**Session 1: React fundamental, JSX and Props**

**S1-E05: React Source Files**

نکات:

* ما در react با DOM سروکار نداریم و با Virtual DOM سرو کار داریم.
* ما در کل پروژه فقط و فقط یک فایل index.html داریم که همان DOM هست.
* داخل فولدر public، فونت ها، imageها و icon ها و موارد عمومی پروژه رو قرار میدیم.
* در ری اکت برای کال کردن فانکشن App به این صورت مینویسیم:

<App/>

**S1-E6: Separation of componenet**

App پدر همه ی کامپوننت هاست.

نکته:

* ما در یک فایل، میتونیم چندین export داشته باشیم و فقط یک export default میتونیم داشته باشیم
* ما 2 جور Import داریم:

1: import default

2:Named import که از {} استفاده میکنیم

**S1-E7: Component in react.js**

Component چیست؟

* کامپوننت ها ، بلاک ها یا اجزاء سازنده اپ ما هستند.
* شامل logic وdata و appearance آن قسمت از برنامه هستند.
* Reusable هستند و میتونن به صورت تو در تو استفاده بشن
* Component ها میتونن دیتا رو انتقال بدن

اگر parent-child باشن میتونن ولی اگر sibling باشن نمیتونن

* Componenet های ما یا Function هستند یا class
* بصورت pascal case نوشته میشن (ProductListOne)
* ما در React فراخوانی دیگه نداریم
* Component ها میتونن بهم دیتا پاس بدن

**S1-E8: CourseCard component markup**

jSX چیست؟

این مواردی که ما در کامپوننت ها return میکنیم jsx هست

ساختارش شبیه HTML هست.(JS and XML)

**S1-E9: Extract CourseList Component**

در این قسمت فقط یک قسمتی از کد را که میشد بصورت جداگانه درآورد، در یک کامپوننت مجزا نوشتیم.

**S1-E10: Render list in react.js**

در این قسمت یک تکه دیگر از کد را تبدیل به کامپوننتی به نام CouorseCard کردم.

**S1-E11: JSX in React.js**

React در پشت صحنه از Babel استفاده میکنه برای اینکه HTML ما رو در پشت صحنه به کد هایی حاوی

React.createElement() تبدیل کنه.

Declarative syntax چیست؟

توصیف میکنه که کامپوننت ما چجوریه، چجوری کار میکنه و چه شکل و شمایلی باید داشته باشه

این کامپوننتی که ما در نهایت یک JSX رو ریترن میکنه، UI و Logic برنامه رو مشخص میکنه

پس JSX یک extension هست که میتونیم توش HTML,CSS,JS بنویسیم.

Imperative چیست؟ Declarative چیست؟

In Declarative : What we want to do 

ما در React روش Declarative رو داریم.یعنی کد های ما به پیچیدگی vanilla JS نیست.

In Imperative: How to do things

ما در vanilla Js از روش imperative استفاده میکردیم.

یعنی دونه دونه المنت ها رو سلکت میکردیم،بعد به DOM،append میکردیم و...

Imperative کد هامون سخت و پیچیده میشه

نکته:

در حالتی که ما با React کار میکنیم ما هیچ وقت DOM رو تاچ نمیکنیم؛ یعنی از createElement و … استفاده نمیکنیم.

**S1-E12: Props in component**

Props ها

* دیتاهایی هستند که از کامپوننت پدر به فرزند منتقل میشه که بتونیم شکل و شمایل کامپوننت رو کنترل کنیم
* و در نهایت component ما بر اساس این بتونه دیتاهایی dynamic ای رو داخل خودش داشته باشه.
* مثل پارامتر هایی بود که ما به فانکشن ها پاس می دادیم.
* Props های ما میتونن هر نوع دیتایی باشن مثل Array ,function ,number ,object,other components
* بر اساس این props ها، کامپوننت ها میتونن توسط پدر، از نظر شکل و شمایل و عملکرد کنترل بشن
* Props ها یک سری آبجکت هستند

**S1-E13: Extract CourseCard component**

در این قسمت کامپوننت CourseCard که شامل دو قسمت CourseCardBody و CourseCardFooter بود را بصورت کامپوننت های جداگانه در خود فایل کامپوننت نوشتیم نه در یک فایل مجزا.

**S1-E14: Key in React.Js**

چرا Key ها در زمانی که استفاده میشه حتما باید یونیک باشه؟و چرا نباید از Index استفاده کرد؟

سناریویی رو در نظر بگیرید که روی یک سری آیتم map زدیم و طبیعتا هر کدام از محصولات باید یک key منحصر به فرد داشته باشند تا ری اکت بعدا یک سر نخ از اون محصولی که میخوایم روش یه کاری انجام بدیم (مثل ویرایش یا حذف و...) داشته باشه.

حالا قاعده کلی این هست که این key رو تا جایی که میشه باید id در نظر بگیریم نه index.

