1. React это библиотека для разработки интерфесов, созданная фейсбуком
2. Angularjs предназначен для развязки логики приложения с помощью DOM операций и нацелена на динамическое обновление страницы
3. React-нужен
4. Объект – это
5. ассоциативный массив

**Ассоциативный массив** – это массив, в котором обращение к значению осуществляется по ключу.

1. Функция-это
2. Чистая функция ??

При одних и тех же данных, дает один и тот же результат, те она предсказумема, ее поведение детерминировано (предопределенно) и не зависит от внешних переменных, все что надо получает в параметрах и все что делает можем увидить в ретурне. И ее можно легко протестировать

1. Иммутабельность
2. Идемпотентность
3. Детерминированность
4. Компонента это учатсок кода который представляет часть веб страницы
5. JSX разметка
6. Map
7. Filter
8. Литерал
9. Литерал объекта
10. Колбэк-функция
11. Отличие элемента от компоненты
12. Стор
13. Props
14. Стор
15. NodeJS
16. Замыкание
17. Хуки
18. CRUD
19. **Функиональные компоненты** компоненты созданные как функциии, называются функциональными. Они принимают объект со свойствами как первый аргумент, и также начинаются с большой буквы.

На вопрос "когда их стоит использовать?" ответ очень простой. Всегда, когда компонент не хранит в себе состояние. Другими словами, большинство компонентов в проекте должно быть именно функциональными.

В остальном они ведут себя точно так же, как и компоненты на классах.

1. **«Библиотека**» - набор функций и способов решения функциональности проекта, для которого пишется код. Программисту не нужно всегда писать код с нуля. Он может воспользоваться библиотеками функций, которые были прописаны другими до него.
2. Фреймворк это программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.
3. Главным **отличием библиотеки от фрейма** является инверсия контроля. Если программист контролирует ситуацию при вызове функции или метода, то фреймворк самостоятельно вызывает код, написанный программистом.

«Фреймворк» отличается от понятия библиотеки тем, что библиотека может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений. В то время как «фреймворк» диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию — «каркас», который нужно будет расширять и изменять, согласно указанным требованиям.

Библиотека просто даёт Вам набор функций, которые Вы можете использовать когда и где хотите.  
Фреймворк обычно жёстко контролирует структуру приложения, ограничивает Ваши возможности (в том числе, и выстрелить себе в ногу). Фреймворк это тоже библиотека, но не простая, а задающая развитие проекта при его использовании.

1. **Angular** – это фреймворк для создания крупномасштабных, высокопроизводительных и простых в обслуживании веб-приложений.
2. **single page application, SPA** это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript.
3. **UI** – user interface, пользовательский интерфейс . Взаимодействие человека с сайтом
4. **ReturnType** описываем то, что вернет нам функция.
5. **DOM** – это Document Object Model  HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.

ДОМ – объектная модуль документа. Это совокупность объектов (ввиде html тегов) которые образут структуру html документа.

1. VanillaJS – это использование простого JavaScript без каких-либо дополнительных библиотек, таких как jQuery. Люди используют этот термин как шутку, чтобы напомнить другим разработчикам, что многое можно сделать в наши дни без дополнительных библиотек JavaScript.
2. **Что происходит при запуске React** : React вызывает функциональную компенту, которая отдает JSX, которая транспилируется при помощи комплиятора Babel и создается virtualDom и далее на основе виртуал Дом создается ДОМ браузера. Если необходимо изменить элементы веб-страницы, то изменения вначале вносятся в виртуальный DOM. Потом новое состояние виртуального DOM сравнивается с текущим состоянием. И если эти состояния различаются, то React находит минимальное количество манипуляций, которые необходимы до обновления реального DOM до нового состояния и производит их. В итоге такая схема взаимодействия с элементами веб-страницы работает гораздо быстрее и эффективнее, чем если бы мы работали из JavaScript с DOM напрямую.
3. Компилятор, такой как **Babel**. Он позволяет писать современный код JavaScript, который будет работать даже в старых браузерах.
4. Менеджер пакетов, такой как Yarn или npm. Он позволяет вам использовать обширную экосистему сторонних пакетов и легко устанавливать или обновлять их.
5. Библиотека реакт явл декларативной

Императивный подъод говорит как имено прийти к результату, т.е. конкретные шаги.

