# **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

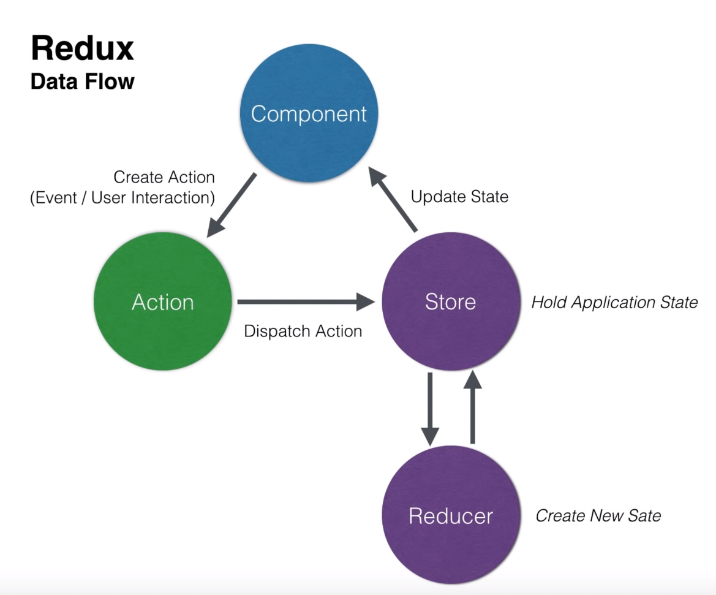
При наличии нескольких хранилищ состояние нашего приложения функционирует по схеме выше, но поток данных является однонаправленным

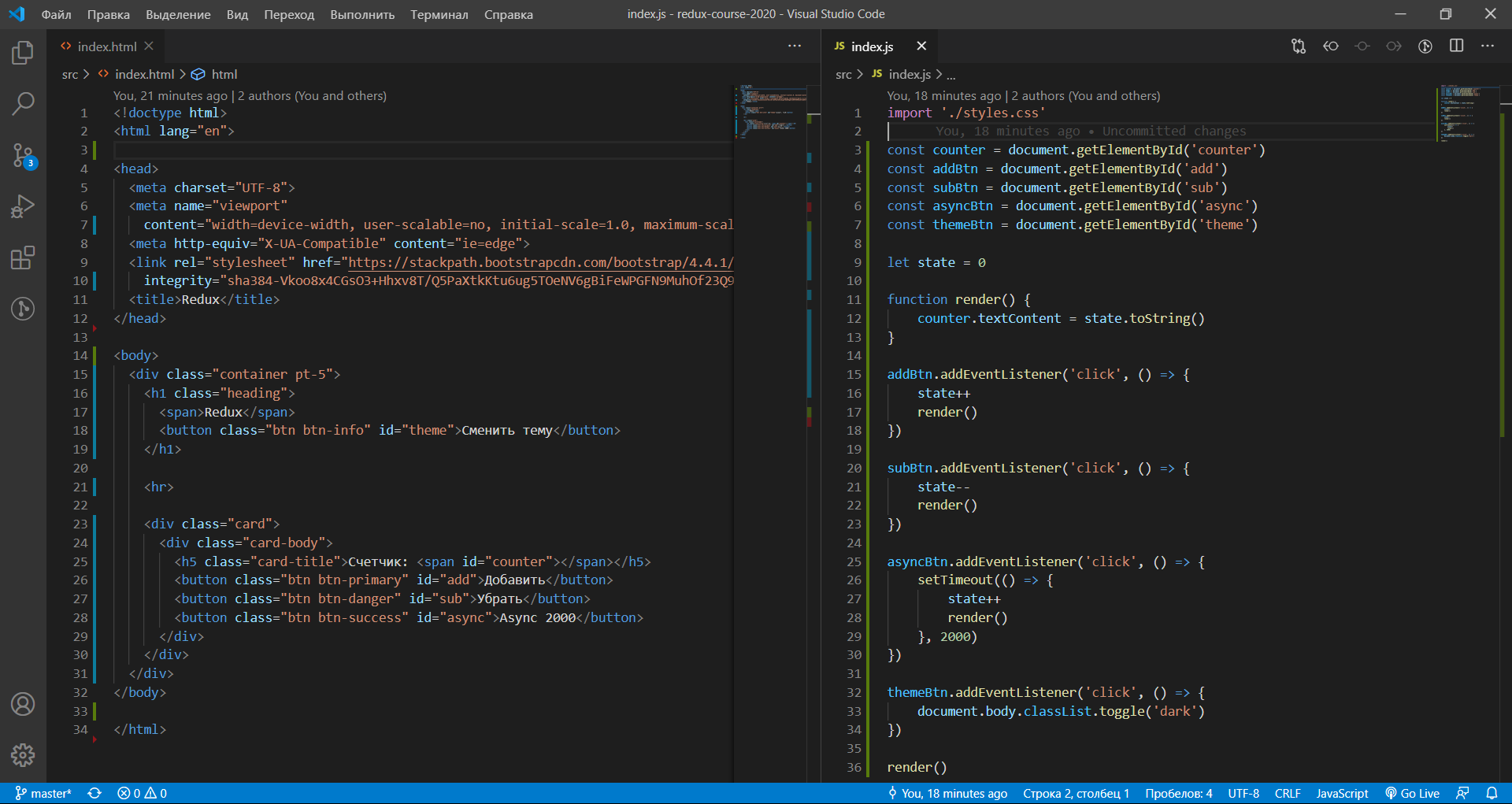
Flux имеет несколько хранилищ

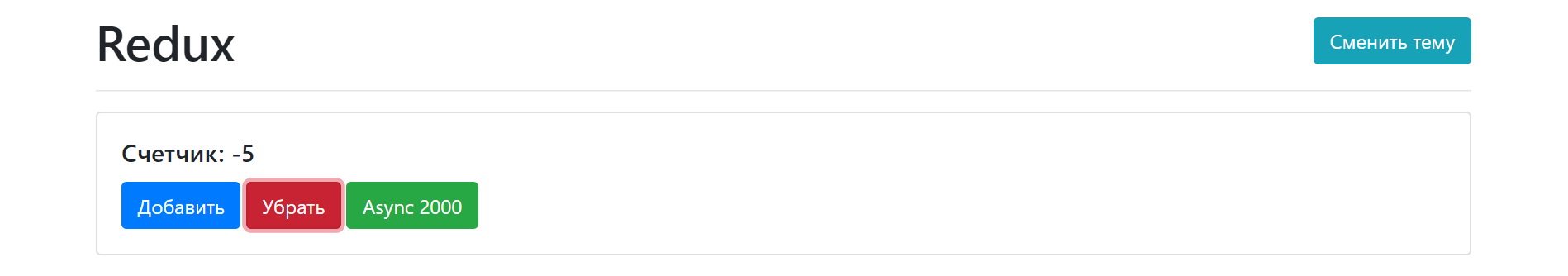
**Redux** – предсказуемый контейнер состояний для приложений JavaScript. Redux также придерживается однонаправленного потока, но он полностью отличается от Flux, т.к Flux может иметь множество потоков

