**ReactJS**

**Components: Basic Idea**

1. Какво са компонентите?

- **Функции**, които създават/генерират HTML

- Чрез компонентите можем да си разделим едно приложение, така че да има **динамично регенериране** на HTML

- Можем да композираме компонентите заедно и така да създадем цялостно приложение

\*ПРОЦЕС: Имаме много функции, които генерират много HTML и са свързани помежду си → HTML-ът стига до DOM дървото (браузърът взима HTML-а и го трансформира в DOM дърво)

\*Когато искаме да направим промяна в структурата на HTML-а, реално ние генерираме целия HTML отгоре до долу и го подаваме на браузъра, който трябва да генерира DOM дървото отначало. Това е неефективно и затова React предоставя така наречения VDOM/

2. Какво е VDOM (Virtual DOM)?

- **Репрезентация** на DOM дървото (второ DOM дърво), което е много по-леко, по-простичко, с много по-малко данни и с много по-малки обекти

- Изключително бързо, защото е по-малко

- Състои се от React елементи, които отговарят на HTML елементите от DOM дървото

- Когато имаме някаква промяна се ъпдейтва VDOM-a и след това прави нещо като **difference checker** (следене за промени) с DOM-a, вижда къде има ъпдейт и тогава прави специфична промяна в DOM дървото - за точно определения ъпдейт. По този начин **НЕ** се пререндерира отначало цялото DOM дърво

3. Какво са props и states?

- Начини да запазваме и **репрезентираме** данни, които можем да подаваме на компонента - единият начин е чрез **props**, а другият чрез **state**

- Оказват влияние на начина, по който компонентът ще генерира крайния резултат, който искаме да видим на екрана

- Чрез тях можем да направим компонентите **динамични**

- Чисти **JS обекти**

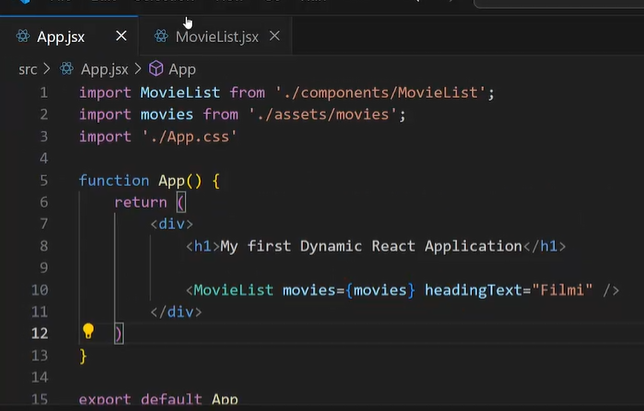
4. Какво са props?

- Най-лесният начин да подаваме **данни** на компонент

- Подават се от parent към children

- **НЕ** можем да променим стойностите по референция, тоест можем да вземем обекта, но не можем да го променим вътрешно

- Props като аргумент е винаги **обект**

Начин, по който можем да подадем данни чрез props на някакъв компонент:  


- Тук имаме **масив от обекти**,

съдържащи информация за

филми. Подаваме ги чрез

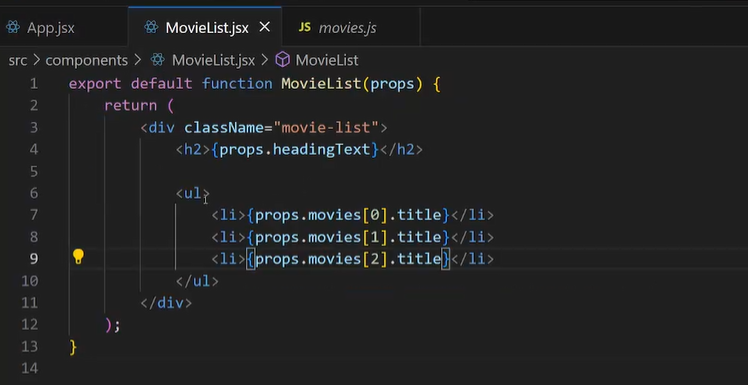
**props на компонента** и когато

отидем в него можем да

достъпим всеки един обект от

масива.

