

به نام خدا

مجموعه سوالات استخدامی ریاكت React.js

نویسنده: Sudheer Jonna
مترجم: جعفر رضایی و تیم ماریوتک

مجموعه سوالات استخدامی ری اکت

اگه خوشتون اومد به گیت‌ها بمون مراجعه کنین و بهمون 🌟 بدین. اگر هم قصد مشارکت داشتید خیلی خوشحال می‌شیم 😊

دانلود کتاب به فرمت‌های PDF/Epub

می‌تونید از بخش ریلیزهای گیت هاب دانلود کنین ([این لینک](#)).

فهرست

ردیف	سوال
	هسته ری اکت
۱	ری اکت چیه؟
۲	اصلی‌ترین ویژگی‌های ری اکت چیا هستن؟
۳	JSX چیه؟
۴	تفاوت‌های Element و Component چیه؟
۵	تو ری اکت چطوری کامپوننت می‌سازیم؟
۶	کی باید از Class Component بجای Function Component استفاده کنیم؟
۷	Pure Components چیه؟
۸	state تو ری اکت چیکار می‌کنه؟
۹	props تو ری اکت چیکار می‌کنه؟

ردیف	سوال
۱۰	تفاوت state و props چیه؟
۱۱	چرا نباید state رو مستقیماً آپدیت کنیم؟
۱۲	هدف از متدهای callback توی استفاده از setState چیه؟
۱۳	تفاوت بین نحوه مدیریت رویداد HTML و React چیه؟
۱۴	چطوری متد یا event رو به تابع callback توی JSX bind کنیم؟
۱۵	چطوری میشه یک مقدار رو به یه تابع callback یا eventHandler پاس بدیم؟
۱۶	Synthetic events (رویدادهای مصنوعی) تو ری اکت چیا هستن؟
۱۷	عبارات شرطی درون خطی چیه؟
۱۸	props های "key" چی هستن و مزایای استفاده از آنها در آرایه عناصر چیه؟
۱۹	کاربرد ref ها چیه؟
۲۰	چطوری از ref استفاده کنیم؟
۲۱	forward ref چیه؟
۲۲	بین callback refs و تابع findDOMNode کدوم رو ترجیح میدی؟
۲۳	چرا Ref های متنی منقضی محسوب می شوند؟
۲۴	Virtual DOM چیه؟
۲۵	Virtual DOM چطوری کار می کنه؟
۲۶	تفاوت بین Shadow DOM و Virtual DOM چیه؟
۲۷	React Fiber چیه؟
۲۸	هدف اصلی React Fiber چیه؟
۲۹	کامپوننت های کنترل شده چی هستن؟
۳۰	کامپوننت های کنترل نشده چی هستن؟

ردیف	سوال
۳۱	تفاوت‌های بین createElement و cloneElement چیا هستن؟
۳۲	مفهوم lift state up یا مدیریت state در لول بالاتر رو توضیح میدی؟
۳۳	فازهای مختلف از lifecycle کامپوننت چیا هستن؟
۳۴	متدهای lifecycle کامپوننت چیا هستن؟
۳۵	کامپوننت‌های Higher-Order چی هستن؟
۳۶	چطوری می‌تونیم props proxy برای کامپوننت‌های HOC ایجاد کنیم؟
۳۷	context چیه؟
۳۸	children prop چیه؟
۳۹	چطوری میشه تو React کامنت نوشت؟
۴۰	چرا توی کامپوننت‌های کلاس باید توی constructor تابع super رو با مقدار props صدا بزنینم؟
۴۱	reconciliation چیه؟
۴۲	چطوری با یه اسم داینامیک set state کنیم؟
۴۳	یه اشتباه رایج برای مدیریت توابع eventها که باعث میشه با هر رندر توابع مجدد ساخته بشن چی هستش؟
۴۴	تابع lazy که برای lazy load استفاده میشه رو می‌تونیم به صورت name export خروجی بگیریم؟
۴۵	چرا ری‌اکت از className بجای class استفاده می‌کنه؟
۴۶	fragmentها چی هستن؟
۴۷	چرا fragmentها از تگ‌های div بهترن؟
۴۸	توی ری‌اکت portalها چیکار می‌کنن؟
۴۹	کامپوننت stateless چیه؟
۵۰	کامپوننت stateful چیه؟

ردیف	سوال
۵۱	چطوری prop های کامپوننت رو اعتبارسنجی کنیم؟
۵۲	مزایای React چیه؟
۵۳	محدودیت های React چیه؟
۵۴	error boundary ها توی ری اکت نسخه 16 چیکار می کنن؟
۵۵	چطوری از error boundary ها توی نسخه ۱۵ ریکت مدیریت شدن؟
۵۶	روش های پیشنهادی برای type checking چیه؟
۵۷	کاربرد پکیج react-dom چیه؟
۵۸	کاربرد متد render از پکیج react-dom چیه؟
۵۹	ReactDOMServer چیه؟
۶۰	چطوری از InnerHtml توی ری اکت استفاده کنیم؟
۶۱	چطوری توی ری اکت استایل دهی می کنیم؟
۶۲	تفاوت event های ری اکت چیه؟
۶۳	اگه توی constructor بیاییم و setState کنیم چی میشه؟
۶۴	تاثیر استفاده از ایندکس به عنوان key چیه؟
۶۵	نظرت راجع به استفاده از setState توی متد componentWillMount چیه؟
۶۶	اگه از prop توی مقداردهی اولیه state استفاده کنیم چی میشه؟
۶۷	چطوری کامپوننت رو با بررسی یه شرط رندر می کنیم؟
۶۸	چرا وقتی prop ها رو روی یه DOM Elemnt می آیم spread می کنیم باید مراقب باشیم؟
۶۹	چطوری از decorator ها توی ری اکت استفاده کنیم؟
۷۰	چطوری یه کامپوننت رو memoize می کنیم؟
۷۱	چطوری باید Server-Side Rendering یا SSR رو توی ری اکت پیاده کنیم؟

ردیف	سوال
۷۲	چطوری حالت production رو برای ری اکت فعال کنیم؟
۷۳	CRA چیه و چه مزایایی داره؟
۷۴	ترتیب اجرا شدن متدهای life cycle چطوره؟
۷۵	کدوم متدهای life cycle توی نسخه 16 ری اکت منسوخ شدن؟
۷۶	کاربرد متد getDerivedStateFromProps چیه؟
۷۷	کاربرد متد getSnapshotBeforeUpdate چیه؟
۷۸	آیا هوک‌ها جای render props و HOC رو می‌گیرن؟
۷۹	روش توصیه شده برای نام‌گذاری کامپوننت‌ها چیه؟
۸۰	روش توصیه شده برای ترتیب متدها در کلاس کامپوننت‌ها چیه؟
۸۱	کامپوننت تعویض کننده یا switching چیه؟
۸۲	چرا نیاز میشه به تابع setState یه فانکشن callback پاس بدیم؟
۸۳	حالت strict توی ری اکت چیکار می‌کنه؟
۸۴	Mixin های ری اکت چی هستن؟
۸۵	چرا isMounted آنتی پترن هست و روش بهتر انجامش چیه؟
۸۶	پشتیبانی ری اکت از pointer event چطوره؟
۸۷	چرا باید اسم کامپوننت با حرف بزرگ شروع بشه؟
۸۸	آیا prop های custom توی ری اکت پشتیبانی میشن؟
۸۹	تفاوت های constructor و getInitialState چیه؟
۹۰	می‌تونیم یه کامپوننت رو بدون setState ری‌رندر کنیم؟
۹۱	تفاوت های فراخوانی super (-) و super(props) توی کلاس کامپوننت های ری اکت چیه؟
۹۲	چطوری توی JSX حلقه یا همون لوپ رو داشته باشیم؟

ردیف	سوال
۹۳	توی attribute ها چطوری به prop دسترسی داشته باشیم؟
۹۴	چطوری یه PropTypes برای آرایه ای از object ها با shape داشته باشیم؟
۹۵	چطوری class های یه المنت رو به صورت شرطی رندر کنیم؟
۹۶	تفاوت های React و ReactDOM چیه؟
۹۷	چرا ReactDOM رو از React جدا کردن؟
۹۸	چطوری از label تو ری اکت استفاده کنیم؟
۹۹	چطوری می تونیم چندتا object از استایل های درون خطی رو با هم ترکیب کنیم؟
۱۰۰	چطوری با resize شدن مرورگر یه ویو رو ری رندر کنیم؟
۱۰۱	تفاوت متدهای setState و replaceState چیه؟
۱۰۲	چطوری به تغییرات state گوش بدیم؟
۱۰۳	روش توصیه شده برای حذف یک عنصر از آرایه توی state چیه؟
۱۰۴	امکانش هست که ری اکت رو بدون رندر کردن HTML استفاده کنیم؟
۱۰۵	چطوری میشه با ری اکت یه JSON به شکل beautify شده نشون داد؟
۱۰۶	چرا نمی تونیم prop رو آپدیت کنیم؟
۱۰۷	چطوری می تونیم موقع لود صفحه روی یه input فوکوس کنیم؟
۱۰۸	روش های ممکن برای آپدیت کردن object توی state چیا هستن؟
۱۰۹	چرا توابع به جای object در setState ترجیح داده می شوند؟
۱۱۰	چطوری می تونیم نسخه ری اکت جاری رو توی محیط اجرایی بفهمیم؟
۱۱۱	روش های لود کردن polyfill توی CRA چیا هستن؟
۱۱۲	توی CRA چطوری از https به جای http استفاده کنیم؟
۱۱۳	توی CRA چطوری میشه از مسیرهای طولانی برای ایمپورت جلوگیری کرد؟

ردیف	سوال
۱۱۴	چطوری میشه Google Analytics رو به react-router اضافه کرد؟
۱۱۵	چطوری یه کامپوننت رو هر ثانیه به روز کنیم؟
۱۱۶	برای استایل‌دهی‌های درون خطی چطوری باید پیشوندهای مخصوص مرورگرها رو اضافه کرد؟
۱۱۷	چطوری کامپوننت‌های ری‌اکت رو با es6 می‌تونیم import و export کنیم؟
۱۱۸	استثنایی که برای نام‌گذاری کامپوننت اجازه استفاده از حرف کوچک رو میده چیه؟
۱۱۹	چرا تابع سازنده کلاس کامپوننت یکبار صدا زده میشه؟
۱۲۰	توی ری‌اکت چطوری مقدار ثابت تعریف کنیم؟
۱۲۱	چطوری توی برنامه event کلیک شدن رو trigger کنیم؟
۱۲۲	آیا استفاده از async/await توی ری‌اکت ممکنه؟
۱۲۳	ساختار پوشه‌بندی معروف برا ری‌اکت چطوره؟
۱۲۴	پکیج‌های مشهور برای انیمیشن چیا هستن؟
۱۲۵	مزایای ماژول‌های style چیه؟
۱۲۶	معروف‌ترین lint‌های ری‌اکت کدوما هستن؟
۱۲۷	چطوری باید توی کامپوننت درخواست api call بزنیم؟
۱۲۸	render props چیه؟
	رووتر ری‌اکت
۱۲۹	React Router چیه؟
۱۳۰	ارتباط React Router و کتابخونه history چیه؟
۱۳۱	کامپوننت‌های router توی نسخه ۴ چیا هستن؟
۱۳۲	هدف از متدهای push و replace توی history چیه؟
۱۳۳	چطوری توی برنامه به route خاص جابجا بشیم؟

ردیف	سوال
۱۳۴	چطوری میشه query پارامترها رو توی ری اکت روتر نسخه ۴ گرفت؟
۱۳۵	دلیل خطای "Router may have only one child element" چیه؟
۱۳۶	چطوری میشه به متد history.push پارامتر اضافه کرد؟
۱۳۷	چطوری میشه صفحه ۴۰۴ ساخت؟
۱۳۸	توی ری اکت روتر نسخه ۴ چطوری میشه history رو گرفت؟
۱۳۹	چطوری بعد از لاگین به شکل خودکار ریدایرکت کنیم؟
	چند زبانگی در ری اکت
۱۴۰	React-Intl چیه؟
۱۴۱	اصلی ترین ویژگی های React Intl چیا هستن؟
۱۴۲	دو روش فرمت کردن توی React Intl چیا هستن؟
۱۴۳	چطوری از FormattedMessage به عنوان یه placeholder میشه استفاده کرد؟
۱۴۴	چطوری میشه locale فعلی رو توی React Intl بدست آورد؟
۱۴۵	چطوری با استفاده از React Intl یه تاریخ رو فرمت بندی کنیم؟
	تست کردن ری اکت
۱۴۶	توی تست ری اکت Shallow Renderer چیه؟
۱۴۷	پکیج TestRenderer توی ری اکت چیه؟
۱۴۸	هدف از پکیج ReactTestUtils چیه؟
۱۴۹	Jest چیه؟
۱۵۰	مزایای jest نسبت به jasmine چیا هستن؟
۱۵۱	یه مثال ساده از تست با jest بزن؟
	React Redux

ردیف	سوال
۱۵۲	Flux چیه؟
۱۵۳	Redux چیه؟
۱۵۴	مبانی اصلی ریداکس چیا هستن؟
۱۵۵	کاستی‌های redux نسبت به flux چیا هستن؟
۱۵۶	تفاوت‌های mapDispatchToProps و mapStateToProps چی هست؟
۱۵۷	توی ریدیوسر می‌تونیم یه action رو dispatch کنیم؟
۱۵۸	چطوری میشه خارج از کامپوننت میشه store ریداکس دسترسی داشت؟
۱۵۹	اشکالات پترن MVW چیا هستن؟
۱۶۰	تشابهی بین Redux و RxJS هست؟
۱۶۱	چطوری میشه یه اکشن رو موقع لود dispatch کرد؟
۱۶۲	چطوری از متد connect از پکیج react-redux استفاده می‌کنیم؟
۱۶۳	چطوری میشه state ریداکس رو ریست کرد؟
۱۶۴	هدف از کاراکتر @ توی decorator متد connect چیه؟
۱۶۵	تفاوت‌های context و React Redux چیه؟
۱۶۶	چرا به توابع state ریداکس reducer میگن؟
۱۶۷	توی redux چطوری میشه api request زد؟
۱۶۸	آیا لازمه همه state همه کامپوننت‌هامونو توی ریداکس نگهداری کنیم؟
۱۶۹	روش صحیح برای دسترسی به store ریداکس چیه؟
۱۷۰	تفاوت‌های component و container توی ریداکس چی هست؟
۱۷۱	هدف از constant ها تا typeها توی ریداکس چیه؟
۱۷۲	روش‌های مختلف برای نوشتن mapDispatchToProps چیه؟

ردیف	سوال
۱۷۳	کاربرد پارامتر ownProps توی mapStateToProps و mapDispatchToProps چیه؟
۱۷۴	ساختار پوشه‌بندی ریشه ریداکس اکثرا چطوره؟
۱۷۵	redux-saga چیه؟
۱۷۶	مدل ذهنی redux-saga چطوره؟
۱۷۷	تفاوت افکتهای call و put توی redux-saga چی هست؟
۱۷۸	Redux Thunk چیه؟
۱۷۹	تفاوت‌های redux-saga و redux-thunk چیا هستن؟
۱۸۰	Redux DevTools چیه؟
۱۸۱	ویژگی‌های Redux DevTools چیا هستن؟
۱۸۲	سلکتورهای ریداکس چی هستن و چرا باید ازشون استفاده کنیم؟
۱۸۳	Redux Form چیه؟
۱۸۴	اصلی‌ترین ویژگی‌های Redux Form چیه؟
۱۸۵	چطوری میشه چندتا middleware به ریداکس اضافه کرد؟
۱۸۶	چطوری میشه توی ریداکس initial state تعریف کرد؟
۱۸۷	تفاوت‌های Relay با Redux چیا هستن؟
	React Native
۱۸۸	تفاوت‌های React و React Native چیا هستن؟
۱۸۹	چطوری میشه برنامه React Native رو تست کرد؟
۱۹۰	چطوری میشه توی React Native لاگ کرد؟
۱۹۱	چطوری میشه React Native رو دیباگ کرد؟
	کتابخانه‌های مورد استفاده با ری‌اکت

ردیف	سوال
۱۹۲	کتابخانه reselect چیه و چطوری کار می‌کنه؟
۱۹۳	Flow چیه؟
۱۹۴	تفاوت‌های Flow و PropTypes چیا هستن؟
۱۹۵	چطوری از آیکون‌های font-awesome توی ری‌اکت استفاده کنیم؟
۱۹۶	React Dev Tools چیه؟
۱۹۷	چرا توی کروم devtools برای فایل‌های local لود نمیشه؟
۱۹۸	چطوری از Polymer توی React استفاده کنیم؟
۱۹۹	مزایای React نسبت به Vue.js چیا هستن؟
۲۰۰	تفاوت‌های React و Angular چیا هستن؟
۲۰۱	چرا تب React در DevTools نشان داده نمی‌شود؟
۲۰۲	Styled components چیه؟
۲۰۳	یه مثال از Styled Components می‌تونن بگی؟
۲۰۴	Relay چیه؟
۲۰۵	چطوری میشه از تایپ اسکریپت توی create-react-app استفاده کرد؟
	متفرقه
۲۰۶	اصلی‌ترین ویژگی‌های کتابخانه reselect چیا هستن؟
۲۰۷	یه مثال از کارکرد کتابخانه reselect بزن؟
۲۰۸	توی Redux اکشن چیکار می‌کنه؟
۲۰۹	استاتیک شی با کلاس‌های ES6 در React کار می‌کنه؟
۲۱۰	ریداکس رو فقط با ری‌اکت میشه استفاده کرد؟
۲۱۱	برای استفاده از Redux به ابزار build خاصی احتیاج داریم؟

ردیف	سوال
۲۱۲	مقادیر پیش فرض ریداکس فرم چطوری تغییرات رو از state می گیرن؟
۲۱۳	توی PropTypes های ری اکت چطوری میشه برای یه prop چند نوع داده مجاز مشخص کرد؟
۲۱۴	می تونیم فایل svg رو به عنوان کامپوننت import کنیم؟
۲۱۵	چرا استفاده از توابع ref callback درون خطی توصیه نمیشه؟
۲۱۶	render hijacking توی ری اکت چیه؟
۲۱۷	پایده سازی factory یا سازنده HOC چطویه؟
۲۱۸	چطوری به یه کامپوننت ری اکت عدد پاس بدیم؟
۲۱۹	لازمه همه state ها رو توی ریداکس مدیریت کنیم؟ لزومی به استفاده از state داخلی داریم؟
۲۲۰	هدف از متد registerServiceWorker توی ری اکت چیه؟
۲۲۱	تابع memo ری اکت چیه؟
۲۲۲	تابع lazy ری اکت چیه؟
۲۲۳	چطوری با استفاده از تابع setState از رندر غیرضروری جلوگیری کنیم؟
۲۲۴	توی نسخه ۱۶ ری اکت چطوری میشه آرایه، Strings و یا عدد رو رندر کنیم؟
۲۲۵	چطوری میشه از تعریف ویژگی در کلاس کامپوننت استفاده کرد؟
۲۲۶	hook ها چی هستن؟
۲۲۷	چه قوانینی برای هوک ها باید رعایت بشن؟
۲۲۸	چطوری میشه از استفاده درست هوک ها اطمینان حاصل کرد؟
۲۲۹	تفاوت های Flux و Redux چیا هستن؟
۲۳۰	مزایای ری اکت روتر نسخه ۴ چیه؟
۲۳۱	می تونی راجع به متد componentDidCatch توضیح بدی؟

ردیف	سوال
۲۳۲	در چه سناریویی error boundary خطا رو catch نمی‌کنه؟
۲۳۳	چرا نیازی به error boundaries برای event handler ها نیست؟
۲۳۴	تفاوت بلوک try catch و error boundary چیه؟
۲۳۵	رفتار خطاهای uncaught در ری‌اکت 16 چیه؟
۲۳۶	محل مناسب برای قرار دادن error boundary کجاست؟
۲۳۷	مزیت چاپ شدن stack trace کامپوننت‌ها توی متن ارور boundary ری‌اکت چیه؟
۲۳۸	متدی که در تعریف کامپوننت‌های class الزامیه؟
۲۳۹	نوع‌های ممکن برای مقدار بازگشتی متد render چیا هستن؟
۲۴۰	هدف اصلی از متد constructor چیه؟
۲۴۱	آیا تعریف متد سازنده توی ری‌اکت الزامیه؟
۲۴۲	Default prop ها چی هستن؟
۲۴۳	چرا نباید تابع setState رو توی متد componentWillUnmount فراخوانی کرد؟
۲۴۴	کاربرد متد getDerivedStateFromError چیه؟
۲۴۵	کدوم متدها و به چه ترتیبی در طول ری‌رندر فراخوانی میشن؟
۲۴۶	کدوم متدها موقع error handling فراخوانی میشن؟
۲۴۷	کارکرد ویژگی displayName چیه؟
۲۴۸	سایپورت مرورگرها برای برنامه ری‌اکتی چطوره؟
۲۴۹	هدف از متد unmountComponentAtNode چیه؟
۲۵۰	code-splitting چیه؟
۲۵۱	مزایای حالت strict چیه؟
۲۵۲	Fragment های دارای key هستن؟

ردیف	سوال
۲۵۳	آیا ری اکت از همه‌ی attribute های HTML پشتیبانی می‌کند؟
۲۵۴	محدودیت‌های HOC ها چی هستن؟
۲۵۵	چطوری میشه forwardRefs رو توی DevTools دیباگ کرد؟
۲۵۶	مقدار یه props کامپوننت کی true میشه؟
۲۵۷	NextJS چیه و ویژگی‌های اصلیش چیا هستن؟
۲۵۸	چط، وی کی‌تونیم یه تابع event handler رو به یه کامپوننت پاس بدیم؟
۲۵۹	استفاده از توابع arrow برای متدهای render خوبه؟
۲۶۰	چطوری از اجرای چندباره یه تابع جلوگیری کنیم؟
۲۶۱	JSX چطوری از حمله‌های Injection جلوگیری می‌کند؟
۲۶۲	چطوری element های رندر شده رو آپدیت کنیم؟
۲۶۳	چرا prop ها read only هستن؟
۲۶۴	چرا می‌گیم تابع setState از طریق merge کردن state را مدیریت می‌کند؟
۲۶۵	چطوری می‌تونیم به متد event handler پارامتر پاس بدیم؟
۲۶۶	چطوری از رندر مجدد کامپوننت‌ها جلوگیری کنیم؟
۲۶۷	شرایطی که بدون مشکل پرفورمنس بتونیم از ایندکس به عنوان key استفاده کنیم چی هست؟
۲۶۸	key های ری اکت باید به صورت عمومی منحصر بفرد باشن؟
۲۶۹	گزینه‌های محبوب برای مدیریت فرم‌ها توی ری اکت چیا هستن؟
۲۷۰	مزایای کتابخانه فرمیک نسبت به redux form چیه؟
۲۷۱	چرا اجباری برای استفاده از ارث‌بری توی ری اکت نیست؟ مزیتی داره؟
۲۷۲	می‌تونیم از web components توی برنامه ری اکت استفاده کنیم؟
۲۷۳	dynamic import چیه؟

ردیف	سوال
۲۷۴	loadable component چیست؟
۲۷۵	کامپوننت suspense چیست؟
۲۷۶	چطوری به ازای route می‌تونیم code splitting داشته باشیم؟
۲۷۷	یه مثال از نحوه استفاده از context میزنی؟
۲۷۸	هدف از مقدار پیش فرض توی context چیست؟
۲۷۹	چطوری از contextType استفاده می‌کنین؟
۲۸۰	consumer چیست؟
۲۸۱	چطوری مسائل مربوط به پرفورمنس با context رو حل می‌کنین؟
۲۸۲	هدف از forward ref توی HOC ها چیست؟
۲۸۳	توی کامپوننت‌ها می‌تونیم پراپ ref داشته باشیم؟
۲۸۴	چرا در هنگام استفاده از ForwardRef نیاز به احتیاط بیشتری در استفاده از کتابخانه های جانبی داریم؟
۲۸۵	چطوری بدون استفاده از ES6 کلاس کامپوننت بسازیم؟
۲۸۶	استفاده از ری‌اکت بدون JSX ممکن است؟
۲۸۷	الگوریتم‌های diffing ری‌اکت چیستن؟
۲۸۸	قوانینی که توسط الگوریتم‌های diffing پوشش داده می‌شوند کدام هستن؟
۲۸۹	چه موقعی نیاز هست که از ref ها استفاده کنیم؟
۲۹۰	برای استفاده از render prop لازمه که اسم prop رو render بزاریم؟
۲۹۱	مشکل استفاده از render props با pure component چیست؟
۲۹۲	چطوری با استفاده از render props می‌تونیم HOC ایجاد کنیم؟
۲۹۳	تکنیک windowing چیست؟
۲۹۴	توی JSX یه مقدار falsy رو چطوری چاپ کنیم؟

ردیف	سوال
۲۹۵	یه مورد استفاده معمول از portals مثال میزنی؟
۲۹۶	توی کامپوننت‌های کنترل نشده چطوری مقداری پیش فرض اضافه کنیم؟
۲۹۷	stack موردعلاقه شما برای کانفیگ پروژه ری‌اکت چیه؟
۲۹۸	تفاوت DOM واقعی و Virtual DOM چیه؟
۲۹۹	چطوری Bootstrap رو به یه برنامه ری‌اکتی اضافه کنیم؟
۳۰۰	می‌تونن یه لیسستی از معروف‌ترین وب‌سایت‌هایی که از ری‌اکت استفاده می‌کنن رو بگی؟
۳۰۱	استفاده از تکنیک CSS In JS تو ری‌اکت توصیه میشه؟
۳۰۲	لازمه همه کلاس کامپوننت‌ها رو تبدیل کنیم به هوک؟
۳۰۳	چطوری میشه با هوک‌های ری‌اکت دیتا fetch کرد؟
۳۰۴	هوک‌ها همه موارد کاربرد کلاس‌ها رو پوشش میدن؟
۳۰۵	نسخه پایدار ری‌اکت که از هوک پشتیبانی می‌کنه کدومه؟
۳۰۶	چرا از حالت destructuring آرایه برای useState استفاده می‌کنیم؟
۳۰۷	منابعی که باعث معرفی ایده هوک‌ها شدن چیا بودن؟
۳۰۸	چطوری به API‌های ضروری اجزای وب دسترسی پیدا کنیم؟
۳۰۹	formik چیه؟
۳۱۰	middlewareهای مرسوم برای مدیریت ارتباط‌های asynchronous توی Redux چیا هستن؟
۳۱۱	مروگرها کد JSX رو متوجه میشن؟
۳۱۲	Data flow یا جریان داده ری‌اکت رو توضیح میدی؟
۳۱۳	react scripts چیه؟
۳۱۴	ویژگی‌های create react app چیه؟

ردیف	سوال
۳۱۵	هدف از متد renderToNodeStream چیه؟
۳۱۶	MobX چیه؟
۳۱۷	تفاوت‌های بین Redux و MobX چیا هستن؟
۳۱۸	لازمه قبل از شروع ری‌اکت ES6 رو یاد گرفت؟
۳۱۹	Concurrent Rendering چیه؟
۳۲۰	تفاوت بین حالت async و concurrent چیه؟
۳۲۱	می‌تونیم از آدرس‌های دارای url جاوااسکریپت در ری‌اکت 16.9 استفاده کرد؟
۳۲۲	هدف از پلاگین eslint برای هوک‌ها چیه؟
۳۲۳	تفاوت‌های Declarative و Imperative توی ری‌اکت چیه؟
۳۲۴	مزایای استفاده از تایپ اسکریپت با ری‌اکت چیه؟

Core React

۱. ری‌اکت چیه؟

ری‌اکت یه کتابخونه متن‌باز هست که برای ساختن رابط کاربری مخصوصا برنامه‌های تک صفحه‌ای استفاده میشه. از این کتابخونه برای مدیریت لایه view توی برنامه‌های وب و موبایل استفاده میشه. توسط [Jordan Walke](#) تولید شده که یه مهندس نرم‌افزار توی شرکت فیس‌بوک هستش. اولین بار سال ۲۰۱۱ و روی برنامه اینستاگرام مورد استفاده قرار گرفت

↑ فهرست مطالب

۲. اصلی‌ترین ویژگی‌های ری‌اکت چیا هستن؟

اصلی‌ترین ویژگی‌های ری‌اکت اینا هستن:

- از **VirtualDOM** به جای RealDOM استفاده می‌کنه چون هزینه تغییرات RealDOM زیاده (یعنی پیدا کردن DOM Element و حذف یا به روز رسانی با سرعت کمتری انجام میشه)
- از **SSR(server side rendering)** پشتیبانی می‌کنه
- از جریان داده ها یا data binding به صورت **یک طرفه (unidirectional)** پیروی می‌کنه
- برای توسعه view از UI کامپوننت‌های **reusable/composable** استفاده می‌کنه

[↑ فهرست مطالب](#)

۳. JSX چیه؟

JSX یه افزونه با سینتکسی شبیه به XML برای ECMAScript است (مخفف *Javascript* XML). اگه بخوایم ساده بگیریم وظیفه اش اینه که سینتکسی ساده تر از `React.createElement` در اختیارتون قرار میده، شما میتونید *Javascript* رو در کنار ساختاری شبیه به HTML داشته باشید. تو مثال زیر می‌بینید که نوشته داخل تگ `h1` مثل یک تابع *Javascript* به تابع `render` تحویل داده میشه.

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{"Welcome to React world!"}</h1>
      </div>
    );
  }
}
```

[↑ فهرست مطالب](#)

۴. تفاوت‌های Component و Element چیه؟

Element یک شی ساده است که وظیفه داره اون چیزی که روی صفحه نمایش داده میشه رو توصیف کنه حالا ممکنه به صورت یک *DOM node* باشه یه به صورت *component* های دیگه. *Elements* میتونن شامل *Elements* دیگه به عنوان *props* باشند. ساختن یک

Element در React کار ساده و کم دردرسریه اما وقتی که ساخته شد هیچ وقت همیشه تغییرش داد.

تو مثال زیر یک شی که توسط React Element ساخته شده رو میبینیم :

```
const element = React.createElement("div", { id: "login-btn" }, "Login");
```

تابع `React.createElement` که توی قطعه کد بالا میبینید یه `object` شبیه به این برمیگردونه:

```
{
  type: 'div',
  props: {
    children: 'Login',
    id: 'login-btn'
  }
}
```

و آخرش هم با استفاده از `ReactDOM.render` می‌تونیم توی `DOM` رندر کنیم

```
<div id="login-btn">Login</div>
```

درحالیکه یه **component** میتونه به روشهای مختلفی ساخته بشه. میتونه یه `class` باشه با یه متد `render`. یا حتی به عنوان یه جایگزین ساده‌تر به صورت یک تابع تعریف بشه. در هر دو حالت کامپوننت ساخته شده `props` رو به عنوان ورودی دریافت می‌کنه و یه خروجی رو به صورت یه `JSX tree` برمی‌گردونه. به مثال زیر دقت کنیم که چطور با استفاده از یه تابع و `JSX` یک کامپوننت ساخته میشه:

```
const Button = ({ onLogin }) => (
  <div id="login-btn" onClick={onLogin}>
    Login
  </div>
);
```

`JSX` به `transpile` `React.createElement` همیشه :

```
const Button = ({ onLogin }) =>
  React.createElement(
    "div",
    { id: "login-btn", onClick: onLogin },
    "Login"
  );
```

۵. تو ری اکت چطوری کامپوننت می‌سازیم؟

تو سوال قبل یه اشاره کوچیک کردیم که دوتا راه برای ساختن کامپوننت وجود داره. ۱. **Function Components**: این ساده‌ترین راه برای ساختن یه کامپوننته. یه *Pure Javascript Function* رو در نظر بگیرید که Props که خودش یه object هست رو به عنوان پارامتر ورودی میگیره و یه React Element به عنوان خروجی برمی‌گردونه مثل همین مثال پایین:

```
function Greeting({ message }) {  
  return <h1>{`Hello, ${message}`}</h1>;  
}
```

۲. **Class Components**: شما میتونید از class که در ES6 به جاواسکریپت اضافه شده برای این کار استفاده کنیم. کامپوننت مثال قبلی رو اگه بخواییم با class پیاده سازی کنیم اینجوری میشه:

```
class Greeting extends React.Component {  
  render() {  
    return <h1>{`Hello, ${this.props.message}`}</h1>;  
  }  
}
```

فقط یادتون نره تو این روش متد `render` یه جورایی required میشه.

↑ فهرست مطالب

۶. کی باید از Class Component بجای Function Component استفاده کنیم؟

اگه کامپوننت نیاز به *state* یا *lifecycle methods* داشت از کلاس کامپوننت‌ها استفاده میکنیم در غیر این صورت میریم سراغ فانکشن کامپوننتها. با این حال از ورژن 16.8 ری اکت به بعد و با اضافه شدن هوکها به فانکشن کامپوننت‌ها، شما میتونید از *state* یا *lifecycle method* یا تمامی فیچرهایی که قبلا فقط در کلاس کامپوننت‌ها قابل استفاده بود توی فانکشن کامپوننتتون استفاده کنید

↑ فهرست مطالب

۷. Pure Components چیست؟

`React.PureComponent` دقیقاً مثل `React.Component` می‌مونه فقط تنها تفاوتی که داره اینه که برخلاف `Component` خودش به صورت خودکار متد `shouldComponentUpdate` رو هندل می‌کنه. وقتی که props یا state در کامپوننت تغییر می‌کنه، `PureComponent` به مقایسه سطحی روی props و state انجام می‌ده (shallow comparison) در حالیکه `Component` این مقایسه رو به صورت خودکار انجام نمیده و به طور پیش فرض کامپوننت هر بار که `shouldComponentUpdate` فراخوانی بشه re-render میشه. بنابراین توی `Component` باید این متد override بشه.

↑ فهرست مطالب

۸. state توی اکت چیکار می‌کنه؟

`State` در هر کامپوننت یه آبجکتی که یه سری اطلاعات که در طول عمر کامپوننت ما ممکنه تغییر کنه رو در خودش ذخیره می‌کنه. ما باید تمام تلاشمون رو بکنیم که stateمون در ساده ترین حالت ممکن باشه و تاجایی که می‌تونیم تعداد کامپوننت‌هایی که stateful هستن رو کاهش بدیم. به عنوان مثال بیایید یه کامپوننت `User` رو که یه state داره بسازیم:

```
class User extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      message: "Welcome to React world",
    };
  }

  render() {
    return (
      <div>
        <h1>{this.state.message}</h1>
      </div>
    );
  }
}
```



state is used for internal communication inside a Component

State و Props بهم شبیه هستن ولی State کاملاً در کنترل کامپوننت هستن و فقط مختص به همون کامپوننت هستن (private). یعنی state ها در هیچ کامپوننتی به غیر از اونی که مالک state هست در دسترس نخواهند بود.

[↑ فهرست مطالب](#)

۹. props تو ری اکت چیکار می‌کنه؟

Props ورودی کامپوننتها هستن. میتونن یه مقدار ساده یا یه شی شامل یه مجموعه مقدار باشن که در لحظه ایجاد کامپوننت و بر اساس یه یه قاعده نام گذاری که خیلی شبیه به HTML-tag attributes هست، به کامپوننت پاس داده میشن. در واقع اینها داده‌هایی هستن که از کامپوننت پدر به فرزند تحویل داده میشن. هدف اصلی وجود Props در ری اکت ایجاد ساختارهای زیر در یک کامپوننته:

- 1 - پاس دادن مقادیر به کامپوننت شما
- 2 - trigger کردن یک متد در زمان تغییر state
- 3 - استفاده از مقادیر داخل متد `render (this.props.reactProps` به عنوان مثال ، یه کامپوننت با استفاده از `reactProps` میسازیم:

```
<Element reactProp={"1"} />
```

این `reactProps` (یا هر چیزی که شما اسمشو میزارید) در نهایت تبدیل به یک property خواهد شد که داخل `props object` ، که داخل تمامی کامپوننت های `react` از ابتدا وجود داره ، قرار میگیره. و به شکل زیر قابل دسترس هست

```
props.reactProp
```

```
this.props.reactProp
```

↑ فهرست مطالب

۱۰. تفاوت state و props چیست؟

هر دو javascript plain object هستند . هر دو وظیفه دارن مقادیری که روی render تاثیر گذار هست رو نگه داری کنن اما عملکردشون با توجه به کامپوننت متفاوت خواهد بود. Props شبیه به پارامترهای ورودی یک فانکشن، به کامپوننت پاس داده میشن در حالیکه state شبیه به متغیرهایی که داخل فانکشن ساخته شدن ، توسط خود کامپوننت ایجاد و مدیریت میشه.

↑ فهرست مطالب

۱۱. چرا نباید state رو مستقیما آپدیت کنیم؟

اگه یه بار تلاش کنید که مستقیما state رو آپدیت کنید متوجه میشید که کامپوننت شما مجددا render نمیشه.