به این خاطر که اگر key مون رو index در نظر بگیریم با اتفاقاتی مثل حذف یکی از محصولات یا sort محصولات، شماره index ها تغییر میکنه و به این ترتیب کل محاسبات ما تغییر میکنه.

مگر اینکه جایی از Index استفاده کنیم که بدونیم جاش قرار نیست تغییر کنه.

**Session 2: state, Events and Form submission**

**S02-E02:Events In React Way**

وقتی داریم یک event مینویسیم باید بصورت callback function بنویسیم چرا که میخواهیم بعدا اجرا شود.

<button

  onMouseEnter={() => alert("Mouse Entered!")}

  key={tab.id}

  className={activetab === tab.id ? "active" : ""}

>

 <span>{tab.title}</span>

 <span className="tab-indicator"></span>

</button>

طبق یک قاعده کلی:

هر وقت خواستیم به handler هامون یک چیزی پاس بدیم، اون رو بصورت arrow function مینویسیم.

function App() {

  const activetab = 1;

  const handleActive = (id)  => {

   console.log("clicked!!" , id)

  }

  return (

    <div className="tab">

      <div className="tab\_\_header">

        {tabData.map((tab) => (

          <button

            onClick={() => handleActive(tab.id)}

            key={tab.id}

            className={activetab === tab.id ? "active" : ""}

          >

            <span>{tab.title}</span>

            <span className="tab-indicator"></span>

          </button>

        ))}

      </div>

   <div className="tab\_\_content">{tabData[activetab - 1].content} </div>

    </div>

  );

}

export default App;

در کد بالا، id رو از داخل Button (arrow func) پاس دادیم به handleActive.

**S02-E03:What is state in React.js ?**

کامپوننت های ما یک سری data در خودشون دارند که این data شامل state و props هست.

این دیتا ها در کنار logic ای که ما مینویسیم UI برنامه ما را می سازند.

نکته:

یعنی خروجی ما بر اساس state و props هست که کنترل میشه.

State و props چه تفاوت هایی باهم دارند؟

* State یک دیتای internal هست ولی props یک دیتای external هست چون از پدر به فرزند منتقل میشه.
* State ها حتما و حتما توسط خود اون کامپوننت (که صاحب اون state هست) باید update بشه ولی props ها توسط کامپوننت پدر update میشن

و میشه اینطوری گفت که props ها استیت کامپوننت پدر هستند و چون استیت یک کامپوننت دیگر هستند باید توسط خود اون کامپوننت آپدیت بشن.

* Props ها Immutable یا read only هستند و ما اجازه Update کردن اون ها رو نداریم ولی state هارو در داخل اون برنامه و داخل اون کامپوننت میتونیم آپدیت کنیم.

نکات:

* کامپوننت های ما فقط زمانی re-render میشن که یا state و یا props اون ها تغییر کنه.
* State ها دیتاهای internal اون کامپوننت هستند که که ما، بین render های متوالی بهش دسترسی داریم.
* State ها، مغز یا memory اون کامپوننت هستند که بین render های متوالی نیاز هست که به این دیتا دسترسی داشته باشیم که بتونیم اطلاعات state مون رو آپدیت کنیم.

**S02-E04:What is state in React.js ?**

چرا به state ها نیاز داریم؟

ما به state ها نیاز داریم چون :

* اطلاعات کامپوننتمون رو توش ذخیره میکنیم
* در render های متوالی بهش دسترسی داشته باشیم
* با آپدیت شدن state بتونیم re-render رو داشته باشیم
* و در نهایت، کامپوننت UI ای که داره رو Update میکنه

React reacts to state changes

نکته:

هوک ها رو در داخل nested function ها یا داخل condition ها یا داخل loop ها نمیتونیم بذاریم.

**S02-E05:Do not mutate state in react.js**

نکته:

ما اجازه نداریم به صورت دستی state های خودمون رو تغییر بدیم.باید از setter function(setState func) کمک بگیریم.

State ها اگر تغییر کنن منجر به re-render شدن و در نهایت منجر به تغییر UI میشن.

حالا واقعا چرا نباید mutate کرد؟

Obj ها reference type هستند.وقتی آبجکت ما (state) ما mutate میشه، رفرنس ما تغییر نمیکنه و وقتی ما یک پراپرتی اون رو تغییر میدیم ولی رفرنس اون آبجکت ثابت هست و همون رفرنس قبلی رو داره.

راه حل چیه؟

وقتی ما یک property از یک obj رو mutate میکنیم فقط داریم property هاش رو تغییر میدیم و بنابراین رفرنس تغییر نمیکنه.

برای اینکه Re-render اتفاق بیوفته تو بحث obj ها و Array ها ما باید یک Obj یا یک array جدید ایجاد کنیم که یک دیتای جدید با یک رفرنس جدید ساخته بشه.

