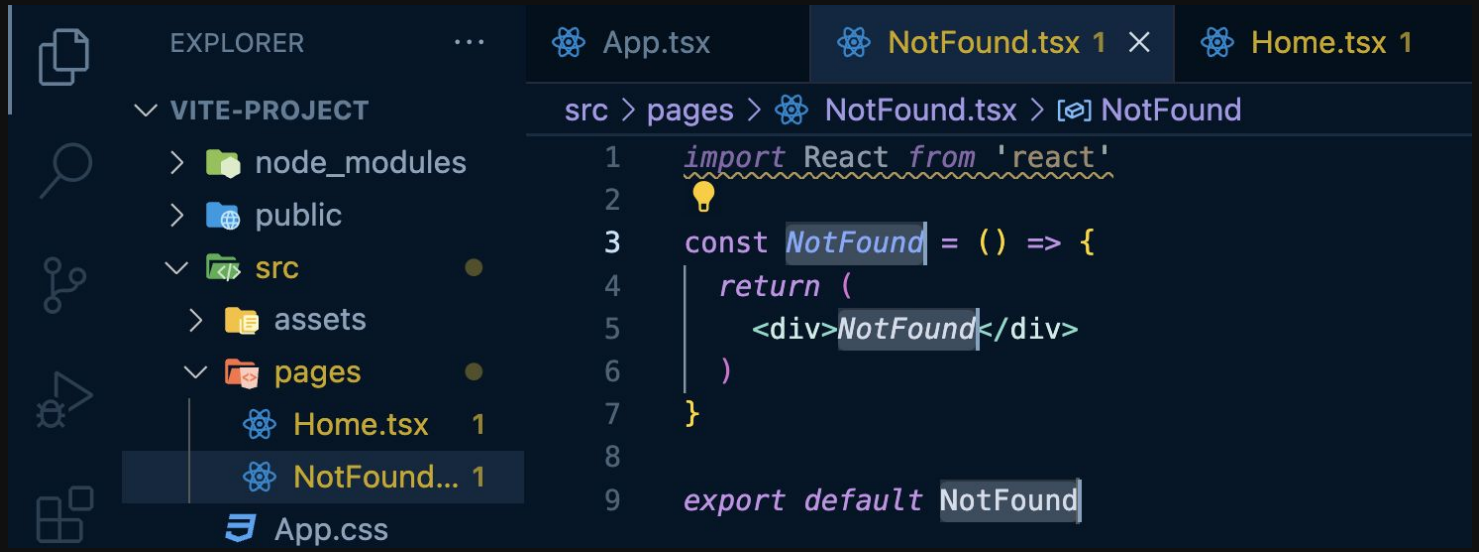


และ ทำการสร้าง Route ได้เลย โดยใน element ให้เราใส่หน้าที่ต้องการไว้

src > App.tsx > ...

```tsx
import './App.css'
import { HashRouter, Route, Routes } from 'react-router-dom'
import Home from './pages/Home'

function App() {

  return (
    <div>
      <h1>Hello Vite + React!</h1>
      <Routes>
        <Route path="/" element={<Home/>}/>
      </Routes>
    </div>
  )
}
```

กรณีที่มีหน้าอื่น ๆ เช่น หน้า Error แล้วเราจะจัดการก็สามารถ Custom เองได้ลักษณะนี้ และ ลองสร้างหน้า NotFound ได้ในลักษณะเดียวกัน

