Я(1)Redux - библиотека для работы с состоянием приложения. Проще говоря, это некоторое хранилище данных. Название читается как Redux и составлено из двух слов: reduce и flux. Reduce — это функция, которая приводит большую структуру данных к одному значению. Flux — архитектура приложения, при которой данные передаются в одну сторону. Инструмент основан на этих двух понятиях, поэтому они вынесены в название. Главные отличия Redux от остальных подходов — это то, что всё состояние приложения хранится в единственном месте.

*СВЕТА(2)Для чего нужен Redux*

• для управления состоянием приложения, работающего с большим количеством данных;

• удобной замены встроенных средств работы с состоянием в React;

• более легкого масштабирования приложения, его преобразования под разные задачи;

• избавления от ошибок, связанных с беспорядком в объекте состояния;

• предсказуемости и понятности работы приложения;

• более простой отладки и доработки;

• повышения производительности и работоспособности программы.

*Я(3)Преимущества Redux*

Легкая работа с состоянием. Redux реализован просто, мало весит и при этом эффективно управляет состоянием. Его основную функциональность можно уместить в десять строчек кода. Пользоваться Redux удобнее, чем самостоятельно писать менеджер состояний.

Упрощение масштабирования. Redux часто используют в связке с React, потому что в React не очень удобный встроенный алгоритм для управления состояниями. Его особенности приводят к тому, что приложение становится тяжело масштабировать. Redux решает эту проблему.

Множество полезных инструментов. Для Redux существуют вспомогательные библиотеки и инструменты, которые сильно облегчают работу с приложениями. Компоненты модифицируют модель, описанную выше, или помогают работать с ней более гибко.

*СВЕТА(4)Какую проблему решает Redux*

Есть некоторое дерево компонентов, в корне которого лежит компонент App. В нем находится еще несколько компонентов, которые также содержат какие-нибудь компоненты. **(тык)** И может возникнуть такая ситуация, когда состояние из этого компонента **(тык)** может понадобиться в этом компоненте. Да, можно вынести это состояние в родительскую компоненту и передавать через параметры, т.е. через пропсы. Но в больших проектах это сильно запутывает код, становится сложно искать какие-либо данные.

Я(5)Данный рисунок наглядно демонстрирует, как с этой проблемой справляется Redux. Состояние выносится во внешнюю зависимость и каждый компонент получает какие-то данные уже из этого состояния. Во-первых, состояние которое используется в одном компоненте можно использовать и в другом. А во-вторых, просто хорошей практикой считается хранить данные и логику, которая с ними взаимодействует, отдельно от компонента.

*СВЕТА(6)А теперь самое интересное. Как это работает.*

Итак, смоделируем ситуацию. **(тык)** В реальной жизни есть банк и он хранит какие-то деньги. В Redux этот банк называется store. Мы знаем, что банк выполняет определенные функции. Например, мы можем положить деньги на счет или же их снять. **(тык)** И такие действия в Redux называются actions. Т.е. action определяет то, как мы изменяем данные. Т.е. мы деньги можем либо взять, либо положить. **(тык)** Но мы не можем просто прийти в банк, открыть ячейку и забрать деньги. Но зато мы можем попросить работника банка снять нам со счета деньги. И в Redux этот работник называют диспетчер, он же dispatch. Прежде, чем как-то изменить состояние, а именно снять или положить деньги, мы должны напрямую обратиться к нему и передать action.

ЯНо тут тоже не так все просто. Это всего лишь диспетчер и он также не может открыть ячейку и достать оттуда деньги. Для этого существует какая-то система, в которую он должен обратиться, чтобы получить для нас деньги. **(тык)** И эта система называется reducer. Сам dispatch никак с деньгами не взаимодействует. Грубо говоря, ему дают бумажку или по-другому action. На этой бумажке написано: "Сними деньги". Он берет этот action, идет к reducer, и передает этот action в reducer. На этом его работа заканчивается. А вот вся логика при работе с деньгами, т.е. с данными в нашем приложении, находится уже внутри reducer. Reducer знает все возможные actions и внутри него определено, что деньги надо либо снять, либо добавить. И вот уже сам reducer напрямую меняет состояние.

СВЕТА(7)И так подведем итог. Есть actions - это набор каких-то действий, по изменению данных. И есть диспетчер, который эти actions принимает. Диспетчер берет actions и передает его в reducer. Reducer понимает язык actions и каким-либо образом изменяет состояние, т.е. данные. И мы уже этими данными пользуемся в своих React-компонентах. И по сути единственное отличие того же банка от Redux в том, что Redux работает именно с данными, а не с реальными деньгами. Это могут быть абсолютно любые данные (о пользователе, списках файлов и др.).

Я(8)Подчеркнем, что состояние приложения нельзя изменять напрямую.

То есть вот так: store.number += 1 — делать нельзя. Чтобы изменить store, нужно сгенерировать событие с описанием изменения. Это делается с помощью store.dispatch(). Объект, который передаётся в dispatch(), action. может содержать любые поля, которые будут описывать изменение, единственное обязательное поле — type. (9)Все изменения обрабатываются функцией Reducer и она должна быть чистой. Чистая функция — это функция, которая всегда для одного и того же входного значения возвращает одинаковый результат + не изменяет ничего снаружи себя.

Она обрабатывает каждый action, который происходит в приложении. Функция принимает на вход есть текущее состояние приложения и action, который только что произошёл, и для каждого типа action изменяет состояние таким образом, как считает нужным. Обратите внимание, что состояние здесь не изменяется напрямую. Вместо этого возвращается новое состояние, которым Redux уже сам заменяет старое.

СВЕТА10 и 11 слады – код!!!