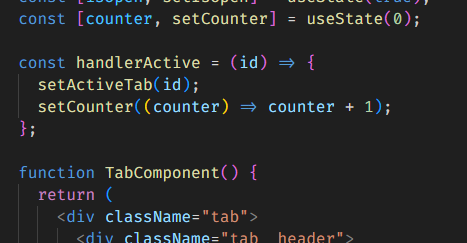
**S02-E06:Update state based on previous state in React.js**

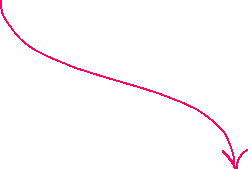
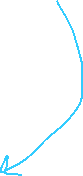
setState هایی که مینویسیم Async هستند. یعنی در آنِ واحد انجام نمیشن.

نکته:

وقتی مقدار state بعدی به state قبلی وابسته است، باید بیایم از روش callback method استفاده کنیم.



قلهمه جا میتونیم از callback func استفاده کنیم حتی جایی که به مقدار قبلیش وابسته نباشه.



اون counter ای که به عنوان آرگومان ورودی میگیره رو در داکیومنت ها با عنوان prevState میشناسند و این مقدار state قبلی ما هست.

مقدار state بعدی ما counter (با فلش آبی) هست.

**S02-E08:Note App project intro**

**S02-E11:Controlled component in React.js**

Controlled component ها چی هستند؟

Controlled component ها یا controlled element ها المنت هایی هستند که مقادیر اون ها توسط state ها مدیریت و کنترل میشه.

e.target اشاره میکنه به المنتی که اون event رو Trigger میکنه.(اون المنتی که اون event رو اجرا میکنه)

**Session 3: Thinking in React-State Management**

**S03-E02:State in React.js**

* **When** to create state
* **What** **type** of state is necessary?
* **Where** to place state
* **Data flow** in app

State ها به 2 مدل تقسیم بندی میشن:

* Local state

بعضی از state ها هستند که فقط در داخل اون compoenet ای که داریم ازش استفاده میکنیم قابل استفاده هستند.

یا compoenet خودش و یا نهایتا چند تا کامپوننت های زیرینش میتونن (با props) از state اش استفاده کنن. (خیلی محدود)

* Global state

یک سری از دیتاها هستند مثل دیتای سفارش های کاربر،اطلاعات کاربر و یا اطلاعات سبد خرید کاربر که ما اگر بین هر کدوم از صفحات application ، navigate کنیم و بچرخیم این اطلاعات رو میخوایم.

یک سری از state ها یا data ها بر اساس نیازمون در خیلی از کامپوننت ها باید بهش دسترسی داشته باشیم.

روش میدیریت state های گلوبال :

مفهومی که خود react در اختیار ما قرار داده به نام Context

یک library جانبی به نام Redux

و یا Reduc toolkit

نکته:

بصورت یک قانون سرانگشتی، ما همیشه با state های local شروع میکنیم به توسعه برنامه و هر جایی از برنامه که احساس کردیم اون state باید global باشه تدیلش میکنیم به Global. قرار نیست هر چیزی که دلمون خواست یک state گلوبال باشه

چه زمانی state ایجاد میکنیم؟

* اگر اون دیتایی که ما داریم در طول عمر پروژه تغییر کنه
* اگر اون دیتا رو نمیتونیم از state یا props های دیگه محاسبه اش کنیم.

(مثال: ما دو تا number داریم: number a و number b

حالا یک number c هم میخوایم از معدل این ها داشته باشیم.حالا چون میتونیم این عدد c رو از میانگین دو عدد دیگر بدست آوریم،پس دیگه state نیست و اصطلاحا derived state یا state مشتق شده هست.در ایجور مواقع نیازی به ایجاد state جدید نیست.

* این دیتا باید موجب Re-render شدن اون کامپوننت بشه و اصطلاحا UI ما Update بشه

**Where to place state?**

1. نسخه static پروژه رو ایجاد میکنیم
2. بر اساس نیازمون state جدید رو ایجاد میکنیم
3. حالا باید از خودمون سوال بپرسیم که آیا این state فقط و فقط در این component استفاده میشه؟
4. اگر جواب مثبت هست که داخل compoenet میذاریم.
5. اگر جواب خیر هست باید بپرسیم: توسط component های فرزند هم استفاده میشه؟
6. اگر جواب این سوال بله هست پس باید توسط props به فرزندانش هم پاس بدیم.
7. اگر پاسخ این سوال خیر هست باید از خودمون بپرسیم که آیا توسط component های sibling هم مورد استفاده قرار میگیره؟
8. اگر جواب فعلا بله هست ما باید state رو up کنیم به compoenent پدر **به این پروسه میگن Lifting state up**

**S03-E03:Lifting state up**

نکته:

به عنوان یک قانون سر انگشتی باید بدونیم که هر جایی از برنامه، وقتی یک دیتایی رو ، کامپوننت های siblings بهش نیاز داشتن باید اون state رو up کنیم به کامپوننت پدرش .

