**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.

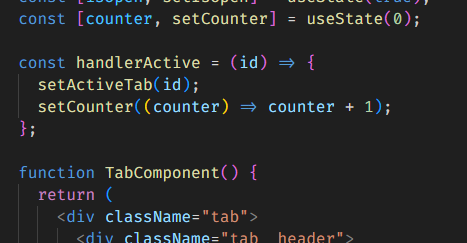
**S02-E06:Update state based on previous state in React.js**

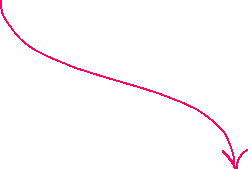
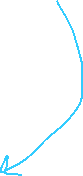
setState هایی که مینویسیم Async هستند. یعنی در آنِ واحد انجام نمیشن.

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.