**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

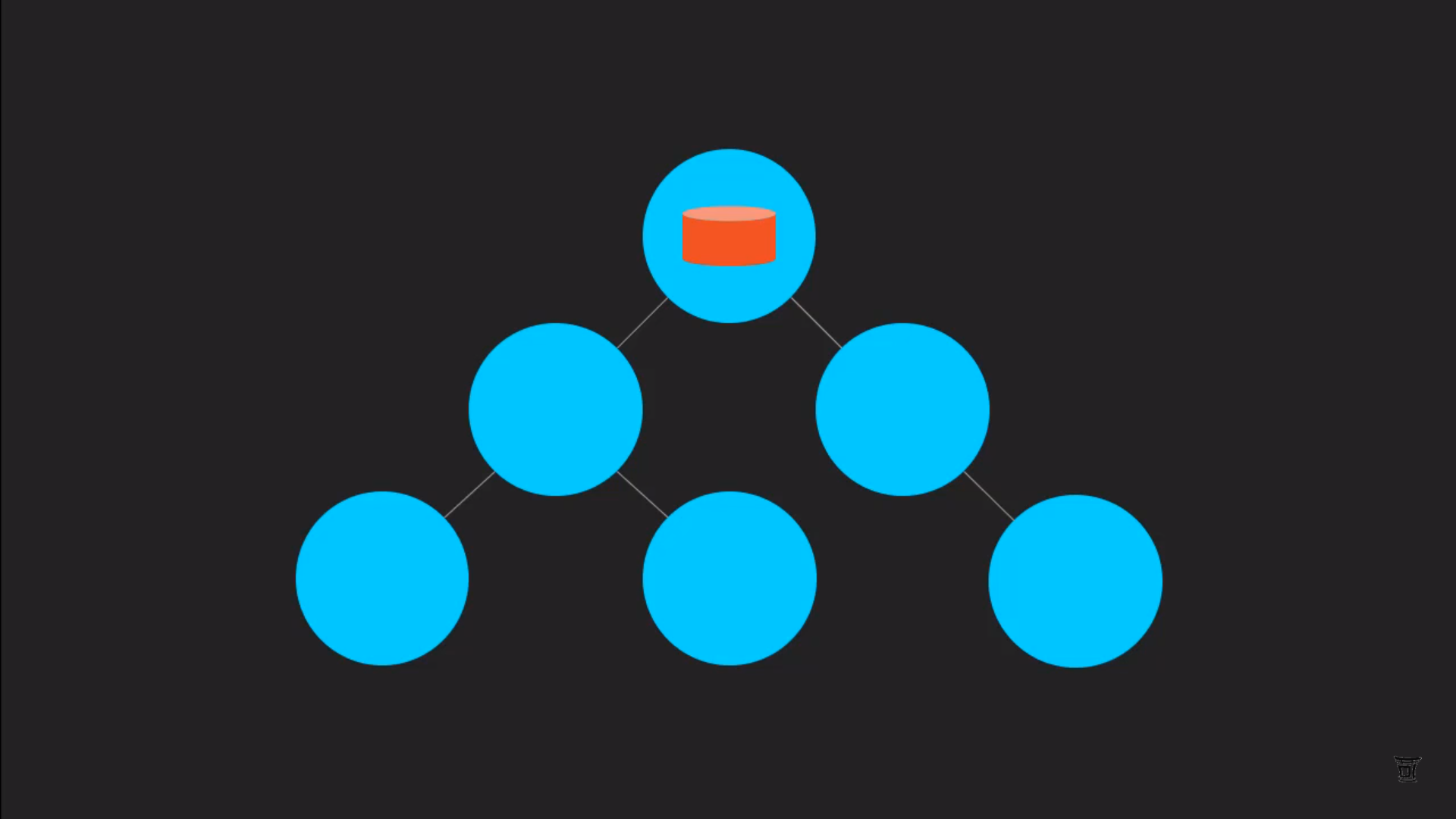
Redux работает по следующей схеме  
  
  
Рассмотрим код ниже, который хранит состояние в переменной state

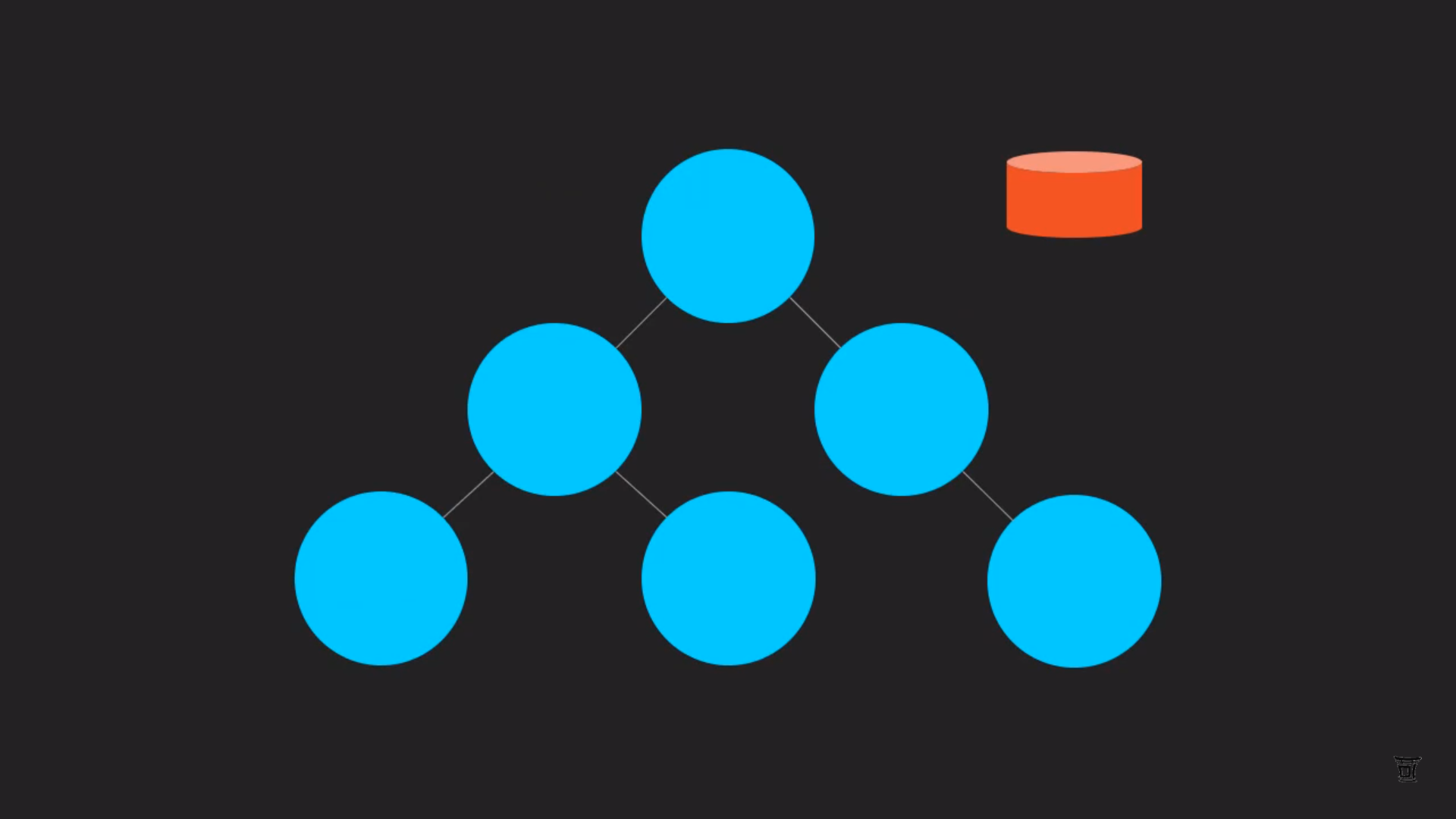


Данное приложение может менять тему, инкрементировать и декрементировать