# **06.08.2020 2 часа – Document JS + 4 часа - Redux Отчет: познакомился с Flux и Redux, разобрал их отличия, написал Redux-функцию reducer**

**Redux** – это отличная структура управления состоянием, обычно используется вместе с React.  
В SPA управление данными со стороны клиентской части намного сложнее, чем кажется. В React можно управлять состоянием, но, когда приложение растет, делать становится это неудобно, так как появляются ошибки и ловушки. Facebook придумал решение для такой проблемы – Flux.

**Flux** – паттерн управления состоянием, который дополняет составные компоненты React, используя однонаправленный поток данных. У Flux очень много хранилищ, и каждое хранилище использует разные небольшие части состояния или данных в нашем приложении. То есть каждый отдельный модуль имеет свое хранилище

**Flux data flow** – поток данных Flux  
1) Пользователь взаимодействует с элементом, элемент запускает действие  
2) Действие отправляет соответствующую функцию, затем эта функция изменяет хранилище  
3) Когда хранилище обновляет свои данные, элементы, которые зависят от этого хранилища, обновляются автоматически. Нам не нужно изменять данные в модулях

Это и есть однонаправленный поток данных. Когда приложение становится больше, несколько хранилищ управляют данными

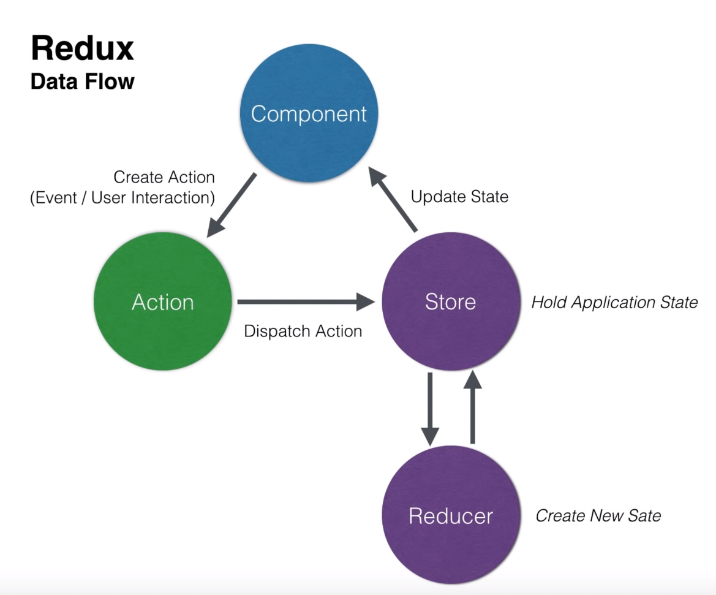
При наличии нескольких хранилищ состояние нашего приложения функционирует по схеме выше, но поток данных является однонаправленным

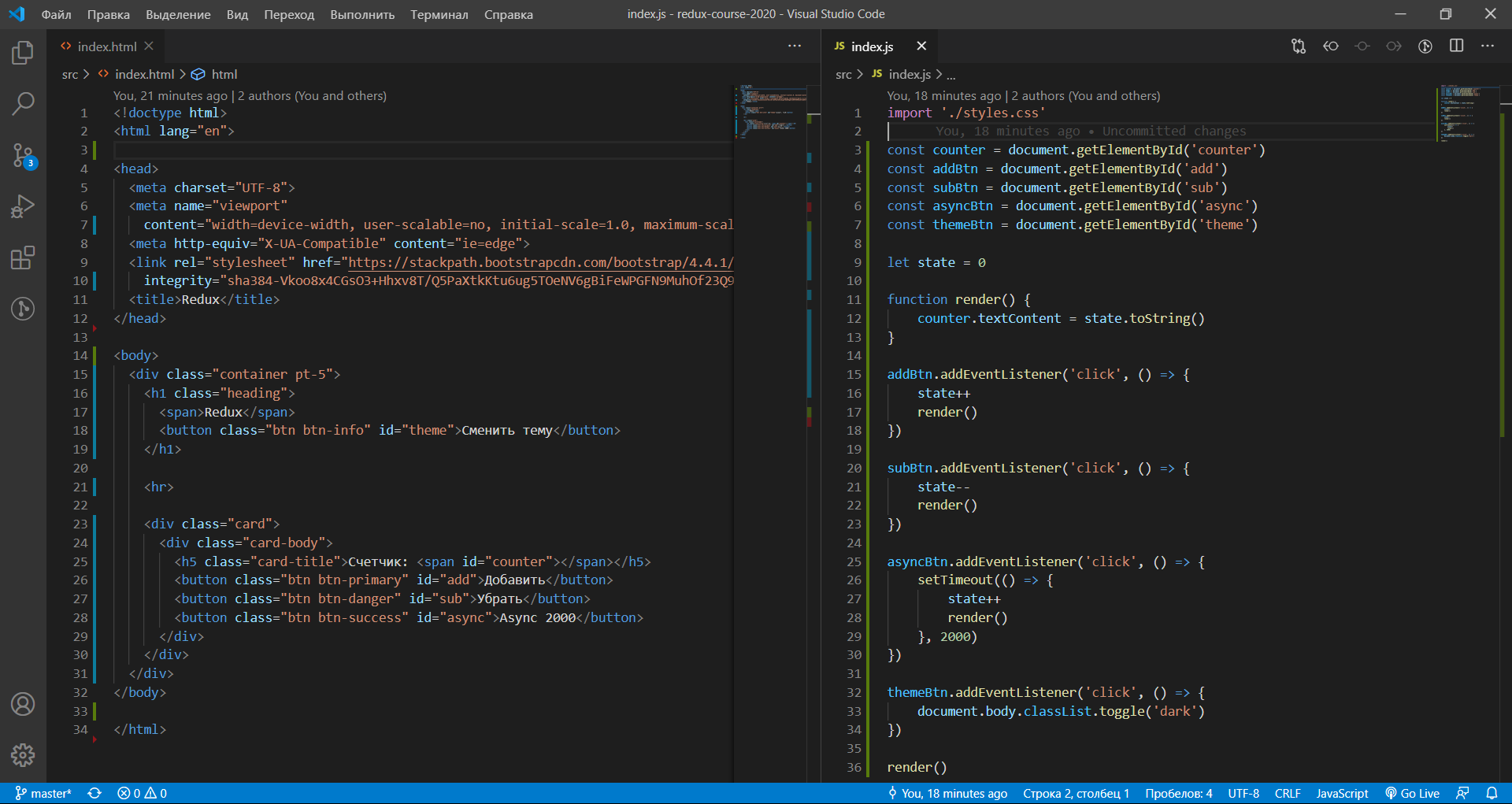
Flux имеет несколько хранилищ

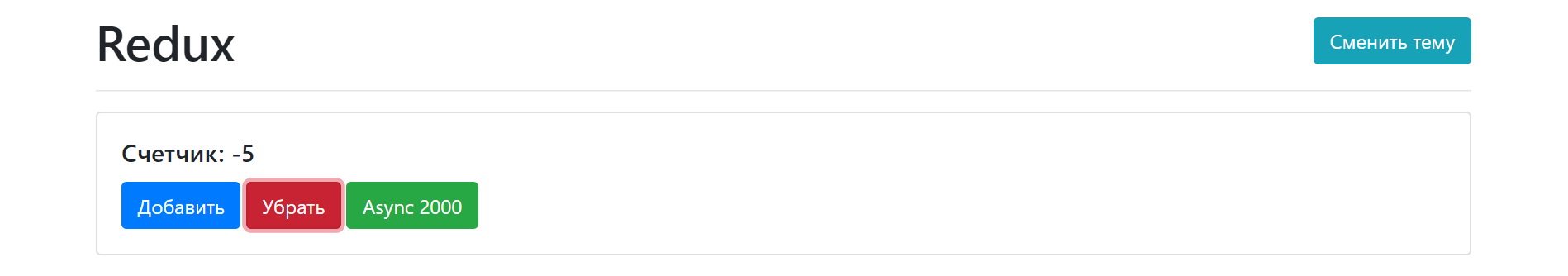
**Redux** – предсказуемый контейнер состояний для приложений JavaScript. Redux также придерживается однонаправленного потока, но он полностью отличается от Flux, т.к Flux может иметь множество потоков