**Използваме къдрави скоби!**



5. Какво е state?

- Съхраняване на **състояние**, което искаме да запазим

- Компонентите могат да си **съхранят** информация в state-а, която да се помни. Например: Когато следващият път се пререндерира даден компонент (например разгъващо се меню), той си казва “аз съм все още expand-нат и искам да се пререндерирам така че все още да бъда expand-нат”

- Има начално състояние в момента, в който компонентът се зарежда в дървото. Този state може да се променя впоследствие (в процеса на използване)

- Това е обект, който е serializable - може да бъде трансформиран с JSON.stringify() и обратно

6. Какво са hooks?

- Механизъм в Реакт, който ни позволява да **добавяме** функционалности и нови способности на компонентите

- Допълнителни **tool-ове**, които можем да закачаме към нашия компонент

7. Какво е useState Hook-a?

- Позволява ни да си запазваме данни, които да стоят между отделните извиквания на функцията

\*Prerender означава функцията да се изпълни отново

**\*ВАЖНО!:** Всеки компонент се prerender-ира, когато някой от state-овете му се промени

8. Какво е conditional rendering?

- Възможност да показваме **различно съдържание** на база state-а и props-а, които получаваме

- Има 4-ри начина:

1. Чрез използване на **return**

2. Чрез **if statement-и** (пишат се над return-а, в който връщаме JSX-а)

3. Чрез **тернарни** оператори

4. Чрез **булеви** оператори

**\*ВАЖНО: Кога се пререндерира компонент?**

- Initial render

- Parent rerender

- Props changed

- State hook change / Custom hook with state

**Components: Deep Dive**

1. Какво е key?

2. Какво е component lifecycle?

- Всеки компонент си има ‘живот’

- 3-те фази:

1. **Mounting** - момент, в който се ражда

2. **Updating** - момент, в който се променя и живее

3. **Unmounting** - момент, в който ‘умира’ и се маха от DOM-а (спира да съществува)

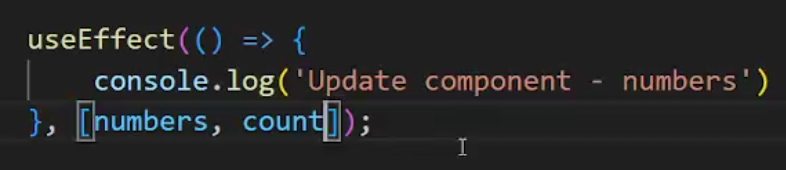
\*При тези фази имаме пререндериране!!

3. Какво е Effect Hook?

- Когато компонентът трябва да свърши нещо полезно по време на съществуването си (страничен ефект, но в добрия смисъл)

- Пример за **effect**: фетчване данни от сървър

В този случай **useEffect** ще се изпълни само когато има промяна при numbers или count. Вторият параметър, който се подава на useEffect е лист и ако искаме да се случва нещо при промяна на някоя променлива, то трябва да я добавим в този лист. Променливите в листа са **dependency-та** (затова и листът се казва **DependencyList**).



4. Какво са CSS modules (модули)?

- CSS файлове, които ни позволяват да групираме класове (и т.н.). Те са **scope-нати** - това означава, че ако стилизираме даден компонент, то тази стилизация ще се отрази само на дадения компонент, без да се прилага за останалите в приложението

- За да създадем модул трябва да имаме: **[name].module.css**

5. Как можем да добавим стилизации (CSS)?

- Има множество начини, няма правилен или грешен

- Импортваме styles от CSS модула и прилагаме името на клас, който сме използвали в този модул на даден елемент. Всички стилизации се запазват в обекта **styles** (може да се казва и по друг начин)

**Forms**

1. Кои са начините за управляване на формите?

