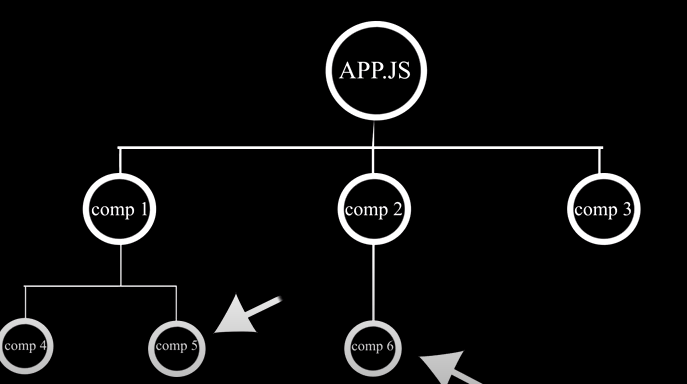
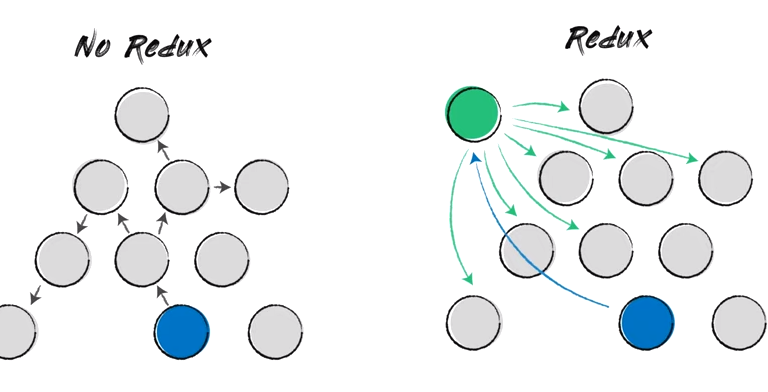
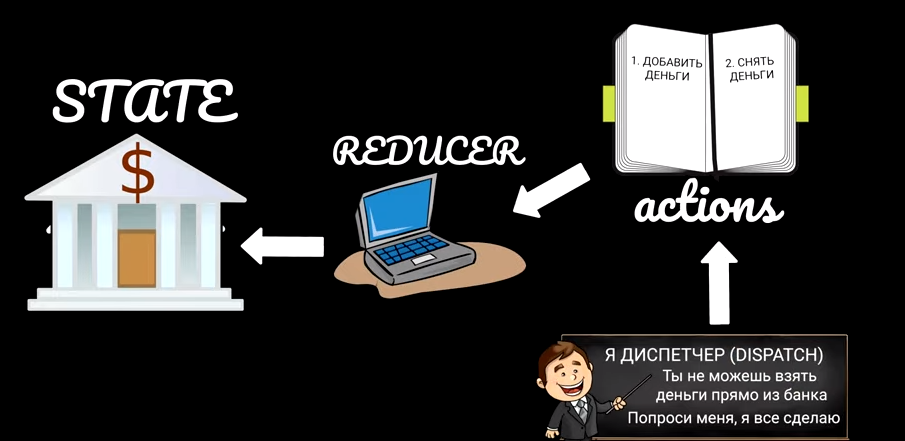
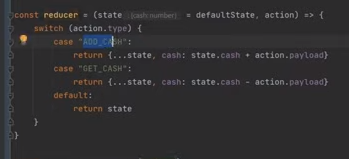
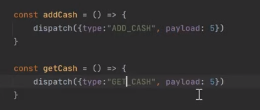
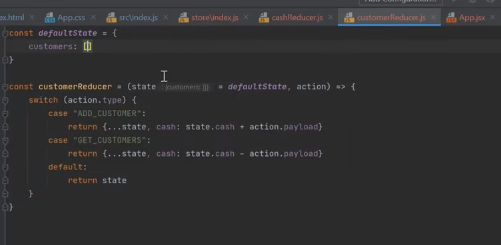
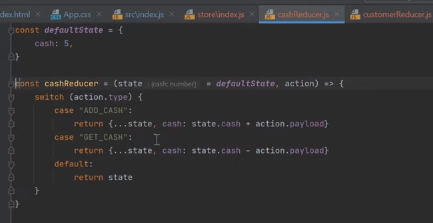
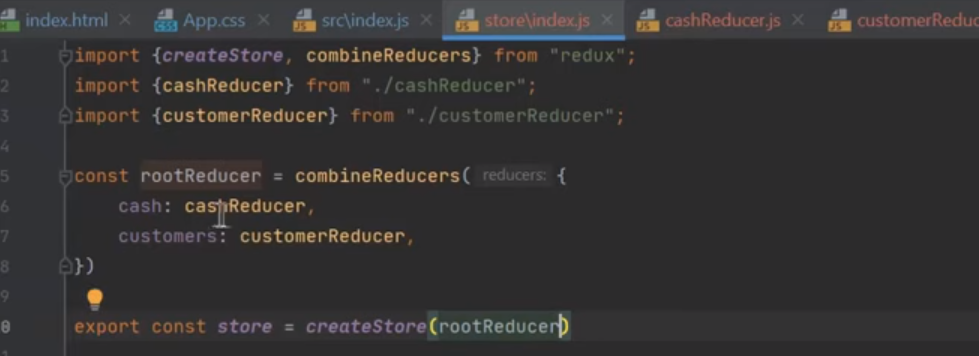
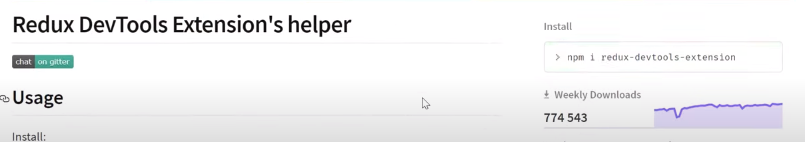
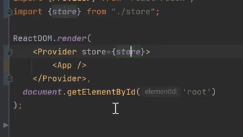
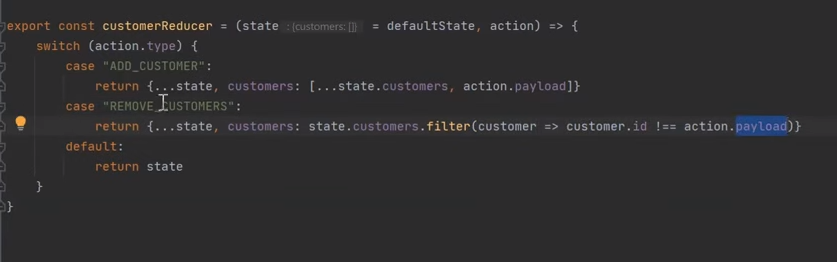
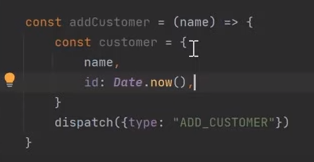
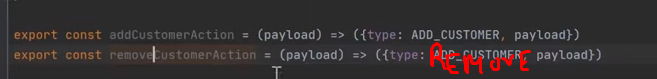
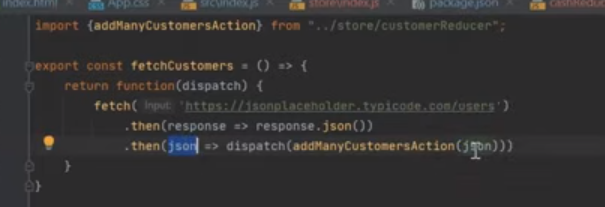
Ulbi TV. Redux playlist.  
  