счетчик, а также делать инкремент с асинхронностью 2000 мс

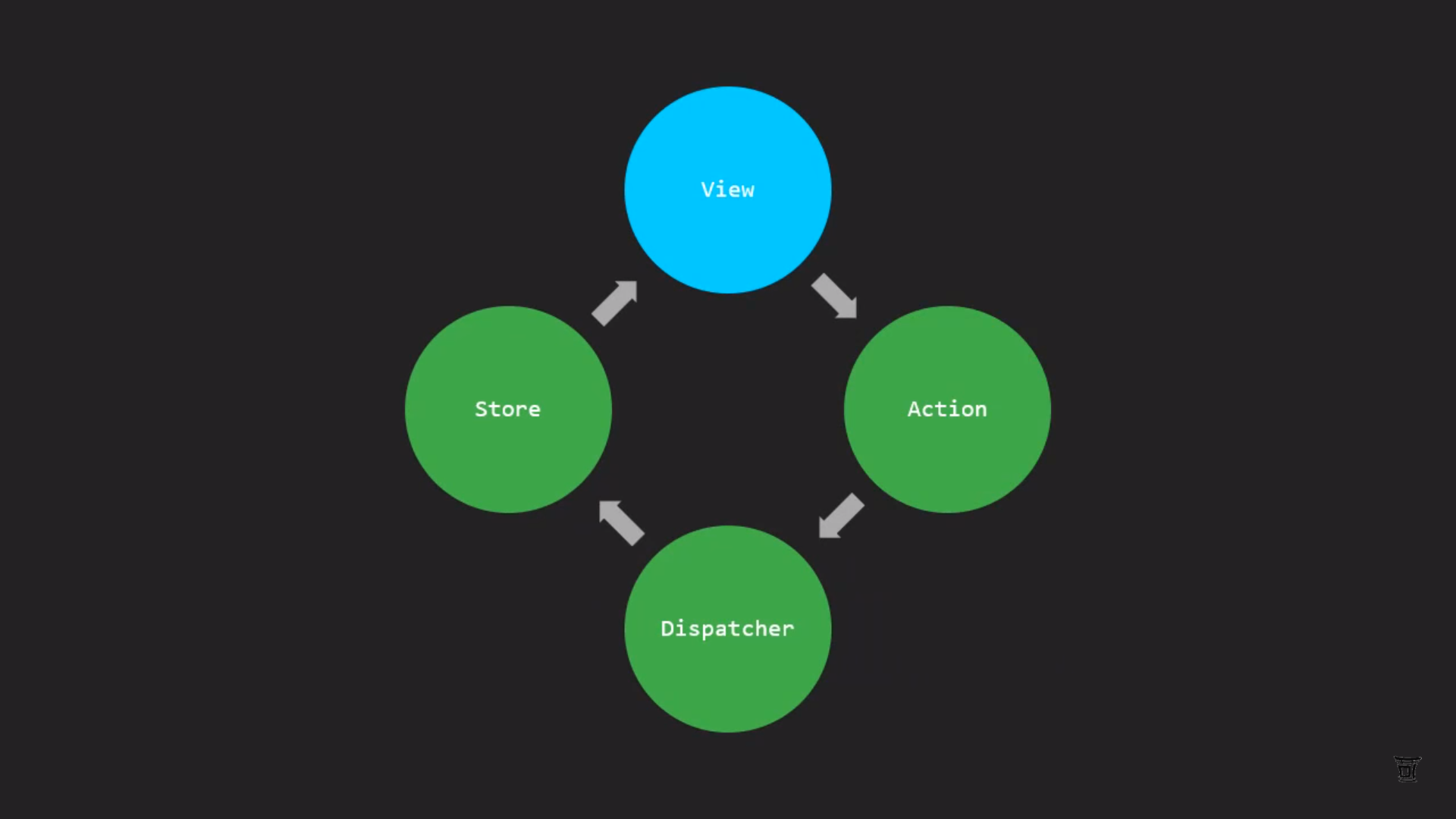
Данное приложение работает достаточно хорошо, но у него есть проблема: когда мы обновляем страницу, наши данные теряются, то есть счетчик принимает начальное значение 0, а тема становится стандартного цвета