```
//Wrong  
this.state.message = "Hello world";
```

به جای اینکه مستقیما state رو آپدیت کنیم می‌تونیم از متد setState در Class Component و از useState در Function Components استفاده کنیم. این متدها یک آپدیت در شی state رو برنامه ریزی و مدیریت میکنن و وقتی تغییر انجام شد کامپوننت شما re-render خواهد شد.

```
//Correct  
this.setState({ message: "Hello World" });
```

```
const [message, setMessage] = React.useState("Hello world");  
  
setMessage("New Hello world");
```

نکته مهم: شما میتونید در سازنده کلاس یا در ورژن جدید جاواسکریپت به عنوان فیلد کلاس هر چیزی رو به شی state خودتون assign کنید.

۱۲. هدف از متدهای callback توی استفاده از setState چیه؟

callback function زمانی که setState تموم شد و کامپوننت مجدداً render شد فراخوانی میشه. از اونجایی که **setState asynchronous** یا همون غیرهمزمانه از callback برای کارهایی استفاده میشه که بعد از تابع setState قراره اجرا بشن. **نکته مهم:** بهتره که به جای callback از lifecycle method ها استفاده کنیم.

```
useState({ name: "John" }, () =>
  console.log("The name has updated and component re-rendered")
);
```

۱۳. تفاوت بین نحوه مدیریت رویداد HTML و React چیه؟

۱. توی HTML، عنوان رخداد حتماً باید با حرف کوچیک شروع بشه یا اصطلاحاً *lowercase* باشه:

```
<button onclick="activateLasers()"></button>
```

ولی توی ری اکت از *camelCase* پیروی می‌کنه:

```
<button onClick={activateLasers}>Test</button>
```

۲. توی HTML می‌تونیم برای جلوگیری از اجرای رفتار پیش‌فرض (preventDefault) یه مقدار **false** برگردونیم:

```
<a href="#" onclick='console.log("The link was clicked."); return false;
```

ولی توی ری اکت برای انجام این مورد حتماً باید از **preventDefault** استفاده بشه :

```
function handleClick(event) {
  event.preventDefault();
  console.log("The link was clicked.");
}
```

۳. توی HTML برای اجرای تابع حتما باید اونو با گذاشتن پرانتزهایی که بعد اسمش میزاریم invoke کنیم ()
ولی توی ری اکت اجباری به گذاشتن () جلوی اسم تابع نیست. (برای مثال به کد اول و تابع "activateLasers" دقت کنید)

↑ فهرست مطالب

۱۴. چطوری متد یا event رو به تابع callback توی JSX bind کنیم؟

سه روش مختلف برای انجام این مورد هست:

۱. **Bind کردن توی Constructor:** توی کلاس‌های جاوااسکریپتی متدها به صورت پیش فرض bound نمیشن. همین موضوع توی کلاس کامپوننت‌های ری اکتی برای متدهای موجود هم رخ میده که اکثرا توی متد سازنده یا همون constructor می‌آییم bind می‌کنیم.

```
class Component extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    // ...
  }
}
```

۲. **استفاده از فیلد عمومی کلاس (public):** اگه از روش اول خوشتون نیاد این روش هم می‌تونه context درست رو موقع callbackها براتون فراهم کنه.

```
handleClick = () => {
  console.log("this is:", this);
};
```

```
<button onClick={this.handleClick}>{"Click me"}</button>
```

۳. **توابع arrow توی callback:** می‌تونین از توابع فلش به شکل مستقیم توی callbackها استفاده کنین.

```
<button onClick={(event) => this.handleClick(event)}>{"Click me"}</button>
```

نکته: اگر متدهای callback به عنوان prop به کامپوننت‌های فرزندشون پاس داده بشن، ممکنه اون کامپوننت‌ها re-rendering‌های ناخواسته‌ای داشته باشن. توی اینگونه موارد روش توصیه شده استفاده از `bind` یا فیلد عمومی کلاس برای مدیریت پرفورمنس هستش.

↑ فهرست مطالب

۱۵. چطوری میشه یک مقدار رو به یه تابع callback یا eventHandler پاس بدیم؟

می‌تونیم از توابع *arrow* استفاده کنیم که با `wrap` کردن دور *event handler* و پاس دادن مقدار بهش نیاز مورد نظرمونو انجام بدیم:

```
<button onClick={() => this.handleClick(id)}>Test</button>
```

این حالت دقیقاً مثل فراخوانی `bind` هستش:

```
<button onClick={this.handleClick.bind(this, id)} />
```

جدا از این روش‌ها، میشه با ایجاد یه *curry*، یه تابع دیگه دور تابع هندلر خودمون `wrap` کنیم و پارامتر رو به اون پاس بدیم:

```
<button onClick={this.handleClick(id)} />  
handleClick = (id) => () => {  
  console.log("Hello, your ticket number is", id);  
};
```

↑ فهرست مطالب

۱۶. Synthetic events (رویدادهای مصنوعی) تو ری‌اکت چیا هستن؟

`SyntheticEvent` یه رخداد *cross-browser* هست که به‌عنوان یه *wrapper* دور *event*‌های اصلی مرورگر هستش. رابط *API* برای کارکردن با اون عیناً مثل رخداد *native*

مرورگرهاست که شامل `stopPropagation` و `preventDefault` می‌شود، با این تفاوت که این رخدادها بر روی همه مرورگرها کار می‌کنند.

[↑ فهرست مطالب](#)

۱۷. عبارات شرطی درون خطی چیست؟

برای بررسی یک شرط می‌تونیم از عبارت شرطی `if` استفاده کنیم، البته عملگرهای درون خطی سه‌گانه (ternary) هم همیشه استفاده کرد که از ویژگی‌های خود `JS` هستند. جدا از این ویژگی‌ها، می‌تونیم هر عبارتی داخل آکولاد و توی `JSX` به اصطلاح `embed` یا ترکیب کنیم و با عملگر منطقی `&&` ترکیب کنیم، مثال پایینی رو ببینید:

```
<h1>Hello!</h1>;
{
  messages.length > 0 && !isLogin ? (
    <h2>You have {messages.length} unread messages.</h2>
  ) : (
    <h2>You don't have unread messages.</h2>
  );
}
```

[↑ فهرست مطالب](#)

۱۸. props های "key" چی هستند و مزایای استفاده از آنها در آرایه عناصر چیست؟

یه `key` به `attribute` ویژه هستند و موقعی که داریم یه آرایه از المان‌ها رو ایجاد می‌کنیم باید بهشون به عنوان `prop` بدیمش. `key`ها به ری‌اکت کمک می‌کنن که بدونن باید کدوم المان رو دقیقاً اضافه، حذف یا به روز کنه. اکثراً از `ID` یا از یه دیتای یونیک به عنوان `key` استفاده می‌کنن:

```
const todoItems = todos.map((todo) => <li key={todo.id}>{todo.text}</li>
```

موقعی که یه آیدی خاص برای المان‌ها نداریم، ممکنه بیاپید و از اندیس یا همون `index` به عنوان `key` استفاده کنید:

```
const todoItems = todos.map((todo, index) => (
  <li key={index}>{todo.text}</li>
));
```

نکته:

۱. استفاده از *index* برای *key* توصیه **نمیشه** چون ممکنه ترتیب عناصر خیلی راحت عوض بشه و این می‌تونه پرفورمنس برنامه رو تحت تاثیر بزاره.
۲. اگه بیایین و لیست مورد نظر رو به جای *li* مثلا با *ی*ه کامپوننت به اسم *ListItem* جایگزین کنین و *prop* مورد نظر *key* رو به جای *li* به اون پاس بدیم، *ی*ه warning توی کنسول خواهیم داشت که میگه *key* پاس داده نشده.

↑ فهرست مطالب

۱۹. کاربرد *ref* ها چیه؟

ref به عنوان *ی*ه مرجع برای دسترسی مستقیم به المان موردنظرمون استفاده میشه . تا حد امکان و توی اکثر مواقع بهتره از اونا استفاده نکنیم، البته خیلی می‌تونن کمک کننده باشن چون دسترسی مستقیمی به DOM element یا instance اصلی component بهمون میدن.

↑ فهرست مطالب

۲۰. چطوری از *ref* استفاده کنیم؟

دو تا روش وجود داره:

۱. استفاده از `React.createRef()` و پاس دادن اون به *element* مورد نظرمون با `ref` attribute .

```
const Component = () => {
  const myRef = React.createRef();

  return <div ref={myRef} />;
};
```

۲. اگر از نسخه ۱۶.۸ به بالاتر هم استفاده می‌کنیم که به اسم `useRef` هست و می‌تونیم به سادگی توی کامپوننت‌های تابعی ازش استفاده کنیم. مثل:

```
const RenderCounter = () => {
  const counter = useRef(0);

  // Since the ref value is updated in the render phase,
  // the value can be incremented more than once
  counter.current = counter.current + 1;

  return <h1>`The component has been re-rendered ${counter} times`</h1>
};
```

↑ فهرست مطالب

۲.۱ forward ref چیه؟

Ref forwarding ویژگی‌ایه که به بعضی از کامپوننت‌ها این اجازه رو میده *ref* دریافت شده رو به کامپوننت فرزند انتقال بدن.

```
const ButtonElement = React.forwardRef((props, ref) => (
  <button ref={ref} className="CustomButton">
    {props.children}
  </button>
));

// Create ref to the DOM button:
const ref = React.createRef();
<ButtonElement ref={ref}>{"Forward Ref"}</ButtonElement>;
```

↑ فهرست مطالب

۲.۲ بین `callback refs` و تابع `findDOMNode` کدوم رو ترجیح میدی؟

ترجیح داده میشه که از *callback refs* به جای `findDOMNode` استفاده کنیم، چون `findDOMNode` از پیشرفت‌های خاص ری‌اکت در آینده جلوگیری می‌کنه. رویکرد **legacy** استفاده از `findDOMNode`:

```
class MyComponent extends Component {
  componentDidMount() {
    findDOMNode(this).scrollIntoView();
  }

  render() {
    return <div></div>;
  }
}
```

رویکرد توصیه شده:

```
class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.node = createRef();
  }
  componentDidMount() {
    this.node.current.scrollIntoView();
  }

  render() {
    return <div ref={this.node} />;
  }
}
```

[↑ فهرست مطالب](#)

۲۳. چرا Ref های متنی منقضی محسوب می شوند؟

اگر قبلاً با ری اکت کار کرده باشید، ممکنه با یه API قدیمی تر آشنا باشید که توی اون ویژگی ref یه رشته ست، مثل `{'ref='textInput}` و گره DOM به عنوان `this.refs.textInput` قابل دسترسیه. We advise against it because *string refs have below issues, and are considered legacy. String refs were removed in React v16*

۱. اونا ری اکت رو وادار میکنن که عناصر در حال اجرا رو پیگیری کنه. این مساله یکم مشکل سازه چون باعث میشه خطاهای عجیب و غریب وقتی که ماژول ری اکت توی باندل کپی میشه ایجاد بشه.
۲. قابل انعطاف نیستن – اگر یه کتابخونه یه ref رو روی فرزند انتقال داده شده قرار بده، کاربر نمیتونه یه ref دیگه ای رو روی اون قرار بده. ref های برگشتی کاملاً ترکیب شده هستن.

۳. با *یه آنالیزگر استاتیک* مثل Flow کار نمی‌کنن. در حقیقت Flow نمی‌تونه حدس بزنه که فریم‌ورک چه کاری روی برا `this.refs`، مثل نوع داده اون (که ممکنه متفاوت باشه). `ref`‌های callback دار بیشتر با آنالیزگرها سازگارترن.

۴. اون طور که اکثر مردم از الگوی "render callback" انتظار دارند کار نمی‌کند (برای مثال `</ {DataTable renderRow={this.renderRow}>`)

```
class MyComponent extends Component {
  renderRow = (index) => {
    // This won't work. Ref will get attached to DataTable rather than M
    return <input ref={"input-" + index} />;

    // This would work though! Callback refs are awesome.
    return <input ref={(input) => (this["input-" + index] = input)} />;
  };

  render() {
    return <DataTable data={this.props.data} renderRow={this.renderRow}
  }
}
```

[↑ فهرست مطالب](#)

۲۴. Virtual DOM چیه؟

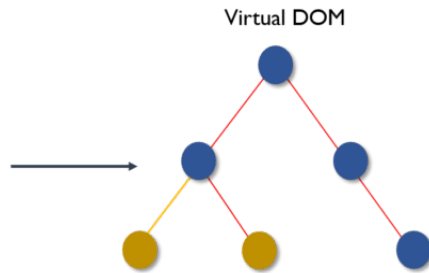
Virtual DOM (VDOM) یه نماینده in-memory از *DOM* واقعی هستش. این نماینده از رابط کاربری توی حافظه رم نگهداری میشه و همواره با *DOM* "واقعی" همگام سازی (sync) میشه. این گام بین تابع رندر و نمایش *element* روی صفحه رخ میده و به مجموعه اتفاقاتی که برای مدیریت این موارد انجام میشه *reconciliation* میگن.

[↑ فهرست مطالب](#)

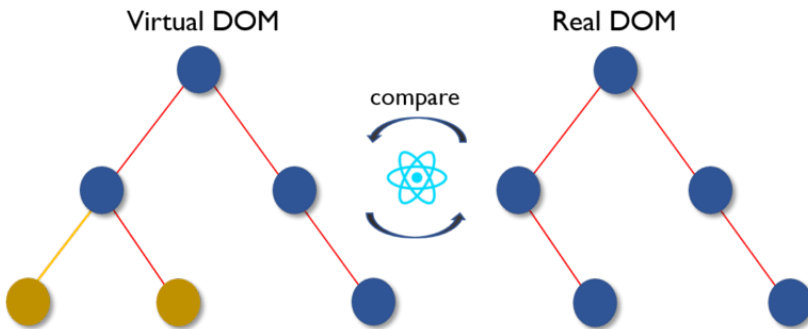
۲۵. Virtual DOM چطوری کار می‌کنه؟

Virtual DOM توی سه مرحله ساده کار می‌کنه.

۱. هر زمان که داده‌های اساسی تغییر می‌کنه، کل رابط کاربری توسط *DOM* مجازی مجددا رندر میشه.

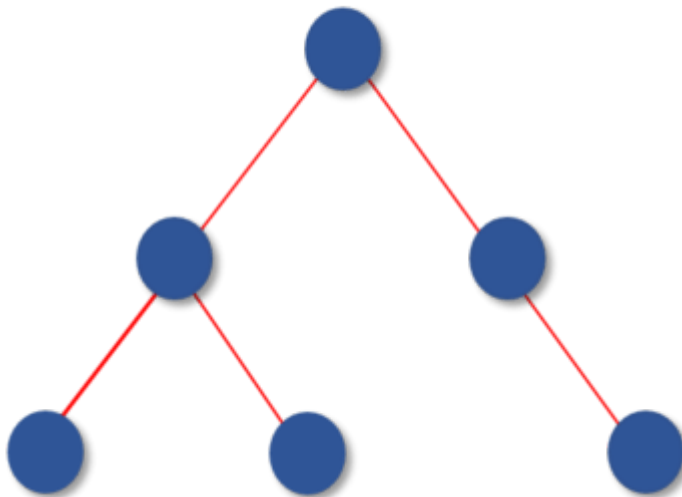


۲. تفاوت بین DOM قبلی و جدید محاسبه میشه.



۳. بعد از انجام محاسبات ، DOM واقعی فقط با مواردی که واقعاً تغییر کردن به روز میشه.

Real DOM (updated)



↑ فهرست مطالب

۲۶. تفاوت بین Virtual DOM و Shadow DOM چیه؟

shadow DOM به تکنولوژی مرورگره که در درجه اول برای تعیین متغیر ها و CSS در کامپوننت وب طراحی شده بود. *virtual DOM* مفهومی که توسط کتابخانه ها در جاوااسکریپت در API های مرورگر اجرا شده.

↑ فهرست مطالب

۲۷. React Fiber چیه؟

Fiber موتور جدید برای عملیات *reconciliation* هست یا میشه گفت که پیاده سازی مجدد الگوریتم هسته ری اکت نسخه ۱۶ هست. هدف پیاده سازی ReactFiber برای بهبود کارکرد توی ناحیه هایی مثل انیمیشن، layout، کار با gesture، قابلیت اینکه عملیات در حال اجرا رو متوقف، قطع یا مجددا فعال کنیم و همینطوری اولویت بندی بروزرسانی های لازم توی DOM رو شامل می شد.

↑ فهرست مطالب

۲۸. هدف اصلی React Fiber چیه؟

هدف پیاده سازی ReactFiber برای بهبود کارکرد توی ناحیه هایی مثل انیمیشن، layout، کار با gesture بود. میشه گفت مهم ترین ویژگی **incremental-rendering** بوده که قابلیت بخش بندی (chunk کردن) عملیات اجرایی و متوقف و اجرا کردن اون توی فریم های مختلف هست.

↑ فهرست مطالب

۲۹. کامپوننت های کنترل شده چی هستن؟

کامپوننتی که عناصر ورودی رو توی فرم های ورودی کاربر کنترل می کنه به عنوان کامپوننت کنترل شده شناخته میشه، هر جهش state به تابع نگهدارنده مرتبط داره. به عنوان مثال، برای نوشتن تمام اسم ها با حروف بزرگ، باید از `handleChange` مثل زیر استفاده کنیم:

```
handleChange(event) {  
  this.setState({value: event.target.value.toUpperCase()})  
}
```

۳۰. کامپوننت‌های کنترل نشده چی هستند؟

کامپوننت‌های کنترل نشده کامپوننت‌هایی هستند که state های خودشان رو به صورت داخلی ذخیره می‌کنند و ما می‌تونیم با استفاده از یک ref از DOM پرس و جو کنیم تا در صورت نیاز مقدار فعلی اونو پیدا کنیم. این یکم شبیه HTML سنتیه. در کامپوننت UserProfile زیر، ورودی `name` با استفاده از ref قابل دسترسیه.

```
class UserProfile extends React.Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.input = React.createRef();
  }

  handleSubmit(event) {
    alert("A name was submitted: " + this.input.current.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          {"Name:"}
          <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

در بیشتر موارد، توصیه میشه که از کامپوننت‌های کنترل شده برای پیاده سازی فرم‌ها استفاده کنیم.

۳۱. تفاوت‌های بین cloneElement و createElement چیا هستند؟

عناصر JSX به توابع `React.createElement` تبدیل می‌شوند تا عناصر ری‌اکتی بسازند که برای نمایش شی‌های استفاده می‌شوند. درحالی‌که `cloneElement` برای کلون کردن یک عنصر و فرستادنش به عنوان `prop` جدید استفاده می‌شود.

↑ فهرست مطالب

۳۲. مفهوم `lift state up` یا مدیریت `state` در لول بالاتر رو توضیح میدی؟

وقتی که کامپوننت‌های مختلف نیاز به یک داده خاص دارند که بین اونا مشترک به‌تره `state` مشترک رو تا حد امکان به نزدیک‌ترین کامپوننت بالایی‌شون انتقال بدیم. این مورد به این معنیه که اگه دو کامپوننت فرزند داریم که به یک `state` مشخص رو دارند مدیریت می‌کنن توی خودشون، اون `state` رو می‌بریم توی کامپوننت والد و بجای مدیریت به `state` توی دوتا کامپوننت، از یک جا و توی کامپوننت والد اون رو مدیریت می‌کنیم.

↑ فهرست مطالب

۳۳. فازهای مختلف از `lifecycle` کامپوننت چیا هستن؟

چرخه حیات کامپوننت سه مرحله داره:

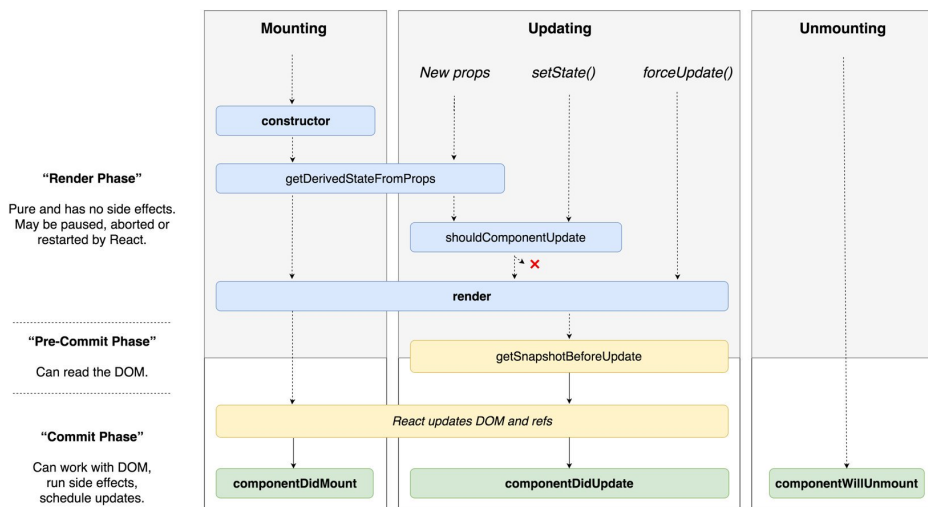
۱. **Mounting**: کامپوننت آماده نصب در `DOM` مرورگر هستش. این مرحله مقاردهی اولیه از متدهای `lifecycle constructor`، `render`، `getDerivedStateFromProps` و `componentDidMount` رو پوشش میده.
۲. **Updating**: در این مرحله، کامپوننت از دو طریق به روزرسانی می‌شه، ارسال `prop`‌های جدید و به روزرسانی `state` از طریق `setState` یا `forceUpdate`. این مرحله متدهای `getDerivedStateFromProps`، `shouldComponentUpdate`، `render`، `getSnapshotBeforeUpdate` و `componentDidUpdate` رو پوشش میده.
۳. **Unmounting**: در مرحله آخر کامپوننت مورد نیاز نیست و از `DOM` مرورگر حذف می‌شه. این مرحله فقط شامل متد `componentWillUnmount` می‌شه. البته بهتره اینجا این نکته رو بگیم که ری‌اکت برای به روز کردن `DOM` به سری فازبندی‌هایی داره که خود اون مرحله رو توی سه تا فاز انجام میده. این پایین به این فازبندی‌ها اشاره می‌کنیم.

۱. **Render** کامپوننت بدون هیچ سایدافکتی رندر میشه. این فقط در مورد کامپوننت های خالص صدقمیکنه و در این مرحله، ری اکت میتونه رندر رو متوقف، حذف یا restart کنه.

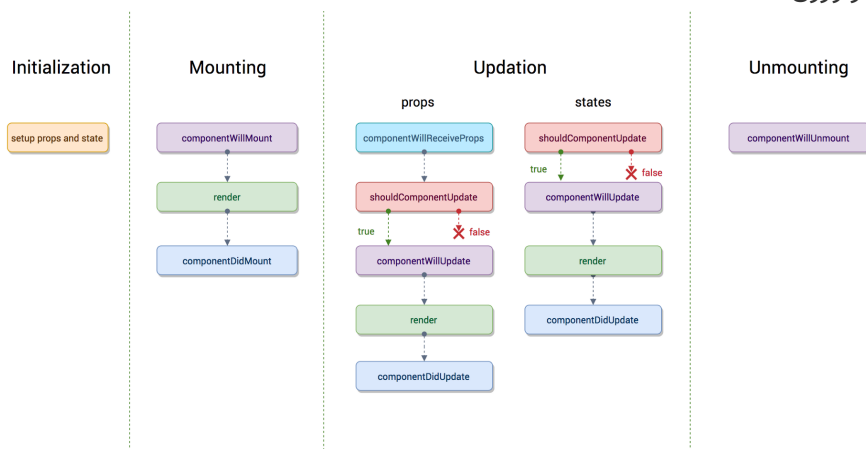
۲. **Pre-commit** قبل از اینکه کامپوننت تغییرات رو روی DOM اعمال کنه، لحظه ای وجود داره که به ری اکت اجازه میده از DOM داخل متد `getSnapshotBeforeUpdate` بخونه.

۳. **Commit** ری اکت با DOM کارمیکنه و lifecycle های آخر رو به ترتیب اجرامیکنه، `componentDidMount` برای نصب، `componentDidUpdate` برای به روزرسانی و `componentWillUnmount` برای غیرفعال کردن.

مراحل ری اکت ۱۶.۳ (یا نسخه تعاملی)



قبل از ورژن ۱۶.۳



↑ فهرست مطالب

۳۴. متدهای lifecycle کامپوننت چیا هستن؟

۱۶.۳ اکت

- **getDerivedStateFromProps**: درست قبل از اینکه Render اجرا بشه فراخوانی میشه و در هر بار render فراخوانی میشه. برای موارد نادری که نیاز داریم از state مشتق بگیریم این متد استفاده میشه. بهتره که اینو بخونید [اگه نیاز داشتن که از state مشتق بگیرین] (<https://reactjs.org/blog/۲۰۱۸/۰۶/۰۷/you-probably-dont-need-derived-state.html>).
- **componentDidMount**: بعد از اولین رندر اجرا میشه و همه درخواست های AJAX، DOM یا بروزرسانی state و تنظیمات event listeners اجرا میشه.
- **shouldComponentUpdate**: تعیین می‌کنه که کامپوننت به روز بشه یا نه. به طور پیش فرض مقدار **true** رو برمی‌گردونه. اگه مطمئن باشیم که کامپوننت بعد از اینکه state یا props به روزرسانی میشه نیازی به رندر شدن نداره، می‌تونیم مقدار **false** رو برگردونیم. اینجا جای خوبی برای بهبود عملکرده چون این امکان رو بهمون میده که اگه کامپوننت prop جدید میگیره از render مجدد جلوگیری کنیم.
- **getSnapshotBeforeUpdate**: درست قبل از رندر مجدد خروجی به DOM اجرا میشه. هر مقداری که توسط این متد برگشت داده میشه به متد **componentDidUpdate** انتقال داده میشه. برای گرفتن اطلاعات از موقعیت اسکرول DOM مفیده.
- **componentDidUpdate**: بیشتر برای به روزرسانی DOM در پاسخ به تغییرات state یا Prop استفاده میشه. این متد زمانی که **shouldComponentUpdate** مقدار **false** رو برگردونه قابل استفاده ست.
- **componentWillUnmount**: این متد برای کنسل کردن همه درخواست های شبکه خروجی یا حذف همه event listener های مرتبط با کامپوننت استفاده میشه.

قبل ورژن ۱۶.۳

- **componentWillMount**: قبل از رندر اجرا میشه و برای پیکربندی سطح برنامه توی کامپوننت ریشه استفاده میشه.
- **componentDidMount**: بعد از اولین رندر اجرا میشه و همه درخواست های AJAX، DOM یا بروزرسانی state و تنظیمات event listeners اجرا میشه.
- **componentWillReceiveProps**: Executed when particular prop updates to trigger state transitions

- **shouldComponentUpdate**: تعیین می‌کند که کامپوننت به روز بشه یا نه. به طور پیش فرض مقدار `true` رو برمی‌گردونه. اگه مطمئن باشیم که کامپوننت بعد از اینکه state یا props به روزرسانی میشه نیازی به رندر شدن نداره، می‌تونیم مقدار `false` رو برگردونیم. اینجا جای خوبی برای بهبود عملکرده چون این امکان رو بهمون میده که اگه کامپوننت prop جدید میگیره از render مجدد جلوگیری کنیم.
- **componentWillUpdate**: قبل از رندر مجدد کامپوننت وقتی که تغییرات state و props توسط `shouldComponentUpdate` مقدار `true` رو برگردونده باشه اجرا میشه.
- **componentDidUpdate**: بیشتر برای به روزرسانی DOM در پاسخ به تغییرات state یا Prop استفاده میشه. این متد زمانی که `shouldComponentUpdate` مقدار `false` رو برگردونه قابل استفاده ست.
- **componentWillUnmount**: این متد برای کنسل کردن همه درخواست های شبکه خروجی یا حذف همه event listener های مرتبط با کامپوننت استفاده میشه.

↑ فهرست مطالب

۳۵. کامپوننت‌های Higher-Order چی هستن؟

کامپوننت با اولویت بالا (HOC) تابعیه که یه کامپوننت میگیره و یه کامپوننت جدید برمی‌گردونه. اصولاً این الگویی که از ماهیت تلفیقی ری‌اکت گرفته شده. ما اینا رو به عنوان کامپوننت های خالص میشناسیم چون میتونن هر کدوم از کامپوننت های فرزندشون رو که به صورت پویا ارائه شدن بپذیرن ولی هیچ کدوم از رفتارهای کامپوننت های ورودی خودشون رو تغییر نمیدن.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

HOC خیلی جاها میتونه استفاده بشه:

۱. استفاده مجدد از کد، منطق و مفهوم bootstrap.

۲. Render hijacking.

۳. مفهوم state و manipulation.

۴. Props manipulation.

↑ فهرست مطالب

۳۶. چطوری می‌تونیم props proxy برای کامپوننت‌های HOC ایجاد کنیم؟

می‌تونیم prop های انتقال داده شده به کامپوننت رو با استفاده از الگوی *props proxy* اضافه یا ویرایش کنیم:

```
function HOC(WrappedComponent) {  
  return class Test extends Component {  
    render() {  
      const newProps = {  
        title: "New Header",  
        footer: false,  
        showFeatureX: false,  
        showFeatureY: true,  
      };  
  
      return <WrappedComponent {...this.props} {...newProps} />;  
    }  
  };  
}
```

[↑ فهرست مطالب](#)

۳۷. context چیه؟

Context روشی رو برای انتقال داده ها بین کامپوننت ها فراهم می‌کنه بدون اینکه بخوایم توی هر سطح به صورت دستی داده ها رو منتقل کنیم. به عنوان مثال، معتبر بودن کاربر، locale preference، UI theme مواردی هستن که توی خیلی از کامپوننت ها باید در دسترس باشن.

```
const { Provider, Consumer } = React.createContext(defaultValue);
```

[↑ فهرست مطالب](#)

۳۸. children prop چیه؟

children (به `prop (this.props.children)` هستش که بهمون اجازه میده کامپوننت ها رو به عنوان داده به کامپوننت های دیگه انقال بدیم، درست مثل `prop` های دیگه‌ای که

استفاده میکنیم. درخت کامپوننت که بین تگ باز و بسته کامپوننت ها قرار داره به اون کامپوننت به عنوان `children prop` پاس داده میشه. تعدادی متد در دسترس توی `react API` برای کار با این `prop` ها وجود داره که شامل `React.Children.map` ، `React.Children.forEach` ، `React.Children.count` ، `React.Children.only` ، `React.Children.toArray` هستش. یه مثال ساده از استفاده از این `children prop` این پایین نوشته شده.

```
const MyDiv = React.createClass({
  render: function () {
    return <div>{this.props.children}</div>;
  },
});

ReactDOM.render(
  <MyDiv>
    <span>{"Hello"}</span>
    <span>{"World"}</span>
  </MyDiv>,
  node
);
```

[↑ فهرست مطالب](#)

۳۹. چطوری میشه تو React کامنت نوشت؟

کامنت ها توی `React/JSX` شبیه به جاوااسکریپت هستن اما کامنت های چند خطی توی آکولاد قرار میگیرن.