**3 принципа Redux**1) Единственный источник истинны  
2) Состояние доступно только для чтения  
3) Изменения можно делать только с помощью чистых функций

В Redux состояние всего приложения хранится в объекте в одном хранилище  
Единственный способ изменить состояние - вызвать action и объект описывающий, что произошло  
Redux работает независимо от React, его можно использовать в JS, Angular, Vue

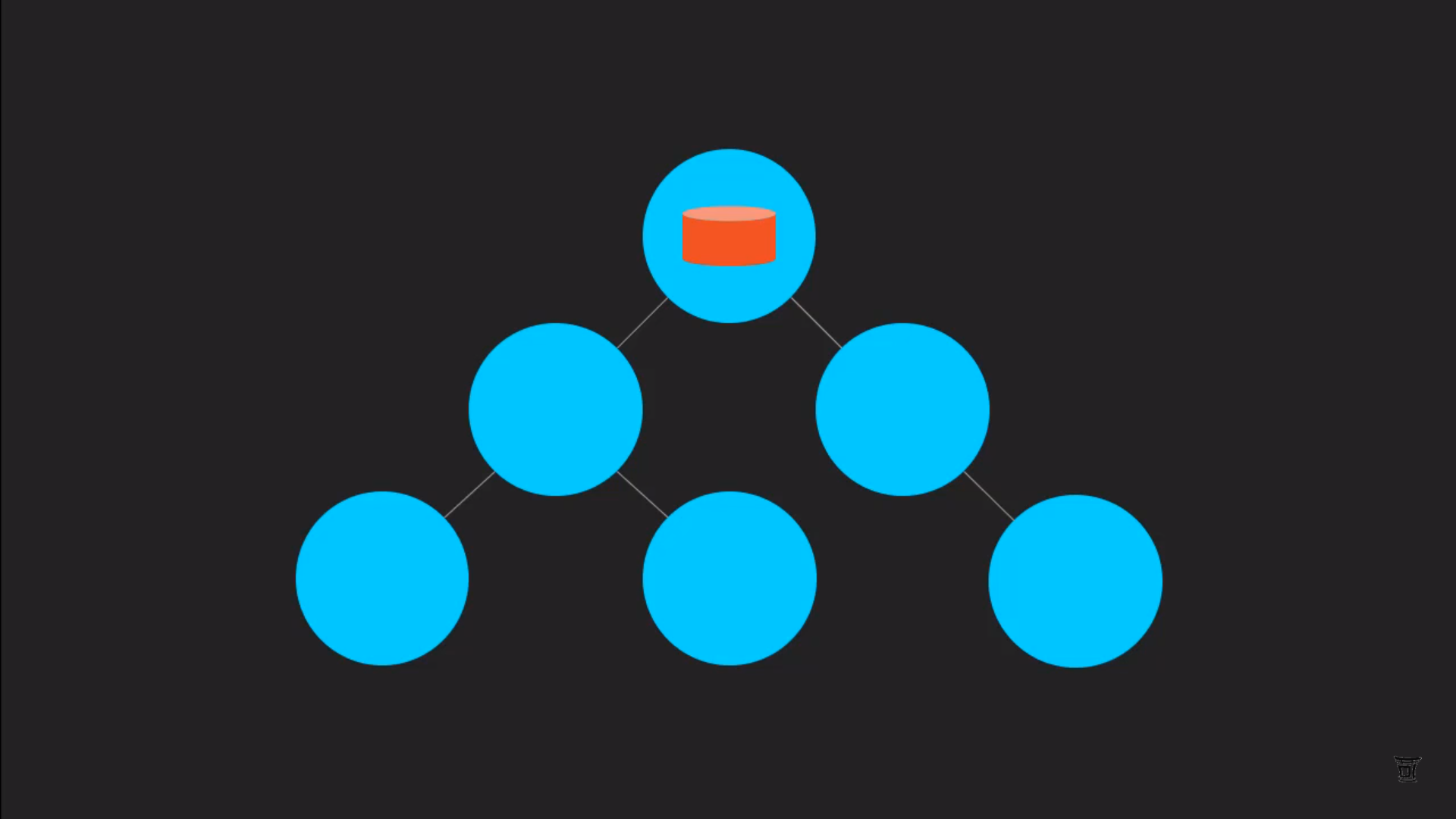
Redux работает по следующей схеме  
  
  