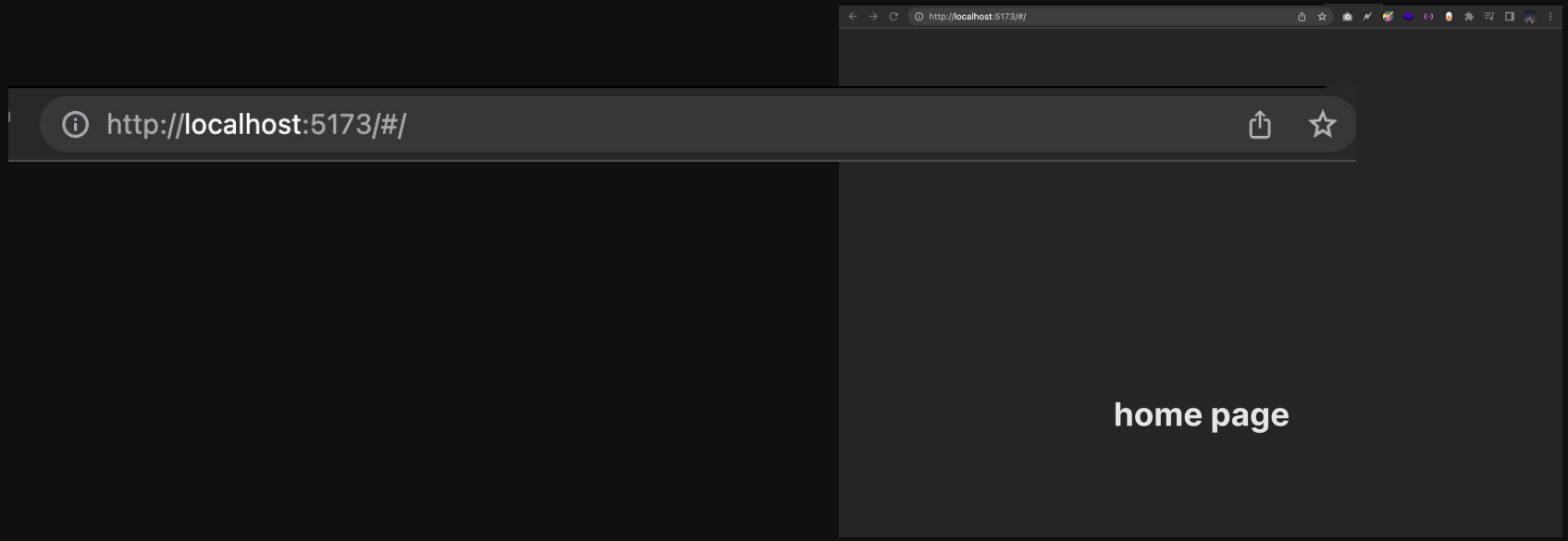


แต่จะแตกต่างกันที่ ระบุเป็น * แทนนั่นเอง

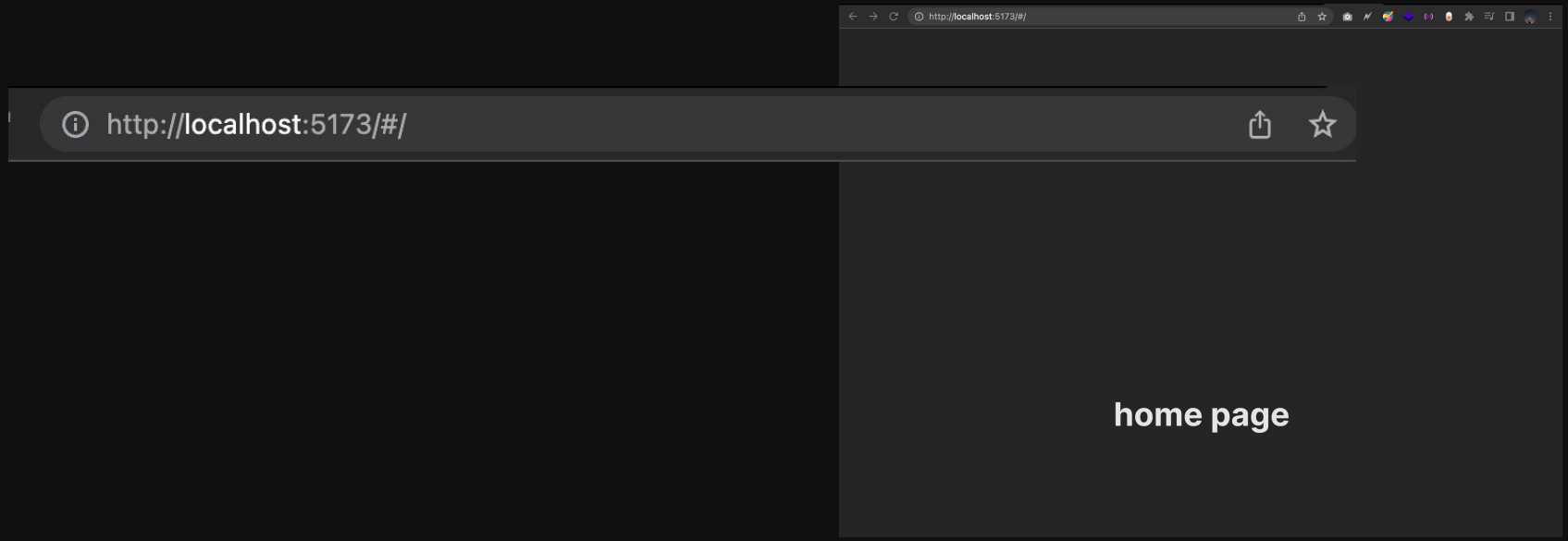
```
1  import './App.css'
2  import { HashRouter, Route, Routes } from 'react-router-dom'
3  import Home from './pages/Home'
4  import NotFound from './pages/NotFound'
5
6  function App() {
7
8    return (
9      <div>
10       <h1>Hello Vite + React!</h1>
11       <Routes>
12         <Route path="/" element={<Home/>}/>
13         <Route path='*' element={<NotFound/>} />
14       </Routes>
15     </div>
16   )
17 }
```