کامنت های تک خطی:

```
<div>
  { /* Single-line comments(In vanilla JavaScript, the single-line comment
    {`Welcome ${user}, let's play React`}
  </div>
```

کامنت های چند خطی:

```
<div>
  { /* Multi-line comments for more than
    one line */}
  {`Welcome ${user}, let's play React`}
</div>
```

۴۰. چرا توی کامپوننت‌های کلاس باید توی constructor تابع super رو با مقدار props صدا بزنیم؟

کلاس constructor تا زمانی که متد `super` صدا زده نشده نمیتونه از `this` استفاده کنه. همین مورد در رابطه با کلاس‌های فرعی ES6 هم صدق می‌کنه. دلیل اصلی انتقال پارامترهای `props` به متد فراخوان `super` دسترسی داشتن به `this.props` توی constructor هستش.

با پاس دادن `props`:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    console.log(this.props); // prints { name: 'John', age: 42 }
  }
}
```

بدون پاس داده شدن `props`:

```
class MyComponent extends React.Component {
  constructor(props) {
    super();

    console.log(this.props); // prints undefined

    // but props parameter is still available
    console.log(props); // prints { name: 'John', age: 42 }
  }

  render() {
    // no difference outside constructor
    console.log(this.props); // prints { name: 'John', age: 42 }
  }
}
```

کد بالا نشون میده که `this.props` فقط توی constructor متفاوت، بیرون از constructor میتونه مثل همه.

۴۱. reconciliation چیست؟

وقتی state یا props به کامپوننت تغییرمی‌کنه، ری‌اکت با مقایسه عنصر تازه return شده و نمونه render شده قبلی تصمیم میگیره که به روزرسانی DOM واقعا ضروریه یا نه. وقتی این دو مقدار با هم برابر نباشه، ری‌اکت به روزرسانی DOM رو انجام میده. به این فرایند *reconciliation* می‌گیم.

↑ فهرست مطالب

۴۲. چطوری با یه اسم داینامیک set state کنیم؟

اگر برای تبدیل کد JSX از ES6 یا babel استفاده میکنین میتونین این کارو با *computed property names* انجام بدین.

```
handleInputChange(event) {  
  this.setState({ [event.target.id]: event.target.value })  
}
```

↑ فهرست مطالب

۴۳. یه اشتباه رایج برای مدیریت توابع event ها که باعث میشه با هر رندر توابع مجدد ساخته بشن چی هستش؟

باید مطمئن باشیم که موقع انتقال یه تابع به عنوان پارامتر تابع صدا زده نمیشه.

```
render() {  
  // Wrong: handleClick is called instead of passed as a reference!  
  return <button onClick={this.handleClick()}>{'Click Me'}</button>  
}
```

در عوض تابع رو بدون پرانتز انتقال بدین:

```
render() {  
  // Correct: handleClick is passed as a reference!  
  return <button onClick={this.handleClick}>{'Click Me'}</button>  
}
```

↑ فهرست مطالب

۴۴. تابع lazy که برای lazy load استفاده میشه رو می‌تونیم به صورت name export خروجی بگیریم؟

نه، تابع `React.lazy` در حال حاضر فقط خروجی پیش فرض رو پشتیبانی می‌کنه. اگه بخوایم ماژول‌هایی رو `import` کنیم که به اونا `exports` گفته میشه، می‌تونیم یه ماژول واسطه تعریف کنیم که اونا رو به عنوان پیش فرض مجدداً تعریف می‌کنه. همچنین تضمین می‌کنه که `tree shaking` همچنان به کار خودش ادامه میده و کامپوننت‌های استفاده نشده رو نمیگیره. بیا این یه کامپوننتی بنویسیم که چندین کامپوننت رو به عنوان خروجی ارائه میده.

```
// MoreComponents.js
export const SomeComponent = /* ... */;
export const UnusedComponent = /* ... */;
```

و کامپوننت `MoreComponents.js` رو در فایل واسطه `IntermediateComponent.js` مجدداً به عنوان خروجی تعریف کنیم.

```
// IntermediateComponent.js
export { SomeComponent as default } from './MoreComponents.js';
```

حالا می‌تونیم ماژول رو با استفاده از تابع `lazy` مثل زیر `import` کنیم.

```
import React, { lazy } from 'react';
const SomeComponent = lazy(() => import('./IntermediateComponent.js'));
```

↑ فهرست مطالب

۴۵. چرا ری‌اکت از `className` بجای `class` استفاده می‌کنه؟

`class` یه کلمه کلیدی توی جاوااسکریپت و `JSX` پسوند جاوااسکریپتیه. دلیل اصلی افتاده ری‌اکت از `className` به جای `class` همینه. یه رشته رو به عنوان `prop` `className` انتقال میدیم.

```
render() {
  return <span className={'menu navigation-menu'}>{'Menu'}</span>
}
```

↑ فهرست مطالب

۴۶. fragment ها چی هستن؟

یه الگوی رایج توی ری اکت وجود داره که برای کامپوننتی استفاده میشه که چندین عنصر رو برمیگردونن. *Fragment* ها این امکان رو فراهم میکنن که بتونیم لیستی از فرزندان رو بدون اضافه کردن نود های اضافی به DOM گروه بندی کنیم.

```
render() {  
  return (  
    <React.Fragment>  
      <ChildA />  
      <ChildB />  
      <ChildC />  
    </React.Fragment>  
  )  
}
```

همچنین یه *shorter syntax* وجود داره اما توی خیلی از ابزار ها پشتیبانی نمیشه:

```
render() {  
  return (  
    <>  
      <ChildA />  
      <ChildB />  
      <ChildC />  
    </>  
  )  
}
```

[↑ فهرست مطالب](#)

۴۷. چرا fragment ها از تگ های div بهترن؟

۱. *Fragment* ها یه کم سریعترن و با ایجاد نکردن *DOM node* اضافی حافظه کمتری استفاده میکنن. این فقط یه مزیت واقعی روی درخت های بزرگ و عمیق داره. بعضی از مکانیسم های *CSS* مثل *Flexbox* و *CSS Grid* روابط والد و فرزندی خاصی دارند و اضافه کردن *div* در وسط ، حفظ طرح مورد نظرمون را دشوار میکنه.
۲. *DOM Inspector* بهم ریختگی کمتری داره.

[↑ فهرست مطالب](#)

۴۸. توی ری‌اکت portal ها چیکار می‌کنن؟

Portal روشی توصیه شده برای رندر کردن فرزند توی DOM هستش و خارج از سلسله مراتب DOM از کامپوننت والد وجود داره.

```
ReactDOM.createPortal(child, container);
```

اولین آرگومان یه فرزند قابل رندر شدن هستش، مثل عنصر، رشته، یا `fragment`. آرگومان دوم عنصر DOM هستش.

↑ فهرست مطالب

۴۹. کامپوننت stateless چیه؟

اگه رفتار یه کامپوننت مستقل از `state` اون کامپوننت باشه بهش کامپوننت `stateless` می‌گیم. می‌تونیم از یه تابع یا یه کلاس برای ساخت کامپوننت های `stateless` استفاده کنیم، اما تا زمانی که ما نیاز داریم از هوک چرخه عمر توی کامپوننت هامون استفاده کنیم باید سراغ کامپوننت های تابع بریم. اگه بخوایم اینجا از کامپوننت های تابع استفاده کنیم فواید زیادی داره، راحت نوشته میشه، فهمیده میشه، تست میشه، سریعتره و می‌تونیم از کلمه کلیدی `this` هم استفاده نکنیم.

↑ فهرست مطالب

۵۰. کامپوننت stateful چیه؟

اگه رفتار یه کامپوننتی به `state` اون کامپوننت وابسته باشه، به عنوان کامپوننت `statefull` شناخته میشه. این کامپوننت ها همیشه جز کلاس کامپوننت ها هستن و شامل یه `state` هستن که توی `constructor` یه مقدار اولیه بهش میدیم.

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    // ...
  }
}
```

ورژن 16.8 ری‌اکت:

هوک‌ها این امکان رو بهمون میدن که بدون نوشتن کلاس‌ها بتونیم از state و ویژگی‌های دیگه ری‌اکت استفاده کنیم.
کامپوننت‌های Equivalent Functional

```
import React, {useState} from 'react';

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

  return (
    // JSX
  )
}
```

↑ فهرست مطالب

۵۱. چطوری prop‌های کامپوننت رو اعتبارسنجی کنیم؟

وقتی برنامه توی حالت *development* یا توسعه هست، ری‌اکت به شکل خودکار تمام prop‌هایی که ما توی کامپوننت استفاده کردیم رو چک می‌کنه تا مطمئن بشه همه‌شون type درستی دارن. اگه هر کدوم از prop‌ها *type* درستی نداشته باشن توی کنسول بهمون warning نشون میده، البته توی حالت *production* این حالت غیر فعاله. prop‌های اجباری با پراپرتی *isRequired* مشخص میشن، همچنین یه سری انواع prop پیش‌فرض وجود دارن که پایین می‌آریمشون :

۱. **PropTypes.number**

۲. **PropTypes.string**

۳. **PropTypes.array**

۴. **PropTypes.object**

PropTypes.func .۵

PropTypes.node .۶

PropTypes.element .۷

PropTypes.bool .۸

PropTypes.symbol .۹

PropTypes.any .۱۰

PropTypes ها رو برای یه کامپوننت تستی به اسم User اینطوری میشه تعریف کرد :

```
import React from "react";
import PropTypes from "prop-types";

class User extends React.Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    age: PropTypes.number.isRequired,
  };

  render() {
    return (
      <>
        <h1>`Welcome, ${this.props.name}`</h1>
        <h2>`Age, ${this.props.age}`</h2>
      </>
    );
  }
}
```

نکته: در ورژن 15.5 ری اکت *propTypes* ها از `React.PropTypes` به کتابخونه `prop-types` انتقال پیدا کردن.

↑ فهرست مطالب

۵۲. مزایای React چیه؟

۱. افزایش عملکرد برنامه با *Virtual DOM*.
۲. خوندن و نوشتن راحتتر کد ها با JSX.
۳. رندر کردن در هر دو سمت کاربر و سرور (*SSR*).
۴. ادغام راحت با فریم ورک ها (*Angular, Backbone*).
۵. امکان نوشتن تست های واحد یا ادغام شده از طریق ابزارهایی مثل *Jest*.

۵۳. محدودیت‌های React چیه؟

۱. ری‌اکت یک کتابخانه برای ساخت لایه view هستش نه یک فریمورک کامل.
۲. وجود یک منحنی یادگیری (سختی یادگیری) برای کسانی که به تازگی می‌خوان برنامه نویسی وب رو یاد بگیرن.
۳. یکپارچه‌سازی ری‌اکت در فریمورک‌های مبتنی بر MVC به یه کانفیگ اضافه‌ای نیاز داره.
۴. پیچیدگی کد با inline templating و JSX افزایش پیدامی‌کنه.
۵. خیلی کامپوننت‌های کوچیک یا boilerplate‌های کوچیک براش ساخته شدن و ممکنه کمی گیج‌کننده باشه.

۵۴. error boundary ها توی ری‌اکت نسخه 16 چیکار می‌کنن؟

Error boundary ها یا به اصطلاح تحت الفظی مرزهای خطا کامپوننت‌هایی هستن که خطاهای جاوااسکریپت رو هرجایی توی درخت فرزندان رخ داده باشن catch میکنن و خطای موردنظر رو log میکنن و علاوه براین می‌تونن یه UI به اصطلاح fallback رو بجای کامپوننت crash شده نشون بدن.

توی یه کلاس کامپوننت با گذاشتن متد `componentDidCatch(error, info)` یا `static getDerivedStateFromError` می‌تونیم یه boundary برای زمانی که خطایی رخ میده درست کنیم. مثل:

```

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  componentDidCatch(error, info) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, info);
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>{"Something went wrong."}</h1>;
    }
    return this.props.children;
  }
}

```

بعدشم همیشه ارزش مثل یه کامپوننت عادی استفاده کرد:

```

<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>

```

نکته : از این ویژگی توی کامپوننت‌های functional همیشه استفاده کرد و در حقیقت احتمالاً نیازی هم بهش ندارین، چون اکثر مواقع برای کل برنامه یه error boundary تعریف می‌کنیم که می‌تونه try..catch باشه.

[↑ فهرست مطالب](#)

۵۵. چطوری از error boundary ها توی نسخه ۱۵ ریکت مدیریت شدن؟

ری‌اکت توی نسخه 15 با استفاده از متد `unstable_handleError` error boundary ها رو مدیریت کرده .

این متد توی نسخه 16 به `componentDidCatch` تغییر کرده.

↑ فهرست مطالب

۵۶. روش‌های پیشنهادی برای type checking چیه؟

به طور معمول ما از کتابخانه `propTypes` ها (در ورژن ۱۵.۵ ری‌اکت `React.PropTypes` به پکیج `react-types` انتقال پیدا کرده) برای چک کردن نوع `prop` در برنامه‌های ری‌اکت استفاده می‌کنیم. برای برنامه‌ای با کدهای بیشتر توصیه میشه از `static type_checker` هایی مثل `flow` یا `typescript` استفاده بشه که چک کردن رو در زمان کامپایل انجام میده و ویژگی‌های مثل `auto-completion` رو ارائه میده.

↑ فهرست مطالب

۵۷. کاربرد پکیج react-dom چیه؟

پکیج `react-dom` متدهای `DOM-specific` یا مخصوص `DOM` رو ارائه میده که میتونه توی سطوح بالای برنامه شما استفاده بشه. اکثر کامپوننت‌ها نیازی به استفاده از این ماثول‌ها ندارن. تعدادی از متدهای این پکیج این‌ها هستن:

۱. متد `render`

۲. متد `hydrate`

۳. متد `unmountComponentAtNode`

۴. متد `findDOMNode`

۵. متد `createPortal`

↑ فهرست مطالب

۵۸. کاربرد متد render از پکیج react-dom چیه؟

این متد برای رندرکردن کامپوننت پاس داده شده، توی یه المنت `DOM` که به عنوان `container` پاس داده شده استفاده میشه و یه رفرنس به کامپوننت برمی‌گردونه. اگه کامپوننت ری‌اکت قبلاً توی `container` مورد نظر رندر شده باشه با یه `update` فقط `DOM`هایی که نیاز به به روز شدن دارن رو رندر می‌کنه.

```
ReactDOM.render(element, container[, callback])
```

اگر پارامتر سوم که به `callback` هست پاس داده بشه، هر موقع که رندر یا به‌روزرسانی انجام بشه اون تابع هم اجرا میشه.

[↑ فهرست مطالب](#)

۵۹. `ReactDOMServer` چیه؟

`ReactDOMServer` این امکان رو بهمون میده که کامپوننت‌ها رو به صورت استاتیک رندر کنیم (معمولا روی `node server` استفاده میشه). `ReactDOMServer` عمدتا برای پیاده سازی سمت سرور استفاده میشه (SSR).

۱. متد `renderToString`

۲. متد `renderToStaticMarkup`

برای مثال ممکنه یه سرور روی `node` بسازین که ممکنه `Express`، `Hapi` یا `Koa` باشه و متد `renderToString` رو برای تبدیل کردن کامپوننت `root` به `html` اجرا کنید و نتیجه بدست اومده رو به عنوان `response` به کلاینت پاس بدین.

```
// using Express
import { renderToString } from "react-dom/server";
import MyPage from "./MyPage";

app.get("/", (req, res) => {
  res.write(
    "<!DOCTYPE html><html><head><title>My Page</title></head><body>"
  );
  res.write('<div id="content">');
  res.write(renderToString(<MyPage />));
  res.write("</div></body></html>");
  res.end();
});
```

[↑ فهرست مطالب](#)

۶۰. چطوری از `innerHTML` توی ری‌اکت استفاده کنیم؟

ویژگی `dangerouslySetInnerHTML` جایگزین ری‌اکت واسه استفاده از `innerHTML` توی `DOM` مرورگره و کارکردش درست مثل `innerHTML` هستش، استفاده از این ویژگی

به خاطر حملات XSS (cross-site-scripting) ریسک بالایی داره. برای این کار باید به آبجکت `innerHTML` به عنوان `key` و به متن `html` به عنوان `value` به این `prop` بفرستیم (یا شاید همون پاس بدیم). توی مثال پایینی کامپوننت از ویژگی `dangerouslySetInnerHTML` برای قرار دادن HTML استفاده کرده.

```
function createMarkup() {  
  return { __html: "First &middot; Second" };  
}  
  
function MyComponent() {  
  return <div dangerouslySetInnerHTML={createMarkup()} />;  
}
```

↑ فهرست مطالب

۶۱. چطوری توی ری اکت استایل دهی می کنیم؟

`attribute` پیش فرض مورد استفاده برای استایل دهی `style` هستش که به `object` جاوااسکریپت رو به عنوان مقدار قبول می کنه که همه `property` های اون بجای `css` عادی `camelCase` هستن. این روش با استایل دهی عادی توی جاوااسکریپت به کم متفاوت و بهینه تر و امن تره، چون جلوی حفره های امنیتی مثل XSS رو میگیره.

```
const divStyle = {  
  color: "blue",  
  backgroundImage: "url(" + imgUrl + ")",  
};  
  
function HelloWorldComponent() {  
  return <div style={divStyle}>Hello World!</div>;  
}
```

↑ فهرست مطالب

۶۲. تفاوت event های ری اکت چیه؟

رویدادهای `handling` در المان های ری اکت به سری تفاوت های نحوی دارن :
۱. `event handler` های ری اکت به جای حروف کوچک به صورت حروف بزرگ نامگذاری شدن.

۲. با JSX ما به تابع رو به جای رشته به عنوان event handler پاس میدیم.

↑ فهرست مطالب

۶۳. اگه توی constructor بیاییم و setState کنیم چی میشه؟

وقتی از `setState` استفاده می‌کنیم، جدا از اینکه به یه آبجکت استیتی اختصاص داده میشه ری‌اکت اون کامپوننت و همه فرزندان اون کامپوننت رو دوباره رندر می‌کنه. ممکنه این ارور رو بگیرین: شما فقط می‌تونید کامپوننت `mount` شده یا در حال `mount` رو به روز رسانی کنید. پس باید بجای `setState` از `this.state` برای مقداردهی state توی constructor استفاده کنیم.

↑ فهرست مطالب

۶۴. تاثیر استفاده از ایندکس به عنوان key چیه؟

key ها باید پایدار، قابل پیش بینی و منحصر به فرد باشن تا ری‌اکت بتونه المان‌ها رو قابل رهگیری کنه. تو کد زیر key هر عنصر براساس ترتیبی که توی لیست داره مقدار قرار می‌گیره و به داده‌هایی که میگیرن ربطی نداره. این کار بهینه سازی‌هایی که می‌تونه توسط ری‌اکت انجام بشه رو محدود می‌کنه.

```
todos.map((todo, index) => <Todo {...todo} key={index} />);
```

اگه از داده‌های همون element به عنوان کلید بخوایم استفاده کنیم، مثلاً `todo.id`. چونکه همه ویژگی‌هایی که یه کلید باید داشته باشه رو داره، هم استیبله و هم منحصر به فرد، توی این حالت ری‌اکت می‌تونه بدون اینکه لازم باشه دوباره همه المنت‌ها رو ارزیابی کنه رندر رو انجام بده.

```
todos.map((todo) => <Todo {...todo} key={todo.id} />);
```

↑ فهرست مطالب

۶۵. نظرت راجع به استفاده از setState توی متد componentWillMount چیه؟

توصیه همیشه که از مقدار دهی اولیه غیر هم زمان در متد `componentWillMount` استفاده نشه. `componentWillMount` درست قبل از `mount` شدن اجرا میشه و قبل از متد `render` صدا زده میشه بنابراین `setState` کردن توی این متد باعث `re-render` شدن نمیشه. باید از ایجاد هر سایه افکتی توی این متد خودداری کنیم و دقت کنیم که اگه مقدار دهی اولیه غیر هم زمانی داریم این کار رو توی متد `componentDidMount` انجام بدیم نه در متد `componentWillMount`.

```
componentDidMount() {
  axios.get(`api/todos`)
    .then((result) => {
      this.setState({
        messages: [...result.data]
      })
    })
}
```

↑ فهرست مطالب

۶۶. اگه از `prop` توی مقداردهی اولیه `state` استفاده کنیم چی میشه؟

اگه `prop` های یه کامپوننت بدون اینکه اون کامپوننت رفرش بشه تغییر کنه، مقدار جدید اون `prop` نمایش داده نمیشه چون تابع `constructor`، `state` جاری اون کامپوننت رو به روز رسانی نمیکنه، مقدار دهی اولیه `state` از `prop` ها فقط زمانی که کامپوننت برای بار اول ساخته شده اجرا میشه. کامپوننت زیر مقدار به روزرسانی شده رو نشون نمیده :

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      records: [],
      inputValue: this.props.inputValue,
    };
  }

  render() {
    return <div>{this.state.inputValue}</div>;
  }
}
```

استفاده از prop ها توی متد render مقدار رو به روز رسانی می‌کنه :

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      record: [],
    };
  }

  render() {
    return <div>{this.props.inputValue}</div>;
  }
}
```

[↑ فهرست مطالب](#)

۶۷. چطوری کامپوننت رو با بررسی یه شرط رندر می‌کنیم؟

بعضی وقتا ما می‌خوایم کامپوننت‌های مختلفی رو بسته به بعضی state ها رندر کنیم. JSX مقدار `false` یا `undefined` رو رندر نمی‌کنه، بنابراین ما می‌تونیم از `short-circuiting` شرطی برای رندر کردن بخش مشخصی از کامپوننتتون استفاده کنیم در صورتی که اون شرط مقدار `true` رو برگردونده باشه.

```
const MyComponent = ({ name, address }) => (
  <div>
    <h2>{name}</h2>
    {address && <p>{address}</p>}
  </div>
);
```

اگه به یه شرط `if-else` نیاز داریم از `ternary operator` استفاده کنیم.

```
const MyComponent = ({ name, address }) => (
  <div>
    <h2>{name}</h2>
    {address ? <p>{address}</p> : <p>{"Address is not available"}</p>}
  </div>
);
```

[↑ فهرست مطالب](#)

۶۸. چرا وقتی prop ها رو روی یه DOM Elemnt می‌آییم spread می‌کنیم باید مراقب باشیم؟

وقتی ما prop ها رو spread می‌کنیم این کار با ریسک اضافه کردن اتریبیوت‌های HTML انجام میدیم که این کار خوبی نیست، به جای این کار می‌تونیم از `rest...` استفاده کنیم که فقط prop های مورد نیاز رو اضافه می‌کنه.

```
const ComponentA = () => (  
  <ComponentB isDisplay={true} className={"componentStyle"} />  
);  
  
const ComponentB = ({ isDisplay, ...domProps }) => (  
  <div {...domProps}>{"ComponentB"}</div>  
);
```

[↑ فهرست مطالب](#)

۶۹. چطوری از decorator ها توی ری‌اکت استفاده کنیم؟

می‌تونیم کلاس کامپوننت ها رو *decorate* کنیم، که درست مثل پاس دادن کامپوننت ها به تابع هستش. **Decorator** ها روش قابل خواندن و انعطاف پذیرتری برای تغییر فانکشنالیتی کامپوننت ها هستن.

```

@setTitle("Profile")
class Profile extends React.Component {
  //....
}

/*
  title is a string that will be set as a document title
  WrappedComponent is what our decorator will receive when
  put directly above a component class as seen in the example above
*/
const setTitle = (title) => (WrappedComponent) => {
  return class extends React.Component {
    componentDidMount() {
      document.title = title;
    }

    render() {
      return <WrappedComponent {...this.props} />;
    }
  };
};

```

نکته: Decorator ها ویژگی‌هایی هستند که در حال حاضر به ES7 اضافه نشدن، ولی توی پیشنهاد stage 2 هستند.

[↑ فهرست مطالب](#)

۷۰. چطوری یه کامپوننت رو memoize می‌کنیم؟

در حال حاضر کتابخانه‌هایی وجود داره که memoize هستند و میتونن توی کامپوننت‌های تابع استفاده بشن، به عنوان مثال کتابخونه `moize` میتونه یه کامپوننت رو توی بقیه کامپوننت ها memoize کنه.

```

import moize from "moize";
import Component from "../components/Component"; // this module exports a

const MemoizedFoo = moize.react(Component);

const Consumer = () => {
  <div>
    {"I will memoize the following entry:"}
    <MemoizedFoo />
  </div>;
};

```

به روز رسانی: توی ورژن 16.6.0 ری اکت، `React.memo` رو داریم که کارش اینه که یه کامپوننت با الویت بالاتر فراهم می‌کنه که کامپوننت رو تا زمانی که prop ها تغییر کنن memoize می‌کنه. برای استفاده ازش کافیه زمان ساخت کامپوننت از `React.memo` استفاده کنیم.

```
const MemoComponent = React.memo(function MemoComponent(props) {  
  /* render using props */  
});  
// OR  
export default React.memo(MyFunctionComponent);
```

↑ فهرست مطالب

۷۱. چطوری باید Server-Side Rendering یا SSR رو توی ری اکت پیاده کنیم؟

ری اکت در حال حاضر به رندر سمت نود سرور مجهزه، یه ورژن خاصی از DOM رندر در دسترسه که دقیقاً از همون الگوی سمت کاربر پیروی می‌کنه.

```
import ReactDOMServer from "react-dom/server";  
import App from "./App";  
  
ReactDOMServer.renderToString(<App />);
```

خروجی این روش یه HTML معمولی به عنوان رشته ست که داخل body صفحه به عنوان ریسپانس سرور قرار می‌گیره. در سمت کاربر، ری اکت محتوای از قبل رندر شده رو تشخیص میده و به صورت یکپارچه اونا رو انتخاب می‌کنه.

↑ فهرست مطالب

۷۲. چطوری حالت production رو برای ری اکت فعال کنیم؟

میشه از پلاگین `DefinePlugin` که روی وب‌پک قابل استفاده هست استفاده کرد و مقدار `NODE_ENV` رو روی `production` ست کرد، با اینکار خطاهای اضافی یا اعتبارسنجی `propTypes` ها روی پروداکشن غیرفعال میشه جدای این موارد، کدهای نوشته

شده بهینه‌سازی می‌شن و مثلاً کدهای بلااستفاده اینا حذف می‌شن و در نتیجه سرعت بهتری رو می‌توته به برنامه بده چون سایز bundle ایجاد شده کوچیکتر خواهد بود.

↑ فهرست مطالب

۷۳. CRA چیه و چه مزایایی داره؟

ابزار `create-react-app` CLI این امکان رو بهمون میده که برنامه‌های ری‌اکت رو سریع و بدون مراحل پی‌کردنی بسازیم و اجرا کنیم. حالا بیاین برنامه `Todo` رو با استفاده از `CRA` بسازیم :

```
# Installation
$ npm install -g create-react-app

# Create new project
$ create-react-app todo-app
$ cd todo-app

# Build, test and run
$ npm run build
$ npm run test
$ npm start
```

این شامل همه اون چیزیه که ما واسه ساختن یه برنامه ری‌اکت لازم داریم :

۱. `ES6`، `JSX`، `React` و روند پشتیبانی `syntax`.
۲. `Language extras beyond ES6 like the object spread operator`.
۳. `Autoprefixed CSS`، بنابراین نیازی به `-webkit` یا پیشنوندهای دیگه‌ای نداریم.
۴. یه اجرا کننده تست تعاملی با پشتیبانی داخلی برای `coverage reporting`.
۵. یه سرور `live development` که اشتباهات معمول رو بهمون هشدار میده.
۶. یه بیلد اسکریپت برای باندل کردن `js`، `css` و تصاویر برای `production` همراه با `hash` ها و `sourcemap`.

↑ فهرست مطالب

۷۴. ترتیب اجرا شدن متدهای `life cycle` چطوره؟

وقتی به نمونه‌ای از کامپوننت ساخته میشه و داخل DOM اضافه میشه، متدهای lifecycle به ترتیب زیر صدا زده میشن.

۱. متد `constructor`

۲. متد `static getDerivedStateFromProps`

۳. متد `render`

۴. متد `componentDidMount`

↑ فهرست مطالب

۷۵. کدوم متدهای life cycle توی نسخه 16 ری اکت منسوخ شدن؟

متدهای lifecycle روش‌های ناامن کدنویسی هستن و با رندر `async` مشکل بیشتری پیدا میکنن.

۱. متد `componentWillMount`

۲. متد `componentWillReceiveProps`

۳. متد `componentWillUpdate`

تو ورژن 16.3 ری اکت این متدها با پیشوند `_UNSAFE` متمایز شدن و نسخه اصلاح نشده تو ورژن 17 ری اکت حذف میشه.

↑ فهرست مطالب

۷۶. کاربرد متد `getDerivedStateFromProps` چیه؟

بعد از اینکه یه کامپوننت بلا فاصله به خوبی قبل `rerender` شد، متد جدید استاتیک `getDerivedStateFromProps` صدا زده میشه.

این متد یا `state` آپدیت شده رو به صورت یه آبجکت برمی گردونه یا `null` رو برمی گردونه که معنیش اینه `prop`های جدید به آپدیت شدن `state` نیازی ندارن.

```
class MyComponent extends React.Component {
  static getDerivedStateFromProps(props, state) {
    // ...
  }
}
```

متد `componentDidUpdate` تمام مواردی که توی متد `componentWillReceiveProps` هست رو پوشش میده.

[↑ فهرست مطالب](#)

۷۷. کاربرد متد `getSnapshotBeforeUpdate` چیه؟

متد جدید `getSnapshotBeforeUpdate` بعد از آپدیت‌های DOM صدا زده میشه. مقدار برگشتی این متد به عنوان پارامتر سوم به متد `componentDidUpdate` پاس داده میشه.

```
class MyComponent extends React.Component {
  getSnapshotBeforeUpdate(prevProps, prevState) {
    // ...
  }
}
```

متد `componentDidUpdate` تمام مواردی که توی متد `componentWillUpdate` استفاده میشه رو پوشش میده.

[↑ فهرست مطالب](#)

۷۸. آیا هوک‌ها جای `render props` و `HOC` رو می‌گیرن؟

کامپوننت‌های با اولویت بالاتر یا همون هوک‌ها و `render prop`‌ها هر دوشون فقط یه `child` رو رندر می‌کنن ولی هوک‌ها روش راحت‌تری رو ارائه میدن که از تو در تو بودن توی درخت کامپوننت‌ها جلوگیری می‌کنه.

[↑ فهرست مطالب](#)

۷۹. روش توضیحه شده برای نام‌گذاری کامپوننت‌ها چیه؟

برای نام‌گذاری کامپوننت‌ها توصیه میشه که از مرجع به جای `displayName` استفاده کنیم. استفاده از `displayName` برای نام‌گذاری کامپوننت:

```
export default React.createClass({
  displayName: "TodoApp",
  // ...
});
```

روش توصیه شده:

```
export default class TodoApp extends React.Component {
  // ...
}
```

[↑ فهرست مطالب](#)

۸.۰ روش توصیه شده برای ترتیب متدها در کلاس کامپوننت‌ها چیه؟

ترتیب توصیه شده متدها از *mounting* تا *render stage*:

۱. متدهای `static`
۲. متد `constructor`
۳. متد `getChildContext`
۴. متد `componentWillMount`
۵. متد `componentDidMount`
۶. متد `componentWillReceiveProps`
۷. متد `shouldComponentUpdate`
۸. متد `componentWillUpdate`
۹. متد `componentDidUpdate`
۱۰. متد `componentWillUnmount`
۱۱. `event handler` مثل `onClickSubmit` یا `onChangeDescription`
۱۲. متدهای دریافت کننده برای رندر مثل `getSelectReason` یا `getFooterContent`
۱۳. متدهای رندر اختیاری مثل `renderNavigation` یا `renderProfilePicture`
۱۴. متد `render`

[↑ فهرست مطالب](#)

۸.۱. کامپوننت تعویض کننده یا switching چیه؟

یه کامپوننت *switcher* کامپوننتی‌ه که یکی از چندتا کامپوننت موردنظر رو رندر می‌کنه. لازمه که برای تصمیم‌گیری بین کامپوننت‌ها از `object` جاوااسکریپتی استفاده کنیم. برای مثال، کدپایین با بررسی `prop` موردنظر `page` بین صفحات مختلف سوییچ می‌کنه:

```
import HomePage from "./HomePage";
import AboutPage from "./AboutPage";
import ServicesPage from "./ServicesPage";
import ContactPage from "./ContactPage";

const PAGES = {
  home: HomePage,
  about: AboutPage,
  services: ServicesPage,
  contact: ContactPage,
};

const Page = (props) => {
  const Handler = PAGES[props.page] || ContactPage;

  return <Handler {...props} />;
};

// The keys of the PAGES object can be used in the prop types to catch o
Page.propTypes = {
  page: PropTypes.oneOf(Object.keys(PAGES)).isRequired,
};
```

↑ فهرست مطالب

۸.۲. چرا نیاز میشه به تابع `setState` یه فانکشن `callback` پاس بدیم؟

دلیلش اینه که `setState` یه عملیات `async` یا ناهمزمانه. `state` ها در ری‌اکت به دلایل عملکردی تغییر می‌کنن، بنابراین یه `state` ممکنه بلافاصله بعد از اینکه `setState` صدا زده شد تغییر نکنه. یعنی اینکه وقتی `setState` رو صدا می‌زنیم نباید به `state` جاری اعتماد کنیم چون نمی‌تونیم مطمئن باشیم که اون `state` چی میتونه باشه. راه حلش اینه که یه تابع رو با `state` قبلی به عنوان یه آرگومان به `setState` پاس بدیم.