Рассмотрим код ниже, который хранит состояние в переменной state

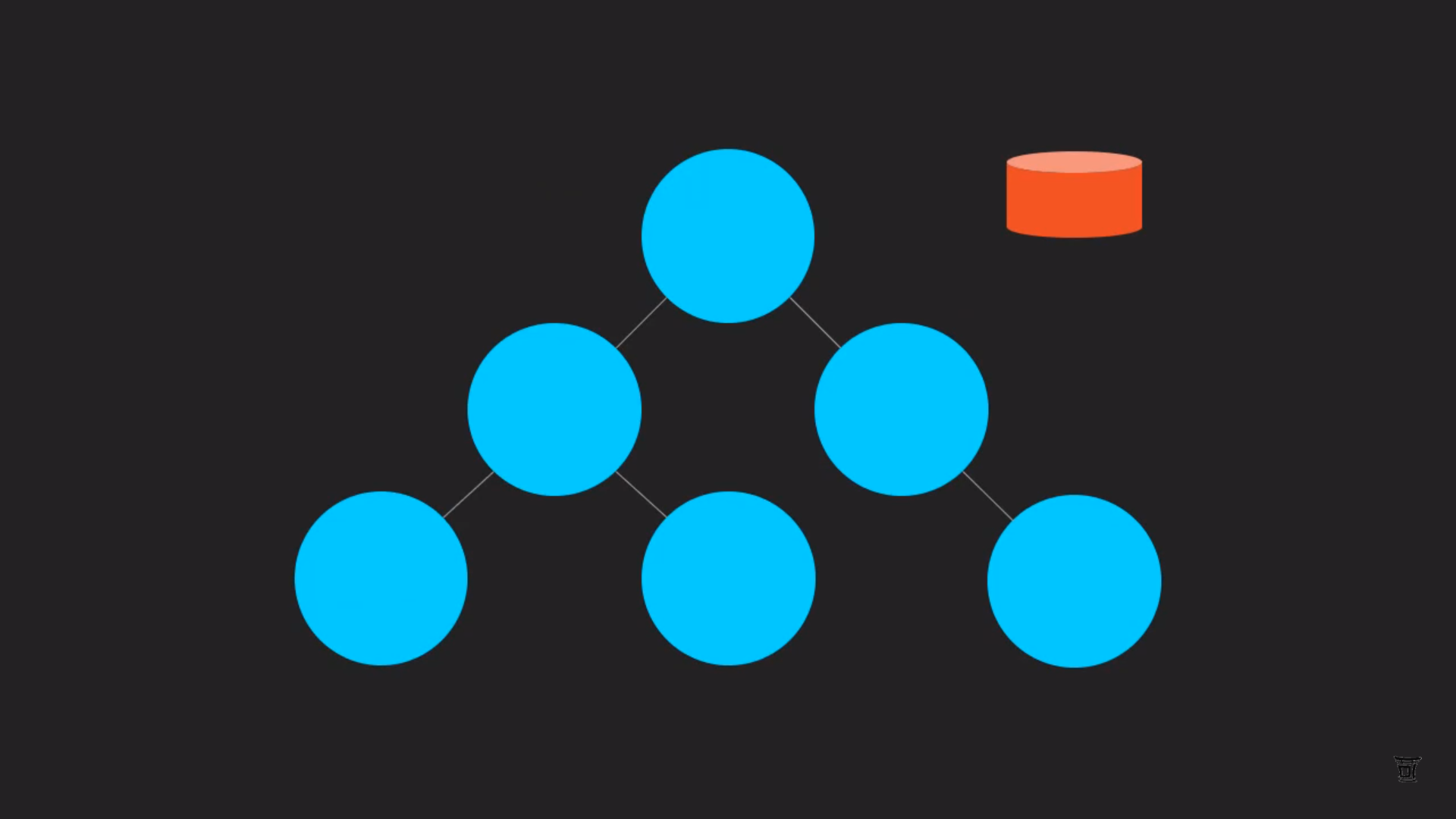


Данное приложение может менять тему, инкрементировать и декрементировать счетчик, а также делать инкремент с асинхронностью 2000 мс

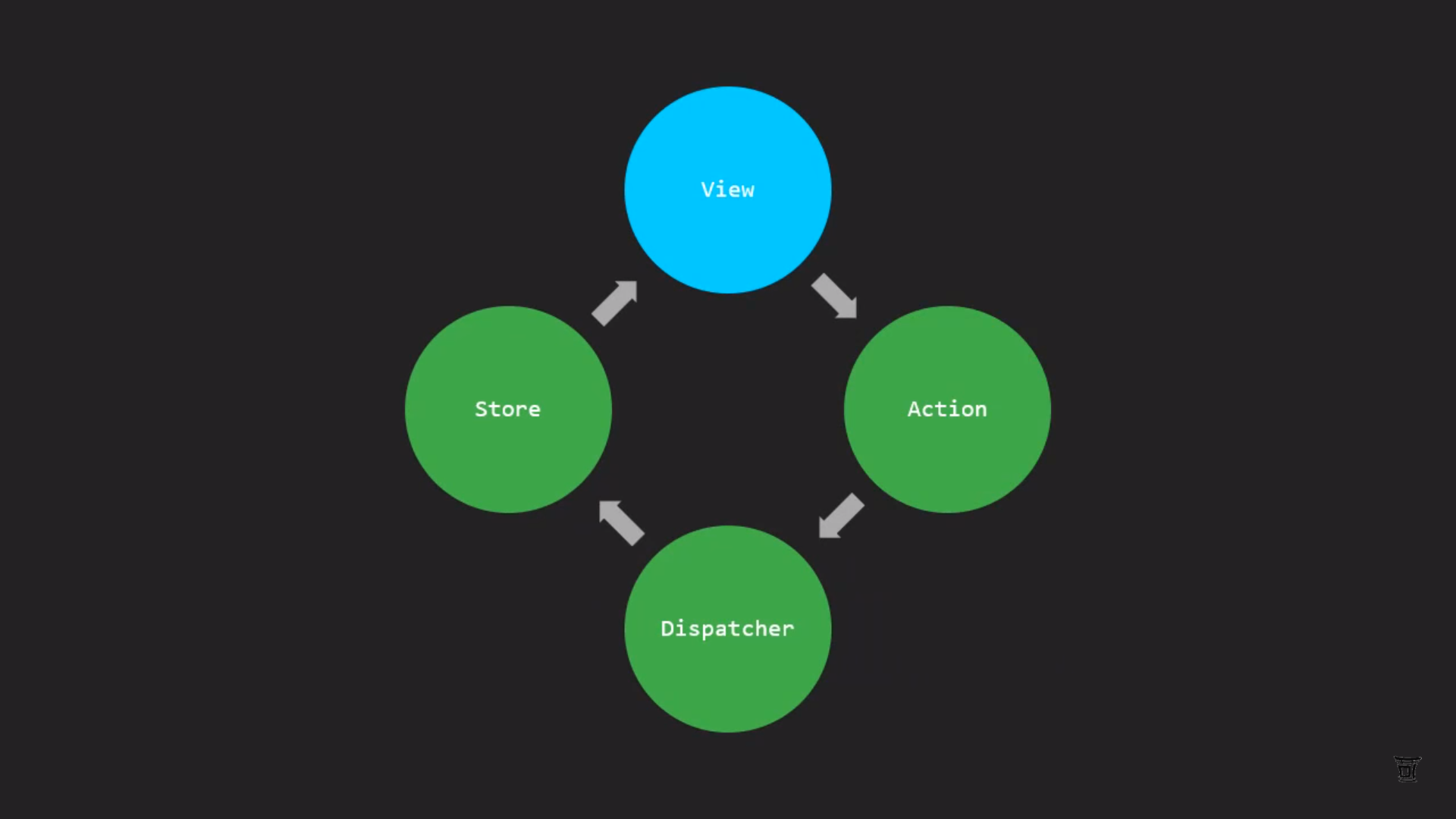
Данное приложение работает достаточно хорошо, но у него есть проблема: когда мы обновляем страницу, наши данные теряются, то есть счетчик принимает начальное значение 0, а тема становится стандартного цвета