แล้วลองทดสอบกันดู

ความจริงคือต้องเข้าผ่าน /#/ ก่อนนะคร้าบ



ตัวนี้จะเรียกว่า HashRouter



ปกติแล้วเราจะพบ Router หลัก ๆ อยู่ 2 ตัว ก็คือ HashRouter กับ Browser Router

243

+50

`BrowserRouter`

It uses history API, i.e. it's unavailable for legacy browsers (IE 9 and lower and contemporaries). Client-side React application is able to maintain clean routes like *example.com/react/route* but needs to be backed by web server. Usually this means that web server should be configured for single-page application, i.e. same `index.html` is served for */react/route* path or any other route on server side. On client side, `window.location.pathname` is parsed by React router. React router renders a component that it was configured to render for */react/route*.

Additionally, the setup may involve server-side rendering, `index.html` may contain rendered components or data that are specific to current route.

`HashRouter`

It uses URL hash, it puts no limitations on supported browsers or web server. Server-side routing is independent from client-side routing.

Backward-compatible single-page application can use it as *example.com/#/react/route*. The setup cannot be backed up by server-side rendering because it's */* path that is served on server side, *#/react/route* URL hash cannot be read from server side. On client side, `window.location.hash` is parsed by React router. React router renders a component that it was configured to render for */react/route*, similarly to `BrowserRouter`.

Most importantly, `HashRouter` use cases aren't limited to SPA. A website may have legacy or search engine-friendly server-side routing, while React application may be a widget that maintains its state in URL like *example.com/server/side/route#/react/route*. Some page that contains React application is served on server side for */server/side/route*, then on client side React router renders a component that it was configured to render for */react/route*, similarly to previous scenario.

Ref :
https://stackoverflow.com/questions/51974369/what-is-the-difference-between-hashrouter-and-browserrouter-in-react

BrowserRouter

Uses everything after the domain(eg .com, .in, .io etc) as the path. For example, if the url is `localhost:3000/home`, then `/home` is the path/route. This router uses the HTML 5 History API to keep the UI in sync with the the path.

BrowserRouter is used for doing client side routing with URL segments. You can load a top level component for each route. This helps separate concerns in your app and makes the logic/data flow more clear. Let's see how this can be implemented in code using `react-router`:

```
1  import { BrowserRouter, Route } from 'react-router-dom';
2
3  function App() {
4      return (
5          <BrowserRouter>
6              <Route path="/home" exact component={Home} />
7          </BrowserRouter>
8      );
9  }
```

browser_router_example.ts hosted with ❤ by GitHub — view raw

> *We'll look at the advantages and disadvantages when we compare the routers later.*

https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78

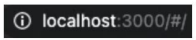
HashRouter

This is similar (visually) to the browser router, with the only difference being that a `#` is added between the domain and the path. Let's say the url is `localhost:3000/#/home`, then the `/home` is the route. Basically, it uses everything after the `#` as the path. Now, we'll see how to implement this:

```
1  import { HashRouter, Route } from 'react-router-dom';
2
3  function App() {
4      return (
5          <HashRouter>
6              <Route path="/home" exact component={Home} />
7          </HashRouter>
8      );
9  }
```

hash_router_example.ts hosted with ❤ by GitHub view raw

We have an `App()` component in which we are returning a `HashRouter` with a route. If you run our little app in the browser, you will find out that react router automatically adds a `#` before the `path`.



https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78

และ เอาจริง ๆ ยังมีอีก 2 ตัว

MemoryRouter

It keeps the history of your “URL” in memory (does not read or write to the address bar). It’s useful in tests and non-browser environments like React Native.

```
1  import { MemoryRouter, Route } from 'react-router';
2
3  function App() {
4      return (
5          <MemoryRouter>
6              <Route path="/home" exact component={Home} />
7          </MemoryRouter>
8      );
9  }
```

memory_router_example.ts hosted with ❤ by GitHub view raw

> *The component will change when you click on a link but the url will remain the same*

https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78

StaticRouter

This is useful in server-side rendering scenarios when the user isn't actually clicking around, so the location never actually changes. Hence, the name: `static`. It's also useful in simple tests when you just need to plug in a location and make assertions on the render output.

. . .

https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78

Comparing Routers

- Browser Router

The browser router is often very complicated to deploy to servers. It uses history API, so it's unavailable for legacy browsers (IE 9 and lower and contemporaries.. but who cares about that those, we are modern folks here 😎).