بیاین فرض کنیم مقدار اولیه count صفر هستش. بعد از سه عملیات پشت هم، مقدار count فقط یکی افزایش پیدا می‌کنه.

```
// assuming this.state.count === 0
this.setState({ count: this.state.count + 1 });
this.setState({ count: this.state.count + 1 });
this.setState({ count: this.state.count + 1 });
// this.state.count === 1, not 3
```

اگه ما به تابع به `setState` پاس بدیم، مقدار count به درستی افزایش پیدا می‌کنه.

```
this.setState((prevState, props) => ({
  count: prevState.count + props.increment,
}));
// this.state.count === 3 as expected
```

[↑ فهرست مطالب](#)

۸۳. حالت strict توی ری‌اکت چیکار می‌کنه؟

`React.StrictMode` یه کامپوننت مفید برای هایلایت کردن مشکلات احتمالی توی برنامه ست. `<StrictMode>` درست مثل `<Fragment>` هیچ المان DOM اضافه‌ای رو زدن نمی‌کنه، بلکه warning ها و additional checks رو برای فرزندان اون کامپوننت فعال می‌کنه. این کار فقط در حالت *development* فعال میشه.

```
import React from "react";

function ExampleApplication() {
  return (
    <div>
      <Header />
      <React.StrictMode>
        <div>
          <ComponentOne />
          <ComponentTwo />
        </div>
      </React.StrictMode>
      <Footer />
    </div>
  );
}
```

توی مثال بالا، *strict mode* فقط روی دو کامپوننت `<ComponentOne>` و `<ComponentTwo>` اعمال میشه.

↑ فهرست مطالب

۸۴. Mixin های ری اکت چی هستن؟

Mixin ها روشی برای جدا کردن کامپوننت‌هایی با عملکرد مشترک هستن. *Mixin* ها نباید استفاده بشن و میتونن با کامپوننت‌های با اولویت بالا (*HOC*) یا *decorator* ها جایگزین بشن. یکی از بیشترین کاربردهای *mixin* ها `PureRenderMixin` هستش. ممکنه تو بعضی از کامپوننت ها برای جلوگیری از *re-render* های غیر ضروری وقتی *prop* ها و *state* با مقادیر قبلی شون برابر هستن از این *mixin* ها استفاده کنیم:

```
const PureRenderMixin = require("react-addons-pure-render-mixin");

const Button = React.createClass({
  mixins: [PureRenderMixin],
  // ...
});
```

↑ فهرست مطالب

۸۵. چرا `isMounted` آنتی پترن هست و روش بهتر انجامش چیه؟

کاربرد اصلی متد `isMounted` برای جلوگیری از فراخوانی `setState` بعد از `unmount` شدن کامپوننت هستش چونکه باعث ایجاد یه خطا میشه. خطاش یه چیزی مثل اینه:

```
Warning: Can only update a mounted or mounting component. This u
...
```

```
</span>
```

توی کلاس کامپوننت‌ها هم این شکلی بعضا جلوشو می‌گرفتن

```
<span align="left" dir="ltr">
```

```
```javascript
if (this.isMounted()) {
 this.setState({...})
}
```

دلیل اینکه این روش توصیه نمیشه اینه که خطایی رو که ری‌اکت بهمون میده

البته توی نسخه‌های جدید ری‌اکت این کار رو خیلی ساده‌تر میشه انجام داد و فقط کافیه یه هوکی بنویسیم که یه `ref` رو مقداردهی می‌کنه و بعد با بررسی اون `ref` میشه فهمید که کامپوننت `mount` شده یا نه، مثلاً:

```
export const useIsMounted = () => {
 const componentIsMounted = useRef(true);
 useEffect(
 () => () => {
 componentIsMounted.current = false;
 },
 []
);
 return componentIsMounted;
};
```

یا حتی یه پکیجی ساخته شده به اسم `ismounted` که می‌تونه بهمون کمک کنه که متوجه بشیم کامپوننت `mount` شده یا نه. ولی حواسمون باشه که ازش درست استفاده کنیم.

[↑ فهرست مطالب](#)

## ۸۶. پشتیبانی ری‌اکت از `pointer event`ها چگونه؟

*pointer Event* ها به روش واحدی رو برای هندل کردن همه ی ایونت‌های ورودی ارائه میدن.

در زمان‌های قدیم ما از موس استفاده میکردیم و برای هندل کردن ایونت‌های مربوط به اون از event listener ها استفاده میکردیم ولی امروزه دستگاه‌های زیادی داریم که با داشتن موس ارتباطی ندارن، مثل قلم‌ها یا گوشی‌های صفحه لمسی.

باید یادمون باشه که این ایونت‌ها فقط تو مرورگرهایی کار میکنن که مشخصه *Pointer Events* رو پشتیبانی میکنن.

ایونت‌های زیر در *React DOM* در دسترس هستن:

۱. `onPointerDown`

۲. `onPointerMove`

۳. `onPointerUp`

۴. `onPointerCancel`

۵. `onGotPointerCapture`

۶. `onLostPointerCapture`

۷. `onPointerEnter`

۸. `onPointerLeave`

۹. `onPointerOver`

۱۰. `onPointerOut`

↑ فهرست مطالب

## ۸۷. چرا باید اسم کامپوننت با حرف بزرگ شروع بشه؟

اگه ما با استفاده از JSX کامپوننتمون رو رندر می‌کنیم، اسم کامپوننت باید با حرف بزرگ شروع بشه در غیر این صورت ری‌اکت خطای تگ غیر قابل تشخیص رو میده. این قرارداد به خاطر اینه که فقط عناصر HTML و تگ‌های SVG می‌تونن با حرف کوچک شروع بشن.

```
} class SomeComponent extends Component
Code goes here //
{
```

می‌تونیم کلاس کامپوننت‌هایی که با حرف کوچک شروع میشن رو هم تعریف کنیم ولی وقتی داریم ایمپورت می‌کنیم باید شامل حروف بزرگ هم باشن:

```
class myComponent extends Component {
 render() {
 return <div />;
 }
}

export default myComponent;
```

وقتی داریم تو یه فایل دیگه‌ای ایمپورت می‌کنیم باید با حرف بزرگ شروع بشه:

```
import MyComponent from "./MyComponent";
```

[↑ فهرست مطالب](#)

## ۸۸. آیا prop های custom توی ری‌اکت پشتیبانی میشن؟

بله. در گذشته ری‌اکت attribute های DOM های ناشناخته رو نادیده می‌گرفت، اگه JSX رو با یه ویژگی‌ای نوشته بودیم که ری‌اکت تشخیص نمیداد، اونو نادیده می‌گرفت. به عنوان مثال:

```
<div mycustomattribute={"something"} />
```

در ری‌اکت ورژن 15 یه div خالی توی DOM رندر می‌کنیم:

```
<div />
```

در ری‌اکت ورژن 16 هر attribute ناشناخته‌ای توی DOM از بین میره:

```
<div mycustomattribute="something" />
```

این برای attribute های غیر استاندارد مرورگرهای خاص، DOM API های جدید و ادغام با کتابخانه‌های third-party مفیده.

[↑ فهرست مطالب](#)

## ۸۹. تفاوت‌های constructor و getInitialState چیه؟

وقتی داریم از کلاس‌های ES6 استفاده می‌کنیم باید state رو توی constructor مقاردهی اولیه کنیم و وقتی از `React.createClass` استفاده می‌کنیم باید از متد

`getInitialState` استفاده کنیم.

استفاده از کلاس‌های ES6:

```
class MyComponent extends React.Component {
 constructor(props) {
 super(props);
 this.state = {
 /* initial state */
 };
 }
}
```

استفاده از `React.createClass`:

```
const MyComponent = React.createClass({
 getInitialState() {
 return {
 /* initial state */
 };
 },
});
```

**نکته:** `React.createClass` در ورژن 16 ری‌اکت حذف شده و به جای اون همیشه از کلاس‌های ساده جاوا اسکریپت استفاده کرد.

[↑ فهرست مطالب](#)

## ۹۰. می‌تونیم به کامپوننت رو بدون `setState` ری‌رندر کنیم؟

در حالت پیش فرض، وقتی `state` یا `prop` کامپوننت تغییرمی‌کنه، کامپوننت دوباره رندر میشه. اگه متد `render` به داده‌های دیگه‌ای وابسته باشه، می‌تونیم با فراخوانی متد `forceUpdate` به ری‌اکت بگیم که این کامپوننت نیازه که دوباره رندر بشه.

```
component.forceUpdate(callback);
```

توصیه میشه که از متد `forceUpdate` استفاده نکنیم و توی `render` فقط از `this.props` و `this.state` استفاده کنیم.

[↑ فهرست مطالب](#)

## ۹۱. تفاوت‌های فراخوانی `super()` و `super(props)` توی کلاس کامپوننت‌های ری‌اکت چیه؟

اگه بخوایم به `this.props` توی `constructor` دسترسی پیدا کنیم باید `prop` ها رو از طریق متد `super` پاس بدیم. استفاده از `super(props)` :

```
class MyComponent extends React.Component {
 constructor(props) {
 super(props);
 console.log(this.props); // { name: 'John', ... }
 }
}
```

استفاده از `super` :

```
class MyComponent extends React.Component {
 constructor(props) {
 super();
 console.log(this.props); // undefined
 }
}
```

بیرون از `constructor` هردو متد مقادیر یکسانی رو برای `this.props` نشون میدن.

[↑ فهرست مطالب](#)

## ۹۲. چطوری توی `JSX` حلقه یا همون لوپ رو داشته باشیم؟

خیلی ساده می‌تونیم از `Array.prototype.map` با سینتکس `ES6 arrow function` استفاده کنیم، برای مثال آرایه‌ای از آیتم‌های یه آبجکت توی آرایه‌ای از کامپوننت‌ها نوشته میشه:

```
<tbody>
 {items.map((item) => (
 <SomeComponent key={item.id} name={item.name} />
))}
</tbody>
```

می‌تونیم با استفاده از حلقه `for` تکرار رو انجام بدیم:

```
<tbody>
 for (let i = 0; i < items.length; i++) {
 <SomeComponent key={items[i].id} name={items[i].name} />
 }
</tbody>
```

به خاطر اینکه تگ های JSX داخل *function calls* تبدیل میشن ما نمی تونیم از statement ها داخل عبارات استفاده کنیم.

[↑ فهرست مطالب](#)

## ۹۳. توی attribute ها چطوری به prop دسترسی داشته باشیم؟

رایکت یا JSX داخل یه attribute درون یابی متغیر رو پشتیبانی نمیکنه. مثال زیر کار نمیکنه:

```

```

اما ما می تونیم هر عبارت JS رو داخل کرلی براکس به عنوان مقدار کلی attribute قرار بدیم.

عیارت زیر کارمیکنه:

```

```

با استفاده از *template strings* هم کارمیکنه:

```

```

[↑ فهرست مطالب](#)

## ۹۴. چطوری یه PropType برای آرایه ای از object ها با shape داشته باشیم؟

اگه بخوایم آرایه ای از آبجکت ها رو به یه کامپوننت با شکل خاصی پاس بدیم، از `React.PropTypes.arrayOf` به عنوان یه آرگومان برای `React.PropTypes.shape` استفاده میکنیم.



```

ReactComponent.propTypes = {
 arrayWithShape: React.PropTypes.arrayOf(
 React.PropTypes.shape({
 color: React.PropTypes.string.isRequired,
 fontSize: React.PropTypes.number.isRequired,
 })
).isRequired,
};

```

[↑ فهرست مطالب](#)

## ۹۵. چطوری class های یه المنت رو به صورت شرطی رندر کنیم؟

نباید از کرلی براکس داخل " استفاده کنیم چون به عنوان یه رشته در نظر گرفته میشه.

```
<div className="btn-panel {this.props.visible ? 'show' : 'hidden'}">
```

به جاش می تونیم کرلی براکس رو به بیرون انتقال بدیم. (فراموش نکنیم که از space بین className ها استفاده کنیم).

```
<div className={`btn-panel ` + (this.props.visible ? 'show' : 'hidden')}>
```

با استفاده از *Template strings* هم کار می کنه:

```
<div className={`btn-panel ${this.props.visible ? 'show' : 'hidden'}}>
```

[↑ فهرست مطالب](#)

## ۹۶. تفاوت های React و ReactDOM چیه؟

بسته ری اکت شامل `React.createElement` ، `React.Component` ، `React.Children` و `React.helper` های دیگه که مربوطه به کلاس کامپوننت ها و المنت ها هستش. می تونیم اینا رو به عنوان isomorphic یا universal helpers که واسه ساختن کامپوننت ها نیاز داریم، در نظر بگیریم. بسته `react-dom` شامل `ReactDOM.render` میشه و داخل `react-dom/server` می تونیم با استفاده از متد های `ReactDOMServer.renderToString` و `ReactDOMServer.renderToStaticMarkup` *server-side rendering* رو پشتیبانی کنیم.

## ۹۷. چرا ReactDOM رو از React جدا کردن؟

تیم ری‌اکت سعی کرده تمام ویژگی‌های مرتبط با DOM رو جدا کنه و اونو تی‌یه کتابخونه جدا به اسم *ReactDOM* قرار بده. ری‌اکت ورژن ۱۴ اولین نسخه‌ایه که توی اون کتابخونه‌ها از هم جدا میشن. با نگاهی به بعضی از بسته‌های ری‌اکت مثل `react-native`، `react-art`، `react-canvas` و `react-three` مشخص میشه که زیبایی و جوهر ری‌اکت هیچ ربطی به مرورگرها یا DOM نداره. برای ساختن محیط‌های بیشتری که ری‌اکت بتونه رندر بشه، تیم ری‌اکت قصد داره بسته اصلی ری‌اکت رو به دو بخش تقسیم کنه: `react` و `react-dom`.

از این طریق میشه کامپوننت‌هایی نوشت که بین ری‌اکت وب و ری‌اکت نیتیو قابل اشتراک باشه.

## ۹۸. چطوری از label تو ری‌اکت استفاده کنیم؟

اگه سعی کنیم که با استفاده از `for attribute` یه عنصر `<label>` متصل به یه متن رو رندر کنیم، اون وقت ویژگی HTML بودن رو از دست میدیم و یه خطا توی کنسول بهمون نشون میده.

```
<label for={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

از اونجایی که `for` یه کلمه کلیدی رزرو شده توی جاوااسکریپت، به جاش باید از `htmlFor` استفاده کنیم.

```
<label htmlFor={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

## ۹۹. چطوری می‌تونیم چندتا object از استایل‌های درون خطی رو با هم ترکیب کنیم؟

می‌تونیم از *spread operator* در ری‌اکت استفاده کنیم:

```
<button style={{ ...styles.panel.button, ...styles.panel.submitButton }}
 {"Submit"}
</button>
```

اگه داریم از ری‌اکت نیتیو استفاده میکنیم می‌تونیم از نماد آرایه استفاده کنیم:

```
<button style={[styles.panel.button, styles.panel.submitButton]}>
 {"Submit"}
</button>
```

[↑ فهرست مطالب](#)

## ۱۰۰. چطوری با **resize** شدن مرورگر یه ویو رو ری‌رندر کنیم؟

می‌تونید به رخداد **resize** توی **componentDidMount** گوش کنیم و ابعاد (**width** و **height**) رو تغییر بدین. البته حواستون باشه که این **listener** رو باید توی متد **componentWillUnmount** حذفش کنیم.

```

class WindowDimensions extends React.Component {
 constructor(props) {
 super(props);
 this.updateDimensions = this.updateDimensions.bind(this);
 }

 componentWillMount() {
 this.updateDimensions();
 }

 componentDidMount() {
 window.addEventListener("resize", this.updateDimensions);
 }

 componentWillUnmount() {
 window.removeEventListener("resize", this.updateDimensions);
 }

 updateDimensions() {
 this.setState({
 width: window.innerWidth,
 height: window.innerHeight,
 });
 }

 render() {
 return (

 {this.state.width} x {this.state.height}

);
 }
}

```

همین کار رو با استفاده از هوک‌ها هم میشه انجام داد و برای این کار همین کد رو توی `useEffect` می‌نویسیم.

```
const [dimensions, setDimensions] = useState();
useEffect(() => {
 window.addEventListener("resize", updateDimensions);

 function updateDimensions() {
 setDimensions({
 width: window.innerWidth,
 height: window.innerHeight,
 });
 }

 return () => {
 window.removeEventListener("resize", updateDimensions);
 };
}, []);

return (

 {this.state.width} x {this.state.height}

);
```

[↑ فهرست مطالب](#)

## ۱۰۱. تفاوت متدهای `replaceState` و `setState` چیست؟

وقتی که از متد `setState` فعلی و قبلی با هم ترکیب می‌شدند. `replaceState` حالت فعلی رو نشون میده و با `state`ای می‌خواهیم جایگزینش می‌کنه. معمولاً `setState` برای این استفاده می‌شه که بنا به دلیلی بخواییم همه کلیدهای قبلی رو پاک کنیم. البته همیشه بجای استفاده از `replaceState` با استفاده از `setState` بیاییم و `state` رو برابر با `false` یا `null` قرار بدیم.

[↑ فهرست مطالب](#)

## ۱۰۲. چطوری به تغییرات `state` گوش بدیم؟

متدی که معرفی میشه در کلاس کامپوننت‌ها هنگام به روز شدن `state` فراخوانی میشه. با استفاده از این متد میشه `state` و `prop` فعلی رو با مقادیر جدید مقایسه کرده و به سری کار که مدنظر داریم رو انجام بدیم.

```
componentWillUpdate(object nextProps, object nextState)
componentDidUpdate(object prevProps, object prevState)
```

با استفاده از هوک `useEffect` هم این امکان بسادگی قابل انجامه و فقط کافیه به `dependency` های این هوک متغیر مربوط به `state` رو بدیم.

```
const [someState, setSomeState] = useState();
useEffect(() => {
 // code
}, [someState]);
```

↑ فهرست مطالب

## ۱۰۳. روش توصیه شده برای حذف یک عنصر از آرایه توی `state` چیه؟

استفاده از متد `Array.prototype.filter` آرایه‌ها روش خوبییه. برای مثال بیاین یه تابع به اسم `removeItem` برای به روز کردن `state` در نظر بگیریم.

```
removeItem(index) {
 this.setState({
 data: this.state.data.filter((item, i) => i !== index)
 })
}
```

↑ فهرست مطالب

## ۱۰۴. امکانش هست که ری‌اکت رو بدون رندر کردن HTML استفاده کنیم؟

توی نسخه‌های بالاتر از (`16.2=>`) میشه. برای مثال تکه کد پایین یه سری مثال برای رندر کردن یه مقدار غیر `html` ای هست:

```
render() {
 return false
}
```

```
render() {
 return null
}
```

```
render() {
 return []
}
```

```
render() {
 return <React.Fragment></React.Fragment>
}
```

```
render() {
 return <></>
}
```

البته حواستون باشه که return کردن `undefined` کار نخواهد کرد.

[↑ فهرست مطالب](#)

## ۱۰۵. چطوری میشه با ری‌اکت یه JSON به شکل `beautify` شده نشون داد؟

میشه با استفاده از تگ `<pre>` و استفاده از option های متد `JSON.stringify` این کار رو انجام داد:

```
const data = { name: "John", age: 42 };

class User extends React.Component {
 render() {
 return <pre>{JSON.stringify(data, null, 2)}</pre>;
 }
}

React.render(<User />, document.getElementById("container"));
```

[↑ فهرست مطالب](#)

## ۱۰۶. چرا نمی‌تونیم prop رو آپدیت کنیم؟

فلسفه ساختاری ری‌اکت طوریه که prop‌ها باید *immutable* باشن و بالا به پایین و به صورت سلسه‌مراتبی مقدار بگیرند. به این معنی که پدر هر کامپوننت می‌تونه هر مقداری رو به فرزند پاس بده و فرزند حق دستکاری اونو نداره.

[↑ فهرست مطالب](#)

## ۱۰۷. چطوری می‌تونیم موقع لود صفحه روی یه input فوکوس کنیم؟

میشه با ایجاد یه *ref* برای المنت `input` و استفاده از اون توی `componentDidMount` یا `useEffect` این‌کار رو کرد:

```
class App extends React.Component {
 componentDidMount() {
 this.nameInput.focus();
 }

 render() {
 return (
 <div>
 <input defaultValue={"Won't focus"} />
 <input
 ref={(input) => (this.nameInput = input)}
 defaultValue={"Will focus"}
 />
 </div>
);
 }
}

ReactDOM.render(<App />, document.getElementById("app"));
```



```
const App = () => {
 const nameInputRef = useRef();
 useEffect(() => {
 nameInputRef.current.focus();
 }, []);

 return (
 <div>
 <input defaultValue={"Won't focus"} />
 <input ref={nameInputRef} defaultValue={"Will focus"} />
 </div>
);
};
```

[↑ فهرست مطالب](#)

## ۱۰۸. روش‌های ممکن برای آپدیت کردن object توی state چیا هستن؟

۱. فراخوانی متد `setState` با استفاده از یه `object` برای ترکیب شدن اون:

▪ استفاده از `Object.assign` برای ایجاد یه کپی از `object`:

```
const user = Object.assign({}, this.state.user, { age: 42 });
this.setState({ user });
```

\* استفاده از عملگر `*spread*`:

```
const user = { ...this.state.user, age: 42 };
this.setState({ user });
```

۲. فراخوانی `setState` با یه تابع `callback`:

```
this.setState((prevState) => ({
 user: {
 ...prevState.user,
 age: 42,
 },
}));
```

[↑ فهرست مطالب](#)

## ۱۰۹. چرا توابع به جای object در setState ترجیح داده می‌شوند؟

ری‌اکت اجازه ترکیب کردن تغییرات state رو با استفاده از متد `setState` فراهم کرده است که باعث بهبود پرفورمنس میشه. چون `this.props` و `this.state` ممکنه به صورت asynchronous و همزمان به روز بشن، نباید به مقدار اونا برای محاسبه مقدار بعدی اعتماد کرد. برای مثال به این شمارنده که درست کار نمی‌کنه دقت کنیم:

```
// Wrong
this.setState({
 counter: this.state.counter + this.props.increment,
});
```

روش توصیه شده فراخوانی متد `setState` با یه تابع بجای object هست. این تابع مقدار state قبلی رو به عنوان پارامتر اول و prop رو به عنوان ورودی دوم می‌گیره و این تابع رو زمانی که مقادیر ورودیش تغییر پیدا کنن فراخوانی می‌کنه.

```
// Correct
this.setState((prevState, props) => ({
 counter: prevState.counter + props.increment,
}));
```

[↑ فهرست مطالب](#)

## ۱۱۰. چطوری می‌تونیم نسخه ری‌اکت جاری رو توی محیط اجرایی بفهمیم؟

خیلی ساده میشه از مقدار `React.version` برای گرفتن نسخه جاری استفاده کرد.

```
const REACT_VERSION = React.version;

ReactDOM.render(
 <div>{`React version: ${REACT_VERSION}`}</div>,
 document.getElementById("app")
);
```

[↑ فهرست مطالب](#)

## ۱۱۱. روش‌های لود کردن polyfill توی CRA چیا هستن؟

### ۱. import دستی از core-js :

یه فایل ایجاد کنیم و اسمشو بزاریم (یه چیزی مثل) `polyfills.js` و توی فایل `index.js` بیایید import کنیمش. کد `npm install core-js` یا `yarn add core-js` رو اجرا کنیم و ویژگی‌هایی که لازم داریم رو از `corejs` بارگذاری کنیم.

```
import "core-js/fn/array/find";
import "core-js/fn/array/includes";
import "core-js/fn/number/is-nan";
```

### ۲. استفاده از سرویس Polyfill :

از سایت `CDN polyfill.io` واسه گرفتن مقدار شخصی سازی شده براساس مرورگر هر فرد استفاده کنیم و خیلی ساده یه خط کد به `index.html` اضافه کنیم:

```
<script src="https://cdn.polyfill.io/v2/polyfill.min.js?features=default"
```

توی تکه کد فوق ما برای polyfill کردن `Array.prototype.includes` درخواست دادیم.

↑ فهرست مطالب

## ۱۱۲. توی CRA چطوری از https به جای http استفاده کنیم؟

لازمه که کانفیگ `HTTPS=true` رو برای env جاری ست کنیم. میشه فایل `package.json` بخش `scripts` رو به شکل پایین تغییر داد:

```
"scripts": {
 "start": "set HTTPS=true && react-scripts start"
}
```

یا حتی `set HTTPS=true && npm start`

↑ فهرست مطالب

## ۱۱۳. توی CRA چطوری میشه از مسیرهای طولانی برای ایمپورت جلوگیری کرد؟

یه فایل به اسم `env`. توی مسیر اصلی پروژه ایجاد می‌کنیم و مسیر مورد نظر خودمون رو اونجا می‌نویسیم:

```
NODE_PATH=src/app
```

بعد از این تغییر سرور `develop` رو ریستارت می‌کنیم بعدش دیگه می‌تونیم هر چیزی رو از مسیر `src/app` بارگذاری کنیم و لازم هم نباشه مسیر کاملشو بهش بدیم.

[↑ فهرست مطالب](#)

## ۱۱۴. چطوری میشه Google Analytics رو به react-router اضافه کرد؟

یه `listener` به `history` object اضافه می‌کنیم تا بتونیم لود شدن صفحه رو `track` کنیم :

```
history.listen(function (location) {
 window.ga("set", "page", location.pathname + location.search);
 window.ga("send", "pageview", location.pathname + location.search);
});
```

[↑ فهرست مطالب](#)

## ۱۱۵. چطوری یه کامپوننت رو هر ثانیه به روز کنیم؟

لازمه که از `setInterval` استفاده کنیم تا تغییرات رو اعمال کنیم و البته حواسمون هست که موقع `unmount` این `interval` رو حذف کنیم که `memory leak` نشه.

```
componentDidMount() {
 this.interval = setInterval(() => this.setState({ time: Date.now() })),
}

componentWillUnmount() {
 clearInterval(this.interval)
}
```

```
let interval;
useEffect(() {
 interval = setInterval(() => this.setState({ time: Date.now() })), 1000

 return () => clearInterval(interval);
}, []);
```

[↑ فهرست مطالب](#)

## ۱۱۶. برای استایل‌دهی‌های درون خطی چطوری باید پیشوندهای مخصوص مرورگرها رو اضافه کرد؟

ری‌اکت به شکل اتوماتیک پیشوندهای مخصوص مرورگرها رو اعمال نمی‌کنه . لازمه که تغییرات رو به شکل دستی اضافه کنیم.

```
<div
 style={{
 transform: "rotate(90deg)",
 WebkitTransform: "rotate(90deg)", // note the capital 'W' here
 msTransform: "rotate(90deg)", // 'ms' is the only lowercase vendor p
 }}
/>
```

[↑ فهرست مطالب](#)

## ۱۱۷. چطوری کامپوننت‌های ری‌اکت رو با es6 می‌تونیم import و export کنیم؟

لازمه که از default برای export کردن کامپوننت‌ها استفاده کنیم

```
import React from "react";
import User from "user";

export default class MyProfile extends React.Component {
 render() {
 return <User type="customer">...</User>;
 }
}
```

با استفاده از شناساگر export کامپوننت MyProfile قراره یه عضو از ماژول فعلی میشه و برای import کردن لزومی به استفاده از عنوان این کامپوننت نیست.

[↑ فهرست مطالب](#)

## ۱۱۸. استثنایی که برای نام‌گذاری کامپوننت اجازه استفاده از حرف کوچک رو میده چیه؟

همه کامپوننت‌های ری‌اکت لازم هست که با حرف بزرگ شروع بشن ولی در این مورد نیز یکسری استثناها وجود داره. تگ‌هایی که با property و عملگر dot کار می‌کنن به عنوان کامپوننت‌های با حرف کوچک تلقی می‌شن. For example the below tag can be compiled to a valid component

```
render(){
 return (
 <obj.component /> // `React.createElement(obj.component)`
)
}
```

[↑ فهرست مطالب](#)

## ۱۱۹. چرا تابع سازنده کلاس کامپوننت یکبار صدا زده میشه؟

الگوریتم *reconciliation* ری‌اکت بعد از رندر کردن کامپوننت با بررسی رندرهای مجدد، بررسی می‌کنه که این کامپوننت قبلاً رندر شده یا نه و اگه قبلاً رندر شده باشه بر روی همون instance قبلی رندر رو انجام میده و instance جدیدی ساخته نمیشه پس تابع سازنده هم تنها یکبار صدا زده میشه.

[↑ فهرست مطالب](#)

## ۱۲۰. توی ری‌اکت چطوری مقدار ثابت تعریف کنیم؟

می‌تونیم از فیلد `ES7 استاتیک` برای تعریف ثابت استفاده کنیم.

```
class MyComponent extends React.Component {
 static DEFAULT_PAGINATION = 10;
}
```

فیلدهای استاتیک بخشی از فیلدهای کلاس توی پروپوزال stage 3 هستن.

↑ فهرست مطالب

## ۱۲۱. چطوری توی برنامه event کلیک شدن رو trigger کنیم؟

می‌تونیم از ref برای بدست آوردن رفرنس `HTMLInputElement` مورد نظر استفاده کنیم و object بدست اومده رو توی یه متغیر یا property نگهداری کنیم، بعدش از اون رفرنس می‌تونیم برای اعمال رخداد کلیک استفاده کنیم که `HTMLInputElement.click` رو فراخوانی می‌کنه. این فرآیند توی دو گام قابل انجام هستش:

۱. ایجاد ref توی متد `render`:

```
<input ref={(input) => (this.inputElement = input)} />
```

۲. اعمال رخداد click توی event handler:

```
this.inputElement.click();
```

↑ فهرست مطالب

## ۱۲۲. آیا استفاده از `async/await` توی ریاکت ممکنه؟

اگه بخواهیم از `async / await` توی ریاکت استفاده کنیم، لازمه که `Babel` و پلاگین `transform-async-to-generator` رو استفاده کنیم. توی React Native اینکار با `Babel` و یه سری transform‌ها انجام میشه.

↑ فهرست مطالب

## ۱۲۳. ساختار پوشه‌بندی معروف برا ریاکت چطوره؟

دو روش معروف برای پوشه‌های ریاکت وجود دارد:

### ۱. گروه بندی براساس ویژگی یا route:

یک روش معروف قراردادن فایل‌های JS، CSS و تست‌ها کنارهم به ازای هر ویژگی یا route هست

```
common/
├ Avatar.js
├ Avatar.css
├ APIUtils.js
└ APIUtils.test.js
feed/
├ index.js
├ Feed.js
├ Feed.css
├ FeedStory.js
├ FeedStory.test.js
└ FeedAPI.js
profile/
├ index.js
├ Profile.js
├ ProfileHeader.js
├ ProfileHeader.css
└ ProfileAPI.js
```

### ۲. گروه‌بندی بر اساس ماهیت فایل:

یک سبک مشهور دیگر گروه‌بندی فایل‌ها براساس ماهیت اونهاست

```
api/
├ APIUtils.js
├ APIUtils.test.js
├ ProfileAPI.js
└ UserAPI.js
components/
├ Avatar.js
├ Avatar.css
├ Feed.js
├ Feed.css
├ FeedStory.js
├ FeedStory.test.js
├ Profile.js
├ ProfileHeader.js
└ ProfileHeader.css
```

↑ فهرست مطالب



## ۱۲۴. پکیج‌های مشهور برای انیمیشن چیا هستند؟

*React Motion* و *React Transition Group*، *React Spring* پکیج‌های مشهور برای انیمیشن برای ری‌اکت هستند.

↑ فهرست مطالب

## ۱۲۵. مزایای ماژول‌های style چیه؟

خیلی توصیه میشه که از استایل‌دهی‌های سخت و مستقیم برای کامپوننت‌ها پرهیز کنیم. هر مقداری که فقط در یک کامپوننت خاصی مورد استفاده قرار می‌گیره، بهتره که درون همون فایل لود بشه.

برای مثال، این استایل‌ها می‌تونن تو یه فایل دیگه انتقال پیدا کنن:

```
export const colors = {
 white,
 black,
 blue,
};

export const space = [0, 8, 16, 32, 64];
```

و توی موقعی که نیاز داریم از اون فایل مشخص لود کنیمشون:

```
import { space, colors } from "./styles";
```

↑ فهرست مطالب

## ۱۲۶. معروف‌ترین linterهای ری‌اکت کدوما هستند؟

ESLint یه linter برای JavaScript هستش. یه سری کتابخونه برای کمک به کدنویسی تو سبک‌های مشخص و استاندارد برای eslint وجود داره. یکی از معروف‌ترین پلاگین‌های موجود `eslint-plugin-react` هست.

به صورت پیش‌فرض این پلاگین یه سری از best practice ها رو برای کدهای نوشته شده بررسی می‌کنه. با مجموعه‌ای از قوانین برای . پلاگین مشهور دیگه

`eslint-plugin-jsx-a11y` هستش، که برای مسائل معروف در زمینه accessibility

کمک می‌کنه. چرا که JSX یه سینتکس متفاوت‌تری از HTML ارائه می‌کنه، مشکلاتی که ممکنه مثلاً با `alt` و `tabindex` پیش میاد رو با این پلاگین میشه متوجه شد.

[↑ فهرست مطالب](#)

## ۱۲۷. چطوری باید توی کامپوننت درخواست `api call` بزنیم؟

می‌تونیم از کتابخونه‌های `AJAX` مثل `Axios` یا حتی از `fetch` که به صورت پیش‌فرض تو مرورگر وجود داره استفاده کنیم. لازمه که توی `Mount` درخواست `API` رو انجام بدیم و برای به روز کردن کامپوننت می‌تونیم از `setState` استفاده کنیم تا داده بدست اومده رو توی کامپوننت نشون بدیم. برای مثال، لیست کارمندان از `API` گرفته میشه و توی `state` نگهداری میشه:

```

class MyComponent extends React.Component {
 constructor(props) {
 super(props);
 this.state = {
 employees: [],
 error: null,
 };
 }

 componentDidMount() {
 fetch("https://api.example.com/items")
 .then((res) => res.json())
 .then(
 (result) => {
 this.setState({
 employees: result.employees,
 });
 },
 (error) => {
 this.setState({ error });
 }
);
 }

 render() {
 const { error, employees } = this.state;
 if (error) {
 return <div>Error: {error.message}</div>;
 } else {
 return (

 {employees.map((employee) => (
 <li key={employee.name}>
 {employee.name}-{employee.experience}

))}

);
 }
 }
}

```

```
const MyComponent = () => {
 const [employees, setEmployees] = useState([]);
 const [error, setError] = useState(null);

 useEffect(() => {
 fetch("https://api.example.com/items")
 .then((res) => res.json())
 .then(
 (result) => {
 setEmployees(result.employees);
 },
 (error) => {
 setError(error);
 }
);
 }, []);

 return error ? (
 <div>Error: {error.message}</div>
) : (

 {employees.map((employee) => (
 <li key={employee.name}>
 {employee.name}-{employee.experience}

))}

);
};
```

↑ فهرست مطالب

## ۱۲۸. render props چیست؟

**Render Props** یک تکنیک ساده برای به اشتراک گذاری کد بین کامپوننت‌هاست که با استفاده از یک prop که به تابع دادم انجام می‌شود. کامپوننت زیر از همین روش برای پاس دادن یک React element استفاده می‌کند.