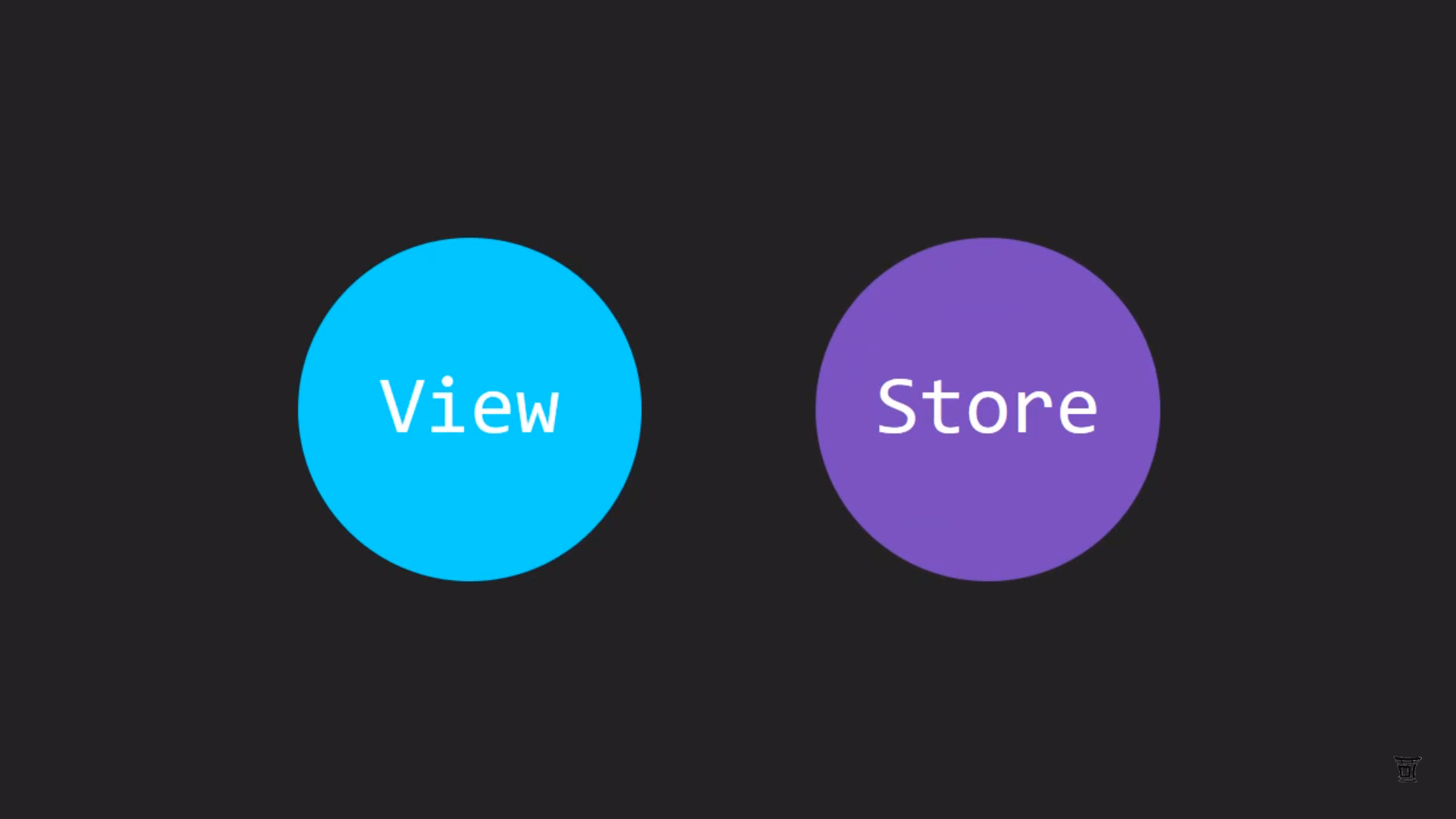
**Введение в Flux и Redux**

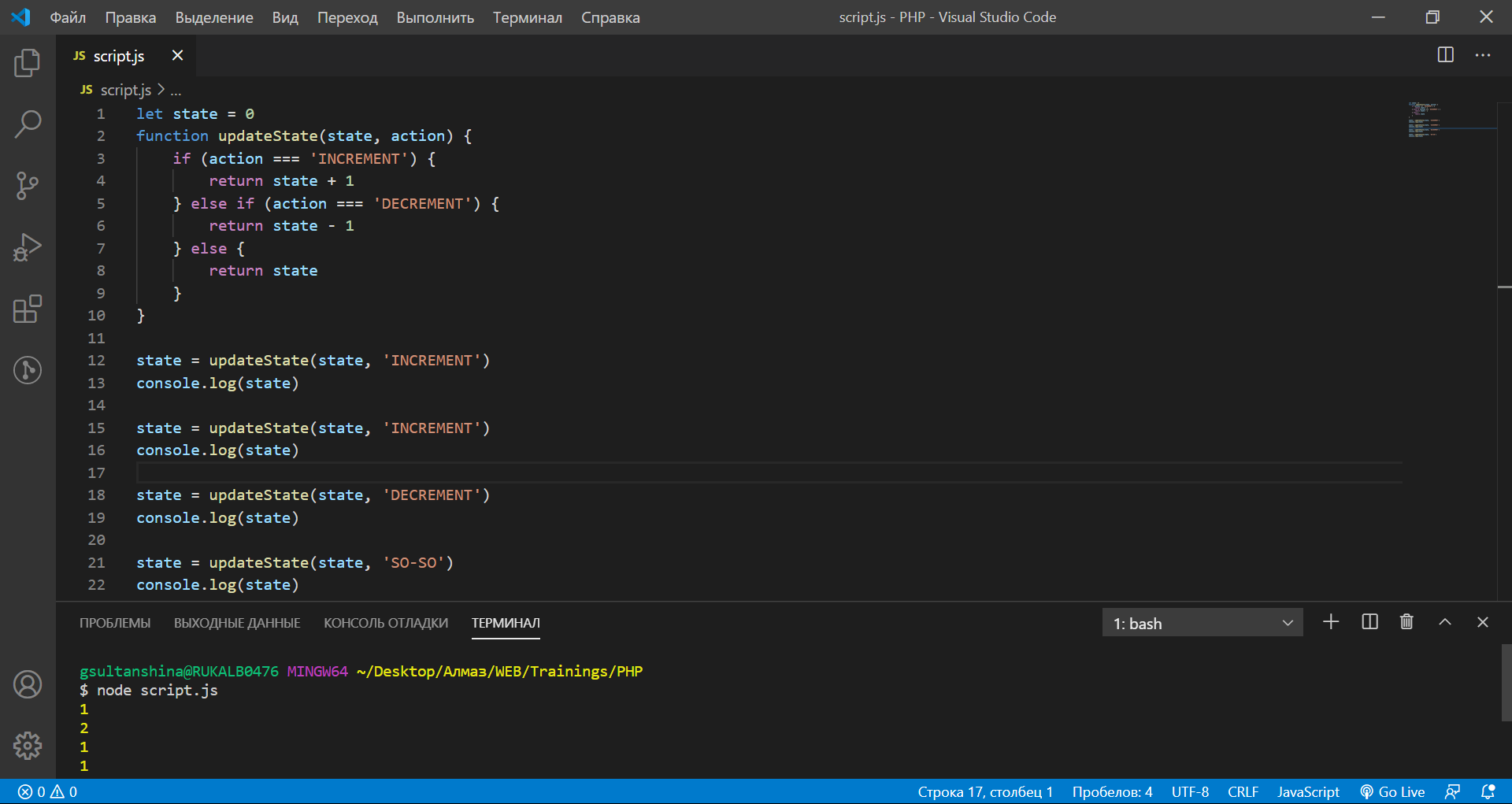
В React состояние рекомендуется размещать как можно выше, в главном компоненте  
При такой архитектуре данные спускаются вниз по свойствам  
Для изменения состояния компоненты сообщают своим родителям об обновлении состояния с помощью передаваемых функций  
Далее основной компонент обновляет состояние и вновь отправляет данные вниз  
Для многих приложений это является нормальным, но с расширением приложения появляется больше компонент, и такая архитектура является не очень удобной

Разработчики Facebook, столкнувшись с такой проблемой, придумали новую архитектуру Flux  
Главной идеей Flux является деление состояния от компонентов. Таким образом любой компонент может взаимодействовать с состоянием напрямую

Flux состоит из 4 частей:  
1) view – пользовательский интерфейс (в React – это компонента)  
2) store – хранилище. Здесь находится состояние приложения  
3) dispatcher – диспетчер. Сообщает хранилищу о каком то событии, передает необходимую инфу  
4) action – действие или событие, которое происходит в приложении