Client-side React application is able to maintain clean routes like *example.com/react/route* but needs to be backed by a web server. A traditional web server evaluates requests like this:

In case of a single page application, it's just a single `index.html` file which requires some extra configuration on the hosting server.

> *However, these days a lot of hosting providers assume that we use this kind of routing and simplify the process.*

We can also use server side rendering with this router, `index.html` may contain rendered components or data that are specific to current route.

https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78

- Hash Router

Enough of browser routers, let's talk hash routers. We can quite easily configure our server to ignore the `#` and server the entire html file regardless of the route. This hash will be used by our routes to show the appropriate content.

Also, `HashRouter` use cases aren't limited to SPA. A website may have legacy or search engine-friendly server-side routing, while React application may be a widget that maintains its state in URL like *example.com/server/side/route#/react/route*.

https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78

- Memory Router

Memory routers will refresh/change the component but no the url. This probably will leave the user hanging and he or she might think that his computer has hanged or there is an issue with the site which is certainly not pleasant for the user.

On the other hand, this is great for environments in which the url is not shown like React Native, Testing Environments etc

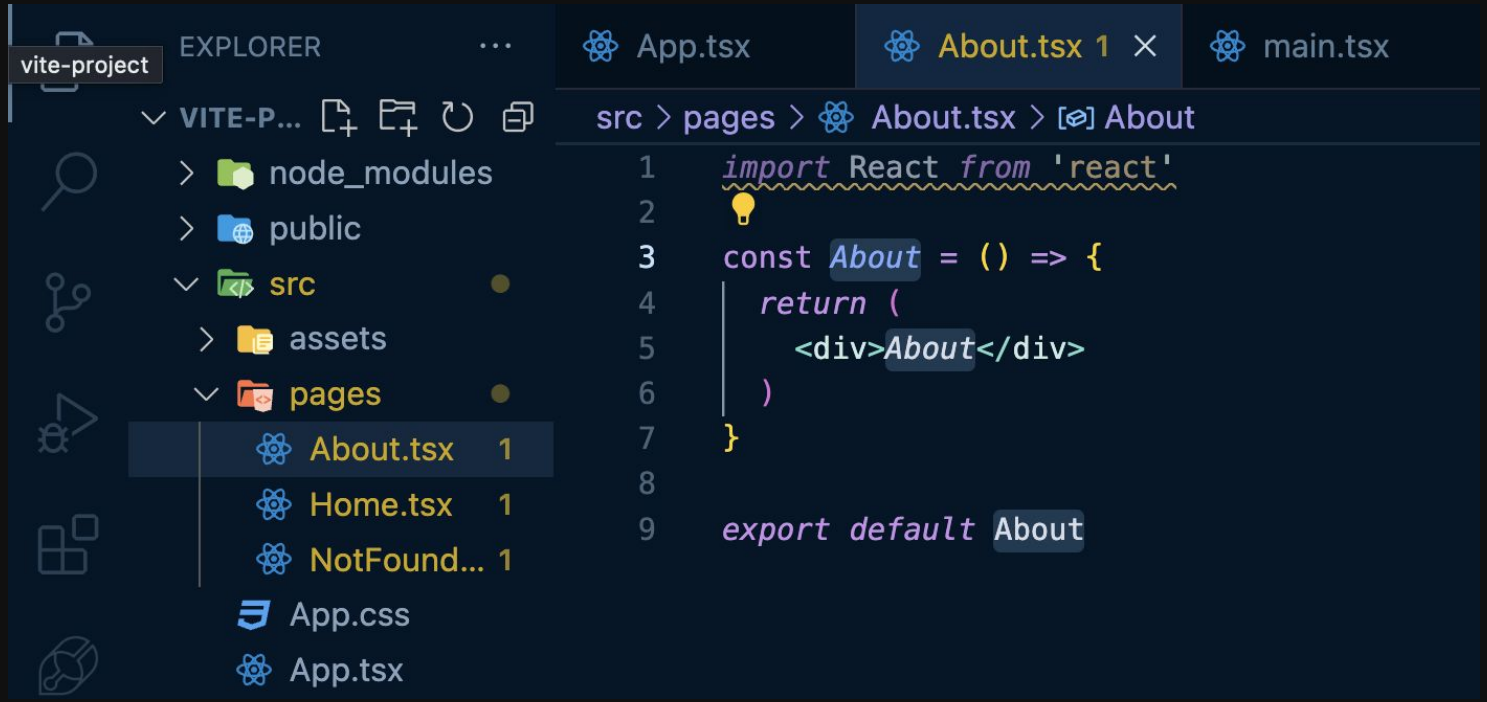
- Static Router

Static routing can be used to define an exit point from a router when no other routes are available or necessary which is called a `default route`. Think of this as a fallback route(404s). Static routing can be used for small networks that require only one or two routes.

https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78

กรณีที่เราต้องการสร้างหน้าถัดไป ?