یعنی یه دیتایی رو در کامپوننت sibling داریم که کامپوننت sibling دیگه میخواد ازش استفاده کنه. باید چی کار کرد؟

* Lifting state up to their father

ما مجاز به انتقال sibling ها بهم نیستیم چون ما دیتا ها رو از پدر به فرزند انتقال میدیم.

**S03-E04:Update Lifted State in Parent**

نکته:

این props ها میتونن هر چیزی و از نوعی باشند. حتی ما میتونیم فانکشن هارو هم از props پاس بدیم.

نکته:

طبق یک قانون سرانگشتی (که البته گاهی هم اینطوری نیست!) میگه که هرجا state ها بودند همون جاstate رو update کنیم.

**S03-E05:Notelist Markup**

در این قسمت، پارت NoteList را ساختیم و NoteItem را هم نوشتیم.

**S03-E06:Deleting Note From Child Component**

نکته:

Filter , reduce , map , slice : این متدها mutate نمیکنه state رو و یک آرایه جدید با رفرنس جدید return میکنه

Shift , unshift , pop , push , splice این ها متد هایی هستند که نباید روی state ها استفاده بشن و state های ما رو mutate میکنن.

**S03-E07:Toggle Note to complete**

Select options , text inputs , radio , checkbox همه اینها onChange رو دارند.

**S03-E09: Sort Notes**

نکته:

Select options , radio inputs , check box , text inputs , check area

برای همه ی اینها onChange رو استفاده میکنیم.

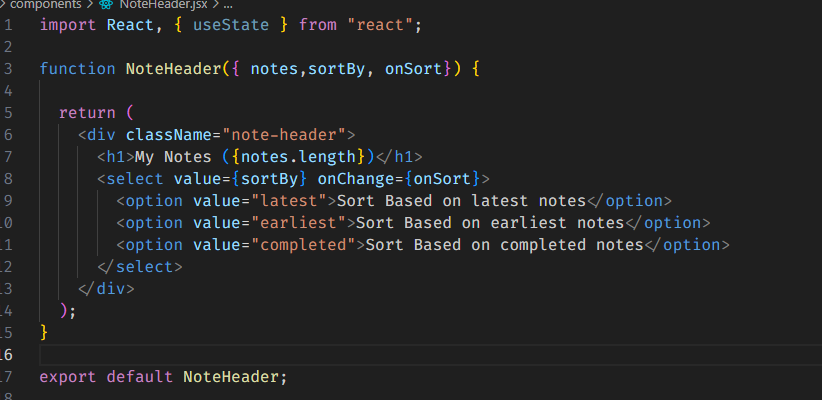
برای مواقعی که نیاز هست مقدار اون input رو ذخیره کنیم داخل برنامه میومدیم از approach controlled component استفاده میکردیم.

در controlled component ها 3 تا قدم داشتیم:

1. ایجاد یک state جدید
2. برای اون input یک value با مقدار state مینویسیم
3. نوشتن onChange

نکته:

هنگام نوشتن select option ها، داخل تگ Select باید Option ها رو بنویسیم. به option ها value میدیم و این Value قراره در event.target.value بهش دسترسی داشته باشیم.

چیزی هم که داخل option مینویسیم مقداری هست که در select input نمایش داده میشه.

نکته:

این نکته مهم هست که ما نباید روی state مون متدی بزنیم که mutate میکنه.اول باید بیایم یک clone از state ها بگیریم و بعد، از اون cloned state بیایم روش اون متد رو استفاده کنیم.

مثلا ما نمیتونیم از متد sort روی state ها استفاده کنیم چون اون رو Mutate میکنه.

یک راه استفاده از متد slice هست و یک راه دیگر استفاده از clone کردن هست با spread.

[…state] یا {…state}

با این کار یک shallow coppy از اون state میگیریم.

خلاصه جلسه:

در این جلسه، ما state رو نوشتیم ولی به دلیل اینکه کامپوننت sibling هم اون دیتا رو میخواست، این دیتا رو گذاشتیم داخل parent خودش یعنی App.

نکته دیگر این جلسه این بود که ما یک clone از state مون گرفتیم چون میخواستیم از متد sort استفاده کنیم و همچنین نمیخواستیم اون رو mutate کنیم.

**S03-E10: Children Props**

این یک مفهوم خیلی مهم در ری اکت هست.

وقتی کاربر وارد صفحه میشه یک سری پیغام هایی بهش نمایش داده میشه.به این پیعام ها میگن toast.

Childern Props چیست؟

بین کامپوننت ها ما میتونیم محتوایی رو بذاریم که بهش میگن children props. چطوری در یک کامپوننت دیگر بگیریمش؟

مثل props های دیگه با نام children حتما باید باشه و میتونیم destructure اش کنیم.

