**ReactJS - terminology and questions for exam**

**Introduction to React and JSX**

1. Предимствата на React

- **Component-base** - всеки компонент може да управлява собствен state

- **Open-source**

- Декларативен (**Declarative**) - например не създаваме елементите един по един, а пишем **JSX**

- **Native** support - може да се изгражда мобилен UI

- Бързо изпълнение (**Fast** performance)

\*Декларативен - казваш какво искаш като резултат

\*Императивен - казваш как искаш да се случи това, което очакваш като резултат

2. Какво е JSX?

- **Superset** език, който се използва в Реакт

- Уникален начин, по който **HTML** и **JS** се миксират

3. Композиция (composition)

- Компонентите **могат** да се нестват

- Имената на компонентите винаги започват с главна буква - Реакт така ги разпознава като компоненти

- Информация се предава чрез **props**

**React Components - Basic Idea**

1. Предимства на компонентите

- Позволяват ни да разделяме UI-а на **независими** и **преизползваеми** части

- Можем да дефинираме компоненти чрез функции и класове

2. Какво връща функционалният компонент?

- Връща **React Element**

3. Какво са props?

- Информацията, която подаваме на компонент, събрана в **обект**

- Компонентът **НЕ** може да променя собствените си props

- props.children ни дава всички деца, които са между отварящия и затварящия таг

4. Какво е state?

- Определя как компонентът се рендерира и как се държи

- Позволява ни да създаваме **динамични** компоненти

5. Какво е hook?

- Специална функция, която ни позволява да се ‘закачим’ за React функционалност/feature

- **Само** за функционалните компоненти! (за клас компонентите се използват HOC-ове)

6. Какво е useState?  
 - Позволява ни да добавим **state** в нашия функционален компонент

- Initial state аргумента, който подаваме в useState(), се използва само при първото рендериране

7. Event handling

- Event-те се наименуват с **camelCase**

- Не е нужно да използваме addEventListener на DOM елемент след като е създаден - трябва просто да предоставим **listener** (например onClick={clickHandler})

8. Какво е SyntheticEvent?

- **Wrapper** на оригиналния event в браузъра

- Предоставя допълнителна информация към оригиналния browser event

- Пример: event.preventDefault(), или както често пишем **e.preventDefault()**

9. Какво е conditional rendering?

- Работи по същия начин както в JS - тернарни оператори, булеви оператори, if/switch, multiple returns

10. Дебъгване

- Най-лесният начин за дебъгване е чрез browser extension-a **React Developer Tools**:

**More info:** <https://react.dev/learn/react-developer-tools>

**React components - Deep Dive**

1. Какво е Virtual DOM?

- **Виртуална** репрезентация на DOM-а, която се пази в паметта

- В паметта имаме структура от данни (дърво) от **Реакт елементи** (а те са обикновени обекти)

- Състои се от **Реакт елементи**

2. Как да итерираме през лист?

- Използваме **map()** и слагаме уникални **key-ове** на всеки елемент

- Key-овете служат за **идентификация** - помагат на Реакт да идентифицира кои елементи се променят, кои се добавят или изтриват

- Ако елементът няма key ще получим **warning**, но това **няма** да спре работата ни

3. Component Lifecycle

- Има **3 фази**:

- **Mounting** - когато елементите се **създават** и **добавят** в DOM-а

- **Updating** - когато компонентът се **пререндерира** (промяна в props или state)

- **Unmounting** - когато елементът се **премахва** от DOM-a

4. Как е useEffect()?