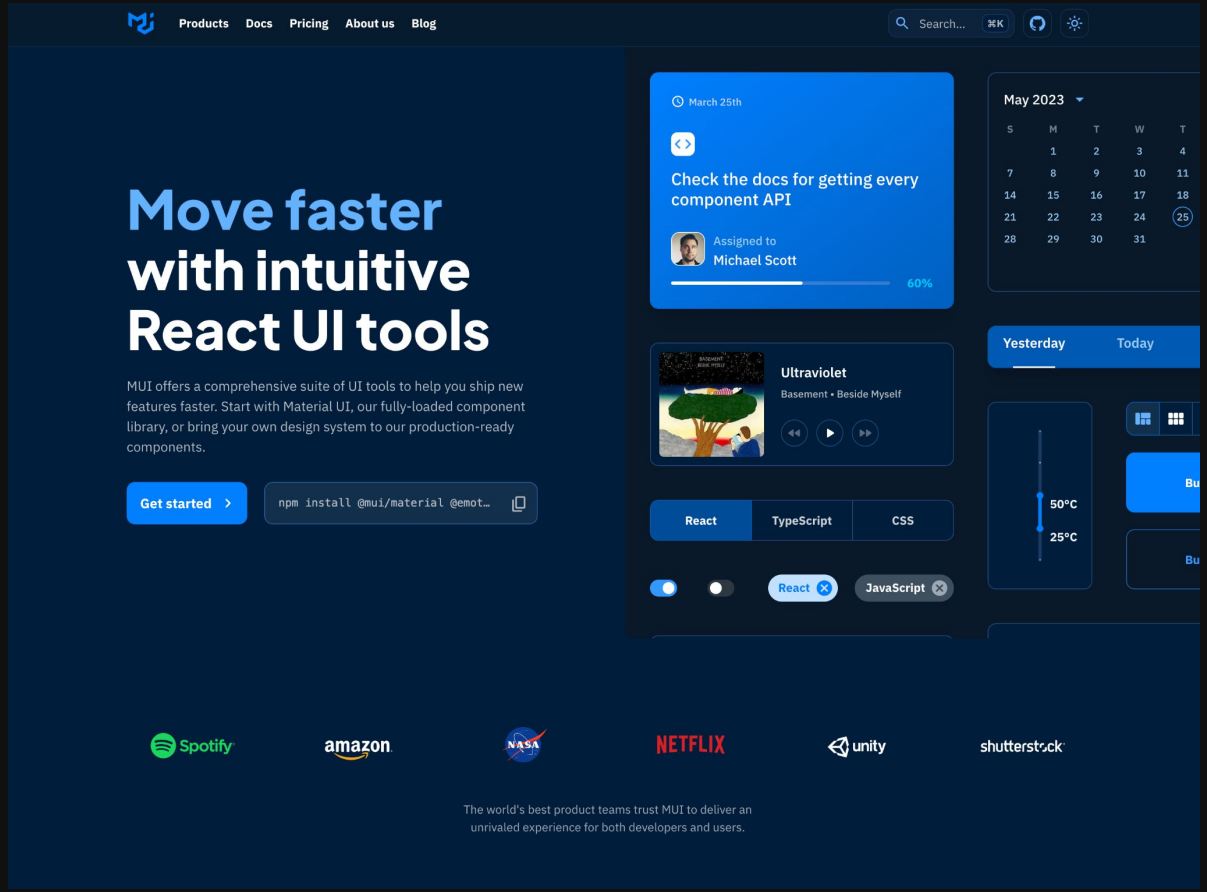
ให้เราสร้าง About.tsx ใน pages เหมือนเดิม



ต่อมาใน App.tsx ก็ทำในลักษณะเดิมนั่นเอง

```tsx
import './App.css'
import { HashRouter, Route, Routes } from 'react-router-dom'
import Home from './pages/Home'
import NotFound from './pages/NotFound'
import About from './pages/About'

function App() {

  return (
      <Routes>
        <Route path="/" element={<Home/>}/>
        <Route path="*" element={<NotFound/>} />
        <Route path="/about" element={<About/>} />
      </Routes>
  )
}
```