در بین این کامپوننت که یک دیتایی اضافه میکنیم (children)، کامپوننت میتونه Reusable و Dynamic باشه.

یکی از موارد استفاده اش در کامپوننت های layout هست که در واقع یک قسمت از برنامه ثابت هست ولی بینش تغییر میکنه در اینجا از children prop استفاده میکنیم.

**S03-E11:Accordion component V1**

A component is **controlled** when it’s managed by its parent using props

A component is **uncontrolled** when it’s managed by its own local state

نکته:

در این قسمت میایم هر کدوم از accordion item هارو در یک کامپوننت جداگانه از هم می نویسیم.چرا؟!

چون میخواهیم اینا مستقل از هم عمل کنن. مثلا ممکنه آکاردئون ما 4 تا آیتم داشته باشه.

ما روی اولی که میزنیم باز میشه ولی بقیه آیتم ها تغییری نمیکنن؛پس نتیجه میگیریم که **هر کدام state خودشونو دارن**.

پس ما میایم AccordionItem رو یک کامپوننت مجزا در نظر میگیریم.

const data = [

  {

    id: 1,

    title: "Accordion One",

    text: " Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus excepturi iure corrupti repellendus beatae, praesentium exercitationem ratione hic quod minima, eaque dignissimos placeat consequuntur est quisquam accusamus temporibus? Molestias, aperiam!",

  },

  {

    id: 2,

    title: "Accordion Two",

    text: " Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus excepturi iure corrupti repellendus beatae, praesentium exercitationem ratione hic quod minima, eaque dignissimos placeat consequuntur est quisquam accusamus temporibus? Molestias, aperiam!",

  },

  {

    id: 3,

    title: "Accordion Three",

    text: " Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus excepturi iure corrupti repellendus beatae, praesentium exercitationem ratione hic quod minima, eaque dignissimos placeat consequuntur est quisquam accusamus temporibus? Molestias, aperiam!",

  },

];

function Accordion() {

  return (

    <div className="accordion">

      {data.map((item) => (

        <AccordionItem key={item.id} item={item} />

      ))}

    </div>

  );

}

export default Accordion;

const AccordionItem = ({ item }) => {

  const [isOpen, setIsOpen] = useState("false");

  return (

    <div className={`accordion-item ${isOpen ? "accordion\_\_expanded" : ""}`}>

      <div className="accordion-item\_\_header" onClick={() => setIsOpen(!isOpen)}>{item.title}</div>

      {isOpen && <div className="accordion-item\_\_content">{item.text}</div>}

    </div>

  );

};

برای ساخت state جدید این سوالات رو از خودمون باید بپرسیم:

1. آیا دیتایی که نیاز داریم از state ها یا props های دیگه ساخته میشه؟
2. آیا این دیتا در طول عمر برنامه تغییر میکنه؟
3. آیا تغییر این state یا این دیتا در نهایت موجب تغییر UI اپلیکیشن ما میشه؟

اگر جوابمون به هر 3 سوال بالا "بله" بود نتیجه میگیریم که باید یک state جدید ایجاد کنیم.

Template literal:

<div className={`accordion-item ${isOpen ? "accordion\_\_expanded" : ""}`}></div>

توضیح کلاس CSS :

.accordion-item.accordion\_\_expanded .accordion-item\_\_content {

  padding: 1rem;

  opacity: 1;

  max-height: 100vh;

  transition: all 0.2s ease-in-out;

}

یعنی آیتمی که کلاس های accordion-item و کلاس accordion\_\_expanded رو هم زمان باهم داره، داخلش یک آیتمی هست با کلاس accordion-item\_\_content .به اون این استایل هارو بده.

**S03-E12:Dynamic Style in chevron**

استایل دهی به کامپوننت

روش اول: در این قسمت، استایل دهی به یک کامپوننت را بصورت inline انجام دادیم:

        <ChevronDownIcon

          style={{

            width: "1.2rem",

            transition: "all 0.2s ease-out",

            rotate: isOpen ? "180deg" : "0deg",

          }}

        />

2 تا {} تودرتو میذاریم.اولیش برای نوشتن JS هست و دومیش برای نوشتن Obj استایل ها در داخل js.

روش دوم: این هست که بر اساس true false شدن استیت رو تغییر میدیم

روش سوم: این هست که یک کلاس رو به لیست کلاس های اون element اضافه میکنیم و بر این اساس میگیم هر وقت این کلاس رو گرفت، فلان استایلش رو تغییر بده.

**S03-E13:Accordion Compoenet V2**

ما در این قسمت از lift state up استفاده کردیم تا به state اش در کامپوننت های دیگر دسترسی داشته باشیم. یعنی استیت رو به کامپوننت پدرش انتقال دادیم.

در جاهایی که محتوای داخلی اون کامپوننت تغییر میکنه، گزینه مناسب این هست که از children prop ها استفاده کنیم.

**S04-E02:How to split component**

When to create new component?