- **Hook**, който ни позволява да правим **операции** във функционални компоненти - например fetch-ване на данни

- Получава два аргумента - **функция**, която изпълнява “side effect-а” и **dependency array**, който е optional

5. CSS модули

- Файлове, в които всички class names и animation names са scope-нати **локално**

- Дефинициите в .module.css файл обхващат конкретен компонент, а **НЕ** глобално

6. fetch()

- Осигурява ни лесен начин, по който да фечваме ресурси/данни **асинхронно**

- Връща **promise**

**Forms**

1. Форми

- Има **два** стандартни начина да се handle-ват форми:

- **Контролирани** компоненти

- **Неконтролирани** компоненти

2. Неконтролирани компоненти

- Стойностите се управляват от **DOM-а**

3. Контролирани компоненти

- Препоръна техника за имплементиране на форми

- Form data-та се контролира от **React**

- Стойностите се пазят в **state**

- **Custom** handlers за change и submit event-и

4. Какво е useRef()?

- Hook, който ни дава да запазваме state

- 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-side Routing?

- Основен елемент от писането на **SPA** апликации

- Позволява навигация **БЕЗ** презареждане на страницата

- По-добър **UX** (user experience)

2. React Router

- API за React апликации

- Предоставя ни начин, по който да имплементираме **рутиране**

3. Навигация с <Link></Link>

- **Предотвратява** презареждане на страницата

4. Какво е useParams()?

- Начин, по който можем да вземем **параметрите**, подадени в Route-а

5. Какво е useLocation()?  
 - Служи, за да върне **текущата локация** на реакт компонент

6. Разлика между NavLink и Link

- NavLink ни позволява да създаваме линкове, но с **допълнителни** възможности - например има способността да знае дали линкът ни е в ‘активен’ или ‘pending’ state

7. AbortController

- Използва се за **cancel-ване** на продължителни fetch заявки

- Предотвратява потенциални **memory leaks** и **ненужен network usage**

8. Lazy loading

- Помага за **оптимизацията** на **изпълнението** на React апликации

- Помага за зареждането на страницата **бързо** и представя ограничен content на потребителя (само този, който е нужен за интеракцията)

**React Hooks**

1. Какво са hook-овете?

- Идеята е да се ‘expose-нат’ функционалности във функционални компоненти

- Преизползваеми и независими за всеки компонент

- Споделяне на функционалност, а **НЕ** данни между компонентите

- Започват с **use** (useState, useEffect, useContext…)

2. Custom hooks

- Понякога е нужно да използваме stateful логика, която няколко компонента да могат да използват

- Custom hook-ът е JS функция, чието име започва с “**use**”

- Ние решаваме какво да приема като аргументи и какво да return-ва

**\*ВАЖНО:** **НЕ** викаме hook-ове в цикли, condition-и или нестнати функция. Викаме ги в top level-a.

**\*ВАЖНО:** **НЕ** викаме hook-ове от обикновени JS функции, а от React функционални компоненти или от други custom hook-ове.

3. Context

- Позволява ни да подаваме данни (data) по компонентното дърво **без** да подаваме props ръчно постоянно

- Направено е за споделяне на данни (data), които са **приети за глобални**. Например: текущият аутентикиран потребител

- Използва се, когато дадени данни трябва да могат да бъдат достъпвани от много компоненти

- Чрез използването на Context API, **избягваме prop drilling**

4. Providers

- Отговорни са за **дефинирането на контекста** и предаването на данните надолу по компонентното дърво

**Advanced Techniques**

1. Higher-Order Components (HOC)

- Функция, която приема компонент и **връща нов компонент**

- Може да се използва както за function based компоненти, така и за class based компоненти. Но hook-овете могат да се използват **само** за function based компоненти

2. Какво е useReducer()?

- **Алтернатива** на useState()

- Прилича на useState() hook-а, но за **по-сложна** логика

3. Error Boundaries

- React компоненти, които catch-ват **JS грешки**

- **НЕ** catch-ват грешки за:

- Event handlers

- Asynchronous code

- Server-side rendering

- Един компонент става error boundary, ако дефинира **static getDerivedStateFromError** и **componentDidCatch**

- **Само** клас компонент може да бъде error boundary

4. Какво е JEST?

- JavaScript **unit testing** framework

- Използва се от **Facebook** за тестване на services и Реакт апликации