MUI



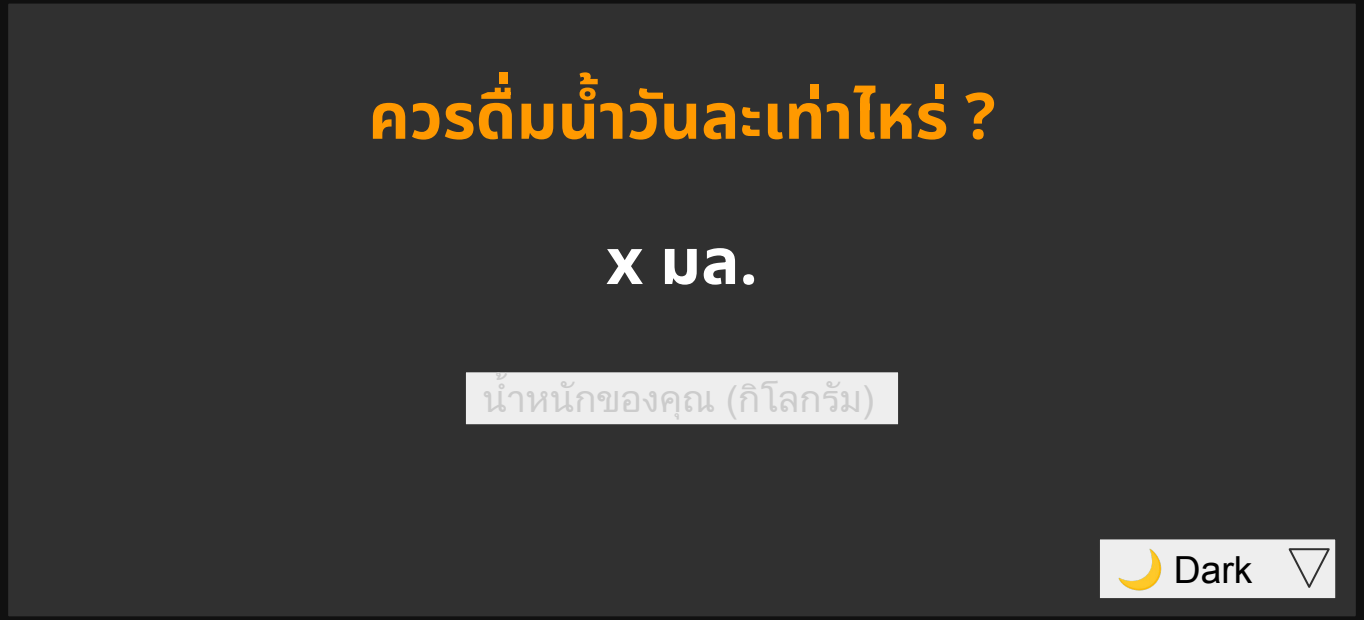
วิธีการใช้งาน MUI

1. สร้างโปรเจกต์ด้วย Vite
2. npm install vite
3. ลองทดสอบ npm run dev
4. ใช้คำสั่ง npm install @mui/material @emotion/react @emotion/styled
5. ทดสอบกับโค้ด Demo (หน้าถัดไป)

```jsx
import { useState } from 'react';
import reactLogo from './assets/react.svg';
import './App.css';
import { Button } from '@mui/material';


function App() {
  const [count, setCount] = useState(0);

  return (
    <div className="App">
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src="/vite.svg" className="logo" alt="Vite logo" />
        </a>
        <a href="https://reactjs.org" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React + + Typescript + MUI 5</h1>
      <Button color="secondary">Secondary</Button>
      <Button variant="contained" color="success">
        Success
      </Button>
      <Button variant="outlined" color="error">
        Error
      </Button>

      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </div>
  );
}

export default App;
```

ตอบใน “ส่งการบ้าน”