```
<DataProvider render={({data}) => <h1>`Hello ${data.target}`</h1>} />
```

کتابخانه‌هایی مثل React Router و DownShift از این پترن استفاده می‌کنند.

## React Router

## ۱۲۹. React Router چیست؟

React Router یک کتابخانه قدرتمند برای جابجایی سریع بین صفحات و flowهای مختلفه که برپایه ری اکت نوشته شده و امکان sync کردن آدرس وارد شده با صفحات رو توی محیطهای مختلف فراهم می‌کنه.

↑ فهرست مطالب

## ۱۳۰. ارتباط React Router و کتابخانه history چیست؟

React Router یک wrapper روی کتابخانه `history` هستش که اعمال اجرایی بر روی `window.history` رو با استفاده از ابجکتهای `hash` و `browser` مدیریت می‌کنه. البته این کتابخانه یک نوع دیگه از `history` ها به اسم `memory history` رو هم معرفی می‌کنه که برای محیطهایی که به صورت عمومی از `history` پشتیبانی نمی‌کنن کاربرد داره. مثل محیط توسعه برنامه موبایل با (React Native) یا محیطهای `unit test` و `Nodejs`.

↑ فهرست مطالب

## ۱۳۱. کامپوننت‌های router توی نسخه ۴ چیا هستن؟

React Router v4 سه نوع مختلف از کامپوننت روتر (`<Router>`) رو معرفی می‌کنه :

۱. `<BrowserRouter>`

۲. `<HashRouter>`

۳. `<MemoryRouter>`

کامپوننت‌های فوق به ترتیب `hash`، `browser`، و `memory history` درست می‌کنن. React Router v4 ساخت `history` را براساس context ارائه شده به ابجکت router انجام می‌دهد.

↑ فهرست مطالب

## ۱۳۲. هدف از متدهای `push` و `replace` توی history چیست؟

هر شیء از `history` دو متد برای جابجایی ارائه می‌دهد.

۱. `push`

۲. `replace`

اگر به `history` به عنوان یک آرایه از مسیرهای بازدید شده نگاه کنیم، `push` یک جابجایی جدید به مسیر اضافه می‌کند و `replace` مسیر فعلی را با یک مسیر جدید جابجا می‌کند.

↑ فهرست مطالب

## ۱۳۳. چطوری توی برنامه به `route` خاص جابجا بشیم؟

روش‌های مختلفی برای جابجایی در برنامه و توسط `cd` وجود دارد.

۱. استفاده از تابع مرتبه بالاتر (`withRouter`) `higher-order` :

متد `withRouter` آبجکت `history` را به عنوان یک `prop` به کامپوننت

اضافه می‌کند. در این `prop` دسترسی به متدهای `push` و `replace`

بسادگی می‌تونه مسیریابی بین کامپوننت `ro` فراهم کنه و نیاز به `context` رو رفع کنه.

```
import { withRouter } from "react-router-dom"; // this also works with '

const Button = withRouter(({ history }) => (
 <button
 type="button"
 onClick={() => {
 history.push("/new-location");
 }}
 >
 {"Click Me!"}
 </button>
));
```

۲. استفاده از کامپوننت `<Route>` و پترن `render props` :

کامپوننت `<Route>` همون `prop` که متد `withRouter` به کامپوننت می‌ده رو به کامپوننت می‌ده.

```
import { Route } from "react-router-dom";

const Button = () => (
 <Route
 render={({ history }) => (
 <button
 type="button"
 onClick={() => {
 history.push("/new-location");
 }}
 >
 {"Click Me!"}
 </button>
)}
 />
);
```

### ۳. استفاده از context:

استفاده از این مورد توصیه نمی‌شود و ممکنه به زودی deprecate شود.

```
const Button = (props, context) => (
 <button
 type="button"
 onClick={() => {
 context.history.push("/new-location");
 }}
 >
 {"Click Me!"}
 </button>
);

Button.contextTypes = {
 history: React.PropTypes.shape({
 push: React.PropTypes.func.isRequired,
 }),
};
```

### ۴. استفاده از هوک‌های موجود:

هوک‌هایی برای دسترسی به history و params در این کتابخانه وجود دارد  
مثل useHistory:

```
const Page = (props, context) => {
 const history = useHistory();
 const location = useLocation();
 const { slug } = useParams();

 return (
 <button
 type="button"
 onClick={() => {
 history.push("/new-location");
 }}
 >
 {"Click Me!"}
 </button>
);
};
```

[↑ فهرست مطالب](#)

## ۱۳۴. چطوری همیشه query پارامترها رو توی ری‌اکت روتر نسخه ۴ گرفت؟

ساده‌ترین راه برای دسترسی به param‌های آدرس استفاده از هوک useParams هست.

```
const { slug } = useParams();

console.log(`slug query param`, slug);
```

[↑ فهرست مطالب](#)

## ۱۳۵. دلیل خطای "Router may have only one child element" چیه؟

باید کامپوننت Route رو توی بلاک <Switch> قرار بدیم چون <Switch> چون Switch باعث میشه که منحصرًا یک کامپوننت در صفحه لود بشه. اولش لازمه که Switch رو import کنیم:

```
import { Switch, Router, Route } from "react-router";
```



بعدش روت‌ها رو توی بلاک `<Switch>` تعریف می‌کنیم:

```
<Router>
 <Switch>
 <Route {/* ... */} />
 <Route {/* ... */} />
 </Switch>
</Router>
```

[↑ فهرست مطالب](#)

## ۱۳۶. چطوری میشه به متد `history.push` پارامتر اضافه کرد؟

موقع جابجایی می‌تونیم یه `object` به `history` پاس بدیم که یه سری گزینه‌ها رو برامون قابل کانفیگ می‌کنه:

```
this.props.history.push({
 pathname: "/template",
 search: "?name=sudheer",
 state: { detail: response.data },
});
```

این کانفیگ‌ها یکیش `search` هست که می‌تونه پارامتر موردنظر ما رو به مسیر مورد نظر بفرسته.

[↑ فهرست مطالب](#)

## ۱۳۷. چطوری میشه صفحه ۴۰۴ ساخت؟

کامپوننت `<Switch>` اولین فرزند `<Route>` ای که با درخواست موجود تطابق داشته باشه رو رندر می‌کنه. از اونجایی که یه `<Route>` بدون `path` یا با `path *` همیشه مطابق با درخواست است، پس هنگام خطای ۴۰۴ این مورد برای رندر استفاده میشه.

```
<Switch>
 <Route exact path="/" component={Home} />
 <Route path="/user" component={User} />
 <Route component={NotFound} />
</Switch>
```

[↑ فهرست مطالب](#)

## ۱۳۸. توی ری‌اکت روتر نسخه ۴ چطوری همیشه history رو گرفت؟

۱. می‌تونیم یه ماژول درست کنیم که `object history` رو می‌ده و هرجایی خواستیم از این فایل استفاده کنیم. برای مثال فایل `history.js` رو ایجاد کنید:

```
import { createBrowserHistory } from "history";

export default createBrowserHistory({
 /* pass a configuration object here if needed */
});
```

۲. می‌تونیم از کامپوننت `<Router>` بجای روترهای پیش‌فرض استفاده کنیم. فایل `history.js` بالا رو توی فایل `index.js` لود می‌کنیم:

```
import { Router } from "react-router-dom";
import history from "../history";
import App from "../App";

ReactDOM.render(
 <Router history={history}>
 <App />
 </Router>,
 holder
);
```

۳. البته همیشه از متد `push` مثل آبجکت پیش‌فرض `history` استفاده کنیم:

```
// some-other-file.js
import history from "../history";

history.push("/go-here");
```

[↑ فهرست مطالب](#)

## ۱۳۹. چطوری بعد از لاگین به شکل خودکار ری‌دایرکت کنیم؟

پکیج `react-router` مکان استفاده از کامپوننت `<Redirect>` رو توی `React Router` می‌ده. رندر کردن `<Redirect>` باعث جابجایی به مسیر پاس داده شده بهش میشه. مثل ری‌دایرکت سرور-ساید، مسیر جدید با `path` فعلی جایگزین می‌شه.

```
import React, { Component } from "react";
import { Redirect } from "react-router";

export default class LoginComponent extends Component {
 render() {
 if (this.state.isLoggedIn === true) {
 return <Redirect to="/your/redirect/page" />;
 } else {
 return <div>{"Login Please"}</div>;
 }
 }
}
```

## چندزبانگی ری‌اکت

### ۱۴۰. React-Intl چیه؟

*React Intl* یه کتابخونه برای راحت کردن کار با برنامه‌های چند زبانه‌ست. این کتابخونه از مجموعه‌ای از کامپوننت‌ها و API‌ها برای فرمت‌بندی `date`، `string` و اعداد برای سهولت چندزبانگی استفاده می‌کنه. *React Intl* بخشی از *FormatJS* هست که امکان اتصال به ری‌اکت رو با کامپوننت‌های خودش فراهم می‌کنه.

↑ فهرست مطالب

### ۱۴۱. اصلی‌ترین ویژگی‌های *React Intl* چیا هستن؟

۱. نمایش اعداد با جداکننده‌های مشخص
۲. نمایش تاریخ و ساعت با فرمت درست
۳. نمایش تاریخ بر اساس زمان حال
۴. امکان استفاده از لیبل‌ها توی `string`
۵. پشتیبانی از بیش از ۱۵۰ زبان
۶. اجرا توی محیط مرورگر و `node`
۷. دارا بودن استانداردهای داخلی

↑ فهرست مطالب

## ۱۴۲. دو روش فرمت کردن توی React Intl چیا هستن؟

این کتابخونه از دو روش برای فرمت‌بندی رشته‌ها، اعداد و تاریخ استفاده می‌کنه: کامپوننت‌های ری‌اکتی و API.

```
<FormattedMessage
 id={"account"}
 defaultMessage={"The amount is less than minimum balance."}
/>
```

```
const messages = defineMessages({
 accountMessage: {
 id: "account",
 defaultMessage: "The amount is less than minimum balance.",
 },
});

formatMessage(messages.accountMessage);
```

[↑ فهرست مطالب](#)

## ۱۴۳. چطوری از FormattedMessage به عنوان یه placeholder همیشه استفاده کرد؟

کامپوننت `<...Formatted>` از `react-intl` بجای بازگرداندن string یه المنت برگشت می‌ده و به همین دلیل همیشه ازش به عنوان placeholder یا `alt` و... استفاده کرد. اگه جایی لازم شد یه پیامی رو اینجور جاها استفاده کنیم باید از `formatMessage` استفاده کنیم. می‌تونیم شی `intl` رو با استفاده از `injectIntl` HOC به کامپوننت موردنظر `inject` کنیم و بعدشم می‌تونیم از متد `formatMessage` روی این شی استفاده کنیم.

```
import React from "react";
import { injectIntl, intlShape } from "react-intl";

const MyComponent = ({ intl }) => {
 const placeholder = intl.formatMessage({ id: "messageId" });
 return <input placeholder={placeholder} />;
};

MyComponent.propTypes = {
 intl: intlShape.isRequired,
};

export default injectIntl(MyComponent);
```

[↑ فهرست مطالب](#)

## ۱۴۴. چطوری میشه locale فعلی رو توی React Intl بدست آورد؟

می‌تونیم با استفاده از `injectIntl` locale فعلی رو بگیریم:

```
import { injectIntl, intlShape } from "react-intl";

const MyComponent = ({ intl }) => (
 <div>{`The current locale is ${intl.locale}`}</div>
);

MyComponent.propTypes = {
 intl: intlShape.isRequired,
};

export default injectIntl(MyComponent);
```

[↑ فهرست مطالب](#)

## ۱۴۵. چطوری با استفاده از React Intl به تاریخ رو فرمت‌بندی کنیم؟

با استفاده از `injectIntl` HOC می‌تونیم به متد `formatDate` توی کامپوننت `FormattedDate` خودمون دسترسی داشته باشیم. این متد به صورت داخلی توسط

استفاده همیشه و مقدار string تاریخ فرمت بندی شده رو برمی‌گردونه.

```
import { injectIntl, intlShape } from "react-intl";

const stringDate = this.props.intl.formatDate(date, {
 year: "numeric",
 month: "numeric",
 day: "numeric",
});

const MyComponent = ({ intl }) => (
 <div>`The formatted date is ${stringDate}`</div>
);

MyComponent.propTypes = {
 intl: intlShape.isRequired,
};

export default injectIntl(MyComponent);
```

## تست ری‌اکت

### ۱۴۶. توی تست ری‌اکت Shallow Renderer چیه؟

*Shallow rendering* برای نوشتن یونیت تست توی ری‌اکت کاربرد داره. این روش بهمون این امکان رو میده که به عمق یک مرتبه کامپوننت موردنظرمون رو رندر کنیم و مقدار بازگردانی شده رو بدون اینکه نگران عملکرد کامپوننت‌های فرزند باشیم، ارزیابی کنیم. برای مثال، اگه کامپوننتی به شکل زیر داشته باشیم:

```
function MyComponent() {
 return (
 <div>
 {"Title"}
 {"Description"}
 </div>
);
}
```

می‌تونیم انتظار اجرا به شکل زیر رو داشته باشیم:

```
import ShallowRenderer from "react-test-renderer/shallow";

// in your test
const renderer = new ShallowRenderer();
renderer.render(<MyComponent />);

const result = renderer.getRenderOutput();

expect(result.type).toBe("div");
expect(result.props.children).toEqual([
 {"Title"},
 {"Description"},
]);
```

[↑ فهرست مطالب](#)

## ۱۴۷. پکیج TestRenderer توی ری‌اکت چیه؟

این پکیج یه renderer معرفی می‌کنه که می‌تونیم ازش برای رندر کردن کامپوننت‌ها و تبدیل اونا به یه آبجکت pure JavaScript استفاده کنیم بدون اینکه وابستگی به DOM یا محیط اجرایی موبایلی داشته باشیم. این پکیج گرفتن snapshot از سلسله مرتب view (یه چیزی شبیه به درخت DOM) که توسط ReactDOM یا React Native درست میشه رو بدون نیاز به مرورگر یا jsdom فراهم می‌کنه.

```
import TestRenderer from "react-test-renderer";

const Link = ({ page, children }) => {children};

const testRenderer = TestRenderer.create(
 <Link page={"https://www.facebook.com/"}>{"Facebook"}</Link>
);

console.log(testRenderer.toJSON());
// {
// type: 'a',
// props: { href: 'https://www.facebook.com/' },
// children: ['Facebook']
// }
```

[↑ فهرست مطالب](#)

## ۱۴۸. هدف از پکیج ReactTestUtils چیه؟

پکیج *ReactTestUtils* توی پکیج `with-addons` ارائه شده و اجازه اجرای یه سری عملیات روی DOMهای شبیه‌سازی شده رو برای انجام یونیت تست‌ها ارائه می‌ده.

↑ فهرست مطالب

## ۱۴۹. Jest چیه؟

*Jest* یه فریم‌ورک برای یونیت تست کردن جاوااسکریپت هستش که توسط فیس بوک و براساس *Jasmine* ساخته شده. *Jest* امکان ایجاد اتوماتیک `mock` (دیتا یا مقدار ثابت برای تست) و محیط `jsdom` رو فراهم می‌کنه و اکثراً برای تست کامپوننت‌ها استفاده میشه.

↑ فهرست مطالب

## ۱۵۰. مزایای jest نسبت به jasmine چیا هستن؟

- یه سری برتری‌هایی نسبت به *Jasmine* داره :
- می‌تونه به صورت اتوماتیک تست‌ها رو توی سورس کد پیدا و اجرا کنه
  - به صورت اتوماتیک می‌تونه وابستگی‌هایی که داریم رو `mock` کنه
  - امکان تست کد `asynchronous` رو به شکل `synchronously` فراهم می‌کنه
  - تست‌ها رو با استفاده از یه پیاده‌سازی مصنوعی از `DOM` (`jsdom`) اجرا می‌کنه و بواسطه اون تست‌ها قابلیت اجرا توسط `cli` رو دارن
  - تست‌ها به شکل موازی اجرا می‌شن و می‌تونن توی مدت زمان زودتری تموم شن

↑ فهرست مطالب

## ۱۵۱. یه مثال ساده از تست با jest بزن؟

خب بیاین یه تست برای تابعی که جمع دو عدد رو توی فایل `sum.js` برامون انجام میده بنویسیم:



```
const sum = (a, b) => a + b;

export default sum;
```

یه فایل به اسم `sum.test.js` ایجاد می‌کنیم که تست‌هامون رو توش بنویسیم:

```
import sum from "./sum";

test("adds 1 + 2 to equal 3", () => {
 expect(sum(1, 2)).toBe(3);
});
```

و بعدش به فایل `package.json` بخش پایین رو اضافه می‌کنیم:

```
{
 "scripts": {
 "test": "jest"
 }
}
```

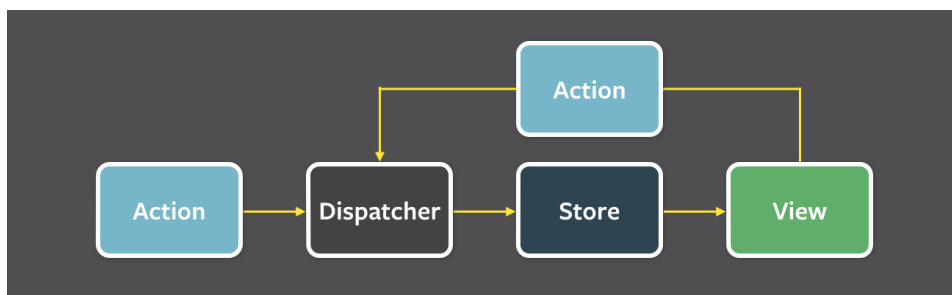
در آخر، دستور `yarn test` یا `npm test` اجرا می‌کنیم و Jest نتیجه تست رو برامون چاپ می‌کنه:

```
$ yarn test
PASS ./sum.test.js
✓ adds 1 + 2 to equal 3 (2ms)
```

## React Redux

### ۱۵۲. Flux چیه؟

*Flux* یه الگوی طراحی برنامه است که به عنوان جایگزینی برای اکثر پترن‌های MVC سنتی به کار میره. در حقیقت یه کتابخونه یا فریم‌ورک نیست و یه معماری برای تکمیل کارکرد ری‌اکت با مفهوم جریان داده یک طرفه (Unidirectional Data Flow) به کار میره. فیس‌بوک از این پترن به شکل داخلی برای توسعه ری‌اکت بهره می‌گیره. جریان کار بین `store`، `dispatcher` و `view`‌های کامپوننت‌ها با ورودی و خروجی مشخص به شکل زیر خواهد بود:



↑ فهرست مطالب

## ۱۵۳. Redux چیست؟

Redux یک state manager (مدیریت کننده حالت) قابل پیش‌بینی برای برنامه‌های جاوااسکریپت است که برپایه دیزاین پترن Flux ایجاد شده. Redux می‌تونه با ری‌اکت یا هر کتابخونه دیگه‌ای استفاده بشه. کم حجمه (حدود ۲ کیلوبایت) و هیچ وابستگی به کتابخونه دیگه‌ای نداره.

↑ فهرست مطالب

## ۱۵۴. مبانی اصلی ریداکس چیا هستن؟

Redux از سه اصل بنیادی پیروی می‌کنه:

۱. **یک مرجع کامل و همواره درست:** حالت موجود برا کل برنامه در یک درخت object و توی یه store نگهداری میشه. این یکی بودن store باعث میشه دنبال کردن تغییرات در طول زمان و حتی دیباگ کردن برنامه ساده‌تر باشه.
۲. **State فقط قابل خواندن است:** تنها روش ایجاد تغییر در store استفاده از action هستش و نتیجه اجرای این action یک object خواهد بود که رخداد پیش اومده رو توصیف می‌کنه. به این ترتیب مطمئن میشیم که تغییرات فقط با action انجام میشن و هر دیتایی توی store باشه توسط خودمون پر شده.
۳. **تغییرات با یه سری تابع pure انجام میشن:** برای مشخص کردن نحوه انجام تغییرات در store باید reducer بنویسیم. Reducerها فقط یه سری توابع pure هستن که حالت قبلی و action رو به عنوان پارامتر می‌گیرن و حالت بعدی رو برگشت میدن.

## ۱۵۵. کاستی‌های redux نسبت به flux چیا هستن؟

بجای گفتن کاستی‌ها بیابین مواردی که می‌دونیم موقع استفاده از Redux بجای Flux داریم رو بگیریم:

۱. باید یاد بگیریم که **mutation انجام ندیم**: Flux در مورد mutate کردن داده

نظری نمی‌دهد، ولی Redux از mutate کردن داده جلوگیری می‌کنه و پکیج‌های مکمل زیادی برای مطمئن شدن از mutate نشدن state توسط برنامه‌نویس ایجاد شده‌اند. این مورد رو میشه فقط برای محیط توسعه با پکیجی مثل `Immutable.js`، `redux-immutable-state-invariant` یا آموزش تیم برای نوشتن کد بدون mutate دیتا محقق کرد.

۲. باید توی انتخاب پکیج‌ها محتاطانه عمل کنید: Flux به شکل خاص کاری برای

حل مشکلاتی مثل `persist`، `undo/redo` کردن داده یا مدیریت فرم‌ها انجام نداده است. در عوض Redux کلی `middleware` و مکمل `store` برای محقق ساختن همچین نیازهای داره.

۳. شاید هنوز به جریان داده خوشگل نداشته باشه در حال حاضر Flux بهمون

اجازه به `type check` استاتیک خوب رو میده ولی Redux هنوز پشتیبانی خوبی نداره براش.

## ۱۵۶. تفاوت‌های `mapStateToProps` و `mapDispatchToProps` چی هست؟

`mapStateToProps` به ابزار برای دریافت به روزشدن‌های `state`‌ها توی کامپوننت هستش (که توسط به کامپوننت دیگه به روز شده):

```
const mapStateToProps = (state) => {
 return {
 todos: getVisibleTodos(state.todos, state.visibilityFilter),
 };
};
```

`mapDispatchToProps` به ابزار برای آوردن `action` برای فراخوانی تو کامپوننت ارائه میده (`action`ی که می‌خواهیم `dispatch` کنیم و ممکنه `state` رو عوض کنه):

```
const mapDispatchToProps = (dispatch) => {
 return {
 onClick: (id) => {
 dispatch(toggleTodo(id));
 },
 };
};
```

توصیه همیشه که همیشه از روش "object shorthand" برای دسترسی به `mapDispatchToProps` استفاده بشه  
 Redux این action رو توی یه تابع دیگه قرار میده که تقریباً همیشه یه چیزی مثل `(...args)`  
 و تابعی که خودش به عنوان `wrapper` ساخته رو به کامپوننت مورد نظر ما میده.

```
const mapDispatchToProps = {
 onClick,
};
```

[↑ فهرست مطالب](#)

## ۱۵۷. توی ریدیوسر می‌تونیم یه action رو dispatch کنیم؟

Dispatch کردن action توی reducer یه **آنتی پترن** محسوب میشه. reducer نباید هیچ **ساید افکتی** داشته باشه، فقط باید خیلی ساده state قبلی و action فعلی رو بگیره و state جدید رو بده. این کار رو اگه با افزودن یه سری listeners و dispatch کردن با تغییرات reducer هم انجام بدیم باز باعث ایجاد action‌های تودرتو میشه و می‌تونه ساید افکت داشته باشه،

[↑ فهرست مطالب](#)

## ۱۵۸. چطوری میشه خارج از کامپوننت میشه store ریداکس دسترسی داشت؟

لازمه که store رو از یه ماژول که با `createStore` ایجاد شده بارگذاری کنیم. البته حواسمون باشه برای انجام این مورد نباید اثری روی window به شکل global ایجاد کنیم.

```
const store = createStore(myReducer);

export default store;
```

↑ فهرست مطالب

## ۱۵۹. اشکالات پترن MVW چیا هستن؟

۱. مدیریت DOM خیلی هزینه‌بر هست و می‌تونه باعث کندی و ناکارآمد شدن برنامه بشه.
۲. بخاطر circular dependencies (وابستگی چرخشی) یه مدل پیچیده بین model ها و view ها ایجاد میشه.
۳. بخاطر تعامل زیاد برنامه تغییرات خیلی زیادی رخ میده (مثل Google Docs).
۴. هیچ روشی ساده و بدون دردسری برای undo کردن (برگشت به عقب) نیست.

↑ فهرست مطالب

## ۱۶۰. تشابهی بین Redux و RxJS هست؟

این دو کتابخونه خیلی متفاوتن و برای اهداف متفاوتی استفاده میشن، ولی یه سری تشابه‌های ریزی دارن.

Redux یه ابزار برای مدیریت state توی کل برنامه‌ست. اکثرا هم به عنوان یه معماری برای ایجاد رابط کاربری استفاده میشه. RxJS یه کتابخونه برای برنامه‌نویسی reactive (کنش گرا) هستش. اکثرا هم برای انجام تسک‌های asynchronous توی جاوااسکریپت به کار میره. می‌تونیم بهش به عنوان یه معماری بجای Promise نگاه کنیم.

Redux هم از الگوی Reactive استفاده می‌کنه چون Store ریداکس reactive هستش. Store میاد action ها رو از دور می‌بینه و تغییرات لازم رو توی خودش ایجاد می‌کنه. RxJS هم از الگوی Reactive پیروی می‌کنه، ولی بجای اینکه خودش این architecture رو بسازه میاد به شما یه سری بلاک‌های سازنده به اسم Observable میده که باهاش بتونید الگوی reactive رو اجرا کنید.

↑ فهرست مطالب

## ۱۶۱. چطوری همیشه به اکشن رو موقع لود dispatch کرد؟

خیلی ساده همیشه اون action رو موقع `mount` اجرا کرد و موقع `render` دیتای مورد نیاز رو داشت.

```
const App = (props) => {
 useEffect(() => {
 props.fetchData();
 }, []);

 return props.isLoading ? (
 <div>{"Loaded"}</div>
) : (
 <div>{"Not Loaded"}</div>
);
};

const mapStateToProps = (state) => ({
 isLoading: state.isLoading,
});

const mapDispatchToProps = { fetchData };

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

[↑ فهرست مطالب](#)

## ۱۶۲. چطوری از متد connect از پکیج react-redux استفاده می‌کنیم؟

- برای دسترسی به دیتای نگهداری شده توی ریداکس باید دو گام زیر رو طی کنیم:
۱. از متد `mapStateToProps` استفاده می‌کنیم و متغیرهای `state` که از `store` می‌خواهیم لود کنیم رو مشخص می‌کنیم.
  ۲. `**` با استفاده از متد `connect` دیتا رو به `props` میدیم `**`، چون دیتایی که این HOC میاره به عنوان `props` به کامپوننت داده میشه. متد `connect` رو هم از پکیج `react-redux` باید بارگذاری کنیم.

```
import React from 'react';
import { connect } from 'react-redux';

const App = props => {
 render() {
 return <div>{props.containerData}</div>
 }
};

const mapStateToProps = state => {
 return { containerData: state.data }
};

export default connect(mapStateToProps)(App);
```

[↑ فهرست مطالب](#)

## ۱۶۳. چطوری همیشه state ریداکس رو ریست کرد؟

لازمه که توی برنامه یه *root reducer* تعریف کنیم که وظیفه معرفی ریدیوسرهای ایجاد شده با `combineReducers` را دارد.

مثلا بیابین `rootReducer` رو برای ست کردن state اولیه با فراخوانی عمل `USER_LOGOUT` تنظیم کنیم. همونطوری که می‌دونیم، به صورت پیش‌فرض ما بنا رو براین می‌زاریم که reducerها با اجرای مقدار `undefined` به عنوان پارامتر اول `initialState` رو برمی‌گردونن و حتی actionش هم مهم نیست.

```
const appReducer = combineReducers({
 /* your app's top-level reducers */
});

const rootReducer = (state, action) => {
 if (action.type === "USER_LOGOUT") {
 state = undefined;
 }

 return appReducer(state, action);
};
```

اگه از پکیج `redux-persist` استفاده می‌کنین، احتمالا لازمه که `storage` رو هم خالی کنین. `redux-persist` یه کپی از دیتای موجود در `store` رو توی `localStorage` نگهداری می‌کنه. اولش، لازمه که یه موتور مناسب برای `storage` بارگذاری کنیم که برای

تجزیه state قبل مقداردهی اون با undefined و پاک کردن مقدارشون مورد استفاده قرار می‌گیره.

```
const appReducer = combineReducers({
 /* your app's top-level reducers */
});

const rootReducer = (state, action) => {
 if (action.type === "USER_LOGOUT") {
 Object.keys(state).forEach((key) => {
 storage.removeItem(`persist:${key}`);
 });

 state = undefined;
 }

 return appReducer(state, action);
};
```

[↑ فهرست مطالب](#)

## ۱۶۴. هدف از کاراکتر @ توی decorator متد connect چیه؟

کاراکتر (symbol) @ در حقیقت یه نماد از جاوااسکریپت برای مشخص کردن decoratorهاست. *Decorators* این امکان رو بهمون میده که بتونیم برای کلاس و ویژگی‌های (properties) اون یادداشت‌ها و مدیریت‌کننده‌هایی رو توی زمان طراحی اضافه کنیم.

بزاریم یه مثال رو برای Redux بزنیم که یه بار از decorator استفاده کنیم و یه بار بدون اون انجامش بدیم.

◦ **Without decorator**



```
import React from "react";
import * as actionCreators from "../actionCreators";
import { bindActionCreators } from "redux";
import { connect } from "react-redux";

function mapStateToProps(state) {
 return { todos: state.todos };
}

function mapDispatchToProps(dispatch) {
 return { actions: bindActionCreators(actionCreators, dispatch) };
}

class MyApp extends React.Component {
 // ...define your main app here
}

export default connect(mapStateToProps, mapDispatchToProps)(MyApp);
```

## ◦ با decorator:

```
import React from "react";
import * as actionCreators from "../actionCreators";
import { bindActionCreators } from "redux";
import { connect } from "react-redux";

function mapStateToProps(state) {
 return { todos: state.todos };
}

function mapDispatchToProps(dispatch) {
 return { actions: bindActionCreators(actionCreators, dispatch) };
}

@connect(mapStateToProps, mapDispatchToProps)
export default class MyApp extends React.Component {
 // ...define your main app here
}
```

مثال‌های بالا تقریباً شبیه به هم هستند فقط یکیشون از decoratorها استفاده می‌کنه و اون یکی حالت عادیه. سینتکس decorator هنوز به صورت پیش‌فرض توی هیچ‌کدوم از runtime‌های جاوااسکریپت فعلاً وجود نداره و هنوز به شکل آزمایشی مورد استفاده قرار می‌گیره ولی پروپوزال افزوده شدنش به زبان در دست بررسیه. خوشبختانه فعلاً می‌تونیم از babel برای استفاده از اون استفاده کنیم.

## ۱۶۵. تفاوت‌های context و React Redux چیست؟

می‌تونیم از **Context** برای استفاده از state توی مراحل داخلی کامپوننت‌های nested استفاده کنیم و پارامترهای مورد نظرمون رو تا هر عمقی که دلخواه‌مون هست ببریم و استفاده کنیم، که البته context برای همین امر به وجود اومده. این درحالیه که **Redux** خیلی قدرتمندتره و یه سری ویژگی‌هایی رو بهمون میده که نداره هنوز. بعلاوه، خود React Redux به شکل داخلی از context استفاده می‌کنه ولی به شکل عمومی این موضوع رو نشون نمیده.

## ۱۶۶. چرا به توابع state ریداکس reducer می‌گن؟

Reducerها همیشه یه مجموعه از stateها رو جمع‌آوری و تحویل میدن (براساس همه actionهای قبلی). برا همین، اونا به عنوان یه سری کاهنده‌های state عمل می‌کنن. هر وقت یه reducer از Redux فراخوانی میشه، state و action به عنوان پارامتر پاس داده میشن و بعدش این state بر اساس action جاری مقادیرش کاهش یا افزایش داده می‌شوند و بعدش state بعدی برگشت داده میشه. یعنی شما می‌تونین یه مجموعه از داده‌ها رو *reduce* که به state نهایی که دلخواهتون هست برسین.

## ۱۶۷. توی redux چطوری میشه api request زد؟

میشه از middleware (میان‌افزار) `redux-thunk` استفاده کرد که اجازه میده بتونیم actionهای async داشته باشیم. بزارین یه مثال از دریافت اطلاعات یه حساب خاص با استفاده از فراخوانی AJAX با استفاده از **fetch API** بزنیم:

```

export function fetchAccount(id) {
 return (dispatch) => {
 dispatch(setLoadingAccountState()); // Show a loading spinner
 fetch(`/account/${id}`, (response) => {
 dispatch(doneFetchingAccount()); // Hide loading spinner
 if (response.status === 200) {
 dispatch(setAccount(response.json)); // Use a normal function to
 } else {
 dispatch(someError);
 }
 });
 });
};

function setAccount(data) {
 return { type: "SET_Account", data: data };
}

```

[↑ فهرست مطالب](#)

## ۱۶۸. آیا لازمه همه state همه کامپوننت‌ها مونو توی ریداکس نگهداری کنیم؟

نه لزومی نداره، دیتای برنامه رو میشه توی store ریداکس نگهداری کرد و مسائل مربوط به UI به شکل داخلی توی state کامپوننت‌ها نگهداری بشن.