Hint: Component is a function that handle one task!

چند تا سوال هست که باید بپرسیم از خودمون:

**Logical** **Seperation**

1. آیا این کامپوننت لاجیک غیر مرتبط دیگه ای رو داخل خودش داره؟ content غیر مرتبط دیگه ای رو داره؟ اگر داره احتمالا میشه جدا نوشتش

**Reusability**

1. آیا امکان این هست که در جای دیگر از آن استفاده کنیم؟

آیا این logic ای که ما داریم نیاز هست که در جای دیگر از آن استفاده شود؟ اگر بله، پس میشه به دلیل reusability بودن اون رو جدا کنیم.

**Complexity**

1. آیا این component تعداد تسک های زیادی رو انجام میده؟

اگر بله پس ...

آیا این compoenent تعداد state و یا props های زیادی دارد؟

Jsx طولانی و یا handler های زیادی داره؟ اگر اینطوری هست پس باید کامپوننت های دیگری را از دل این کامپوننت ایجاد کنیم.

**Personal Preferences**

1. گاهی هم بر اساس ترجیح شخصی تمایل داریم کامپوننت ما به این شکل باشه و مثلا عادت کردم به کامپوننت های کوچک یا حتی برعکس؛ به کامپوننت های بزرگ!

**S04-E03: characterList and Navbar component**

در این قسمت یک سری از کامپوننت ها رو توسعه دادیم و UI رو نوشتیم.

**S04-E04: Character Detail Component**

در این قسمت Detail Componentرا توسعه دادیم.

**S04-E05: Component Categories**

در این قسمت componenet ها رو به 3 دسته تقسیم میکنیم:

1. کامپوننت های بدون state
2. کامپوننت های state دار
3. کامپوننت های container یا structural component

که این دسته همان کامپوننت هایی هستند که کامپوننت های کوچک تر را در خود جای می دهند. و یه جوایی layout هم میگن بهشون.یعنی همون قسمت هایی که وقتی navigate میکنیم بین صفحات مختلف ، ثابت هستن مثل sidebar کنار صفحه.

**S04-E06:Props drilling**

در این قسمت به مشکلی برخورد کردیم که راه حل آن را به قسمت بعدی می سپاریم.

Props drilling

مشکل این بود که برای کامپوننتی که گاها 6 مرتبه پایین تر از جایی بود که state در آن قرار داشت، باید مرحله به مرحله و کامپوننت به کامپوننت props میفرستادیم. حتی بعضی از کامپوونت ها به آن دیتا نیازی نداشتند ولی برای اینکه آن دیتا به کامپوننت مد نظر برسد مجبور شدیم این کار را انجام دهیم که به آن props drilling میگویند.

**S05-E01: Overview**

در این فصل میخواهیم به این پرسش ها پاسخ دهیم.

* How React re-renders components?
* State update batching
* Pure Component
* Rules of render logic

**S05-E02: How React re-render components?**

ری اکت چطوریre-render شدن و در نهایت آپیدت uI رو برامون انجام میده؟

وقتی state یا props یک کامپوننت تغییر پیدا کنه،اون کامپوننت

re-render میشه و اگر یه کامپوننتی Re-render بشه children های اون کامپوننت هم re-render میشن

Diffing process

❌ما اجازه touch کردن DOM یا دسترسی به DOM Api هارو در داخل برنامه نداریم و در React هم این کار رو نمی کنیم.

✔ اما solution ای که react در اختیار ما میذاره استفاده از VDOM یا همون DOM مجازی هست.

پکیج React DOM ای که استفاده میکردیم نقطه اتصالی بود بین DOM و VDOM.

ری اکت یک کپی از DOM رو در داخل VDOM ذخیره میکنه.زمانی که یک state یا یک props ای تغییر میکنه ری اکت میاد اون دیتایی که الان در VDOM آپدیت میکنه رو مقایسه میکنه با ورژن قبلی VDOM که همان ورژن کپی شده از DOM واقعی هست.پس زمانی که ما یک state رو تغییر میدیم یک VDOM جدید رو داریم و این VDOM با اون VDOM (که یک کپی از DOM واقعی ما هست) مقایسه میشه. اینجا react در فاز diffing کاری که انجام میده این هست که VDOM جدید رو با VDOM قبلی مقایسه میکنه و اون قسمتی که تغییر کرده رو re-render میکنه و پس از re-render شدن، ما DOM جدید رو داریم و دقیقا داخل DOM ما اون پارت هایی عوض میشن که تغییر کردن.

برای خلاصه:

VDOM یک کپی از DOM هست که در هر لحظه که state یا props آپدیت میشه، ورژن الان VDOM رو با ورژن قبلی مقایسه میکنه و فقط اون کامپوننت هایی که Re-render میشن در داخل DOM اصلی ما تغییر پیدا میکنن و بقیه کامپوننت ها که در re-rendering نقشی نداشتند، دست نخورده باقی می مانند که این به شدت performance برنامه ما رو بیشتر میکنه.