Декларативный подход говорит какой результат нам нужен

Что делает React таким уникальным:

- в своей работе реакт испольуте предпроцессор jsx. Это такое повесть html и js. Благодоая этому мы сразу можем создавать логику и прописывать разметку в одном месте

-внутри реакте есть алгоритм, который позволяет отслеживать какие части приложения изменились и отслеживать только их (алгоритм сравнения)

-техгология виртуолДОМ

Перерисовка просходит 1) Пропс поменялся 2) Стайт поменялся

Вызов функции 1 ) Перерисовалась родительская компонента 2) Пропс поменялся 3) Стайт поменялся

1. **Хранилище (store)** — это объект, который:

содержит состояние приложения;

отображает состояние через getState();

может обновлять состояние через dispatch();

позволяет регистрироваться (или удаляться) в качестве слушателя изменения состояния через subscribe().Основная идея: есть объект store (хранилище), у которого есть state, и мы можем это хранилище отправлять через метод store.dispatch отправлять action-ы, и он с помощью reducer-ов будет менять свой стейт

Хранилище в приложении всегда уникально.

1. **дженерики** — это возможность создавать компоненты, работающие не только с одним, а с несколькими типами данных".
2. Результат typeof null == "object" – это официально признанная ошибка в языке, которая сохраняется для совместимости. На самом деле null – это не объект, а отдельный тип данных.
3. **useSelector** – принимает в себя некоторую функцию, которую он в качестве аргумента, когда он ее сам вызовет, отдаст ей весь стейт, а мы забираем то что нужно
4. . Отличие get от post:

Гет запрос он нужен для того, чтобы получить от свервера данные

Пост на сервак что-то постим, отправляем на сервер

1. **Жизненый цикл компоненты делится на 4 части:**

Инициализция, монтаж, обновление, размонтирование

Инициализация – это фаза, на которой компонента собирается начать свой путь

Монтаж – это фаза, на котрой наш компонент реакт монтируется в дом

Обновление – это то место, где состояние компоненты изменяется и просходит повторный рендеринг. НА этом этапе данные компоненты стейт и пропс обновлятся на пользовательские события. Это и приводит к повторной визуализации компоненты

Размонтирование – этот метод означает конец жизненого цикла компоненты.

1. Методы жизненого цилка можно разделить на 3 категории:

Mounting(монтирование компоненты в дом дерево)

Updation(обновление компоненты)

Unmoounting(Размонтирование компоненты из ДОМ (удаление компоненты из Браузерного ДОМа)

1. **VAR** не имеет блочной области видимости. Как мы видели в предыдущих примерах, блок кода создает область видимости для переменных, объявленных с помощью ключевых слов const и let. Однако это не работает для переменных, объявленных с помощью ключевого слова var.
2. **Redux** – это библиотека стейт менеджмента. То что позволяет удобно управлять стейтом. **React-redux** библиотека предназначенная для взаимодействия компоненты со стором.
3. **Интерфейс объекта** – методы и свойства, при помощи которых мы можем взаимодействовать с объектом.
4. **ХОК** нужны для того, чтобы удобно переиспользовать код. Connect функция + 2 функции, которые мы должны передать в нее.
5. **Минусы реакта**: первое что весь рендер происходит на клиенте а не на сервере (если я правильно понял), а второе что у реакта нет SEO оптимизации и для этого есть решения
6. **Flux** — это архитектура, которую команда Facebook использует при работе с React. Это не фреймворк, или библиотека, это новый архитектурный подход, который дополняет React и принцип однонаправленного потока данных.
7. **Диспетчер(Dispatcher)** — это менеджер всего этого процесса. Это центральный узел вашего приложения. Диспетчер получает на вход действия и рассылает эти действия (и связанные с ними данные) зарегистрированным обработчикам.

У любой компоненты есть два метода **componentDidMount()**:

Реакт может их вызывать когда ему надо. Компонента ничего не знает, что происходит с jsx который она выплюнула. Для это в комопненте есть методы жизненого цикла, которые реакт может дергать, для того, чтобы сообщить компоненте что с ней произошло. Когда реакт вставил наш jsx в ДОМ браузера он вызовет метод **componentDidMount()**: и сообщит, что наша компонента уже в ДОМе браузера

**componentWillUnmount()**: сообщит, что наша компонента сейчас уберется из ДОМа

сайд эффекты

В процессе работы компонент проходит через ряд этапов жизненного цикла. На каждом из этапов вызывается определенная функция, в которой мы можем определить какие-либо действия:

