1. Какие есть общие особенности у React.Context и MobX?

Ответ:

- React Context и MobX - это два различных инструмента для управления состоянием в React-приложениях. Они имеют некоторые общие особенности:
- Централизованное хранение состояния: Оба инструмента предоставляют возможность централизованного хранения состояния приложения. React Context позволяет передавать данные от родительских компонентов к дочерним без явной передачи через пропсы, а MobX предоставляет глобальный объект состояния (store), к которому могут обращаться все компоненты.
- Подписка на изменения состояния: И React Context, и MobX предлагают механизмы подписки на изменения состояния. В React Context подписка осуществляется через использование Consumer-компонента или useContext хука, а в MobX - через использование наблюдаемых свойств и реактивных реакций.
- Перерисовка компонентов при изменении состояния: Оба инструмента обеспечивают автоматическую перерисовку компонентов, связанных с изменением состояния. React Context обновляет компоненты, которые зависят от контекста, при изменении значения, а MobX обновляет компоненты, использующие наблюдаемые свойства, когда эти свойства изменяются.
- Гибкость: И React Context, и MobX обладают гибкостью в использовании и могут быть адаптированы к различным сценариям. Они позволяют выбрать уровень гранулярности, на котором будет управляться состояние приложения, и могут быть комбинированы с другими инструментами или библиотеками для управления состоянием, например, с Redux.

2. Как можно структурировать наше приложение для работы с менеджерами состояний?

Ответ:

- При структурировании React-приложения для изучения слов с использованием MobX в работе с менеджерами состояний, вы можете использовать следующий подход:

- Определите структуру состояния: Определите, какая информация будет храниться в состоянии вашего приложения. Например, список слов, текущее слово, введенные ответы и т.д. Разбейте состояние на наблюдаемые свойства (observable properties) MobX.

- Создайте store: Создайте MobX store, который будет содержать состояние и методы для его обновления. Store - это класс, управляющий состоянием приложения. Определите наблюдаемые свойства и методы для изменения состояния.

- Разделите компоненты: Разделите интерфейс вашего приложения на отдельные компоненты, каждый из которых будет отвечать за свою часть интерфейса. Компоненты могут быть функциональными или классовыми. Каждый компонент может получать доступ к состоянию через MobX store.

- Подключите компоненты к состоянию: В каждом компоненте подключите необходимые наблюдаемые свойства из MobX store. Используйте декораторы `@observer` или функцию `observer()` для обертки компонентов и автоматической перерисовки при изменении состояния.
- Обновление состояния: В компонентах вызывайте методы MobX store для обновления состояния, например, добавление нового слова, обновление текущего слова или введенного ответа.

- Работа с действиями: Определите действия (actions) в MobX store, которые будут использоваться для обновления состояния. Действия могут быть асинхронными и содержать

логику обработки взаимодействия с пользователем, загрузки данных и т.д. Вызывайте эти действия из компонентов для обновления состояния.

- Использование селекторов: Для получения данных из состояния в компонентах используйте селекторы (selectors). Селекторы - это функции, которые извлекают и трансформируют данные из MobX store для использования в компонентах. Это позволяет абстрагироваться от структуры состояния и делает компоненты более гибкими.

- В конечном итоге, структурирование приложения для работы с MobX включает создание MobX store для хранения состояния, разделение интерфейса на компоненты, подключение компонентов к состоянию и использование действий для обновления состояния.

3. Когда есть смысл подключать в приложение менеджеры состояний?

Ответ:
Менеджеры состояний имеют смысл подключать в приложение, если вы имеете дело с крупными и сложными приложениями, нуждаетесь в управлении глобальным состоянием, сложной бизнес-логикой, синхронизацией с сервером или обработкой асинхронных операций, а также если вам нужна возможность отмены действий или возврата к предыдущему состоянию.

4. Как вы думаете, почему нельзя использовать @observable вместе с методом shouldComponentUpdate?

Ответ:
использование @observable вместе с методом shouldComponentUpdate может вызвать проблемы и нежелательное поведение. MobX уже обрабатывает изменения в @observable свойствах и автоматически обновляет компоненты, связанные с ними. Использование shouldComponentUpdate в этом случае может быть излишним и привести к непредсказуемым результатам. Рекомендуется либо полностью полагаться на MobX для управления перерисовкой компонентов, либо использовать shouldComponentUpdate без использования @observable свойств.

5. Можно ли использовать несколько разных store в одном приложении?

Ответ:Да, вы можете использовать несколько разных хранилищ (store) в одном приложении, используя MobX. В MobX хранилище представляет собой объект, который содержит состояние приложения и логику для его изменения. Каждое хранилище может иметь свою собственную структуру и функциональность в зависимости от требований вашего приложения.

6. Можно ли использовать MobX без декораторов?

Ответ:Да, можно использовать MobX без декораторов. В MobX декораторы - это специальные аннотации, которые применяются к классам, свойствам или методам, чтобы указать MobX, как отслеживать изменения и реагировать на них. Однако, начиная с версии MobX 6, декораторы являются необязательными и могут быть заменены на альтернативный синтаксис.

7. Можно ли использовать MobX без React?

Ответ:
Да, можно использовать MobX без React. MobX является библиотекой для управления состоянием приложения, которая не зависит от конкретной библиотеки пользовательского интерфейса. В официальной документации MobX сказано, что он может быть использован с любым фреймворком или библиотекой для JavaScript.
В контексте React, MobX обычно используется в связке с ним, чтобы упростить управление состоянием компонентов и обеспечить реактивные обновления. Однако вы можете

использовать MobX независимо от React, например, с другими фреймворками, такими как Vue.js или Angular, или даже в чистом JavaScript.