Visualize Re-rendering

1- React Dev tools extension

* Component -> view settings -> highlight updated when component render

2- What change on DOM

* Three dots -> more tools -> rendering -> paint flashing

نکته:



گهگاه ممکنه یک children از اون state یا props استفاده نکرده باشه ولی با این حال react به این نکته دقت نمیکنه و بچه های اون کامپوننت رو هم Re-render میکنه.

**S05-E03:Purecomponents**

Pure function چیست؟

Function ای است که **همیشه** به ازای داشتن input های مشخص، Output های مشخصی داشته باشد و هیچ گونه side Effect ای نداشته باشد.

Impure functions چیست؟

Function ای است که برعکس حالت قبلی، با داشتن input های یکسان، output های متفاوتی خواهیم داشت. این فانکشن ها به عوامل بیرونی وابسته هستند مثل متغیر های public یا اینکه به نتایج فانکش های دیگر وابستگی دارند که میتونه رفتار و خروجی اون هارو تغییر بده.

Pure component

به خاطر بهینه سازی استفاده میشه.

Re-render هایی که state ها یا props های اون component مشابه باشه و تغییر نکنه اون رندر ها رو نادیده میگیره.

به یاد داریم که Re-render شدن یک کامپوننت زمانی اتفاق میوفته که state یا props اون component تغییر کنه؛ پس اگر زمانی state یا props تغییر نکرد خیلی بهتره که ما اون Compoenet رو re-render نکنیم.

حالا در pure component هم همینو داریم که اگر state یا props ما در render های متوالی تغییر نکرد و state و props هم چنان مشابه بود پس ما re-render این کامپوننت رو نادیده بگیریم و از این re-render شدن اضافی جلوگیری کنیم.

مثلا: کامپوننت A در طی re—rendering استیت یا پراپسش تغییر میکنه.حالا این کامپوننت یک سری children هم داره که اون ها هم متعاقبا re-render میشن ولی خب در اونها تغییر state یا props رو نداریم. راه حل برای کامپوننت های children در اینجا چیه؟

استفاده از Pure Compoenent😎

Pure compoenet هارو در class component ها استفاده میکنیم.

<https://dev.to/codeofrelevancy/you-need-to-know-about-pure-functions-impure-functions-in-javascript-57>

ولی در function Compoenet ها از memo استفاده میکنیم.

Memo چیست؟

Memo یک فانکشن هست که در ورودی یک فانکشن میگیره و روی بهینه سازیش کار میکنه.یعنی چی؟!

یعنی نگاه میکنه که اگر اون compoenent ،state یا props اش تغییر نکرده بود تشخیص میده که نیازی به re-render شدن نداره و از این عمل خودداری میکنه.

<https://react.dev/reference/react/memo>

A react compoenet should always have **Pure rendering Logic**.

**S05-E04: Rules Of Render Logic**

اساس rect بر این استوار است که react فرض میکند تمام compoenent ها pure هستند؛ ولی همیشه اینطو نیست و کلا خیلی اوقات پیش میاد که کامپوننتمون Pure نباشه.

Render logic

کد هایی هست که در کف component مینویسیم.مثل state یا jsx ای که مینویسیم در فانکشن کامپوننت.هر زمان که اون کامپوننت Render میشه اون خط کد ها return و اجرا میشن.

ما در event handler function هامون اجازه side effect کردن داریم.

نکته:

چرا ما از side effect در render logicاستفاده نمیکنیم؟

جدا از بحث اینکه نباید mutate کنیم بحث غیر قابل پیش بینی بودن کامپوننت هست.

ما چه زمانی و کجا side effect هارو ایجاد میکنیم؟

Side effect ها در Render logic انجام **نمیشن** .در

function event handler ها

و یک hook ای که React بهمون معرفی کرده به نام UseEffect.

Side effect ها چی هستند؟

هر گونه تغییرات جانبی هستند. مثل :

* Network request ها یا API هایی که میفرستیم سمت بک اند.
* مثل یک timer
* مثل دسترسی به DOM API (مثل querySelector و ...)
* یا مثل mutate کردن یک obj یا muate کردن یک state
* Updating the sreen
* Starting an animation
* Changing data

⛔ این کار هارو در داخل Render logic نباید انجام بدیم.

Do not mutate obj or variables outside the function scope!