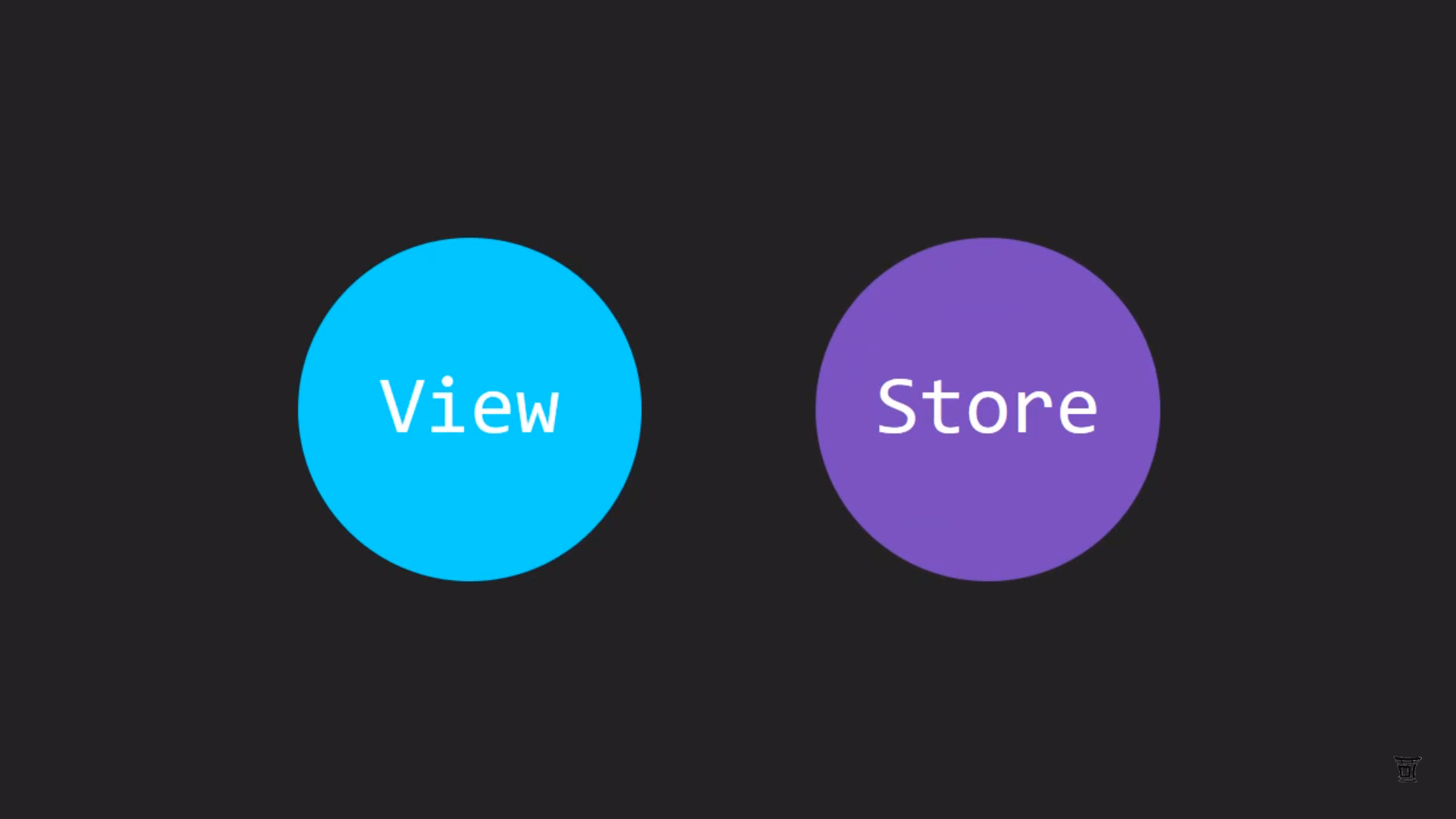
**Введение в Flux и Redux**

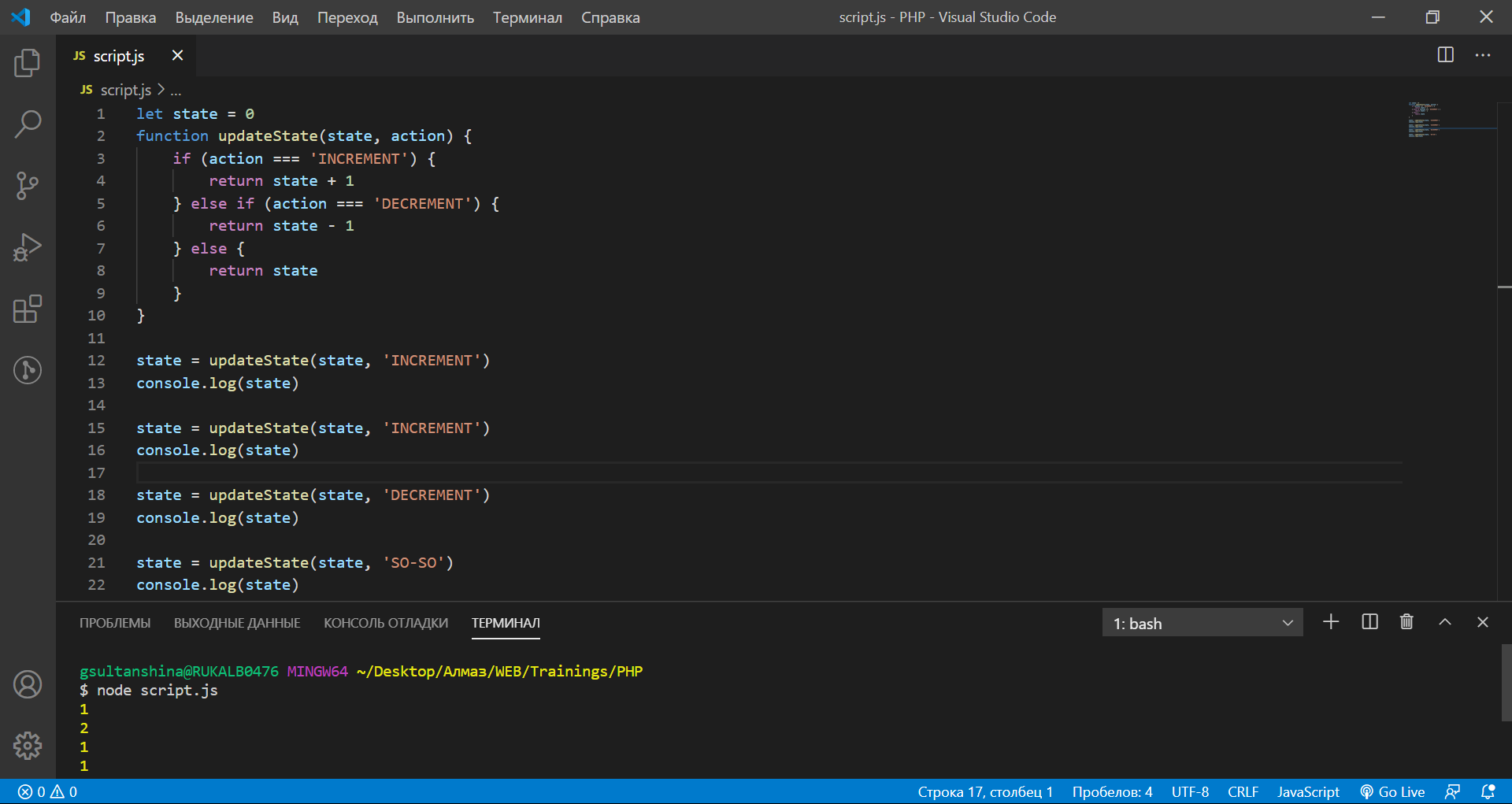
В React состояние рекомендуется размещать как можно выше, в главном компоненте  
При такой архитектуре данные спускаются вниз по свойствам  
Для изменения состояния компоненты сообщают своим родителям об обновлении состояния с помощью передаваемых функций  
Далее основной компонент обновляет состояние 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