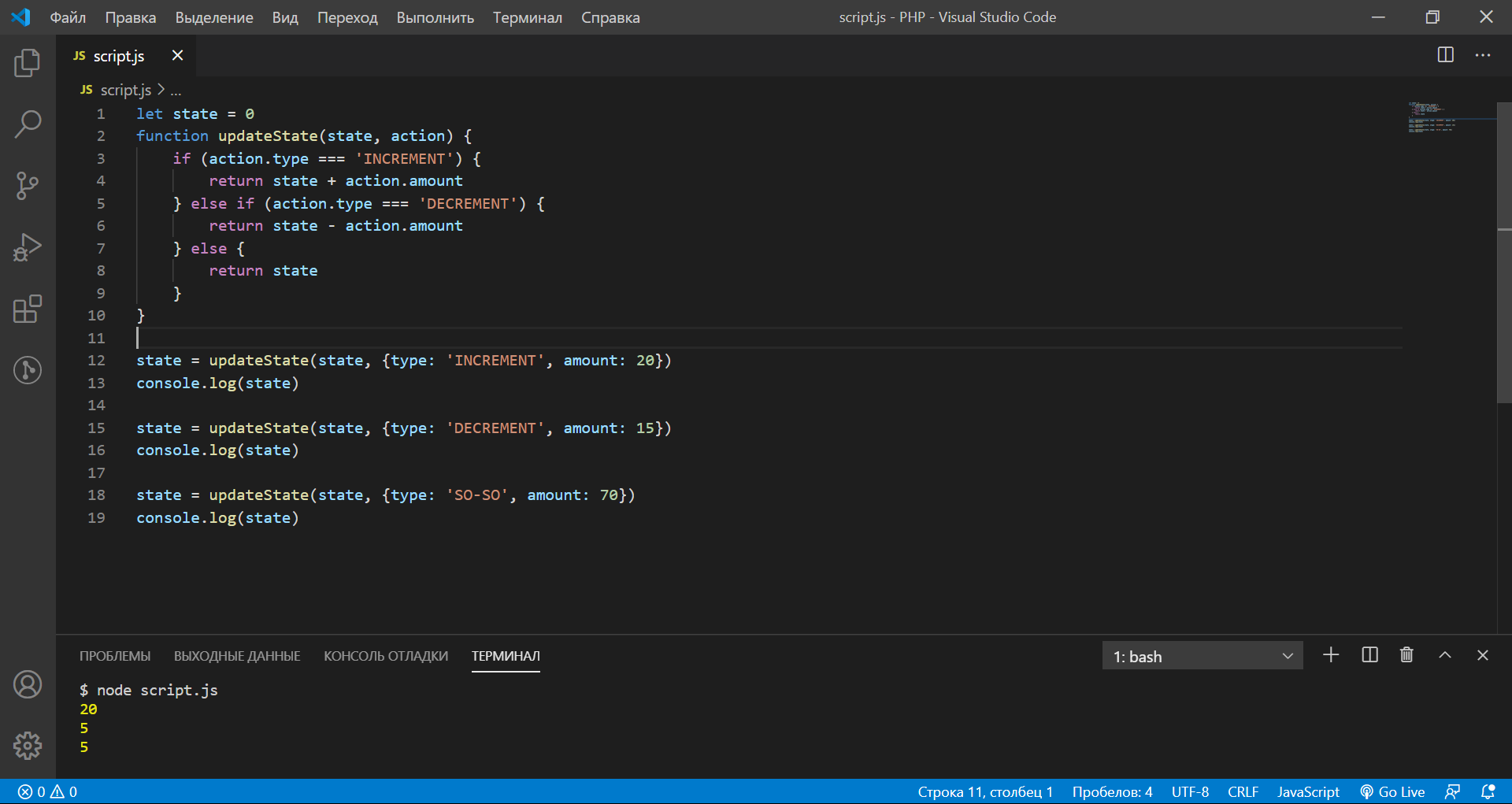
Если в общем, то компонента генерирует действие, диспетчер сообщает об этом хранилищу, хранилище изменяет состояние, данные передаются в компоненту  


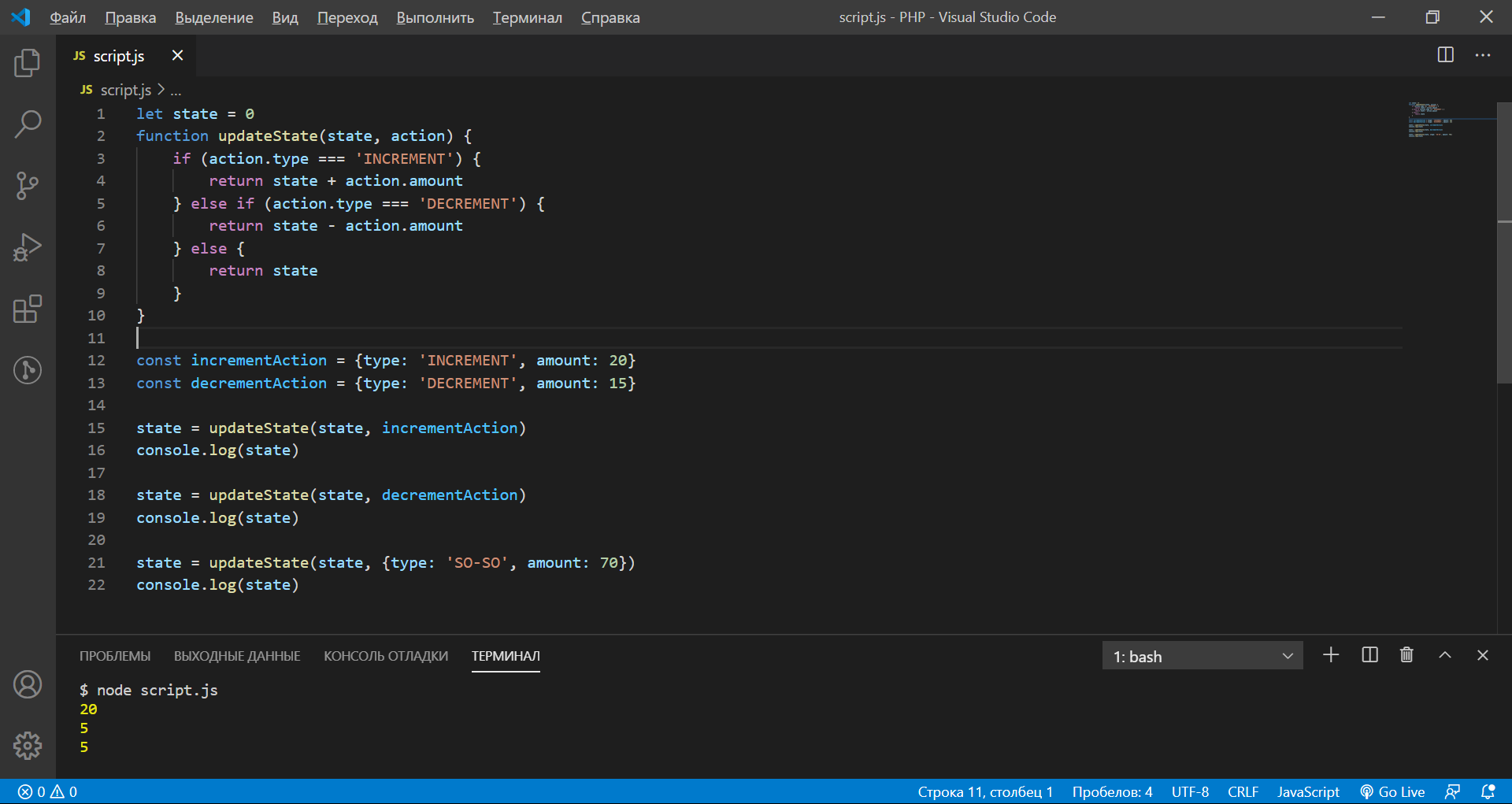
Flux имеет несколько реализаций, но самой популярной из них является Redux. Главной характерной чертой Redux является простота. В нем компонента взаимодействует с хранилищем напрямую. С помощью action компонента сообщает хранилищу, что произошло в приложении. Также в Redux имеется dispatcher, но он является частью хранилища  


**Пишем функцию Reducer (преобразователь)**Данная функция не обновляет исходное состояние, она возвращает новое состояние****