1. **constructor(props)**: конструктор, в котором происходит начальная инициализация компонента
2. **static getDerivedStateFromProps(props, state)**: вызывается непосредственно перед рендерингом компонента. Этот метод не имеет доступа к текущему объекту компонента (то есть обратиться к объкту компоненту через this) и должен возвращать объект для обновления объекта state или значение null, если нечего обновлять.
3. **render()**: рендеринг компонента
4. **componentDidMount()**: вызывается после рендеринга компонента. Здесь можно выполнять запросы к удаленным ресурсам
5. **componentWillUnmount()**: вызывается перед удалением компонента из DOM

Кроме этих основных этапов или событий жизненного цикла, также имеется еще ряд функций, которые вызываются при обновлении состояния после начального рендеринга компонента, если в компоненте происходят обновления:

1. **static getDerivedStateFromProps(props, state)**
2. **shouldComponentUpdate(nextProps, nextState)**: вызывается каждый раз при обновлении объекта props или state. В качестве параметра передаются новый объект props и state. Эта функция должна возвращать true (надо делать обновление) или false (игнорировать обновление). По умолчанию возвращается true. Но если функция будет возвращать false, то тем самым мы отключим обновление компонента, а последующие функции не будут срабатывать.
3. **render()**: рендеринг компонента (если shouldComponentUpdate возвращает true)
4. **getSnapshotBeforeUpdate(prevProps, prevState)**: вызывается непосредственно перед компонента. Он позволяет компоненту получить информацию из DOM перед возможным обновлением. Возвращает в качестве значения какой-то отдельный аспект, который передается в качестве третьего параметра в метод componentDidUpdate() и может учитываться в componentDidUpdate при обновлении. Если нечего возвращать, то возвращается значение null
5. **componentDidUpdate(prevProps, prevState, snapshot)**: вызывается сразу после обновления компонента (если shouldComponentUpdate возвращает true). В качестве параметров передаются старые значения объектов props и state. Третий параметр - значение, которое возвращает метод getSnapshotBeforeUpdate

Mapstatetoprops задача взять из всего стейта, какую-то часть. Дать название и передать в нашу компоненту. Наша компонента сможет взять эту часть через пропсы, обратившись props.название которое мы присвоили

Mapsdispatchtoprops взять метод диспатч, и вернуть объект в котором калбэки, те свойства, которые явл функциями. Наша компнента моет вызвать эти свойства, или задиспачить в наш бизнес. И наш бизнес изменит наш стейт

Provider добавляет стор в контекст с помощью контекстаАЙПИАЙ. Поэтому любая дочерняя компонента может стать потребителем и достать то, что положил родитель(provide)

Конструктор – это конструктор компанента, которая вызывается до того, как компонент будет смонтирован.

**componentDidMount()**: вызывается сразу после монтирование компонента. Хорошо подходит для создания сетевых запросов и настройки подписки компонентов, на различные события.

**componentDidUpdate**(): вызывается сразу, после обновления.Метод повзоялет работать с ДОМ приобновлении компонента.Подходит для выполнения сетевых запросов, которые выполняются на основании результатов сравнения текущих пропсов с предыдущими.

**componentWillUnmount:** вызывает перед размонтированием компоненты. В нем выполняются необходимые сбросы (отмена таймера, а также сетевых запросов и подписок)

**shouldComponentUpdate**: вызывает перед рендером, когда получает новые пропсы или состояние. Используется для производительности, чтобы избежать лишних перерисовак.

**Render** предназначен для возврата разметки

**getDerivedStateFromProps:** вызывает перед методом render возвращает объект для обновления состояния или null чтобы ничего не обновлять.

**getSpnapshotBeforeUpdate:** позволяет компоненту брать некоторую информацию из ДОМ перед ее возможным изменением.

**getDerivedStateFromError:** метод жизненого цикла, вызывается после возникновения ошибки у компанента потомка. Он получает ошибку в качестве параметра и возвращает значение для обновления состояния.

**componentDidCatch()** Этот метод жизненного цикла вызывается после возникновения ошибки у компонента-потомка.

Валера 2

18-18 как напрямую обратиться к браузеру

18-20 контролируем неконтролируемый элемент 18-35

18-39 глобальная и локальная область видимости

18-46 замыкание

18-51 методы жиз цикла

20-45 пост предсф

20-59 ХОК конект

Однонопаравленный поток данных