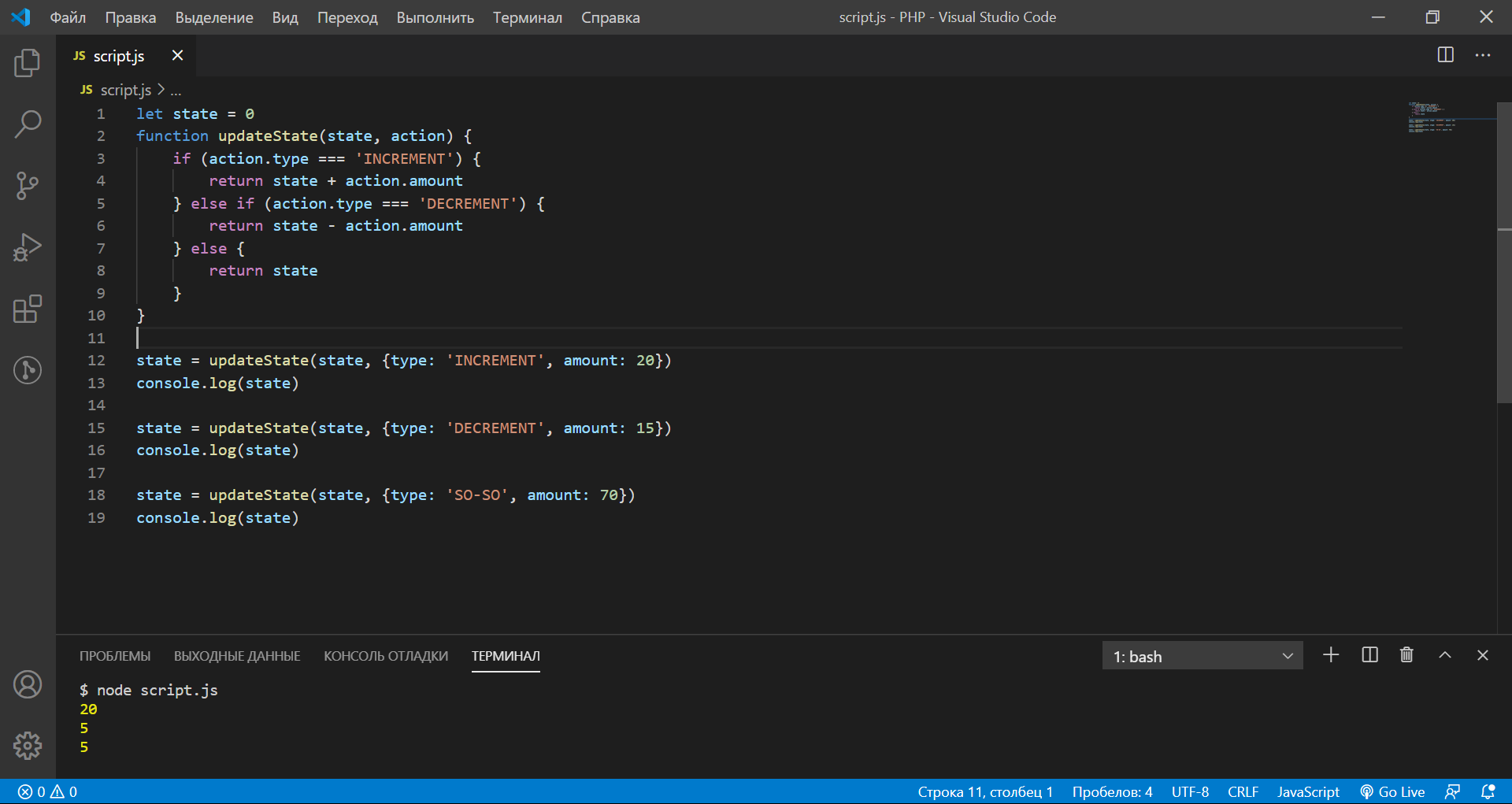
Если в общем, то компонента генерирует действие, диспетчер сообщает об этом хранилищу, хранилище изменяет состояние, данные передаются в компоненту  