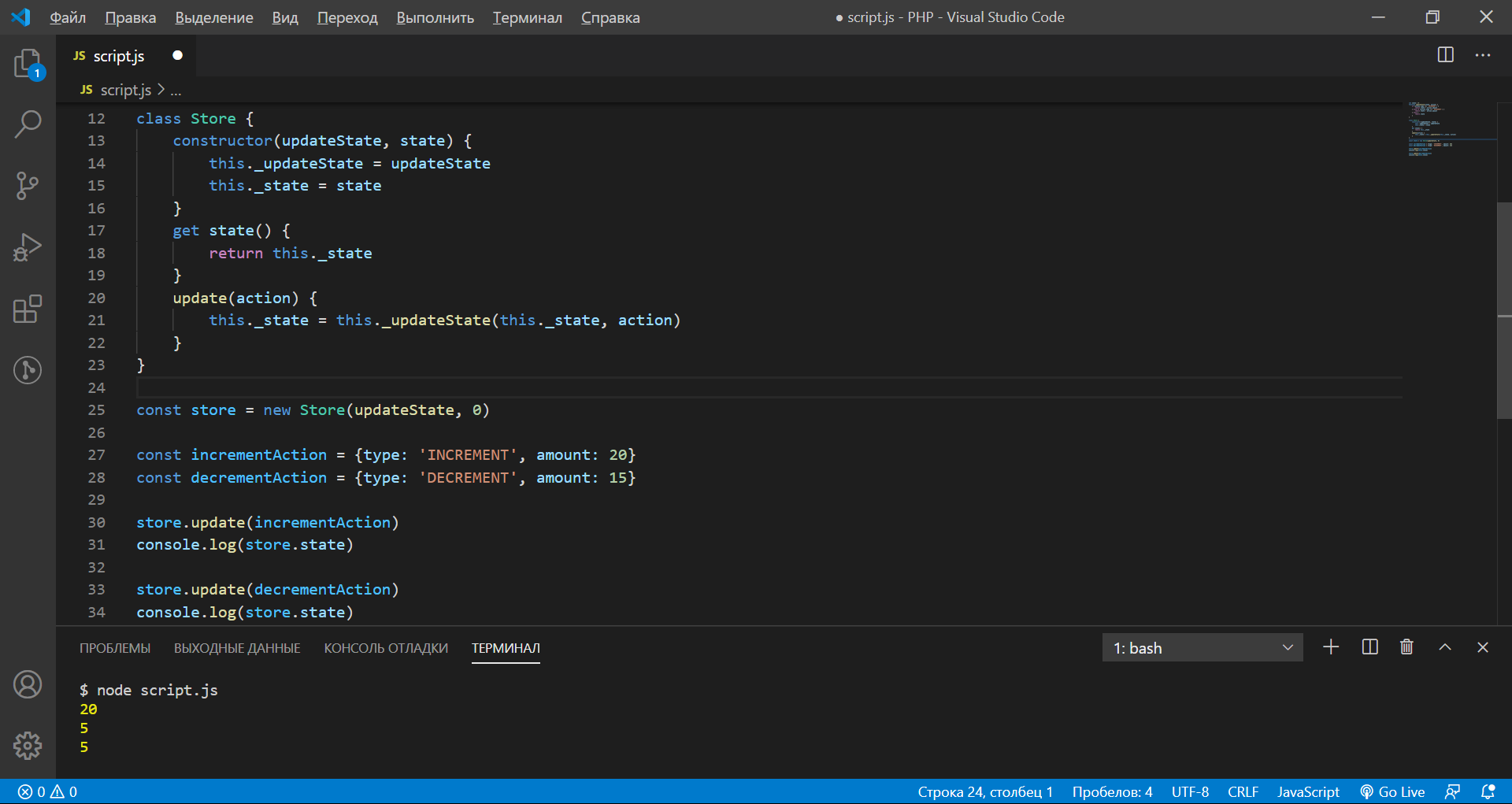
# **07.08.2020 4 часа Отчет: начал изучать официальную документацию Redux**

**Actions**

Перепишем наш код, сделав action более правильным

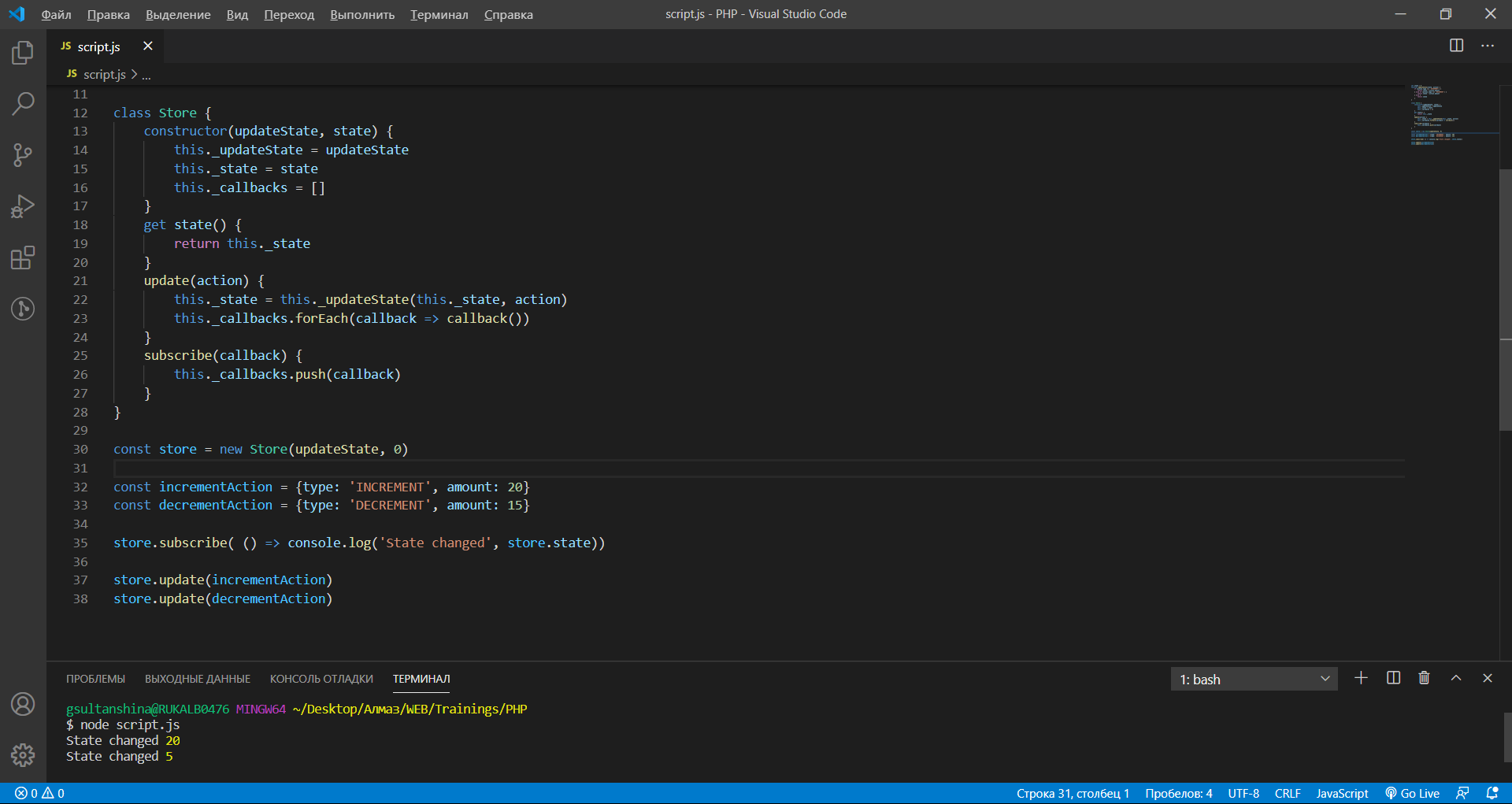
Вынесем объекты в переменную, чтобы их было проще передавать  