[↑ فهرست مطالب](#)

## ۱۶۹. روش صحیح برای دسترسی به store ریداکس چیه؟

بهترین روش برای دسترسی به store و انجام عملیات روی اون استفاده از تابع `connect` هستش که یه کامپوننت جدید ایجاد می‌کنه که کامپوننت جاری توی اون قرار داره و دیتای لازم رو بهش پاس میده. این پترن با عنوان **Higher-Order Components** یا کامپوننت‌های مرتبه بالاتر شناخته میشه و یه روش مورد استفاده برای extend کردن کارکرد کامپوننت‌های ری‌اکتی محسوب میشه. این تابع بهمون این امکان رو میده که state و action‌های مورد نظرمون رو به داخل کامپوننت بیاریم و البته به شکل پیوسته با تغییرات اونا کامپوننت‌مون رو به روز کنیم.

بیا یه مثال از کامپوننت `<FilterLink>` با استفاده از تابع `connect` بزنیم:

```
import { connect } from "react-redux";
import { setVisibilityFilter } from "../actions";
import Link from "../components/Link";

const mapStateToProps = (state, ownProps) => ({
 active: ownProps.filter === state.visibilityFilter,
});

const mapDispatchToProps = (dispatch, ownProps) => ({
 onClick: () => dispatch(setVisibilityFilter(ownProps.filter)),
});

const FilterLink = connect(mapStateToProps, mapDispatchToProps)(Link);

export default FilterLink;
```

بخاطر مسائل مربوط به پرفورمنس سازنده‌های ریداکس همیشه `connect` رو بجای استفاده از context API برای دسترسی به store ریداکس توصیه می‌کنن.

```
class MyComponent {
 someMethod() {
 doSomethingWith(this.context.store);
 }
}
```

[↑ فهرست مطالب](#)

## ۱۷۰. تفاوت‌های component و container توی ریداکس چی هست؟

**Component** یه کامپوننت class یا function هست که لایه ظاهری و مربوط به UI برنامه شما توی اون قرار می‌گیره.

**Container** یه اصطلاح غیررسمی برای کامپوننت‌هایی هست که به store ریداکس وصل شدن. Container ها به *state subscribe* میکنن یا action ها رو *dispatch* می‌کنن و هیچ DOM element ای رو رندر نمی‌کنن بلکه کامپوننت‌های UI رو به عنوان child به روز می‌کنن.

**نکته مهم** : استفاده از این روش تقریباً توی سال ۲۰۱۹ دیگه منقضی محسوب میشه و چون هوک‌های ری‌اکت خیلی راحت می‌تونن دیتا رو توی هر سطح از کامپوننت برامون لود کنن، پس جدا نشدن این دولایه تاثیر چشم‌گیری توی ساده بودن کدها نخواهد داشت و

بعضا حتی می‌تونه کار رو سخت‌تر کنه، پس به عنوان مترجم توصیه می‌کنم این کار رو انجام ندین 😊

↑ فهرست مطالب

## ۱۷۱. هدف از constant ها تا type ها توی ریداکس چیه؟

Constant ها یا موارد ثابت بهتون این اجازه رو میدن که کارکرد یه عملکرد مشخص رو به سادگی توی پروژه پیدا کنید. البته از خطاهای ساده‌ای که ممکنه براتون پیش بیاد هم جلوگیری می‌کنه. مثل خطاهای مربوط به type یا `ReferenceError` ها که ممکنه خیلی راحت رخ بدن.

اکثرا مقادیر ثابت constant رو توی یه فایل مثل ( `constants.js` ) یا ( `actionTypes.js` ) قرار می‌دیم.

```
export const ADD_TODO = "ADD_TODO";
export const DELETE_TODO = "DELETE_TODO";
export const EDIT_TODO = "EDIT_TODO";
export const COMPLETE_TODO = "COMPLETE_TODO";
export const COMPLETE_ALL = "COMPLETE_ALL";
export const CLEAR_COMPLETED = "CLEAR_COMPLETED";
```

توی ریداکس از این مقادیر دوتا جا استفاده میشه:

۱. موقع ساخت `action`:

مثلا فرض می‌کنیم `actions.js`:

```
import { ADD_TODO } from "./actionTypes";

export function addTodo(text) {
 return { type: ADD_TODO, text };
}
```

۲. توی `reducer` ها:

مثلا یه فایل به اسم `reducer.js` رو در نظر بگیرین:

```
import { ADD_TODO } from './actionTypes';

export default (state = [], action) => {
 switch (action.type) {
 case ADD_TODO:
 return [
 ...state,
 {
 text: action.text,
 completed: false,
 },
];
 default:
 return state;
 }
};
```

[↑ فهرست مطالب](#)

## ۱۷۲. روش‌های مختلف برای نوشتن `mapDispatchToProps` چیه؟

چندین روش برای `bind` کردن `action` به متد `dispatch` توی `mapDispatchToProps` هستش که پایین بررسی‌شون می‌کنیم:

```
const mapDispatchToProps = (dispatch) => ({
 action: () => dispatch(action()),
});
```

```
const mapDispatchToProps = (dispatch) => ({
 action: bindActionCreators(action, dispatch),
});
```

```
const mapDispatchToProps = { action };
```

روش سوم خلاصه شده روش اوله که معمولا توصیه میشه.

[↑ فهرست مطالب](#)

## ۱۷۳. کاربرد پارامتر ownProps توی mapStateToProps و mapDispatchToProps چیه؟

اگه پارامتر `ownProps` ارائه شده باشه، `ReactRedux` پارامترهایی که به کامپوننت پاس داده شدن رو به تابع `connect` پاس میده. پس اگه یه کامپوننت `connect` شده مثل کد زیر داشته باشیم:

```
import ConnectedComponent from "../containers/ConnectedComponent";

<ConnectedComponent user={"john"} />;
```

پارامتر `ownProps` توی `mapStateToProps` و `mapDispatchToProps` یه `object` رو خواهد داشت که مقدار زیر رو داره:

```
{
 user: "john";
}
```

می‌تونیم از این مقدار استفاده کنیم تا در مورد مقدار بازگشتی تصمیم بگیریم.

[↑ فهرست مطالب](#)

## ۱۷۴. ساختار پوشه‌بندی ریشه ریداکس اکثراً چطوره؟

اکثر برنامه‌های ریداکسی یه ساختاری مثل این دارند:

۱. **Components**: که برای کامپوننت‌های `dumb` یا فقط نمایشی که به ریداکس وصل نیستند استفاده می‌شود.

۲. **Containers**: که برای کامپوننت‌های `smart` که به ریداکس وصل هستن.

۳. **Actions**: که برای همه `action`‌ها استفاده میشه و هر فایل به بخشی از عملکرد برنامه تعلق داره.

۴. **Reducers**: که برای همه `reducer`‌ها استفاده میشه و هر فایل به یه `state` توی `store` تعلق داره.

۵. **Store**: که برای ساختن `store` استفاده میشه.

این ساختار برای یه برنامه کوچک تا بزرگ کاربرد داره. البته اون بخشی ازش که کامپوننت‌های `dumb` و `smart` یا همون `container` و `component` رو بر طبق وصل شدنشون به ریداکس جدا می‌کردیم تقریباً منقضی محسوب میشه.

## ۱۷۵. redux-saga جیه؟

redux-saga به کتابخانه هست که تمرکز اصلیش برای ایجاد side-effect (چیزهای asynchronous مثل fetch کردن داده و غیرشفاف مثل دسترسی به کش مرورگر) که توی برنامه‌های React/Redux با این روش ساده‌تر و بهتر انجام میشه. پکیج ریداکس ساگا روی NPM هست:

```
$ npm install --save redux-saga
```

## ۱۷۶. مدل ذهنی redux-saga چطوره؟

Saga مثل یه thread جداگانه برای برنامه عمل می‌کنه و فقط برای مدیریت سایید افکت کارایی داره. redux-saga به میان‌افزار برای ریداکس هستش، که به معنی اینه که می‌تونه به صورت اتوماتیک توسط action‌های ریداکس شروع بشه، متوقف بشه و یا کار خاصی انجام بده. این میان‌افزار به کل store ریداکس و action‌هایی که کار می‌کنن دسترسی داره و می‌تونه هر action دیگه‌ای رو dispatch کنه.

## ۱۷۷. تفاوت افکت‌های call و put توی redux-saga چی هست؟

هر دوی افکت‌های call و put سازنده‌های افکت هستن. تابع call برای ایجاد توضیح افکت استفاده میشه که به میان‌افزار دستور میده منتظر call بمونه. تابع put به افکت ایجاد می‌کنه، که به store می‌گه یه action خاص رو فقط اجرا کنه. בזارین یه مثال در مورد عملکرد این دوتا افکت برای دریافت داده یه کاربر بنزیم.



```
function* fetchUserSaga(action) {
 // `call` function accepts rest arguments, which will be passed to `api.fetchUser`
 // Instructing middleware to call promise, its resolved value will be passed to the next saga
 const userData = yield call(api.fetchUser, action.userId);

 // Instructing middleware to dispatch corresponding action.
 yield put({
 type: "FETCH_USER_SUCCESS",
 userData,
 });
}
```

[↑ فهرست مطالب](#)

## ۱۷۸. Redux Thunk چیست؟

میان افزار *Redux Thunk* بهمون این اجازه رو میده که action‌هایی رو بسازیم که به جای action عادی تابع برگردونن thunk می‌تونه به عنوان یه ایجاد کننده delay برای dispatch کردن یه action استفاده کنیم، یا حتی با بررسی یه شرط خاص یه action رو dispatch کنیم. تابعی که توی action استفاده میشه و `dispatch` و `getState` رو به عنوان پارامتر ورودی می‌گیره.

[↑ فهرست مطالب](#)

## ۱۷۹. تفاوت‌های redux-saga و redux-thunk چیست؟

هر دوی *ReduxThunk* و *ReduxSaga* می‌تونن مدیریت سایید افکت‌ها رو به دست بگیرن. توی اکثر سناریوها، Thunk از *Promise* استفاده می‌کنه، درحالی‌که Saga از *Generator*‌ها استفاده می‌کنه. Thunk تقریباً ساده‌تره و promise رو تقریباً همه دولوپرها باهاش آشنا هستن، Sagas/Generators خیلی قوی‌تر هستن و می‌تونن کاربردی‌تر باشن ولی خب لازمه که یاد بگیرینش. هردوی میان‌افزارها می‌تونن خیلی مفید باشن و شما می‌تونین با Thunks شروع کنین و اگه جایی دیدین نیازمندی‌تون رو برآورده نمی‌کنه سراغ Sagas برید.

[↑ فهرست مطالب](#)

## ۱۸۰. Redux DevTools چیه؟

*ReduxDevTools* یه محیط برای مشاهده در لحظه تغییرات ریداکس فراهم می‌کنه و قابلیت اجرای مجدد action و یه رابط کاربری قابل شخصی‌سازی رو فراهم می‌کنه. اگه نمی‌خواهین پکیج ReduxDevTools رو نصب کنید می‌تونین از افزونه ReduxDevTools برای Chrome و Firefox استفاده کنین.

↑ فهرست مطالب

## ۱۸۱. ویژگی‌های Redux DevTools چیا هستن؟

۱. بهتون اجازه میده که اطلاعات هر state و payload پاس داده شده به action رو مشاهده کنین.
۲. بهتون اجازه میده که action‌های اجرا شده رو لغو کنید.
۳. اگه یه تغییری روی کدهای reducer بدین، هر actionی که stage شده رو مجدد ارزیابی می‌کنه.
۴. اگه یه reducers یه خطایی بده، میشه متوجه شد که در طی انجام شدن کدوم action این اتفاق افتاده و خطا چی بوده.
۵. با `persistState` می‌تونین دیباگ روی موقع reload‌های مختلف ذخیره کنید.

↑ فهرست مطالب

## ۱۸۲. سلکتورهای ریداکس چی هستن و چرا باید ازشون استفاده کنیم؟

*Selector* یه سری تابع هستن که state ریداکس رو به عنوان یه پارامتر دریافت می‌کنه و یه سری داده که می‌خواهیم رو به کامپوننت پاس میده. برای مثال، دریافت اطلاعات کاربر از ریداکس با selector زیر فراهم میشه:

```
const getUserData = (state) => state.user.data;
```

↑ فهرست مطالب

## ۱۸۳. Redux Form چیه؟

*ReduxForm* با ری اکت و ریداکس کار می‌کنه تا همه اطلاعات فرم‌ها رو توی *state* ریداکس مدیریت کنیم. *ReduxForm* می‌تونه با *input*های خام *HTML5* هم کار کنه، ولی با فریم‌ورک‌های معروف *UI* مثل *Material*، *ReactWidgets* و *ReactBootstrap* کار کنه.

[↑ فهرست مطالب](#)

## ۱۸۴. اصلی‌ترین ویژگی‌های Redux Form چیه؟

۱. ماندگاری مقادیر فیلدهای فرم توی ریداکس.
۲. اعتبارسنجی (*sync/async*) و ثبت فرم.
۳. فرمت کردن، تجزیه و نرمالسازی مقادیر فیلدها.

[↑ فهرست مطالب](#)

## ۱۸۵. چطوری میشه چندتا *middleware* به ریداکس اضافه کرد؟

می‌تونیم از `applyMiddleware` استفاده کنیم. برای مثال میشه از `redux-thunk` و `logger` به عنوان پارامترهای `applyMiddleware` استفاده کنیم:

```
import { createStore, applyMiddleware } from "redux";
const createStoreWithMiddleware = applyMiddleware(
 ReduxThunk,
 logger
)(createStore);
```

[↑ فهرست مطالب](#)

## ۱۸۶. چطوری میشه توی ریداکس *initial state* تعریف کرد؟

لازم داریم که *state* اولیه رو به عنوان پارامتر دوم به `createStore` پاس بدیم:

```
const rootReducer = combineReducers({
 todos: todos,
 visibilityFilter: visibilityFilter,
});

const initialState = {
 todos: [{ id: 123, name: "example", completed: false }],
};

const store = createStore(rootReducer, initialState);
```

[↑ فهرست مطالب](#)

## ۱۸۷. تفاوت‌های Relay با Redux چیا هستند؟

Relay و Redux توی این مورد که دوتا شونم از یه store استفاده می‌کنن شبیه بهم هستن. تفاوت اصلی این دو اینه که relay فقط state‌هایی رو مدیریت می‌کنه که از سرور تاثیر گرفتن و همه دسترسی‌هایی که به state مربوطه رو با کوئری‌های GraphQL (برای خوندن داده‌ها) و mutationها (برای تغییرات داده) انجام میده. Relay داده‌ها برای شما رو cache می‌کنه و گرفتن داده از سرور رو برای شما بهینه می‌کنه. چون فقط تغییرات رو دریافت میکرد و نه چیز دیگه‌ای.

## React Native

## ۱۸۸. تفاوت‌های React و React Native چیا هستند؟

**React** یه کتابخونه جاوااسکریپتی هست که از اجرای اون روی frontend و اجرای اون روی سرور برای تولید رابط کاربری و برنامه‌های تحت وب پشتیبانی می‌کنه. **React Native** یه فریم‌ورک موبایل هست که کدها رو به کامپوننت‌های native روی موبایل compile می‌کنه و بهمون این اجازه رو میده که برنامه‌های موبایلی (iOS, Android, and Windows) رو با استفاده از جاوااسکریپت بسازیم که از ری‌اکت برای تولید کامپوننت استفاده می‌کنه.

[↑ فهرست مطالب](#)

## ۱۸۹. چطوری همیشه برنامه React Native رو تست کرد؟

ReactNative میتونه توی شبیه‌سازهای سیستم‌عامل‌های موبایلی مثل iOS و Android تست کرد. می‌تونیم برنامه‌های خودمون رو توی برنامه <https://expo.io> (expo) توی گوشی خودمون هم ببینیم که با استفاده از QR-code میتونه به برنامه روی کامپیوتر و گوشی sync کنه، البته باید هر دوی این دستگاه‌ها تو یه شبکه وایرلس باشه.

↑ فهرست مطالب

## ۱۹۰. چطوری همیشه توی React Native لاگ کرد؟

می‌تونیم از `console.warn` ، `console.log` و غیره استفاده کرد. از نسخه ReactNative 0.29 می‌تونیم خیلی ساده کدهای زیر رو اجرا کنیم که لاگ رو توی خروجی ببینیم:

```
$ react-native log-ios
$ react-native log-android
```

↑ فهرست مطالب

## ۱۹۱. چطوری همیشه React Native رو دیباگ کرد؟

۱. برای دیباگ کردن برنامه ری‌اکت native گام‌های زیر رو طی می‌کنیم:
  ۱. برنامه رو توی شبیه‌ساز iOS اجرا می‌کنیم.
  ۲. دکمه‌های `Command + D` رو فشار میدیم و یه صفحه وب توی آدرس `http://localhost:8081/debugger-ui` اجرا میشه.
  ۳. چک‌باکس `On Caught Exceptions` رو برای یه دیباگ بهتر فعال می‌کنیم.
  ۴. دکمه‌های `Command + Option + I` رو برای اجرای `developer-tools` کروم فشار میدیم یا از طریق منوهای `View` و `Developer` و `DeveloperTools` باز می‌کنیمش.
  ۵. حالا می‌تونیم برنامه مورد نظر خودمون رو به راحتی تست کنیم.

## کتابخانه‌های پشتیبانی شده ری‌اکتی و Integrationهاش

## ۱۹۲. کتابخانه reselect چیه و چطوری کار می‌کنه؟

Reselect یه کتابخانه **selector** برای ریداکس هست که از مفهوم *memoization* استفاده می‌کنه. این کتابخانه به صورت اولیه نوشته شده بوده که داده‌های هر برنامه Redux-like یا شبیه ریداکس رو پردازش کنه، ولی نتونسته با هیچ برنامه یا کتابخانه دیگه‌ای گره بخوره. Reselect یه کپی از آخرین inputs/outputs از هر فراخوانی رو نگهداری می‌کنه و فقط زمانی اونو دوباره محاسبه می‌کنه که تغییراتی توی ورودی رخ داده باشه. اگه همون ورودی‌ها دوبار استفاده بشن، Reselect مقدار cache شده رو برمی‌گردونه. memoization و lcache ای که استفاده میشه تا حد زیادی قابل شخصی‌سازی.

↑ فهرست مطالب

## ۱۹۳. Flow چیه؟

Flow یه *static type checker* هستش که طراحی شده تا خطاهای مربوط به نوع‌ها رو توی جاوااسکریپت پیدا کنیم. نوع‌های flow می‌تونه خیلی ریزبینانه‌تر از رویکردهای سنتی بررسی نوع عمل کنه. برای مثال، Flow بهمون کمک می‌کنه که خطاهای مربوط به دریافت `null` توی برنامه رو کنترل کنیم که توی روش‌های سنتی غیرممکنه تقریباً.

↑ فهرست مطالب

## ۱۹۴. تفاوت‌های Flow و PropTypes چیا هستن؟

Flow یه ابزار تجزیه و تحلیل *استاتیک* (static-checker) هستش که از ویژگی‌های بالاتر از زبان استفاده می‌کنه و بهمون کمک می‌کنه که به بخش‌های مختلف برنامه نوع اضافه کنیم و خطاهایی که مرتبط با بررسی نوع‌ها هست رو موقع *compile* ازشون جلوگیری کنیم. PropTypes یه روش بررسی نوع ساده (موقع runtime) هست که روی ری‌اکت اضافه شده. به غیر از نوع‌هایی که به کامپوننت موردنظر به عنوان prop داده شده رو نمی‌تونه بررسی کنه. پس اگه دنبال یه روش برای بررسی نوع منعطف هستیم که توی کل پروژه عمل کنه Flow یا TypeScript روش‌های بهتری هستن.

↑ فهرست مطالب

## ۱۹۵. چطوری از آیکون‌های font-awesome توی ری‌اكت استفاده کنیم؟

گام‌های زیر برای استفاده از font-awesome توی ری‌اكت باید طی بشه:  
۱. پکیج font-awesome رو نصب می‌کنیم:

```
npm install --save font-awesome
```

۲. font-awesome رو توی فایل index.js بارگذاری می‌کنیم:

```
import "font-awesome/css/font-awesome.min.css";
```

۳. از کلاس این فونت توی className های موردنظر استفاده می‌کنیم:

```
render() {
 return <div><i className={'fa fa-spinner'} /></div>
}
```

↑ فهرست مطالب

## ۱۹۶. React Dev Tools چیه؟

*ReactDeveloperTools* بهمون اجازه اینو میده که سلسله مراتب کامپوننت‌های برنامه رو بررسی کنیم و شامل prop و state هم میشه. این مورد به دو روش افزونه (برای Chrome و Firefox) و یه برنامه جانبی مستقل (که با سافاری و مرورگرهای دیگه هم کار می‌کنه) در دسترسه.

پس سه مورد رو می‌تونیم در نظر بگیریم:

۱. افزونه Chrome

۲. افزونه Firefox

۳. برنامه مستقل (Safari، ReactNative و ...)

↑ فهرست مطالب

## ۱۹۷. چرا توی کروم devtools برای فایل‌های local لود نمیشه؟

اگه یه فایل محلی HTML رو توی مرورگر باز کنیم ( ...//:file ) بعدش لازمه که *ChromeExtensions* یا همون افزونه‌های کروم رو باز کنیم و چک‌باکس

## ۱۹۸. چطوری از Polymer توی React استفاده کنیم؟

۱. یه element برای Polymer ایجاد می‌کنیم :

```
<link rel="import" href="../../bower_components/polymer/polymer.html" />
Polymer({
 is: "calender-element",
 ready: function () {
 this.textContent = "I am a calender";
 },
});
```

۲. کامپوننت Polymer رو با تگ‌های HTML ایجاد می‌کنیم و توی داکيومنت html بارگذاری می‌کنیم، برای مثال اونو توی index.html برنامه بارگذاری کنیم:

```
<link
 rel="import"
 href="./src/polymer-components/calender-element.html"
/>
```

۳. از اون element توی فایل JSX استفاده می‌کنیم:

```
import React from "react";

class MyComponent extends React.Component {
 render() {
 return <calender-element />;
 }
}

export default MyComponent;
```

## ۱۹۹. مزایای React نسبت به Vue.js چیا هستن؟



ری اکت مزایای زیر رو نسبت به Vue.js داره:

۱. انعطاف پذیری بیشتری رو توی توسعه برنامه‌های بزرگ بهمون میده.
  ۲. تست کردنش راحت‌تره.
  ۳. برای تولید برنامه‌های موبایلی هم مناسبه.
  ۴. اطلاعات و راهکارهای مختلفی براش توی دسترسه.
- نکته:** لیست موارد فوق صرفاً اظهار نظر شخصی بوده و براساس تجربه حرفه‌ای ممکن است متفاوت باشد. اما به عنوان پارامترهای پایه مفید هستن

\*\*[فهرست](#فهرست)\*\*

## ۲۰۰. تفاوت‌های React و Angular چیا هستن؟

React	Angular
ری اکت یه کتابخونه ست و فقط یه لایه view داره	Angular یه فریم ورک و عملکردش کاملاً MVC هستش
در ری اکت جریان داده ها فقط از یه طریق هستش و به خاطر همین اشکال زدایی راحت تره	در Angular جریان داده ها از دو جهت، یعنی اتصال داده های دوطرفه بین والدین و فرزندان رو داره و به خاطر همین اشکال زدایی سخت تره

**نکته:** لیست موارد فوق صرفاً اظهار نظر شخصی بوده و براساس تجربه حرفه‌ای ممکن است متفاوت باشد. اما به عنوان پارامترهای پایه مفید هستند

[↑ فهرست مطالب](#)

## ۲۰۱. چرا تب React در DevTools نشان داده نمی‌شود؟

موقع بارگیری صفحه، *React DevTools* یه گلوبال به اسم `__REACT_DEVTOOLS_GLOBAL_HOOK__` تنظیم میکنه، بعدش ری اکت موقع مقدار دهی اولیه با اون هوک ارتباط برقرار میکنه. اگه وب سایت از ری اکت استفاده نکنه یا ری اکت نتونه با DevTools ارتباط برقرار کنه اون تب رو نشون نمیده.

[↑ فهرست مطالب](#)

## ۲۰۲. Styled components چیست؟

styled-components به کتابخانه جاوااسکریپت برای طراحی ظاهر برنامه های ری اکت. نقشه برداری بین استایل ها و کامپوننت ها رو حذف میکنه و بهمون این امکان رو میده که CSS واقعی رو با جاوااسکریپت بنویسیم.

[↑ فهرست مطالب](#)

## ۲۰۳. یه مثال از Styled Components می تونی بگی؟

بیاین کامپوننت های `<Title>` و `<Wrapper>` رو با استایل های خاص برای هر کدوم بسازیم.

```
import React from 'react'
import styled from 'styled-components'

// Create a <Title> component that renders an <h1> which is centered, red
const Title = styled.h1`
 font-size: 1.5em;
 text-align: center;
 color: palevioletred;
`

// Create a <Wrapper> component that renders a <section> with some padding
const Wrapper = styled.section`
 padding: 4em;
 background: papayawhip;
`
```

این دو تا متغیر، `Title` و `Wrapper`، کامپوننت هایی هستن که می تونیم مثل هر کامپوننت دیگه ای رندرشون کنیم.

```
<Wrapper>
 <Title>{'Lets start first styled component!'}</Title>
</Wrapper>
```

[↑ فهرست مطالب](#)

## ۲۰۴. Relay چیست؟

Relay به فریم ورک جاوااسکریپت هستش برای ارائه یه لایه داده و ارتباط client-server به برنامه های وب با استفاده از لایه view ری اکت.

[↑ فهرست مطالب](#)

## ۲۰۵. چطوری میشه از تایپ اسکریپت توی create-react-app استفاده کرد؟

با شروع از react-scripts@2.1.0 یا بالاتر، یه پشتیبان داخلی برای typescript وجود داره. میتونیم گزینه `--typescript` رو به صورت زیر منتقل کنیم.

```
npx create-react-app my-app --typescript

or

yarn create react-app my-app --typescript
```

ولی برای ورژن های پایین تر وقتی داریم یه پروژه جدید می سازیم react scripts، گزینه `--scripts-version` رو به عنوان `react-scripts-ts` تنظیم میکنیم. `react-scripts-ts` مجموعه ای از تنظیمات برای گرفتن پروژه `create-react-app` و آوردن TypeScript داخلش هست. حالا ساختار پروژه باید این شکلی باشه:

```
my-app/
├ .gitignore
├ images.d.ts
├ node_modules/
├ public/
├ src/
├ └ ...
├ package.json
├ tsconfig.json
├ tsconfig.prod.json
├ tsconfig.test.json
└ tsconfig.json
```

## متفرقه

## ۲۰۶. اصلی ترین ویژگی های کتابخونه reselect چیا هستن؟

۱. Selector ها داده های مشتق شده رو محاسبه میکنه و به ریداکس اجازه میدن حداقل state های ممکن رو ذخیره کنه.
۲. Selector ها کارامد هستن. یه selector تا وقتی که یکی از آرگومان هاش تغییر نکرده معتبر نیست.
۳. Selector ها قابل ترکیب هستن. اونا می تونن به عنوان ورودی برای بقیه Selector ها استفاده بشن.

## ۲۰۷. یه مثال از کاربرد کتابخونه **reselect** بزن؟

بیاین محاسبات و مقادیر مختلف یه سفارش حمل و نقل رو با استفاده ساده از Reselect انجام بدیم:

```
import { createSelector } from 'reselect'

const shopItemsSelector = state => state.shop.items
const taxPercentSelector = state => state.shop.taxPercent

const subtotalSelector = createSelector(
 shopItemsSelector,
 items => items.reduce((acc, item) => acc + item.value, 0)
)

const taxSelector = createSelector(
 subtotalSelector,
 taxPercentSelector,
 (subtotal, taxPercent) => subtotal * (taxPercent / 100)
)

export const totalSelector = createSelector(
 subtotalSelector,
 taxSelector,
 (subtotal, tax) => ({ total: subtotal + tax })
)

let exampleState = {
 shop: {
 taxPercent: 8,
 items: [
 { name: 'apple', value: 1.20 },
 { name: 'orange', value: 0.95 },
]
 }
}

console.log(subtotalSelector(exampleState)) // 2.15
console.log(taxSelector(exampleState)) // 0.172
console.log(totalSelector(exampleState)) // { total: 2.322 }
```

[↑ فهرست مطالب](#)

## ۲۰۸. توی Redux اکشن چیکار می‌کنه؟

اکشن‌ها آبجکت‌های ساده جاوااسکریپت یا اطلاعاتی هستند که داده‌ها رو از برنامه به store می‌فرستن. اونا تنها منابع اطلاعاتی برای store هستند. اکشن باید به ویژگی type داشته باشه که نوع اکشنی که انجام میشه رو نشون بده. برای مثال اکشنی که نشون میده به آیتم todo جدید اضافه شده:

```
{
 type: 'ADD_TODO',
 text: 'Add todo item'
}
```

[↑ فهرست مطالب](#)

## ۲۰۹. استاتیک شی با کلاس های ES6 در React کار می کنه؟

خیر، استاتیک ها فقط با `React.createClass()` کار میکنند:

```
someComponent= React.createClass({
 statics: {
 someMethod: function() {
 // ..
 }
 }
})
```

اما میتونیم استاتیک ها رو داخل کلاس های ES6 یا خارج از کلاس مثل زیر بنویسیم،

```
class Component extends React.Component {
 static propTypes = {
 // ...
 }

 static someMethod() {
 // ...
 }
}
```

```
class Component extends React.Component {

}

Component.propTypes = {...}
Component.someMethod = function(){...}
```

[↑ فهرست مطالب](#)

## ۲۱۰. ریداکس رو فقط با ری اکت میشه استفاده کرد؟

ریداکس میتونه به عنوان یه ذخیره داده برای لایه UI استفاده بشه. رایج ترین کاربرد ریداکس برای ری اکت و ری اکت نیتیو هستش، ولی اتصالاتی هم برای Angular، Angular 2، Vue، Mithril و موارد دیگه موجوده. ریداکس به راحتی یه مکانیسم اشتراکی ارائه میده که میتونه برای کد های دیگه هم استفاده بشه.

↑ فهرست مطالب

## ۲۱۱. برای استفاده از Redux به ابزار build خاصی احتیاج داریم؟

ریداکس در اصل توی ES6 نوشته شده و برای تولید توی ES5 با Webpack و Babel منتشر شده. ما باید بتونیم بدون توجه به مراحل ساخت جاوااسکریپت از اون استفاده کنیم. ریداکس همینطور یه ساختار UMD ارائه میده که میتونه مستقیم و بدون هیچگونه مراحل ساخت مورد استفاده قرار بگیره.

↑ فهرست مطالب

## ۲۱۲. مقادیر پیش فرض ریداکس فرم چطوری تغییرات رو از state می گیرن؟

باید تنظیمات `enableReinitialize : true` رو اضافه کنیم.

```
const InitializeFromStateForm = reduxForm({
 form: 'initializeFromState',
 enableReinitialize : true
})(UserEdit)
```

اگه `initialValues` prop به روز بشه، فرممون هم به روز میشه.

↑ فهرست مطالب

## ۲۱۳. توی PropTypes های ری اکت چطوری میشه برای یه prop چند نوع داده مجاز مشخص کرد؟

می‌تونیم از یکی از متد های `PropTypes` به اسم `oneOfType` () استفاده کنیم. برای مثال، ویژگی `height` رو می‌تونیم با دو نوع `string` یا `number` مثل زیر تعریف کنیم:

```
Component.propTypes = {
 size: PropTypes.oneOfType([
 PropTypes.string,
 PropTypes.number
])
}
```

[↑ فهرست مطالب](#)

## ۲۱۴. می‌تونیم فایل `svg` رو به عنوان کامپوننت `import` کنیم؟

می‌تونیم `SVG` رو مستقیماً به عنوان یه کامپوننت به جای لود کردنش به عنوان یه فایل ایمپورت کنیم. این ویژگی توی `react-scripts@2.0.0` و ورژن های بالاتر در دسترسه.

```
import { ReactComponent as Logo } from './logo.svg'

const App = () => (
 <div>
 { /* Logo is an actual react component */ }
 <Logo />
 </div>
)
```

**نکته** فراموش نکنیم که موقع ایمپورت کردن از آکولاد استفاده کنیم.

[↑ فهرست مطالب](#)

## ۲۱۵. چرا استفاده از توابع `ref callback` درون خطی توصیه نمیشه؟

اگه `ref callback` به عنوان یه تابع درون خطی تعریف بشه، در طول به روزرسانی دو بار فراخوانی میشه، یه بار با مقدار `null` و بعد دوباره با عنصر `DOM`. این موضوع به خاطر اینه که یه نمونه جدیدی از تابع با هر بار رندر ساخته میشه، پس ری‌اکت باید `ref` قبلی رو پاک کنه و یه نمونه جدید ایجاد کنه.



```

class UserForm extends Component {
 handleSubmit = () => {
 console.log("Input Value is: ", this.input.value)
 }

 render () {
 return (
 <form onSubmit={this.handleSubmit}>
 <input
 type='text'
 ref={(input) => this.input = input} /> // Access DOM input in t
 <button type='submit'>Submit</button>
 </form>
)
 }
}

```

اما انتظار ما این‌ه که وقتی کامپوننت mount شد، ref callback یه بار صدا زده بشه. یه راه حل سریع استفاده از class property syntax ES6 برای تعریف تابع هستش.

```

class UserForm extends Component {
 handleSubmit = () => {
 console.log("Input Value is: ", this.input.value)
 }

 setSearchInput = (input) => {
 this.input = input
 }

 render () {
 return (
 <form onSubmit={this.handleSubmit}>
 <input
 type='text'
 ref={this.setSearchInput} /> // Access DOM input in handle subm
 <button type='submit'>Submit</button>
 </form>
)
 }
}

```

[↑ فهرست مطالب](#)

۲۱۶. render hijacking توی ری‌اکت چیه؟

مفهوم render hijacking توانایی کنترل اینه که چه کامپوننتی خروجی بقیه کامپوننت هاست. در واقع به این معنیه که ما می تونیم با قرار دادن کامپوننت خودمون توی یه کامپوننت با اولویت بالا یه تغییراتی بهش بدیم، مثلاً یه سری prop بهش اضافه کنیم یا تغییرات دیگه ای که باعث تغییر منطق رندر بشه. این در واقع hijacking رو فعال نمیکنه اما با استفاده از HOC این امکان رو فراهم میکنیم که کامپوننت رفتار متفاوتی داشته باشه.