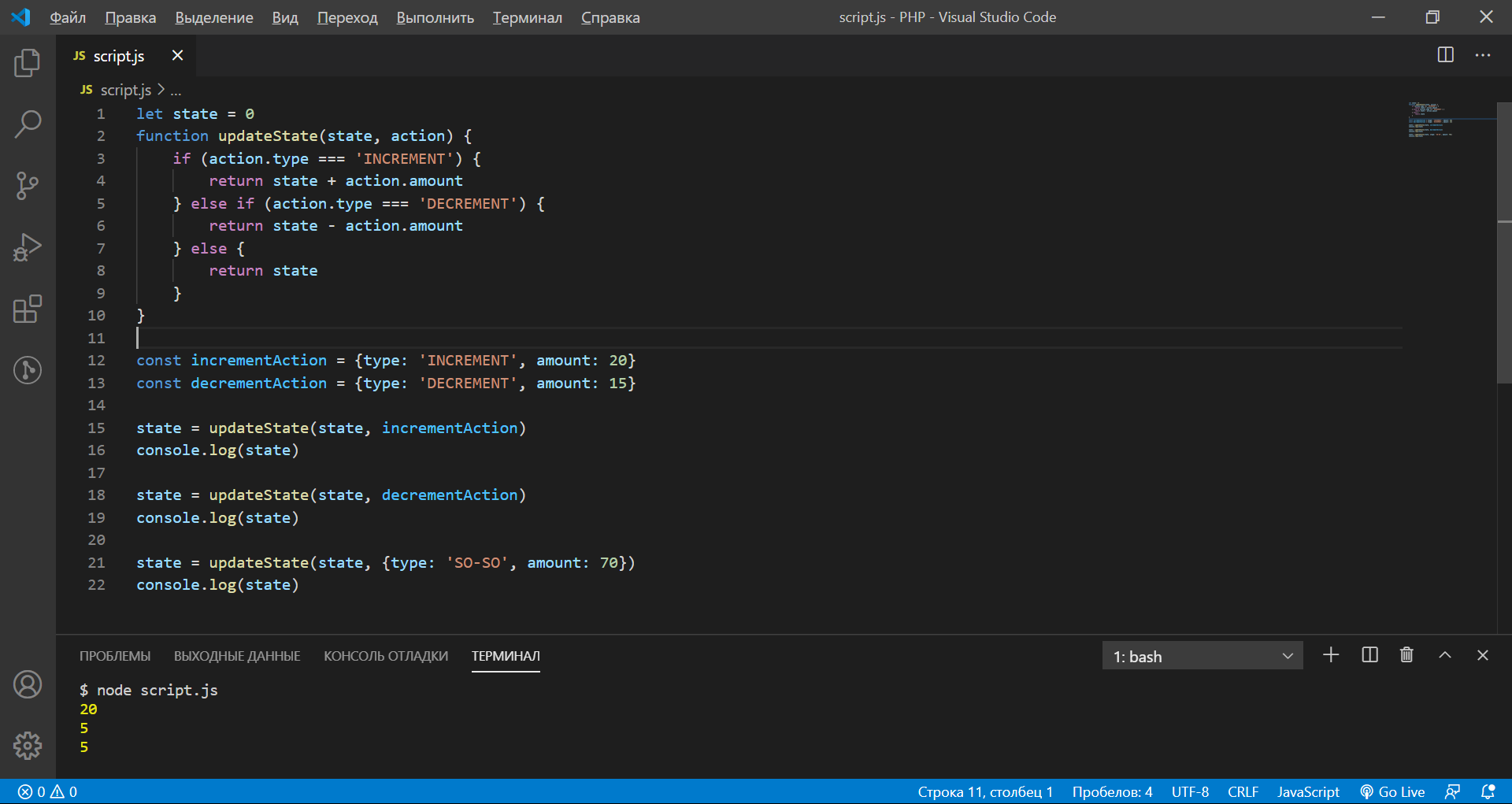
Flux имеет несколько реализаций, но самой популярной из них является Redux. Главной характерной чертой Redux является простота. В нем компонента взаимодействует с хранилищем напрямую. С помощью action компонента сообщает хранилищу, что произошло в приложении. Также в Redux имеется dispatcher, но он является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


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

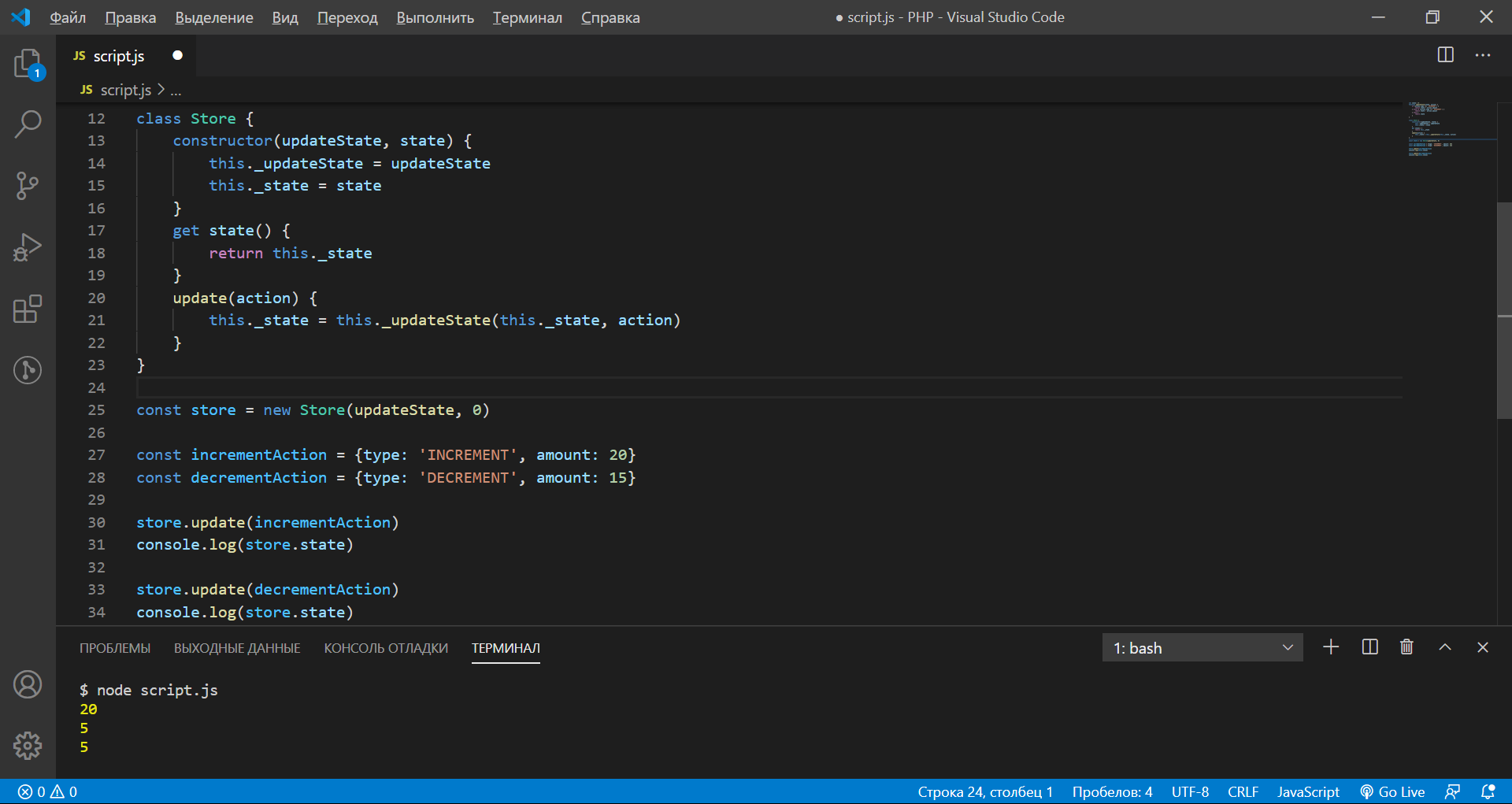
# **07.08.2020 4 часа Отчет: начал изучать официальную документацию Redux**