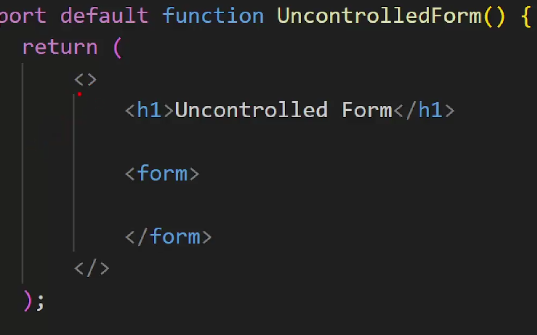
- Те са **2** - **контролирани** компоненти и **неконтролирани** компоненти

- По-лесният начин са неконтролираните форми, но **по-предпочитаният са контролираните**

2. Неконтролирани компоненти

- Стандартни **HTML** форми без никаква React намеса

- Стойностите (value-тата) се менажират от **DOM** дървото (НЕ от VDOM-а)

\*За да “симулираме” присъствието на основен елемент в компонент можем да използваме празен елемент. Той играе ролята на елемент и работи като основен за компонента без да създава HTML елемент накрая:

3. Контролирани компоненти

- Данните и state-а се контролират от **React**, а не от DOM дървото

- Контролираните компоненти са предпочитания вариант

- Взимаме стойностите чрез **useState** и **set** функциите

4. Какво е useRef?

- **Hook**, който отново ни дава възможност да запазваме state - прилича на useState като цел, но не като начин на използване

- Можем да запазваме стойности **в референция**

- Пример за това кога бихме използвали useRef: имаме бутон в отдалечен компонент, но искаме този бутон да управлява някакъв вътрешен state някъде

**Routing**

1. Какво е client-site routing (рутиране)?

- Механизъм, който се изисква да прилагаме за **single page applications**. Ако имаме multi page application, тогава клиентско рутиране **не** ни трябва

2. Какво е React Router?

- Библиотека за React приложения, която ни позволява да имаме **рутиране**

\*Начин, по който лесно можем да имаме route-ове с параметри:

<Route path="/catalog/:category/:userId" component={Catalog}/>

\*Можем да достъпваме тези параметри от компонента чрез useParams():

const {category, userId} = useParams();

\*Дефиниране на nested routes:  


**Advanced Techniques**

1. Context

- **Типът проблем**: Authentication информацията стои в **App**() компонента, за да можем да я предаваме през всички компоненти надолу и да бъде достъпна до тях. Но работата на App() **не** е да се занимава с такъв тип state. Затова трябва да намерим начин по който да изнесем този state, така че да успеем да го “споделим” с компонентното дърво, но и да **не** стои в самия компонент.

- Да извадим state, който да бъде възможен да се **споделя** по компонентите надолу

2. Higher-Order Components (HOC)

- Начин да композираме/**изнесем** логика от компонента си и да направим абстракция на някаква логика, която може да се използва в **няколко различни компонента**

- **Кога бихме искали да използваме HOC-ове?** - В ситуация, където имаме стар application, в който е имало клас компоненти и част от компонентите са мигрирани на function компоненти. Ако искаме да споделим hook с много компоненти, част от които са клас компоненти, то ние ще използваме HOC-ве.

\*Hook-овете **не** могат да се използват с **клас** компоненти и когато искаме да създадем hook за клас компоненти и function компоненти, то ние сме пред дилема и ще използваме **HOC**-ове.

3. Какво е useReducer()?

- **Алтернатива** на useState()

- Reducer - **функция**, която прави промяна в state-а (по-специфична промяна)

- **Кога се използва?** - когато имаме много **сложен** state за менажиране

const [state, dispatch] = useReducer(reducer, initialState)

**state** - текущо актуалният state

**dispatch** - функцията, която прави промяната / променя state-a