หน้าตาตัวอย่าง ผู้เรียนสามารถออกแบบใหม่ให้สวยขึ้นได้



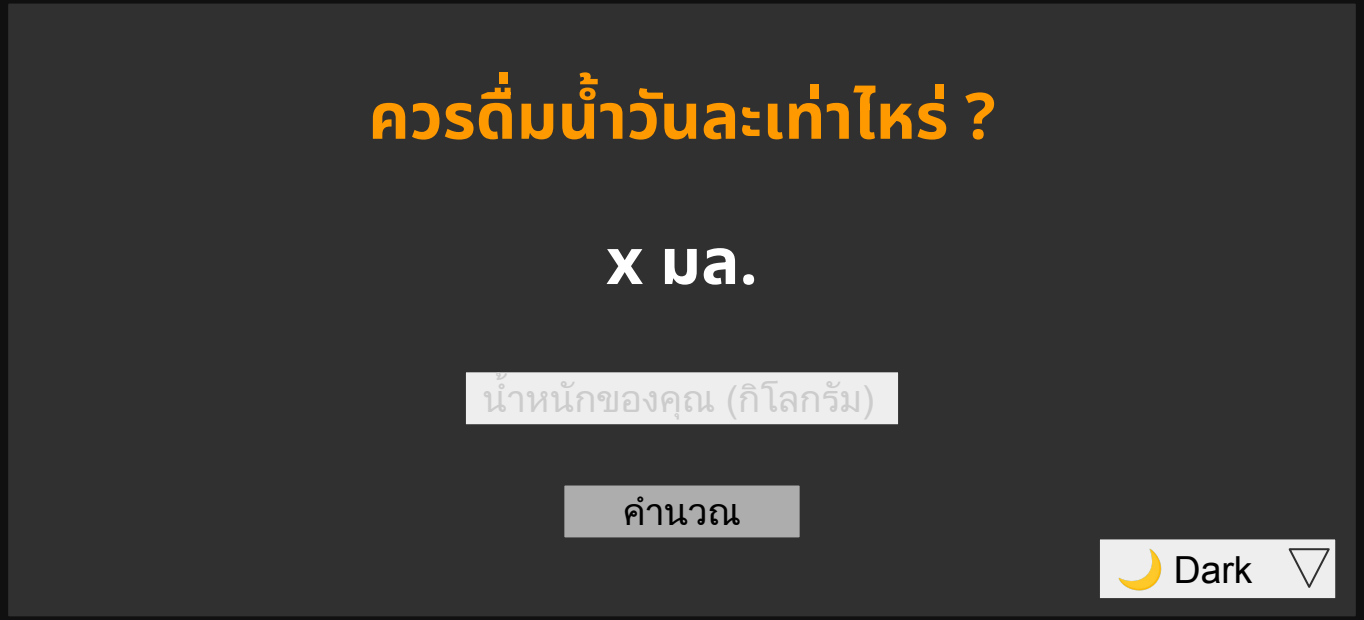
ทำโปรแกรมคำนวณปริมาณน้ำที่ต้องดื่มต่อวัน
โดยคำนวณทันทีเมื่อกรอกข้อมูลลงไป และ สามารถสลับธีมได้

สูตร : น้ำหนัก x 2.2 x 30/2 = ปริมาณน้ำ(มล.)