**Actions**

Перепишем наш код, сделав action более правильным

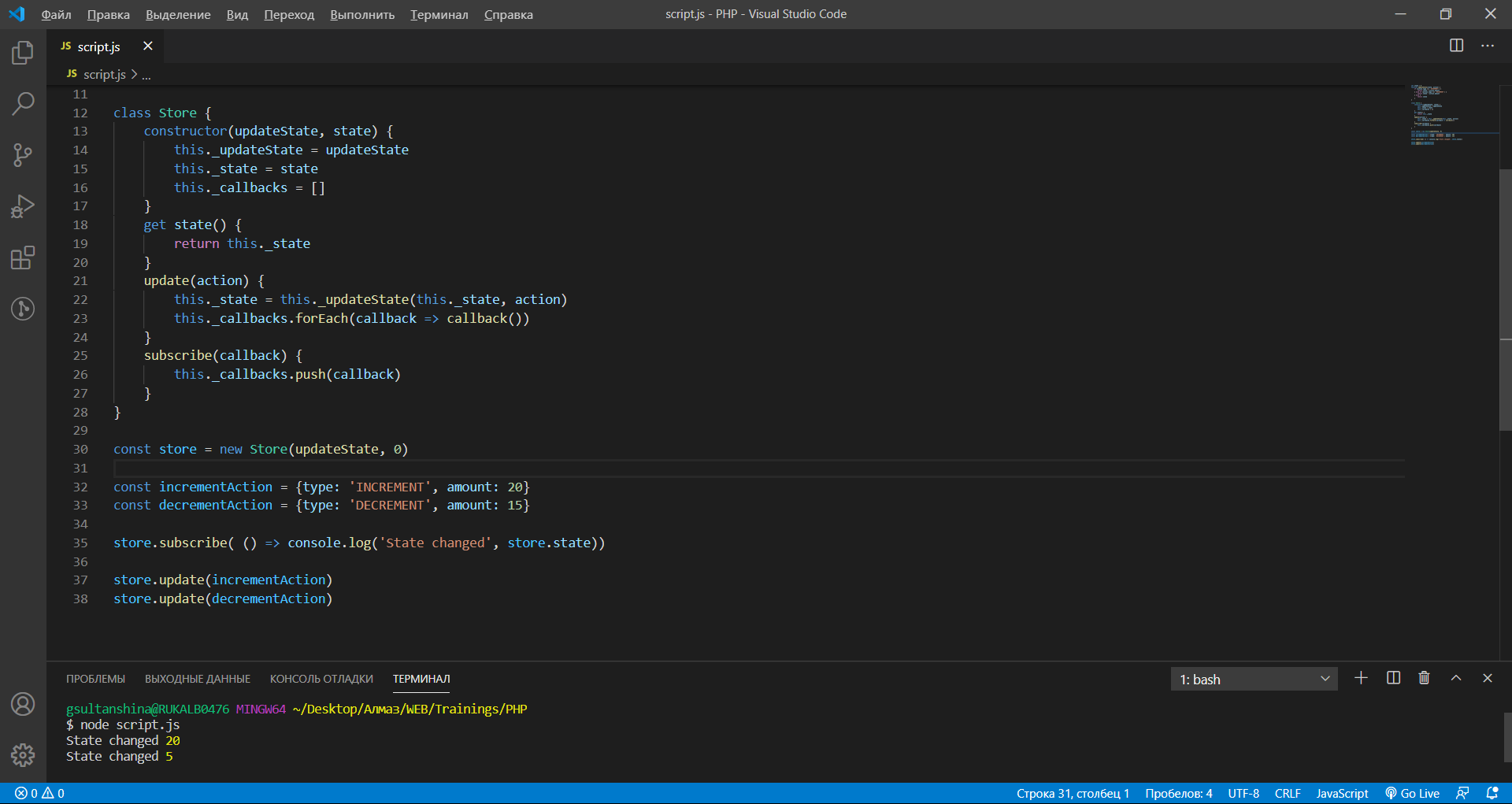
Вынесем объекты в переменную, чтобы их было проще передавать  