**S05-E05: State update Batching**

**Queueing a Series of State Updates**

batch : دسته ای

در این مورد ما مفهوم رو داریم که state update batching،

یا همان Update گروهی state ها که میتونیم چندین state رو در داخل یک صف آپدیت کنیم.

setState ها بصورت گروهی update میشن

React موقعی re-render میکنه که تمام کد های event handler ما خوانده بشه و کارشون تموم بشه و در اینجاست که صفحه رو Re-render میکنه. و فرض کنیم که چندین setState رو در یک eventhandler داریم؛همه ی اینهارو صف میکنه و در آخر این هارو اجرا میکنه.

import { useState } from 'react';

export default function Counter() {

  const [number, setNumber] = useState(0);

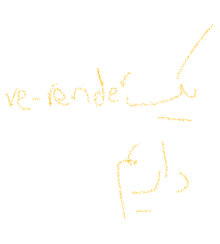
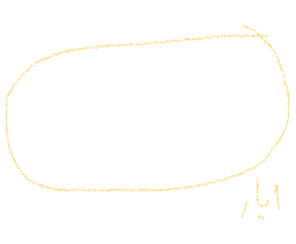
  return (

    <>

      <h1>{number}</h1>

      <button onClick={() => {

        setNumber(number + 1);

        setNumber(number + 1);

        setNumber(number + 1);

      }}>+3</button>

    </>

  )

مثلا اینجا داخل button ما چند بار setState نوشتیم، این کد ها تا انتها خونده میشن و نهایتا re-render **یکبار** انجام میشه.

در اینجا React مثل یک waiter عمل میکنه یعنی ؛ کل سفارش ها رو از مشتری میگیره و به یکباره به آشپزخانه میره. و به این صورت Update گروهی state هارو داره.

فواید Queue batching :

1. این باعث میشه که Performance برنامه بالاتر بره. و سرعت بالاتر
2. و جلوگیری میکنه از ناسازگاری هایی که داخل کامپوننت هست

نکته:

اگر یک state رو خواستیم چندین بار Updtate کنیم و به اصطلاح به state قبلی خودش وابسته بود،پیشنهاد ری اکت استفاده از callback هست.

  import { useState } from 'react';

export default function Counter() {

  const [number, setNumber] = useState(0);

  return (

    <>

      <h1>{number}</h1>

      <button onClick={() => {

        setNumber(n => n + 1);

        setNumber(n => n + 1);

        setNumber(n => n + 1);

      }}>+3</button>

    </>

  )

}

**S06**:Component Life cycle, Side effect and Data fetching

**S06-E01:Overview**

در این فصل به مفاهیمی مثل :

* Component life cycle
* Data fetching
* useEffect hook

می پردازیم.

**S06-E02: Component Life Cycle**

ما سه تا فاز داریم در ری اکت:

1. Mount

* Component rendered for the **first time**
* **Initial state** and **props** created

1. Re-render (updating)

* **State** or **Props** changes
* **Parent** component re-rendered

1. Unmount

* The component is **destroyed** and **removed**
* State and Props are **destroyed**

**S06-E03: How to NOT fetch data in React.js?**

نکته:

ما در render logic نباید setState کنیم.

ما هیچ وقتی نباید بیایم fetch data رو در کف کامپوننت (render logic) بنویسیم؛ چرا که این کار موجب re-render های زیاد و در نتیجه خطا میشه.

Fetch API (back-end requests) , timer , access to DOM , …

همه اینها جزو effect های ما هست و اینهارو نباید در render logic نویسیم.

2 تا راه حل حل داریم:

1. در event handler هامون انجام بدیم
2. با استفاده از یک هوک به نام useEffect

**S06-E04: UseEffect Intro**

useEffect( , ) یک فانکشن هست که return نداره.

2 تا آرگومان میگیره:

آرگومان اول یک تابع هست

آرگومان دوم یک dependency هست.

مثالی از useEffect

useEffect(() => {

  fetch("https://rickandmortyapi.com/api/character")

    .then((res) => res.json())

    .then((res) => setcharacter(res.results.slice(0, 5)));

}, []);

نکته:

کامپوننت ها میتونن چندین useEffect داشته باشند.

چرا باید از useEffect استفاده کنیم؟

گفتیم که برای نوشتن effect ها میشود از event hander ها هم استفاده کرد.پس چرا از useEffect استفاده میکنیم؟

useEffect برای موقعی هست که وقتی برنامه برای اولین دفعه load میشه اون کار خاص رو انجام بده؛ مثل fetch کردن data و نشون دادنش در صفحه.

ولی وقتی ما یک event handler می نویسیم (مثلا برای onclick روی یک button)، اجرا شدن اون event handler وابسته میشه به کلیک کردن روی اون button ؛ و مسئله اینجاست که ما گاهی این وابسته بودن رو نمیخوایم و میخوایم در اولین لود صفحه، اون دیتا برامون نمایش داده بشن.

پس میایم و از useEffect استفاده میکنیم.

اگر bracket dependecy رو حذف کنیم از UseEffect چه اتفاقی میوفته:

وجود [] در آرگومان دوم UseEffect به این معناست که این فانکشن رو (آرگومان اول) رو در فاز mount اجرا کن.

و اگر حذف بشه تعداد Request هایی که App مون میفرسته سمت بک اند خیلی زیاد میشه.