ตอบใน “ส่งการบ้าน”
🔥 CHALLENGE

หน้าตาตัวอย่าง ผู้เรียนสามารถออกแบบใหม่ให้สวยขึ้นได้

BORNTO DEV

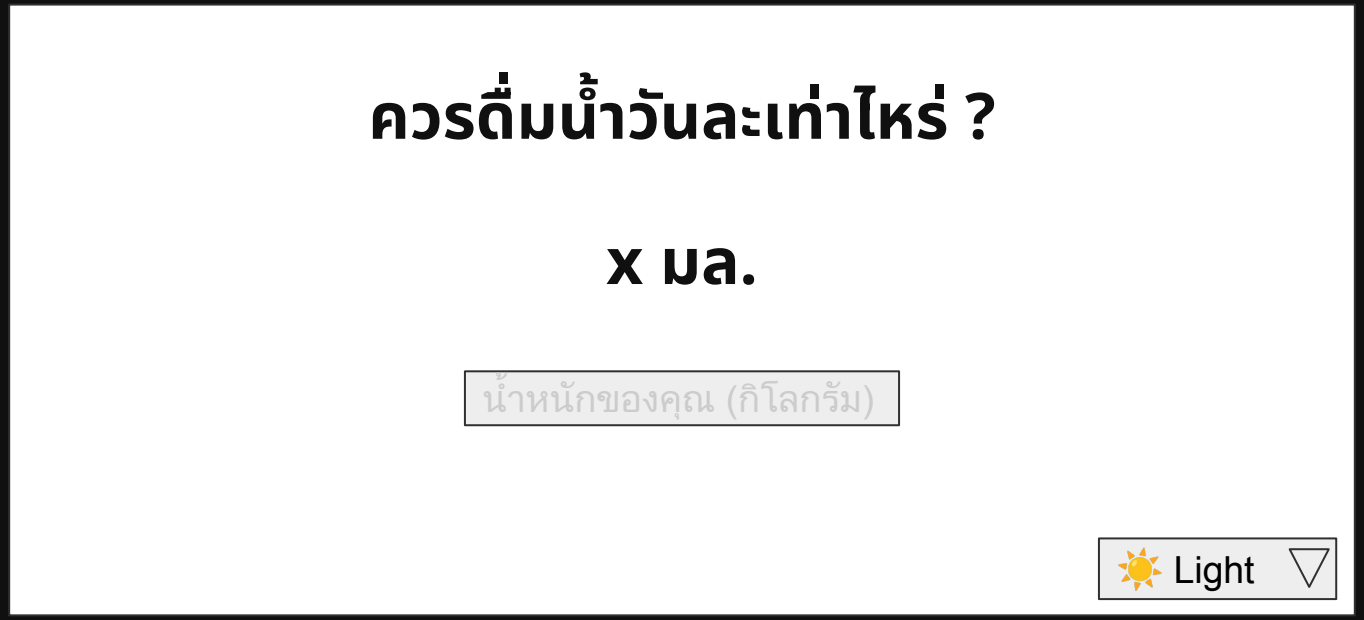


ทำโปรแกรมคำนวณปริมาณน้ำที่ต้องดื่มต่อวัน
โดยคำนวณเมื่อกดปุ่ม และ สามารถสลับธีมได้

สูตร : น้ำหนัก x 2.2 x 30/2 = ปริมาณน้ำ(มล.)

ตอบใน “ส่งการบ้าน”

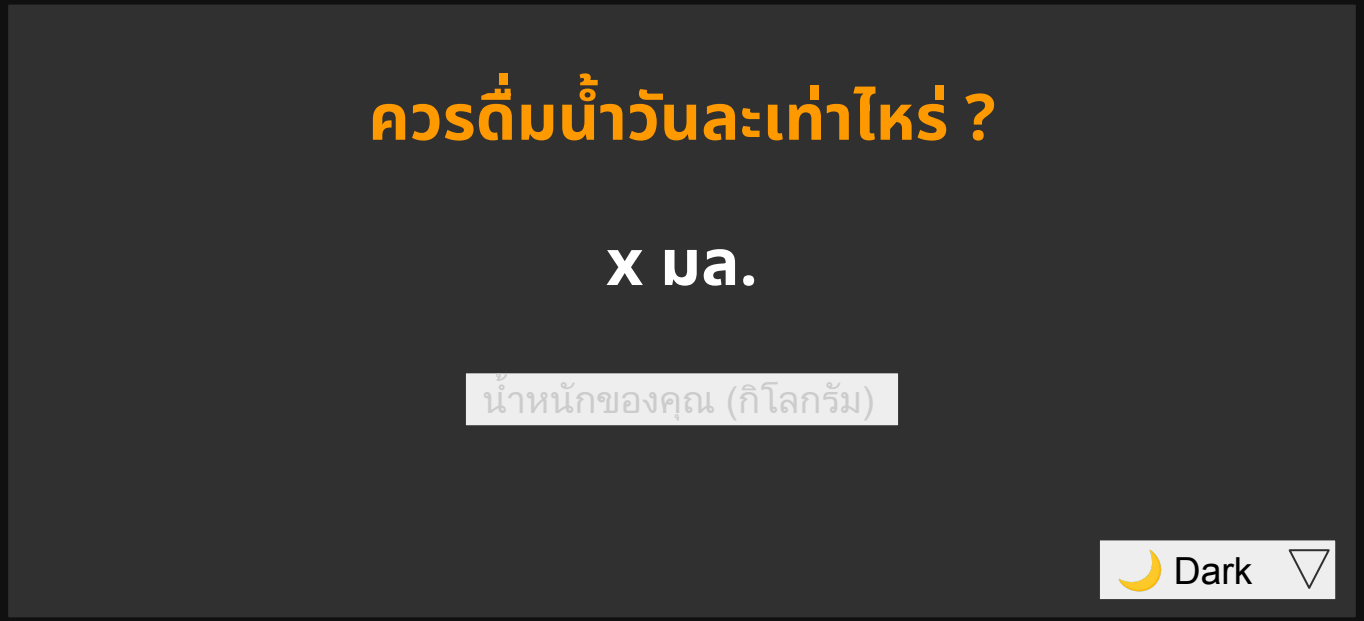
หน้าตาตัวอย่าง ผู้เรียนสามารถออกแบบใหม่ให้สวยขึ้นได้



ทำโปรแกรมคำนวณปริมาณน้ำที่ต้องดื่มต่อวัน
โดยคำนวณทันทีเมื่อกรอกข้อมูลลงไป และ สามารถสลับธีมได้

สูตร : น้ำหนัก x 2.2 x 30/2 = ปริมาณน้ำ(มล.)

ตอบใน “ส่งการบ้าน”



นำ Project Upload ขึ้น Repository ของ GitHub แล้วส่งลิงก์มาในส่งการบ้านได้เลย *อย่าลืมเปิด Public*

สูตร : น้ำหนัก x 2.2 x 30/2 = ปริมาณน้ำ(มล.)