↑ فهرست مطالب

## ۲۱۷. پیاده سازی factory یا سازنده HOC چگونه؟

دو روش اصلی برای اجرای HOC ها توی ری اکت وجود داره. 1. Props Proxy (PP) و 2. Inheritance Inversion (II). اونا روش های مختلفی رو برای اداره کردن *WrappedComponent* دنبال می کنن.

### Props Proxy

تو این روش، متد رندر HOC یه عنصر ری اکت از نوع *WrappedComponent* رو برمی گردونه. ما هم prop هایی که HOC دریافت میکنه رو انتقال میدیم، به خاطر همین بهش میگیم **Props Proxy**.

```
function ppHOC(WrappedComponent) {
 return class PP extends React.Component {
 render() {
 return <WrappedComponent {...this.props}/>
 }
 }
}
```

**\*\*Inheritance Inversion\*\***

In this approach, the returned HOC class (Enhancer) extends the *WrappedComponent*. It is called Inheritance Inversion because instead of the *WrappedComponent* extending some Enhancer class, it is passively extended by the Enhancer. In this way the relationship between them seems **.inverse**

```
function iiHOC(WrappedComponent) {
 return class Enhancer extends WrappedComponent {
 render() {
 return super.render()
 }
 }
}
```

↑ فهرست مطالب

## ۲۱۸. چطوری به یه کامپوننت ری‌اکت عدد پاس بدیم؟

اعداد رو باید از طریق آکولاد همونطور که رشته رو داخل کوتیشن قرار میدیم، انتقال بدیم.

```
React.render(<User age={30} department={"IT"} />, document.getElementById
```

↑ فهرست مطالب

## ۲۱۹. لازمه همه state ها رو توی ریداکس مدیریت کنیم؟ لزومی به استفاده از state داخلی داریم؟

این به تصمیم توسعه دهنده بستگی داره. به عنوان مثال این وظیفه توسعه دهنده ست که بررسی کنه چه نوعی از state ها برنامه رو تشکیل بده و هر state کجا باید قرار بگیره. بعضی از کاربرا ترجیح میدن هر قسمت از داده رو توی ریداکس نگه دارن، تا همیشه یه ورژن پشت سر هم و کنترل شده از برنامه شون رو داشته باشن. یه عده دیگه ترجیح میدن UI State یا non-critical رو نگه دارن، مثل "is this dropdown currently open"، توی state داخلی یه کامپوننت.

اینا قوانینی هستن که تعیین می کنن چه نوع داده ای باید توی ریداکس قرار بگیره

۱. آیا بقیه قسمتی برنامه به این داده ها اهمیت میدن؟

۲. آیا نیازه که بتونیم یه سری داده ها رو از روی این داده های اصلی به دست

بیاریم؟

۳. آیا از این داده ها توی چندین کامپوننت استفاده میشه؟

۴. آیا نیازه که بتونیم یه state رو به یه بازه زمانی خاصی برگردونیم؟

۵. آیا میخوایم داده رو توی حافظه نگه داریم؟ (یعنی به جای درخواست مجدد،

از اطلاعات موجود توی state استفاده کنیم)

## ۲۲۰. هدف از متد registerServiceWorker توی ری اکت چیه؟

ری اکت به صورت پیش فرض و بدون هیچگونه پیکربندی، یه سرویس دهنده برامون ایجاد میکنه. سرویس دهنده یه API وب هستش که در ذخیره کردن asset ها و فایل های دیگه بهمون کمک میکنه تا وقتی کاربر آفلاینه یا سرعت اینترنتش پایینه، بازم بتونه نتایج رو روی صفحه ببینه. به این ترتیب بهمون کمک میکنه تجربه کاربری بهتری ایجاد کنیم و همون چیزیه که باید در مورد service worker ها بدونیم. این ها همه مواردی بود در رابطه با اضافه کردن قابلیت های آفلاین به سایتمون.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

ReactDOM.render(<App />, document.getElementById('root'));
registerServiceWorker();
```

## ۲۲۱. تابع memo ری اکت چیه؟

Class components can be restricted from rendering when their input props are the same using **PureComponent** or **shouldComponentUpdate**. Now you can do the same with function components by wrapping them in **.React.memo**

```
const MyComponent = React.memo(function MyComponent(props) {
 /* only rerenders if props change */
});
```

## ۲۲۲. تابع lazy ری اکت چیه؟

The `React.lazy` function lets you render an dynamic import as a regular component. It will automatically load the bundle containing the `OtherComponent` when the component gets rendered. This must return a Promise which resolves to a module with a default export containing a `.React` component

```
const OtherComponent = React.lazy(() => import("./OtherComponent"));

function MyComponent() {
 return (
 <div>
 <OtherComponent />
 </div>
);
}
```

Note:\*\* `React.lazy` and `Suspense` is not yet available for server-side rendering. If\*\* you want to do code-splitting in a server rendered app, we still recommend `React.Loadable`

[↑ فهرست مطالب](#)

## ۲۲۳. چطوری با استفاده از تابع `setState` از رندر غیرضروری جلوگیری کنیم؟

You can compare current value of the state with an existing state value and decide whether to rerender the page or not. If the values are same then you need to return **null** to stop rerendering otherwise return the latest state value. For example, the user profile information is conditionally rendered as follows,

```
getUserProfile = (user) => {
 const latestAddress = user.address;
 this.setState((state) => {
 if (state.address === latestAddress) {
 return null;
 } else {
 return { title: latestAddress };
 }
 });
};
```

## ۲۲۴. توی نسخه ۱۶ ری اکت چطوری میشه آرایه، Strings و یا عدد رو رندر کنیم؟

**Arrays:** Unlike older releases, you don't need to make sure **render** method return a single element in React16. You are able to return multiple sibling elements without a wrapping element by returning an array. For example, let us take the below list of developers

```
const ReactJSDevs = () => {
 return [
 <li key="1">John,
 <li key="2">Jackie,
 <li key="3">Jordan,
];
};
```

You can also merge this array of items in another array component

```
const JSDevs = () => {
 return (

 Brad
 Brodge
 <ReactJSDevs />
 Brandon

);
};
```

**Strings and Numbers:** You can also return string and number type from the render method

```
render() {
 return 'Welcome to ReactJS questions';
}
// Number
render() {
 return 2018;
}
```

## ۲۲۵. چطوری همیشه از تعریف ویژگی در کلاس کامپوننت استفاده کرد؟

React Class Components can be made much more concise using the class field declarations. You can initialize local state without using the constructor and declare class methods by using arrow functions without the extra need to bind them. Let's take a counter example to demonstrate class field declarations for state without using constructor and methods without ,binding

```
class Counter extends Component {
 state = { value: 0 };

 handleIncrement = () => {
 this.setState((prevState) => ({
 value: prevState.value + 1,
 }));
 };

 handleDecrement = () => {
 this.setState((prevState) => ({
 value: prevState.value - 1,
 }));
 };

 render() {
 return (
 <div>
 {this.state.value}

 <button onClick={this.handleIncrement}>+</button>
 <button onClick={this.handleDecrement}>-</button>
 </div>
);
 }
}
```

[↑ فهرست مطالب](#)

## ۲۲۶. hook ها چی هستن؟

Hooks is a new feature that lets you use state and other React features ,without writing a class. Let's see an example of useState hook example

```
import { useState } from "react";

function Example() {
 // Declare a new state variable, which we'll call "count"
 const [count, setCount] = useState(0);

 return (
 <div>
 <p>You clicked {count} times</p>
 <button onClick={() => setCount(count + 1)}>Click me</button>
 </div>
);
}
```

[↑ فهرست مطالب](#)

## ۲۲۷. چه قوانینی برای هوک‌ها باید رعایت بشن؟

- You need to follow two rules inorder to use hooks
۱. Call Hooks only at the top level of your react functions. i.e, You shouldn't call Hooks inside loops, conditions, or nested functions. This will ensure that Hooks are called in the same order each time a component renders and it preserves the state .of Hooks between multiple useState and useEffect calls
  ۲. Call Hooks from React Functions only. i.e, You shouldn't call .Hooks from regular JavaScript functions

[↑ فهرست مطالب](#)

## ۲۲۸. چطوری میشه از استفاده درست هوک‌ها اطمینان حاصل کرد؟

React team released an ESLint plugin called **eslint-plugin-react-hooks** that enforces these two rules. You can add this plugin to your project using the ,below command



```
npm install eslint-plugin-react-hooks@next
```

And apply the below config in your ESLint config file

```
// Your ESLint configuration
{
 "plugins": [
 // ...
 "react-hooks"
],
 "rules": {
 // ...
 "react-hooks/rules-of-hooks": "error"
 }
}
```

.Note:\*\* This plugin is intended to use in Create React App by default\*\*

[↑ فهرست مطالب](#)

## ۲۲۹. تفاوت‌های Flux و Redux چیا هستن؟

Below are the major differences between Flux and Redux

Redux	Flux
State is immutable	State is mutable
The Store and change logic are separate	The Store contains both state and change logic
There is only one store exist	There are multiple stores exist
Single store with hierarchical reducers	All the stores are disconnected and flat
There is no concept of dispatcher	It has a singleton dispatcher
Container components uses connect function	React components subscribe to the store

## ۲۳۰. مزایای ری‌اکت روتر نسخه ۴ چیه؟

Below are the main benefits of React Router V4 module

۱. In React Router v4(version 4), the API is completely about components. A router can be visualized as a single component() which wraps specific child router components.
۲. You don't need to manually set history. The router module will take care history by wrapping routes with component.
۳. The application size is reduced by adding only the specific router module(Web, core, or native)

## ۲۳۱. می‌تونن راجع به متد `componentDidCatch` توضیح بدی؟

The **componentDidCatch** lifecycle method is invoked after an error has been thrown by a descendant component. The method receives two parameters

۱. error: - The error object which was thrown
۲. info: - An object with a `componentStack` key contains the information about which component threw the error

The method structure would be as follows

```
;(componentDidCatch(error, info
```

## ۲۳۲. در چه سناریوئی `error boundary` خطا رو `catch` نمی‌کنه؟

Below are the cases in which error boundaries doesn't work

۱. Inside Event handlers
۲. Asynchronous code using **setTimeout** or **requestAnimationFrame** callbacks

۳. During Server side rendering

۴. When errors thrown in the error boundary code itself

[↑ فهرست مطالب](#)

## ۲۳۳. چرا نیازی به error boundaries برای event handler نیست؟

Error boundaries do not catch errors inside event handlers. Event handlers don't happen or are invoked during rendering time unlike render method or lifecycle methods. So React knows how to recover these kind of errors in event handlers.

If still you need to catch an error inside event handler, use the regular JavaScript try / catch statement as below

```
class MyComponent extends React.Component {
 constructor(props) {
 super(props);
 this.state = { error: null };
 }

 handleClick = () => {
 try {
 // Do something that could throw
 } catch (error) {
 this.setState({ error });
 }
 };

 render() {
 if (this.state.error) {
 return <h1>Caught an error.</h1>;
 }
 return <div onClick={this.handleClick}>Click Me</div>;
 }
}
```

The above code is catching the error using vanilla javascript try/catch block instead of error boundaries.

[↑ فهرست مطالب](#)

## ۲۳۴. تفاوت بلوک try catch و error boundary چیست؟

Try catch block works with imperative code whereas error boundaries are meant for declarative code to render on the screen

For example, the try catch block used for below imperative code

```
try {
 showButton();
} catch (error) {
 // ...
}
```

,Whereas error boundaries wrap declarative code as below

```
<ErrorBoundary>
 <MyComponent />
</ErrorBoundary>
```

So if an error occurs in a **componentDidUpdate** method caused by a **setState** somewhere deep in the tree, it will still correctly propagate to the closest error boundary

[↑ فهرست مطالب](#)

## ۲۳۵. رفتار خطاهای uncaught در ری اکت 16 چیست؟

In React 16, errors that were not caught by any error boundary will result in unmounting of the whole React component tree. The reason behind this decision is that it is worse to leave corrupted UI in place than to completely remove it. For example, it is worse for a payments app to display a wrong amount than to render nothing

[↑ فهرست مطالب](#)

## ۲۳۶. محل مناسب برای قرار دادن error boundary کجاست؟

The granularity of error boundaries usage is up to the developer based on project needs. You can follow either of these approaches

You can wrap top-level route components to display a generic .error message for the entire application .  
You can also wrap individual components in an error boundary to .protect them from crashing the rest of the application

↑ فهرست مطالب

## ۲۳۷. مزیت چاپ شدن stack trace کامپوننت‌ها توی متن ارور boundary ری‌اکت چیه؟

Apart from error messages and javascript stack, React16 will display the component stack trace with file names and line numbers using error boundary concept. For example, BuggyCounter component displays the component stack trace as below

```
► React caught an error thrown by BuggyCounter. You should fix this error in your code. react-dom.development.js:7708
React will try to recreate this component tree from scratch using the error boundary you provided, ErrorBoundary.
Error: I crashed!
The error is located at:
 in BuggyCounter (at App.js:26)
 in ErrorBoundary (at App.js:21)
 in div (at App.js:8)
 in App (at index.js:5)
```

↑ فهرست مطالب

## ۲۳۸. متدی که در تعریف کامپوننت‌های class الزامیه؟

The render() method is the only required method in a class component. i.e, .All methods other than render method are optional for a class component

↑ فهرست مطالب

## ۲۳۹. نوع‌های ممکن برای مقدار بازگشتی متد render چیا هستن؟

,Below are the list of following types used and return from render method

**React elements:** Elements that instruct React to render a DOM .  
node. It includes html elements such as `<div/>` and user

.defined elements

**Arrays and fragments:** Return multiple elements to render as .۲

Arrays and Fragments to wrap multiple elements

**.Portals:** Render children into a different DOM subtree .۳

**String and numbers:** Render both Strings and Numbers as text .۴

nodes in the DOM

**Booleans or null:** Doesn't render anything but these types are .۵

.used to conditionally render content

[↑ فهرست مطالب](#)

## ۲۴۰. هدف اصلی از متد constructor چیه؟

,The constructor is mainly used for two purposes

To initialize local state by assigning object to this.state . ۱

For binding event handler methods to the instance .۲

,For example, the below code covers both the above cases

```
} (constructor(props
;(super(props
!Don't call this.setState() here //
;{ this.state = { counter: 0
;(this.handleClick = this.handleClick.bind(this
{
```

[↑ فهرست مطالب](#)

## ۲۴۱. آیا تعریف متد سازنده توی ری اکت الزامیه؟

No, it is not mandatory. i.e, If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React .component

[↑ فهرست مطالب](#)

## ۲۴۲. Default prop ها چی هستن؟

The defaultProps are defined as a property on the component class to set the default props for the class. This is used for undefined props, but not for null props. For example, let us create color default prop for the button component

```
class MyButton extends React.Component {
 // ...
}

MyButton.defaultProps = {
 color: "red",
};
```

If props.color is not provided then it will set the default value to 'red'. i.e, Whenever you try to access the color prop it uses default value

```
render() {
 return <MyButton /> ; // props.color will be set to red
}
```

.Note:\*\* If you provide null value then it remains null value\*\*

[↑ فهرست مطالب](#)

## ۲۴۳. چرا نباید تابع setState رو توی متد componentWillUnmount فراخوانی کرد؟

You should not call setState() in componentWillUnmount() because Once a component instance is unmounted, it will never be mounted again

[↑ فهرست مطالب](#)

## ۲۴۴. کاربرد متد getDerivedStateFromError چیه؟

This lifecycle method is invoked after an error has been thrown by a descendant component. It receives the error that was thrown as a

parameter and should return a value to update state. The signature of the lifecycle method is as follows

```
static getDerivedStateFromError(error)
```

Let us take error boundary use case with the above lifecycle method for demonstration purpose

```
class ErrorBoundary extends React.Component {
 constructor(props) {
 super(props);
 this.state = { hasError: false };
 }

 static getDerivedStateFromError(error) {
 // Update state so the next render will show the fallback UI.
 return { hasError: true };
 }

 render() {
 if (this.state.hasError) {
 // You can render any custom fallback UI
 return <h1>Something went wrong.</h1>;
 }

 return this.props.children;
 }
}
```

[↑ فهرست مطالب](#)

## ۲۴۵. کدام متدها و به چه ترتیبی در طول ری‌رندر فراخوانی میشوند؟

An update can be caused by changes to props or state. The below methods are called in the following order when a component is being re-rendered

۱. `(static) getDerivedStateFromProps`
۲. `(should)ComponentUpdate`
۳. `(render)`
۴. `(get)SnapshotBeforeUpdate`
۵. `(component)DidUpdate`



## ۲۴۶. کدوم متدها موقع error handling فراخوانی میشن؟

Below methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component

۱. `()static getDerivedStateFromError`

۲. `()componentDidCatch`

## ۲۴۷. کارکرد ویژگی `displayName` چیه؟

The `displayName` string is used in debugging messages. Usually, you don't need to set it explicitly because it's inferred from the name of the function or class that defines the component. You might want to set it explicitly if you want to display a different name for debugging purposes or when you create a higher-order component.

For example, To ease debugging, choose a display name that communicates that it's the result of a `withSubscription` HOC

```
function withSubscription(WrappedComponent) {
 class WithSubscription extends React.Component {
 /* ... */
 }
 WithSubscription.displayName = `WithSubscription(${getDisplayName(
 WrappedComponent
)})`;
 return WithSubscription;
}
function getDisplayName(WrappedComponent) {
 return (
 WrappedComponent.displayName || WrappedComponent.name || "Component"
);
}
```

## ۲۴۸. سائپورت مرورگرها برای برنامه ری اکتی چگونه؟

React supports all popular browsers, including Internet Explorer 9 and above, although some polyfills are required for older browsers such as IE 9 and IE 10. If you use **es5-shim** and **es5-sham** polyfill then it even support old browsers that doesn't support ES5 methods.

[↑ فهرست مطالب](#)

## ۲۴۹. هدف از متد `unmountComponentAtNode` چیست؟

This method is available from react-dom package and it removes a mounted React component from the DOM and clean up its event handlers and state.

If no component was mounted in the container, calling this function does nothing. Returns true if a component was unmounted and false if there was no component to unmount.

The method signature would be as follows

```
ReactDOM.unmountComponentAtNode(container);
```

[↑ فهرست مطالب](#)

## ۲۵۰. `code-splitting` چیست؟

Code-Splitting is a feature supported by bundlers like Webpack and Browserify which can create multiple bundles that can be dynamically loaded at runtime. The react project supports code splitting via dynamic `import()` feature.

For example, in the below code snippets, it will make moduleA.js and all its unique dependencies as a separate chunk that only loads after the user clicks the 'Load' button.

**moduleA.js**

```
const moduleA = "Hello";

export { moduleA };
```

```
import React, { Component } from "react";

class App extends Component {
 handleClick = () => {
 import("./moduleA")
 .then(({ moduleA }) => {
 // Use moduleA
 })
 .catch((err) => {
 // Handle failure
 });
 };

 render() {
 return (
 <div>
 <button onClick={this.handleClick}>Load</button>
 </div>
);
 }
}

export default App;
```

[↑ فهرست مطالب](#)

## ۲۵۱. مزایای حالت strict چیه؟

The will be helpful in the below cases

۱. Identifying components with **unsafe lifecycle methods**
۲. Warning about **legacy string ref** API usage
۳. Detecting unexpected **side effects**
۴. Detecting **legacy context** API
۵. Warning about deprecated findDOMNode usage

[↑ فهرست مطالب](#)

## ۲۵۲. Fragment های دارای key هستن؟

The Fragments declared with the explicit `<React.Fragment>` syntax may have keys. The general usecase is mapping a collection to an array of fragments as below

```
function Glossary(props) {
 return (
 <dl>
 {props.items.map((item) => (
 // Without the `key`, React will fire a key warning
 <React.Fragment key={item.id}>
 <dt>{item.term}</dt>
 <dd>{item.description}</dd>
 </React.Fragment>
))}
 </dl>
);
}
```

Note: key is the only attribute that can be passed to Fragment. In the future, there might be a support for additional attributes, such as event handlers

[↑ فهرست مطالب](#)

## ۲۵۳. آیا ری اکت از تمامی attribute های HTML پشتیبانی می‌کند؟

As of React 16, both standard or custom DOM attributes are fully supported. Since React components often take both custom and DOM-related props, React uses the camelCase convention just like the DOM APIs. Let us take a few props with respect to standard HTML attributes

```
<div tabIndex="-1" /> // Just like node.tabIndex DOM API
<div className="Button" /> // Just like node.className DOM API
<input readOnly={true} /> // Just like node.readOnly DOM API
```

These props work similarly to the corresponding HTML attributes, with the exception of the special cases. It also support all SVG attributes

[↑ فهرست مطالب](#)

## ۲۵۴. محدودیت‌های HOC ها چی هستن؟

Higher-order components come with a few caveats apart from its benefits.

Below are the few listed in an order

### ۱. Don't Use HOCs Inside the render Method .

It is not recommended to apply a HOC to a component within the .render method of a component

```
} ()render
A new version of EnhancedComponent is created on every render //
EnhancedComponent1 !== EnhancedComponent2 //
;(const EnhancedComponent = enhance(MyComponent
!That causes the entire subtree to unmount/remount each time //
;</ return <EnhancedComponent
{
```

The above code impact performance by remounting a component that causes the state of that component and all of its children to be lost. Instead, apply HOCs outside the component definition so that the resulting component is created only once ۲. **\*\*Static Methods Must Be Copied Over:\*\*** When you apply a HOC to a component the new component does not have any of the static methods of the original component

```
// Define a static method
WrappedComponent.staticMethod = function () {
 /*...*/
};
// Now apply a HOC
const EnhancedComponent = enhance(WrappedComponent);

// The enhanced component has no static method
typeof EnhancedComponent.staticMethod === "undefined"; // true
```

You can overcome this by copying the methods onto the container before returning it

```
function enhance(WrappedComponent) {
 class Enhance extends React.Component {
 /*...*/
 }
 // Must know exactly which method(s) to copy :(
 Enhance.staticMethod = WrappedComponent.staticMethod;
 return Enhance;
}
```

Refs Aren't Passed Through:\*\* For HOCs you need to pass through all props to\*\* the wrapped component but this does not work for refs. This is because ref is not really a prop similar to key. In this case you need to use the React.forwardRef API

↑ فهرست مطالب

## ۲۵۵. چطوری همیشه forwardRefs رو توی DevTools دیباگ کرد؟

**React.forwardRef** accepts a render function as parameter and DevTools uses this function to determine what to display for the ref forwarding component. For example, If you don't name the render function or not using `displayName` property then it will appear as "ForwardRef" in the DevTools

```
const WrappedComponent = React.forwardRef((props, ref) => {
 return <LogProps {...props} forwardedRef={ref} />;
});
```

But If you name the render function then it will appear as

ForwardRef(myFunction)

```
const WrappedComponent = React.forwardRef(function myFunction(props, ref)
 return <LogProps {...props} forwardedRef={ref} />;
});
```

As an alternative, You can also set `displayName` property for `forwardRef` function

```
function logProps(Component) {
 class LogProps extends React.Component {
 // ...
 }

 function forwardRef(props, ref) {
 return <LogProps {...props} forwardedRef={ref} />;
 }

 // Give this component a more helpful display name in DevTools.
 // e.g. "ForwardRef(logProps(MyComponent))"
 const name = Component.displayName || Component.name;
 forwardRef.displayName = `logProps(${name})`;

 return React.forwardRef(forwardRef);
}
```

[↑ فهرست مطالب](#)

## ۲۵۶. مقدار یه props کامپوننت کی true میشه؟

If you pass no value for a prop, it defaults to true. This behavior is available so that it matches the behavior of HTML. For example, below expressions ,are equivalent

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

Note:\*\* It is not recommend using this approach because it can be confused with the\*\*  
({ES۶ object shorthand (example, {name} which is short for {name: name

[↑ فهرست مطالب](#)

## ۲۵۷. NextJS چیه و ویژگی‌های اصلیش چیا هستن؟

Next.js is a popular and lightweight framework for static and server-rendered applications built with React. It also provides styling and ,routing solutions. Below are the major features provided by NextJS

- ۱. Server-rendered by default
- ۲. Automatic code splitting for faster page loads
- ۳. (Simple client-side routing (page based
- ۴. (Webpack-based dev environment which supports (HMR
- ۵. Able to implement with Express or any other Node.js HTTP server
- ۶. Customizable with your own Babel and Webpack configurations

[↑ فهرست مطالب](#)

## ۲۵۸. چطوری کی تونیم یه تابع event handler رو به یه کامپوننت پاس بدیم؟

You can pass event handlers and other functions as props to child components. It can be used in child component as below

```
<button onClick={this.handleClick}>
```

[↑ فهرست مطالب](#)

## ۲۵۹. استفاده از توابع arrow برای متدهای render خوبه؟

Yes, You can use. It is often the easiest way to pass parameters to callback functions. But you need to optimize the performance while using it

```
class Foo extends Component {
 handleClick() {
 console.log("Click happened");
 }
 render() {
 return <button onClick={() => this.handleClick()}>Click Me</button>;
 }
}
```

Note:\*\* Using an arrow function in render method creates a new function each time\*\* the component renders, which may have performance implications



## ۲۶۰. چطوری از اجرای چندباره یه تابع جلوگیری کنیم؟

If you use an event handler such as **onClick** or **onScroll** and want to prevent the callback from being fired too quickly, then you can limit the rate at which callback is executed. This can be achieved in the below possible ways

**Throttling:** Changes based on a time based frequency. For .۱ example, it can be used using `_throttle` lodash function

**Debouncing:** Publish changes after a period of inactivity. For .۲ example, it can be used using `_debounce` lodash function

**RequestAnimationFrame throttling:** Changes based on .۳ requestAnimationFrame. For example, it can be used using `raf-schd` lodash function

## ۲۶۱. JSX چطوری از حمله‌های Injection جلوگیری می‌کنه؟

React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that's not explicitly written in your application. Everything is converted to a string before being rendered. For example, you can embed user input as below

```
const name = response.potentiallyMaliciousInput;
const element = <h1>{name}</h1>;
```

.This way you can prevent XSS(Cross-site-scripting) attacks in the application

## ۲۶۲. چطوری element رندر شده رو آپدیت کنیم؟

You can update UI(represented by rendered element) by passing the newly created element to ReactDOM's render method. For example, lets take a

ticking clock example, where it updates the time by calling render method multiple times

```
function tick() {
 const element = (
 <div>
 <h1>Hello, world!</h1>
 <h2>It is {new Date().toLocaleTimeString()}</h2>
 </div>
);
 ReactDOM.render(element, document.getElementById("root"));
}

setInterval(tick, 1000);
```

[↑ فهرست مطالب](#)

## ۲۶۳. چرا prop ها read only هستند؟

When you declare a component as a function or a class, it must never modify its own props. Let us take a below capital function

```
function capital(amount, interest) {
 return amount + interest;
}
```

The above function is called “pure” because it does not attempt to change their inputs, and always return the same result for the same inputs. Hence, React has a single rule saying "All React components must act like pure functions with respect to their props".

[↑ فهرست مطالب](#)

## ۲۶۴. چرا می‌گیم تابع setState از طریق merge کردن state را مدیریت می‌کنه؟

When you call setState() in the component, React merges the object you provide into the current state. For example, let us take a facebook user with posts and comments details as state variables

```

constructor(props) {
 super(props);
 this.state = {
 posts: [],
 comments: []
 };
}

```

Now you can update them independently with separate `setState()` calls as below

```

componentDidMount() {
 fetchPosts().then(response => {
 this.setState({
 posts: response.posts
 });
 });

 fetchComments().then(response => {
 this.setState({
 comments: response.comments
 });
 });
}

```

As mentioned in the above code snippets, `this.setState({ comments })` updates only `.comments` variable without modifying or replacing `posts` variable

[↑ فهرست مطالب](#)

## ۲۶۵. چطوری می‌تونیم به متد `event handler` پارامتر پاس بدیم؟

During iterations or loops, it is common to pass an extra parameter to an event handler. This can be achieved through arrow functions or `bind` method. Let us take an example of user details updated in a grid

```

onClick={(e) => this.updateUser(userId, e)}>Update User details</button>
onClick={this.updateUser.bind(this, userId)}>Update User details</button>

```

In both the approaches, the synthetic argument `e` is passed as a second argument. You need to pass it explicitly for arrow functions and it forwarded

.automatically for bind method

[↑ فهرست مطالب](#)

## ۲۶۶. چطوری از رندر مجدد کامپوننت‌ها جلوگیری کنیم؟

You can prevent component from rendering by returning null based on .specific condition. This way it can conditionally render component

```
function Greeting(props) {
 if (!props.loggedIn) {
 return null;
 }

 return <div className="greeting">welcome, {props.name}</div>;
}
```

```
class User extends React.Component {
 constructor(props) {
 super(props);
 this.state = {loggedIn: false, name: 'John'};
 }

 render() {
 return (
 <div>
 //Prevent component render if it is not loggedIn
 <Greeting loggedIn={this.state.loggedIn} />
 <UserDetails name={this.state.name}>
 </div>
);
 }
}
```

In the above example, the greeting component skips its rendering section by applying .condition and returning null value

[↑ فهرست مطالب](#)

## ۲۶۷. شرایطی که بدون مشکل پرفورمنس بتونیم از ایندکس به عنوان key استفاده کنیم چی هست؟

- There are three conditions to make sure, it is safe use the index as a key
۱. The list and items are static– they are not computed and do not change
  ۲. The items in the list have no ids
  ۳. The list is never reordered or filtered

[↑ فهرست مطالب](#)

## ۲۶۸. key های ری اکت باید به صورت عمومی منحصر بفرد باشن؟

Keys used within arrays should be unique among their siblings but they don't need to be globally unique. i.e, You can use the same keys with two different arrays. For example, the below book component uses two arrays with different arrays

```
function Book(props) {
 const index = (

 {props.pages.map((page) => (
 <li key={page.id}>{page.title}
))}

);
 const content = props.pages.map((page) => (
 <div key={page.id}>
 <h3>{page.title}</h3>
 <p>{page.content}</p>
 <p>{page.pageNumber}</p>
 </div>
));
 return (
 <div>
 {index}
 <hr />
 {content}
 </div>
);
}
```

[↑ فهرست مطالب](#)

## ۲۶۹. گزینه‌های محبوب برای مدیریت فرم‌ها توی ری‌اکت چیا هستن؟

Formik is a form library for react which provides solutions such as validation, keeping track of the visited fields, and handling form submission.

In detail, You can categorize them as follows

۱. Getting values in and out of form state

۲. Validation and error messages

۳. Handling form submission

It is used to create a scalable, performant, form helper with a minimal API to solve annoying stuff

↑ فهرست مطالب

## ۲۷۰. مزایای کتابخانه فرمیک نسبت به redux form چیه؟

Below are the main reasons to recommend formik over redux form library

۱. The form state is inherently short-term and local, so tracking it in Redux (or any kind of Flux library) is unnecessary

۲. Redux-Form calls your entire top-level Redux reducer multiple times ON EVERY SINGLE KEYSTROKE. This way it increases

input latency for large apps

۳. Redux-Form is 22.5 kB minified gzipped whereas Formik is 12.7 kB

↑ فهرست مطالب

## ۲۷۱. چرا اجباری برای استفاده از ارث‌بری توی ری‌اکت نیست؟ مزیتی داره؟

In React, it is recommend using composition instead of inheritance to reuse code between components. Both Props and composition give you all the flexibility you need to customize a component's look and behavior in an explicit and safe way

Whereas, If you want to reuse non-UI functionality between components, it is suggested to extracting it into a separate JavaScript module. Later components import it and use that function, object, or a class, without .extending it

[↑ فهرست مطالب](#)

## ۲۷۲. می‌تونیم از web components توی برنامه ری‌اِکت استفاده کنیم؟

Yes, you can use web components in a react application. Even though many developers won't use this combination, it may require especially if you are using third-party UI components that are written using Web Components. ,For example, let us use Vaadin date picker web component as below

```
import React, { Component } from "react";
import "./App.css";
import "@vaadin/vaadin-date-picker";
class App extends Component {
 render() {
 return (
 <div className="App">
 <vaadin-date-picker label="When were you born?"></vaadin-date-pi
 </div>
);
 }
}
export default App;
```

[↑ فهرست مطالب](#)

## ۲۷۳. dynamic import چیه؟

The dynamic import() syntax is a ECMAScript proposal not currently part of the language standard. It is expected to be accepted in the near future. You can achieve code-splitting into your app using dynamic import(). Let's take ,an example of addition

**۱. Normal Import**

```
import { add } from "./math";
console.log(add(10, 20));
```

**\*\*Dynamic Import\*\*** ۲

```
import("./math").then((math) => {
 console.log(math.add(10, 20));
});
```

[↑ فهرست مطالب](#)

## ۲۷۴. **loadable component** چي هستن؟

If you want to do code-splitting in a server rendered app, it is recommend to use Loadable Components because React.lazy and Suspense is not yet available for server-side rendering. Loadable lets you render a dynamic ,import as a regular component. Lets take an example

```
import loadable from "@loadable/component";

const OtherComponent = loadable(() => import("./OtherComponent"));

function MyComponent() {
 return (
 <div>
 <OtherComponent />
 </div>
);
}
```

Now OtherComponent will be loaded in a separated bundle

[↑ فهرست مطالب](#)

## ۲۷۵. **suspense** کامپوننت چيه؟

If the module containing the dynamic import is not yet loaded by the time parent component renders, you must show some fallback content while you're waiting for it to load using a loading indicator. This can be done using



**Suspense** component. For example, the below code uses suspense component