**Store**  
На данный момент состояние приложения находится в открытом доступе. Более правильно – спрятать. Это можно сделать несколькими способами, но сейчас воспользуемся классами

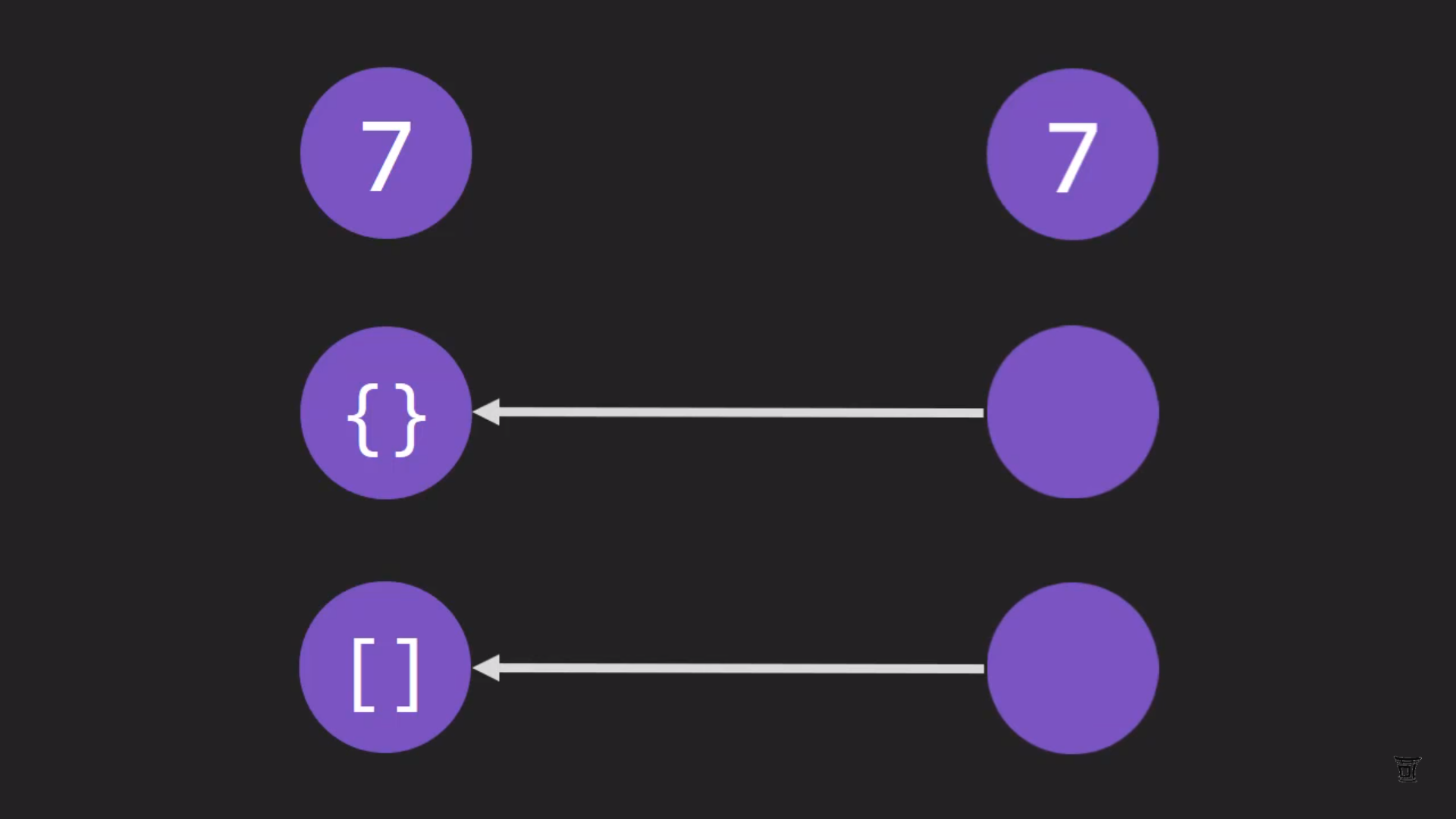


Смысл хранилища заключается не в том, чтобы полностью спрятать состояние, а в том, чтобы ограничить к нему доступ через определенный интерфейc. Также было бы удобно, если при изменении состояния хранилище сообщало об этом всем заинтересованным

**Подписка**Наше хранилище предоставляет метод для уведомления о действиях в приложении. Также хранилище предоставляет состояние с помощью свойства. Так мы никак не можем сообщить внешнему миру о том, что состояние изменилось. Мы можем спрашивать хранилище обновилось ли состояние или нет, но было бы удобнее подписаться на обновления, передав колбэк. Для реализации этого механизма мы воспользуемся паттерном наблюдателя. Все части приложения, которые необходимо знать об обновлении состояния, передадут хранилищу функцию обратного вызова. В свою очередь, хранилище при изменении состояния вызовет все эти функции



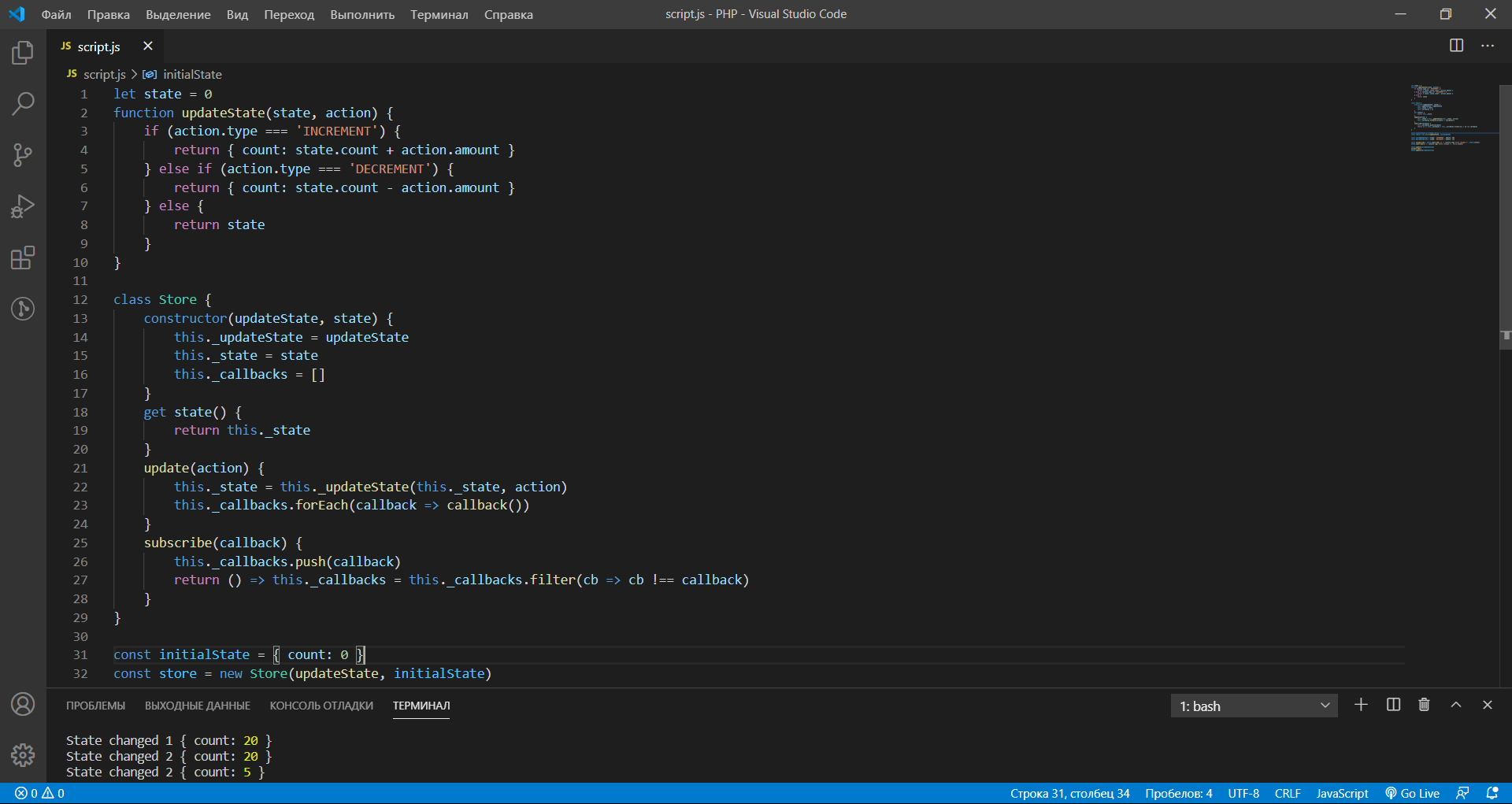
**Состояние**Главными двумя составляющими хранилища являются состояние и функции, которые его изменяют. В данный момент состоянием является простое число. В реальном проекте скорее всего понадобится использовать объект или массив