Какую проблему решает redux?  
  
Состояние из компонента 5 может понадобиться в состоянии 6.  
  
  
Без redux’а нужно было бы выносить состояние в родительский компонент, а потом пробрасывать в нужный через пропсы.  
  
Redux решает эту проблему state хранилищем.  
  
Actions – определяют то, что мы будем делать с данными.  
Передаём в dispatch наш Action.  
Reducer – то, куда dispatcher передаёт action.  
Reducer знает все возможные Actions и он сам изменяет состояние.  
  
  
2) Redux и React. State, reducer, action + redux hooks useDispatch и useSelector.  
  
npm install redux react-redux  
  
Создаём store.  
Первым параметром принимает reducer.  
  
  
  
Reducer простая js функция, принимающая состояние и action.  
  
  
Action – объект type и payload.  
Создаём switch case по action.type, по дефолту возвращает неизмененное состояние.  
- Как мы должны изменять состояние в иных случаях?  
Новый объект с переписанным каким-то полем.  
  
  
Создаём начальное состояние, которое и прокинем в reducer по умолчанию.  
   
  
  
ReactRedux – связь реакт компонентов и редакса.  
Получаем Provider из него, в который оборачиваем наше приложение.  
  
Теперь для изменения состояния нужен dispatch, создаем его с помощью хука useDispatch from react-redux.  
  
Теперь, как это состояние получить?  
Хук useSelector, принимает функцию, которая уже принимает состояние и возвращать нужную переменную.  
  
  
  
Пишем функции для взаимодействия с dispatch.  
  
  
**3) Redux и React. Combine Reducers, redux devtools.**  
  
Проблема: если приложение будет разрастаться и одного reducer’a не будет хватать. Что делать если reducer’ов несколько?  
  
Создадим второй reducer.  
Создадим отдельную папку для работы с данными – Store.  
В ней создадим файлы для каждого reducer.  
И файл index.js в котором будет сам store.  
  
Для каждого reducer’a свой initialState.  
  
Customers reducer.

  
  
CashReducer.  


Теперь объединение reducer’ов.  
Функция combineReducer из redux.  
Создадим новый объект из combineReducer, который в свой очередь принимает объект из reducer’ов.  
  
И передаём теперь этот новый объект параметров в store.  
  
  
Но щас ошибка в браузере, ведь в useSelector пытаемся получить поле из одного состояния.  
  
  
  
Теперь для удобства разработки необходимо будет отслеживать изменение состояния. Это можно сделать с помощью инструментов разработчика.  
  
Кстати при создании store можно передать как middleware, так и инструменты разработчика.  
А для того, чтобы использовать middleware вместе с инструментами разработчика, нужно установить модуль (но он теперь Deprecated).  
  
  
  
Из этого пакета импортируем функцию composeWithDevTools и передадим её вторым параметром в createStore.  
  


Установить redux DevTools расширение в браузер.  
  
  
Store необходимо передать в компонент provider.  
  
  
**4) React и Redux.Action creators. Работа с массивами. Рефакторинг.**  
  
Получаем массив customers с помощью useSelector.  
  
Доделываем логику customerReducer.  
  
Создаем функцию для добавления клиента.  
  
  
  
  
  
Хорошая практика – вынесение названий Action’s в отдельные константы.  
  
  
Создадим функцию actionCreater, возвращающая объект.  
  
  
**5) React и Redux.Action creators. Redux thunk и асинхронные действия.**Для работы с асинхронным кодом потребуется модуль redux-thunk.  
  
Добавляем его в composeWithDevTools, как middleware.  
  
  
Цель – получить с сервера пользователей.  
  
Введём новый action в customerReducer.  
Создадим асинхронную функцию, для работы с сервером. Она должна возвращать новую функцию, которая параметром принимает dispatch.  
  
  
  
Назначаем обработку через клик.  
  