```
const OtherComponent = React.lazy(() => import("./OtherComponent"));

function MyComponent() {
 return (
 <div>
 <Suspense fallback={<div>Loading...</div>}>
 <OtherComponent />
 </Suspense>
 </div>
);
}
```

.As mentioned in the above code, Suspense is wrapped above the lazy component

[↑ فهرست مطالب](#)

## ۲۷۶. چطوری به ازای route می‌تونیم code splitting داشته باشیم؟

One of the best place to do code splitting is with routes. The entire page is going to re-render at once so users are unlikely to interact with other elements in the page at the same time. Due to this, the user experience won't be disturbed. Let us take an example of route based website using libraries like React Router with React.lazy

```
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import React, { Suspense, lazy } from "react";

const Home = lazy(() => import("./routes/Home"));
const About = lazy(() => import("./routes/About"));

const App = () => (
 <Router>
 <Suspense fallback=<div>Loading...</div>>
 <Switch>
 <Route exact path="/" component={Home} />
 <Route path="/about" component={About} />
 </Switch>
 </Suspense>
 </Router>
);
```

.In the above code, the code splitting will happen at each route level

[↑ فهرست مطالب](#)

## ۲۷۷. یه مثال از نحوه استفاده از context میزنی؟

**Context** is designed to share data that can be considered **global** for a tree of React components. For example, in the code below lets manually thread through a “theme” prop in order to style the Button component

```
// Lets create a context with a default theme value "luna"
const ThemeContext = React.createContext("luna");
// Create App component where it uses provider to pass theme value in the tree
class App extends React.Component {
 render() {
 return (
 <ThemeContext.Provider value="nova">
 <Toolbar />
 </ThemeContext.Provider>
);
 }
}
// A middle component where you don't need to pass theme prop anymore
function Toolbar(props) {
 return (
 <div>
 <ThemedButton />
 </div>
);
}
// Lets read theme value in the button component to use
class ThemedButton extends React.Component {
 static contextType = ThemeContext;
 render() {
 return <Button theme={this.context} />;
 }
}
```

[↑ فهرست مطالب](#)

## ۲۷۸. هدف از مقدار پیش فرض توی context چیه؟

The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This can be helpful for testing components in isolation without wrapping them. Below code snippet provides default theme value as Luna

```
const MyContext = React.createContext(defaultValue);
```

[↑ فهرست مطالب](#)

## ۲۷۹. چطوری از contextType استفاده می‌کنیم؟

ContextType is used to consume the context object. The contextType property can be used in two ways

### ۱. contextType as property of class

The contextType property on a class can be assigned a Context object created by `React.createContext()`. After that, you can consume the nearest current value of that Context type using `this.context` in any of the lifecycle methods and render function. Lets assign contextType property on MyClass as below

```
class MyClass extends React.Component {
 componentDidMount() {
 let value = this.context;
 /* perform a side-effect at mount using the value of MyContext */
 }
 componentDidUpdate() {
 let value = this.context;
 /* ... */
 }
 componentWillUnmount() {
 let value = this.context;
 /* ... */
 }
 render() {
 let value = this.context;
 /* render something based on the value of MyContext */
 }
}
MyClass.contextType = MyContext;
```

Static field\*\* You can use a static class field to initialize your contextType using\*\* ۲. public class field syntax

```
class MyClass extends React.Component {
 static contextType = MyContext;
 render() {
 let value = this.context;
 /* render something based on the value */
 }
}
```

↑ فهرست مطالب

## ۲۸۰. consumer چیست؟

A Consumer is a React component that subscribes to context changes. It requires a function as a child which receives current context value as argument and returns a react node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree. Lets take a simple example

```
<MyContext.Consumer>
 {value => /* render something based on the context value */}
</MyContext.Consumer>
```

[↑ فهرست مطالب](#)

## ۲۸۱. چطوری مسائل مربوط به پرفورمنس با context رو حل می‌کنین؟

The context uses reference identity to determine when to re-render, there are some gotchas that could trigger unintentional renders in consumers when a provider's parent re-renders. For example, the code below will re-render all consumers every time the Provider re-renders because a new object is always created for value

```
class App extends React.Component {
 render() {
 return (
 <Provider value={{ something: "something" }}>
 <Toolbar />
 </Provider>
);
 }
}
```

,This can be solved by lifting up the value to parent state

```

class App extends React.Component {
 constructor(props) {
 super(props);
 this.state = {
 value: { something: "something" },
 };
 }

 render() {
 return (
 <Provider value={this.state.value}>
 <Toolbar />
 </Provider>
);
 }
}

```

[↑ فهرست مطالب](#)

## ۲۸۲. هدف از forward ref توی HOC ها چیه؟

Refs will not get passed through because ref is not a prop. It handled differently by React just like **key**. If you add a ref to a HOC, the ref will refer to the outermost container component, not the wrapped component. In this case, you can use Forward Ref API. For example, we can explicitly forward .refs to the inner FancyButton component using the React.forwardRef API

,The below HOC logs all props

```
function logProps(Component) {
 class LogProps extends React.Component {
 componentDidUpdate(prevProps) {
 console.log("old props:", prevProps);
 console.log("new props:", this.props);
 }

 render() {
 const { forwardedRef, ...rest } = this.props;

 // Assign the custom prop "forwardedRef" as a ref
 return <Component ref={forwardedRef} {...rest} />;
 }
 }

 return React.forwardRef((props, ref) => {
 return <LogProps {...props} forwardedRef={ref} />;
 });
}
```

Let's use this HOC to log all props that get passed to our “fancy button” component

```
class FancyButton extends React.Component {
 focus() {
 // ...
 }

 // ...
}
export default logProps(FancyButton);
```

Now let's create a ref and pass it to FancyButton component. In this case, you can set  
.focus to button element

```
import FancyButton from "./FancyButton";

const ref = React.createRef();
ref.current.focus();
<FancyButton label="Click Me" handleClick={handleClick} ref={ref} />;
```

[↑ فهرست مطالب](#)

۲۸۳. توی کامپوننت‌ها می‌تونیم پراپ ref داشته باشیم؟

Regular function or class components don't receive the ref argument, and ref is not available in props either. The second ref argument only exists when you define a component with React.forwardRef call

[↑ فهرست مطالب](#)

## ۲۸۴. چرا در هنگام استفاده از ForwardRef ها نیاز به احتیاط بیشتری در استفاده از کتابخانه های جانبی داریم؟

When you start using forwardRef in a component library, you should treat it as a breaking change and release a new major version of your library. This is because your library likely has a different behavior such as what refs get assigned to, and what types are exported. These changes can break apps and other libraries that depend on the old behavior

[↑ فهرست مطالب](#)

## ۲۸۵. چطوری بدون استفاده از ES6 کلاس کامپوننت بسازیم؟

If you don't use ES6 then you may need to use the create-react-class module instead. For default props, you need to define getDefaultProps() as a function on the passed object. Whereas for initial state, you have to provide a separate getInitialState method that returns the initial state



```
var Greeting = createReactClass({
 getDefaultProps: function () {
 return {
 name: "Jhohn",
 };
 },
 getInitialState: function () {
 return { message: this.props.message };
 },
 handleClick: function () {
 console.log(this.state.message);
 },
 render: function () {
 return <h1>Hello, {this.props.name}</h1>;
 },
});
```

Note:\*\* If you use createReactClass then autobinding is available for all methods.\*\*  
 .i.e, You don't need to use .bind(this) with in constructor for event handlers

[↑ فهرست مطالب](#)

## ۲۸۶. استفاده از ری اکت بدون JSX ممکن است؟

Yes, JSX is not mandatory for using React. Actually it is convenient when you don't want to set up compilation in your build environment. Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children). For example, let us take a greeting example with JSX

```
class Greeting extends React.Component {
 render() {
 return <div>Hello {this.props.message}</div>;
 }
}

ReactDOM.render(
 <Greeting message="World" />,
 document.getElementById("root")
);
```

,You can write the same code without JSX as below

```
class Greeting extends React.Component {
 render() {
 return React.createElement("div", null, `Hello ${this.props.message}`)
 }
}

ReactDOM.render(
 React.createElement(Greeting, { message: "World" }, null),
 document.getElementById("root")
);
```

[↑ فهرست مطالب](#)

## ۲۸۷. الگوریتم‌های diffing ری‌اکت چی هستن؟

React needs to use algorithms to find out how to efficiently update the UI to match the most recent tree. The diffing algorithms is generating the minimum number of operations to transform one tree into another. However, the algorithms have a complexity in the order of  $O(n^3)$  where  $n$  is the number of elements in the tree.

In this case, for displaying 1000 elements would require in the order of one billion comparisons. This is far too expensive. Instead, React implements a heuristic  $O(n)$  algorithm based on two assumptions:

۱. Two elements of different types will produce different trees.
۲. The developer can hint at which child elements may be stable across different renders with a key prop.

[↑ فهرست مطالب](#)

## ۲۸۸. قوانینی که توسط الگوریتم‌های diffing پوشش داده می‌شوند کدام هستن؟

When diffing two trees, React first compares the two root elements. The behavior is different depending on the types of the root elements. It covers the below rules during reconciliation algorithm.

## Elements Of Different Types . 1

Whenever the root elements have different types, React will tear down the old tree and build the new tree from scratch. For example, elements to



or from ,

.to of different types lead a full rebuild

## DOM Elements Of The Same Type . 2

When comparing two React DOM elements of the same type, React looks at the attributes of both, keeps the same underlying DOM node, and only updates the changed attributes. Lets take an example with same ,DOM elements except className attribute

```
<div className="show" title="ReactJS" />
```

```
<div className="hide" title="ReactJS" />
```

Component Elements Of The Same Type:\*\* When a component updates, the\*\* .3 instance stays the same, so that state is maintained across renders. React updates the props of the underlying component instance to match the new element, and calls `componentWillReceiveProps()` and `componentWillUpdate()` on the underlying instance. After that, the `render()` method is called and the diff algorithm recurses on the previous result and the new result. 4. \*\*Recurring On Children:\*\* when recursing on the children of a DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there's a difference. For example, when adding an element at the end of the children, converting between these two .trees works well

```

 first
 second

 first
 second
 third

```

Handling keys:\*\* React supports a key attribute. When children have keys, React\*\* .5 uses the key to match children in the original tree with children in the subsequent ,tree. For example, adding a key can make the tree conversion efficient

```


 <li key="2015">Duke
 <li key="2016">Villanova

 <li key="2014">Connecticut
 <li key="2015">Duke
 <li key="2016">Villanova


```

[↑ فهرست مطالب](#)

## ۲۸۹. چه موقعی نیاز هست که از refها استفاده کنیم؟

- There are few use cases to go for refs
۱. Managing focus, text selection, or media playback .
  ۲. Triggering imperative animations .
  ۳. Integrating with third-party DOM libraries .

[↑ فهرست مطالب](#)

## ۲۹۰. برای استفاده از render prop ها لازم که اسم prop رو render بزاریم؟

Even though the pattern named render props, you don't have to use a prop named render to use this pattern. i.e, Any prop that is a function that a component uses to know what to render is technically a "render prop". Lets take an example with the children prop for render props

```

<Mouse
 children={({mouse}) => (
 <p>
 The mouse position is {mouse.x}, {mouse.y}
 </p>
)}
/>

```

Actually children prop doesn't need to be named in the list of "attributes" in JSX element. Instead, you can keep it directly inside element

```
<Mouse>
 {(mouse) => (
 <p>
 The mouse position is {mouse.x}, {mouse.y}
 </p>
)}
</Mouse>
```

While using this above technique (without any name), explicitly state that children should be a function in your propTypes

```
Mouse.propTypes = {
 children: PropTypes.func.isRequired,
};
```

[↑ فهرست مطالب](#)

## ۲۹۱. مشکل استفاده از render props با pure component ها چیه؟

If you create a function inside a render method, it negates the purpose of pure component. Because the shallow prop comparison will always return false for new props, and each render in this case will generate a new value for the render prop. You can solve this issue by defining the render function as instance method

[↑ فهرست مطالب](#)

## ۲۹۲. چطوری با استفاده از render props می‌تونیم HOC ایجاد کنیم؟

You can implement most higher-order components (HOC) using a regular component with a render prop. For example, if you would prefer to have a withMouse HOC instead of a component, you could easily create one using a regular with a render prop

```
function withMouse(Component) {
 return class extends React.Component {
 render() {
 return (
 <Mouse
 render={({mouse}) => <Component {...this.props} mouse={mouse} />
 />
);
 }
 };
}
```

.This way render props gives the flexibility of using either pattern

[↑ فهرست مطالب](#)

## ۲۹۳. تکنیک windowing چیست؟

Windowing is a technique that only renders a small subset of your rows at any given time, and can dramatically reduce the time it takes to re-render the components as well as the number of DOM nodes created. If your application renders long lists of data then this technique is recommended.

Both react-window and react-virtualized are popular windowing libraries which provides several reusable components for displaying lists, grids, and .tabular data

[↑ فهرست مطالب](#)

## ۲۹۴. توی JSX یه مقدار falsy رو چطوری چاپ کنیم؟

The falsy values such as false, null, undefined, and true are valid children but they don't render anything. If you still want to display them then you need to ,convert it to string. Let's take an example on how to convert to a string

```
<div>My JavaScript variable is {String(myVariable)}.</div>
```

[↑ فهرست مطالب](#)

## ۲۹۵. به مورد استفاده معمول از portals مثال میزنی؟

React portals are very useful when a parent component has overflow: hidden or has properties that affect the stacking context (z-index, position, opacity etc styles) and you need to visually “break out” of its container. For example, dialogs, global message notifications, hovercards, and tooltips.

[↑ فهرست مطالب](#)

## ۲۹۶. توی کامپوننت‌های کنترل نشده چطوری مقداری پیش فرض اضافه کنیم؟

In React, the value attribute on form elements will override the value in the DOM. With an uncontrolled component, you might want React to specify the initial value, but leave subsequent updates uncontrolled. To handle this case, you can specify a **defaultValue** attribute instead of **value**.

```
render() {
 return (
 <form onSubmit={this.handleSubmit}>
 <label>
 User Name:
 <input
 defaultValue="John"
 type="text"
 ref={this.input} />
 </label>
 <input type="submit" value="Submit" />
 </form>
);
}
```

The same applies for `select` and `textArea` inputs. But you need to use **defaultChecked** for `checkbox` and `radio` inputs.

[↑ فهرست مطالب](#)

## ۲۹۷. stack موردعلاقه شما برای کانفیگ پروژه ری اکت چیه؟

Even though the tech stack varies from developer to developer, the most popular stack is used in react boilerplate project code. It mainly uses Redux and redux-saga for state management and asynchronous side-effects, react-router for routing purpose, styled-components for styling react components, axios for invoking REST api, and other supported stack such as webpack, .reselect, ESNext, Babel

You can clone the project <https://github.com/react-boilerplate/react-boilerplate> and start working on any new react project

↑ فهرست مطالب

## ۲۹۸. تفاوت DOM واقعی و Virtual DOM چیه؟

,Below are the main differences between Real DOM and Virtual DOM

Virtual DOM	Real DOM
Updates are fast	Updates are slow
DOM manipulation is very easy	.DOM manipulation is very expensive
You Can't directly update HTML	.You can update HTML directly
There is no memory wastage	It causes too much of memory wastage
It updates the JSX if element update	Creates a new DOM if element updates

↑ فهرست مطالب

## ۲۹۹. چطوری Bootstrap رو به یه برنامه ری اکتی اضافه کنیم؟

Bootstrap can be added to your React app in a three possible ways

۱. Using the Bootstrap CDN

This is the easiest way to add bootstrap. Add both bootstrap CSS and JS resources in a head tag



## ۲. Bootstrap as Dependency

If you are using a build tool or a module bundler such as Webpack, then this is the preferred option for adding Bootstrap to your React application

```
npm install bootstrap
```

## ۳. React Bootstrap Package

In this case, you can add Bootstrap to our React app by using a package that has rebuilt Bootstrap components to work particularly as React components. Below packages are popular in this category

۱. react-bootstrap

۲. reactstrap

[↑ فهرست مطالب](#)

## ۳۰۰. می‌تونن به لیست‌های معروف‌ترین وبسایت‌هایی که از ریاکت استفاده می‌کنن رو بگی؟

این زیر به لیست از 10 وبسایت مشهور که از ریاکت برای فرانت‌اندشون استفاده می‌کنن رو لیست می‌کنیم:

۱. Facebook

۲. Uber

۳. Instagram

۴. WhatsApp

۵. Khan Academy

۶. Airbnb

۷. Dropbox

۸. Flipboard

۹. Netflix

۱۰. PayPal

[↑ فهرست مطالب](#)

## ۳۰۱. استفاده از تکنیک CSS In JS تو ریاکت توصیه میشه؟

ری‌اکت هیچ ایده‌ای راجع به اینکه استایل‌ها چطوری تعریف شدن نداره اما اگه تازه کار باشین می‌تونین از یه فایل جداگانه `*.css` که مثلاً توی پروژه‌های ساده استفاده می‌شد کمک بگیرین و با استفاده از `className` از استایل‌ها استفاده کنین. `CSS In Js` یه بخش از خود ری‌اکت نیست و توسط کتابخونه‌های `third-party` بهش اضافه شده اما اگه می‌خوایین. ازش (`CSS-In-JS`) استفاده کنین کتابخونه `styled-components` می‌تونه گزینه خوبی باشه.

↑ فهرست مطالب

## ۳۰۲. لازمه همه کلاس کامپوننت‌ها رو تبدیل کنیم به هوک؟

نه. ولی می‌تونین از هوک‌ها توی بعضی از کامپوننت‌های قدیمی یا جدید استفاده کنین و سعی کنین باهاش راحت باشین البته برنامه‌ای برای حذف `classes` از ری‌اکت هنوز وجود نداره.

↑ فهرست مطالب

## ۳۰۳. چطوری میشه با هوک‌های ری‌اکت دیتا `fetch` کرد؟

هوک این افکت اسمش `useEffect` هستش و میشه خیلی ساده ازش برای فراخوانی API با استفاده از `axios` استفاده کرد. نتیجه درخواست رو هم خیلی ساده میشه ریخت تو یه `state` داخلی از `component` که وظیفه این ثبت شدن داده رو هم تابع `setter` از `useState` به عهده می‌گیره. خب بزارین یه مثال بزنین که لیست مقالات رو از یه API می‌گیره:

```
import React, { useState, useEffect } from "react";
import axios from "axios";

function App() {
 const [data, setData] = useState({ hits: [] });

 useEffect(async () => {
 const result = await axios(
 "http://hn.algolia.com/api/v1/search?query=react"
);

 setData(result.data);
 }, []);

 return (

 {data.hits.map((item) => (
 <li key={item.objectID}>
 {item.title}

))}

);
}

export default App;
```

دقت کنید که یه آرایه خالی به عنوان پارامتر دوم به هوک effect دادیم که فقط موقع mount شدن درخواست رو بفرسته و لازم نباشه با هر بار رندر درخواست زده بشه، اگ لازم بود با تغییرات یه مقدار (مثلا شناسه مقاله) درخواست API رو مجدداً بزنیم، می‌تونستیم عنوان متغیر رو توی اون آرایه قرار بدیمش و با هر تغییر اون متغیر افکت مجدداً اجرا بشه.

[↑ فهرست مطالب](#)

## ۳۰۴. هوک‌ها همه موارد کاربرد کلاس‌ها رو پوشش میدن؟

هوک‌ها همیشه گفت همه موارد کارکردی کلاس‌ها رو پوشش نمیدن ولی با اضافه شدن هوک‌های جدید برنامه‌های خوبی برای آینده هوک‌ها پیش‌بینی میشه. در حال حاضر هیچ هوکی وجود نداره که کارکرد متدهای **getSnapshotBeforeUpdate** و **componentDidCatch** رو محقق کنه.

## ۳۰۵. نسخه پایدار ری اکت که از هوک پشتیبانی می‌کند کدومه؟

ری اکت حالت پایداری از هوک‌ها رو توی نسخه 16.8 برای پکیج‌های زیر منتشر کرد:

۱. React DOM

۲. React DOM Server

۳. React Test Renderer

۴. React Shallow Renderer

## ۳۰۶. چرا از حالت destructuring آرایه برای useState استفاده می‌کنیم؟

وقتی که با استفاده از هوک `useState` به `state` رو معرفی می‌کنیم، به آرایه دوتایی برمی‌گردونه که اندیس اولش متغیر مورد نظر برای دسترسی به `state` هست و اندیس دوم `setter` یا تغییر دهنده اون `state`. به روش اینه که با استفاده از اندیس‌های آرایه و `[0]` و `[1]` بهشون دسترسی پیدا کنیم ولی به کم ممکنه گیج کننده باشه. ولی با استفاده از حالت `destructuring` خیلی ساده‌تر میشه این کار رو انجام داد. برای مثال دسترسی به `state` با اندیس‌های آرایه این شکلی میشد:

```
var userStateVariable = useState("userProfile"); // Returns an array pair
var user = userStateVariable[0]; // Access first item
var setUser = userStateVariable[1]; // Access second item
```

ولی همون کد با استفاده از `destructuring` آرایه‌ها به شکل پایین درمیداد:

```
const [user, setUser] = useState("userProfile");
```

## ۳۰۷. منابعی که باعث معرفی ایده هوک‌ها شدن چیا بودن؟

ایده معرفی هوک از منابع مختلفی به وجود اومد. این پایین به لیستی ازشون رو میاریم:

۱. تجربه قبلی که با functional API توی پکیج react-future داشتن
۲. تجربه انجمن ری اکت با پراپ render مثل کامپوننت‌های Reaction
۳. متغیرهای state و سلول‌های state توی DisplayScript.
۴. Subscription‌های موجود توی Rxjs.
۵. کامپوننت‌های reducer توی ReasonReact.

↑ فهرست مطالب

## ۳۰۸. چطوری به API‌های ضروری اجزای وب دسترسی پیدا کنیم؟

کامپوننت‌های web اکثراً به عنوان API‌های imperative برای اجرای یه وظیفه خاص قلمداد می‌شن. برای استفاده ازشون باید با استفاده از **ref** که امکان کار با DOM را فراهم می‌کنه بیاییم یه کامپوننت که به شکل imperative کار می‌کنه ایجاد کنیم. ولی اگه از وب کامپوننت‌های کاستوم یا همون third-party استفاده می‌کنیم، بهترین کار نوشتن یه کامپوننت **wrapper** برای استفاده از اون وب کامپوننت هست.

↑ فهرست مطالب

## ۳۰۹. formik چیه؟

- Formik یه کتابخونه ری اکت هست که امکان حل سه مشکل اساسی رو فراهم می‌کنه:
۱. دریافت و مدیریت مقادیر از state
  ۲. اعتبارسنجی و مدیریت خطاها
  ۳. مدیریت ثبت فرم‌ها

↑ فهرست مطالب

## ۳۱۰. middleware‌های مرسوم برای مدیریت ارتباط‌های asynchronous توی Redux چیا هستن؟

یه سری از میان‌افزارهای (middleware) معروف برای مدیریت فراخوانی action‌هایی که به شکل asynchronous توی Redux فراخوانی میشن اینا هستن: **Redux Thunk**، **Redux** و **Promise** و **Redux Saga**.

## ۳۱۱. مرورگرها کد JSX رو متوجه میشن؟

نه، مرورگرها نمی‌تونن کد JSX رو متوجه بشن. مجبوریم که از یه transpiler برای تبدیل کد JSX به کد جاوااسکریپت عادی که مرورگرها متوجه میشن تبدیل کنیم. مشهورترین transpiler در حال حاضر Babel هست که برای اینکار استفاده میشه.

## ۳۱۲. Data flow یا جریان داده ری‌اکت رو توضیح میدی؟

ری‌اکت از روش جریان داده یک طرفه استفاده می‌کنه. استفاده از prop باعث میشه از تکرار موارد بدیهی جلوگیری بشه و درک کردنش ساده‌تر از روش سنتی data-binding دو طرفه باشه.

## ۳۱۳. react scripts چیه؟

پکیج `react-scripts` یه مجموعه از اسکریپت‌هاست که توی `create-react-app` برای ایجاد سریع و ساده پروژه ری‌اکتی ازشون استفاده میشه. دستور `react-scripts start` محیط توسعه کد رو ایجاد می‌کنه و یه سرور براتون استارت می‌کنه که از لود در لحظه و داغ مازول‌ها پشتیبانی می‌کنه.

## ۳۱۴. ویژگی‌های create react app چیه؟

این پایین به یه سری از ویژگی‌های `create-react-app` رو لیست می‌کنیم.

۱. ساپورت کامل از Flow و React، JSX، ES6، Typescript

۲. Autoprefixed CSS

۳. CSS Reset/Normalize

۴. سرور live development

- ۵. به اجرا کننده unit-test که ساپورت built-in برای گزارش coverage داره
- ۶. به اسکریپت build برای bundle کردن فایل‌های CSS، JS و تصاویر که برای استفاده production با قابلیت hash و sourcemap عمل می‌کنه
- ۷. به سرویس ورکر برای استفاده به صورت offline-first که قابلیت استفاده به صورت web-app و pwa رو فراهم می‌کنه

↑ فهرست مطالب

## ۳۱۵. هدف از متد renderToNodeStream چیه؟

متد `ReactDOMServer#renderToNodeStream` برای تولید HTML روی سرور و ارسال اون به درخواست initial کاربر استفاده می‌شه که باعث میشه صفحات سریع‌تر لود بشن. البته علاوه بر سرعت، به موتورهای جستجو این امکان رو میده که وبسایت شما رو به سادگی crawl کنن و SEO سایت بهتر بشه.

**Note:** البته یادتون باشه که این متد توی مرورگر قابل اجرا نیست و فقط روی سرور کار می‌کنه.

↑ فهرست مطالب

## ۳۱۶. MobX چیه؟

MobX یه راه‌حل ساده، scalable برای مدیریت state هست که خیلی قوی تست شده. این روش برای برنامه‌نویسی تابعی کنش‌گرا (TFRP) استفاده می‌شه. برای برنامه‌های ری‌اکتی لازمه که پکیج‌های زیر رو نصب کنین:

```
npm install mobx --save
npm install mobx-react --save
```

↑ فهرست مطالب

## ۳۱۷. تفاوت‌های بین Redux و MobX چیا هستن؟

این پایین به سری از اصلی‌ترین تفاوت‌های Redux و MobX رو می‌گیم:

موضوع	Redux	MobX
تعریف	یه کتابخونه جاواسکریپتی هستش که امکان مدیریت state رو فراهم می‌کنه	یه کتابخونه جاواسکریپتی هستش که امکان مدیریت state به صورت کنش‌گرا رو فراهم می‌کنه
برنامه‌نویسی	به صورت پایه‌ای با ES6 نوشته شده	به صورت پایه‌ای با ES5 نوشته بشه
Store دیتا	فقط یه store برای مدیریت همه داده‌ها وجود داره	بیش از یه store برای ذخیره و مدیریت داده وجود داره
کاربرد	به شکل اساسی برای برنامه‌های پیچیده و بزرگ استفاده میشه	برای برنامه‌های ساده بیشتر کاربرد داره
پرفورمنس	نیاز به یه سری بهبودها داره	پرفورمنس بهتری ارائه میده
چگونگی ذخیره داده	از آبجکت جاواسکریپت به عنوان store استفاده می‌کنه	از observable برای نگهداری داده استفاده می‌کنه

## ↑ فهرست مطالب

## ۳۱۸. لازمه قبل از شروع ری‌اکت ES6 رو یاد گرفت؟

نه، اجبار برای یادگرفتن es2015/es6 برای کار با ری‌اکت وجود نداره. ولی توصیه شدیدی میشه که یاد بگیریدش چون منابع خیلی زیادی هستن که به شکل پیش‌فرض با es6 کار شدن. بزارین یه نگاه کلی به مواردی که الزاما با es6 کارشون رو ذکر کنیم:

۱. Destructuring: برای گرفتن مقادیر prop و استفاده از اونا توی کامپوننت

```
// in es 5
var someData = this.props.someData;
var dispatch = this.props.dispatch;

// in es6
const { someData, dispatch } = this.props;
```

۲. عملگر spread: به پاس دادن prop‌ها به پایین برای کامپوننت‌های فرزند کمک می‌کنه



```
// in es 5
<SomeComponent someData={this.props.someData} dispatch={this.props.dispatch} />

// in es6
<SomeComponent {...this.props} />
```

۳. توابع arrow: کدها رو کم حجم تر می کنه

```
// es 5
var users = userList.map(function (user) {
 return {user.name};
});

// es 6
const users = userList.map((user) => {user.name});
```

[↑ فهرست مطالب](#)

## ۳۱۹. Concurrent Rendering چیست؟

Concurrent rendering باعث میشه برنامه ری اکتی بتونه توی رندر کردن درخت کامپوننت ها به شکل مسئولانه تری عمل کنه و انجام این رندر رو بدون بلاک کردن thread اصلی مرورگر انجام بده. این امر به ری اکت این اجازه رو میده که بتونه اجرا شدن یه رندر طولانی رو به بخش های مرتب شده بر اساس اولویت تقسیم کنه و توی پیک های مختلف رندر رو انجام بده. برای مثال وقتی حالت concurrent فعال باشه، ری اکت یه نیم نگاهی هم به بقیه تسک هایی که هنوز انجام نشدن داره و اگه تسک با اولویت دیگه ای رو ببینه، حالت فعلی که داشت رندر می کرد رو متوقف می کنه و به انجام کار با اولویت تر می رسه. این حالت رو به دو روش میشه فعال کرد:

۱. برای یه بخش از برنامه با wrap کردن کامپوننت توی تگ concurrent :

```
<React.unstable_ConcurrentMode>
 <Something />
</React.unstable_ConcurrentMode>
```

۲. برای کل برنامه با استفاده از createRoot موقع رندر

```
ReactDOM.unstable_createRoot(domNode).render(<App />);
```

[↑ فهرست مطالب](#)

## ۳۲۰. تفاوت بین حالت `async` و `concurrent` چیه؟

هر دوتاشون به یه چیز اشاره می‌کنن. قبلا حالت `concurrent` با عنوان "Async Mode" توسط تیم ری‌اکت معرفی می‌شد. عنوان این قابلیت به این دلیل تغییر پیدا کرد که قابلیت ری‌اکت برای کار روی مرحله‌های با اولویت متفاوت رو نشون بده. همین موضوع جلوی اشتباهات در مورد طرز تفکر راجع به رندر کردن `async` رو می‌گیره.

↑ فهرست مطالب

## ۳۲۱. می‌تونیم از آدرس‌های دارای `url` جاواسکریپت در ری‌اکت 16.9 استفاده کرد؟

آره، میشه از `javascript`: استفاده کرد ولی یه `warning` توی کنسول برامون نشون داده میشه. چون آدرس‌هایی که با `javascript`: شروع میشن خطرناکن و می‌تونن باعث ایجاد باگ امنیتی توی برنامه بشن.

```
const companyProfile = {
 website: "javascript: alert('Your website is hacked')",
};
// It will log a warning
More details;
```

البته بخاطر داشته باشین که نسخه‌های بعدی ری‌اکت قراره بجای `warning` یه ارور برای این مورد `throw` کنن.

↑ فهرست مطالب

## ۳۲۲. هدف از پلاگین `eslint` برای هوک‌ها چیه؟

پلاگین `ESLint` میاد یه سری قوانین برای درست نوشت هوک‌ها رو توی برنامه الزامی می‌کنه. روش تشخیص دادن هوک‌ها هم اینطوره که می‌گه اگه اسم تابعی با "use" شروع بشه و درست بعد اون یه حرف بزرگ بیاد پس اون تابع هوک هستش. این پلاگین در حالت پایه این دوتا شرط رو الزام می‌کنه:

۱. فراخوانی هوک‌ها یا باید داخل یه تابع که عنوانش `PascalCase` هست (منظور یه کامپوننته) یا یه تابع دیگه که مثلا `useSomething` هست (custom هوک) انجام بشه.

۲. هوک‌ها باید توی همه رندها با یه ترتیب مشخص اجرا بشن.

[↑ فهرست مطالب](#)

## ۳۲۳. تفاوت‌های Declarative و Imperative توی ری‌اکت چیه؟

یه کامپوننت ساده UI رو تصور کنین، مثلاً یه دکمه "لایک". وقتی که روش کلیک می‌کنین رنگ از خاکستری به آبی تغییر پیدا می‌کنه و اگه دوباره کلیک کنید باز خاکستری میشه. رویش imperative برای انجام این کار اینطوریه:

```
if (user.likes()) {
 if (hasBlue()) {
 removeBlue();
 addGrey();
 } else {
 removeGrey();
 addBlue();
 }
}
```

لازمه اول بررسی کنیم که چه چیزی رو توی اسکرین داریم نمایش می‌دیم و بعدش ببایم state رو عوض کنیم به حالتی که می‌خواهیم برامون نمایش انجام بشه، توی برنامه‌های بزرگ و واقعی مدیریت این حالت‌ها خیلی می‌تونه سخت باشه. در حالت مقابل، روش declarative می‌تونه اینطوری باشه:

```
if (this.state.liked) {
 return <blueLike />;
} else {
 return <greyLike />;
}
```

چون روش declarative حالت‌ها رو جدا در نظر می‌گیره، این بخش از کد براساس state فقط تصمیم می‌گیره که چه ظاهری رو نمایش بده و به همین دلیل درک کردنش ساده‌تره.

[↑ فهرست مطالب](#)

## ۳۲۴. مزایای استفاده از تایپ اسکریپت با ری‌اکت چیه؟

یه سری از مزایای استفاده از typescript با Reactjs اینا هستن:

۱. می‌تونیم از آخرین ویژگی‌های جاوااسکریپت استفاده کنیم
۲. از interfaceها برای تعریف نوع‌های دلخواه و پیچیده استفاده کنیم
۳. IDE‌هایی مثل VS Code برای TypeScript ساخته شدن
۴. با افزایش خوانایی و Validation از خطاهای ناخواسته جلوگیری کنیم

**↑ فهرست مطالب**