Примитивные данные передаются копированием значения, а объекты копированием ссылки

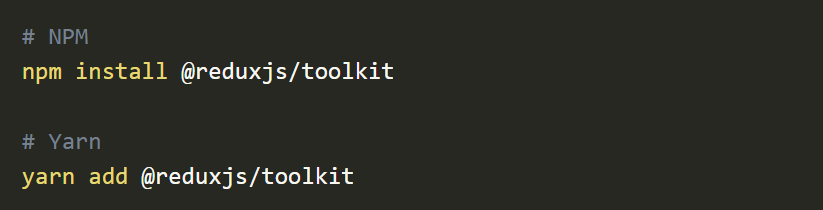
Таким образом, получив объект в аргументе функции, мы будем изменять оригинальный объект

Очень часто проект разбивается на несколько команд, которые занимаются различными частями приложения, но даже если проектом занимается один человек, передавая состояние туда-сюда, то рано или поздно можно нарваться на баги

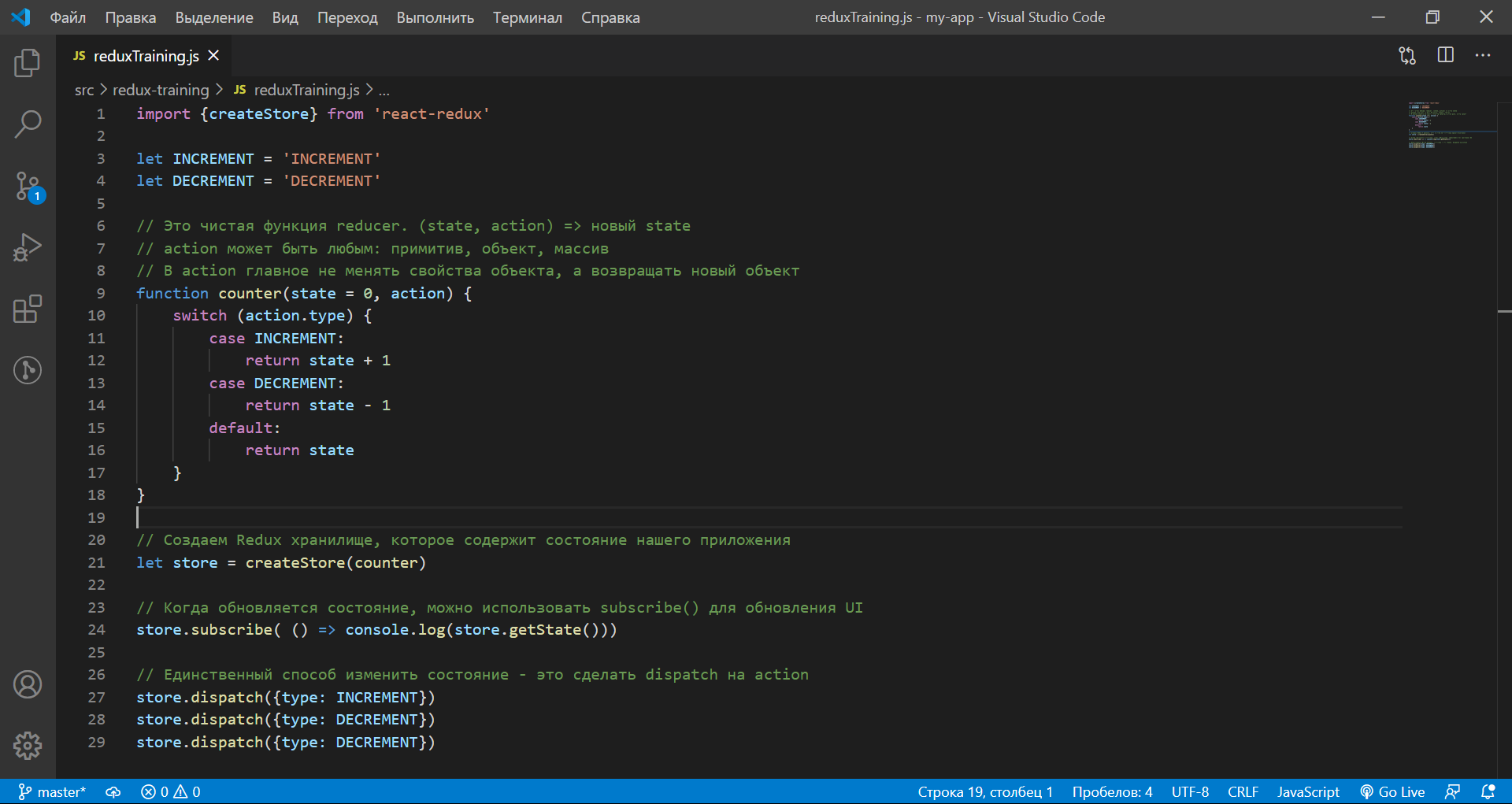
Для решения проблем управления состоянием JS использует функциональное программирование:  
- чистые функции   
- неизменность данных

  
Функция изменяющая состояние (reducer) должна быть чистой. Все данные в приложении помещаются в одно место (в хранилище). Чистые функции возвращают новое состояние

**Официальная документация**Redux – предсказуемый контейнер состояния для JS приложений  
Он помогает писать приложения, которые ведут себя последовательно, и позволяет запускать в разных средах (клиенсткой, серверной, нативной), которые легко тестируются  
Кроме того, он обеспечивает возможность редактирования кода в отладчике  
Redux можно использовать с любой библиотекой, он весит всего лишь 2кб

**Установка  
Redux Toolkit (RTK) –** подход к написанию логики, который рекомендуют разработчики Redux  
Он оборачивается вокруг ядра Redux, содержит необходимые пакеты и функции для создания app

**Create a React Redux App**Чтобы создать новое приложение с помощью React и Redux, рекомендуется использовать шаблон

**Создадим базовый пример  
**

Вместо того, чтобы напрямую изменять состояние, мы указываем мутации, которые могут произойти с action. Затем мы пишем специальную функцию reducer, чтобы решить, как каждое действие будет менять состояние

В обычном Redux приложении существует только одно хранилище, и одна reduce функция  
По мере роста приложения мы разделяем корневой reducer на более мелкие reducer, которые независимо работают на разных частях дерева state