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

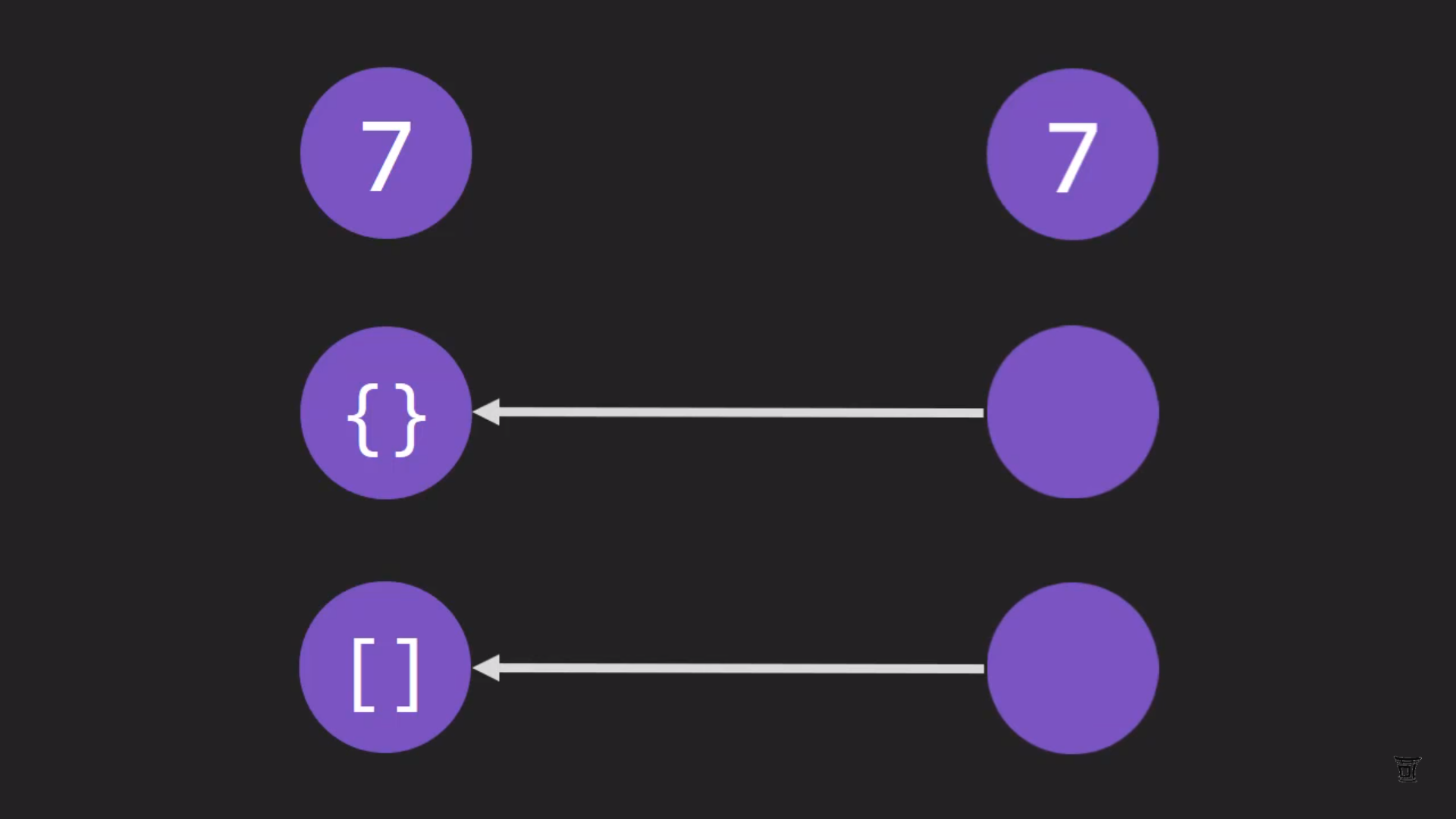


Смысл хранилища заключается не в том, чтобы полностью спрятать состояние, а в том, чтобы ограничить к нему доступ через определенный интерфейc. Также было бы удобно, если при изменении состояния хранилище сообщало об этом всем заинтересованным

**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



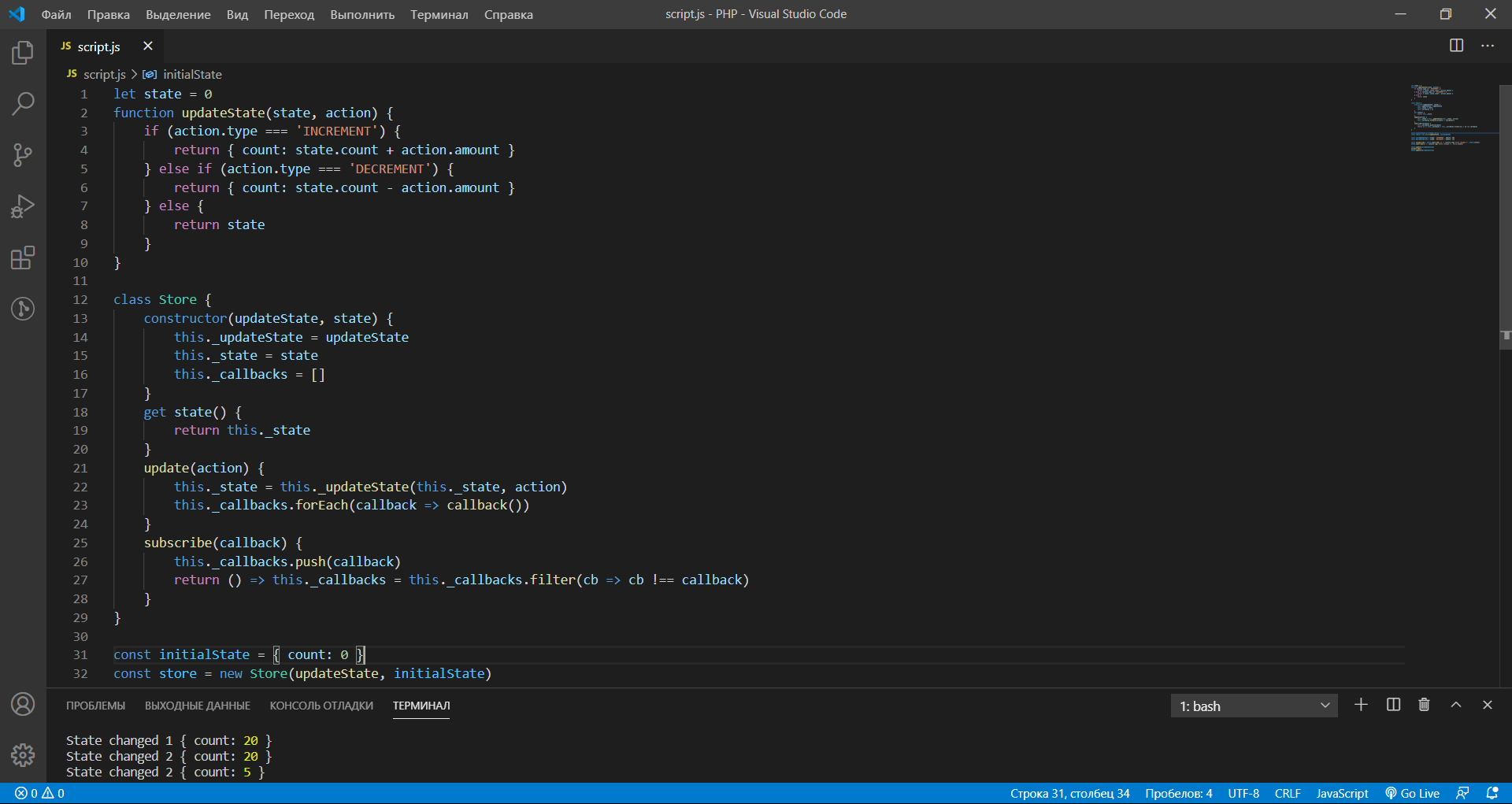
**Состояние**Главными двумя составляющими хранилища являются состояние и функции, которые его изменяют. В данный момент состоянием является простое число. В реальном проекте скорее всего понадобится использовать объект или массив

Примитивные данные передаются копированием значения, а объекты копированием ссылки

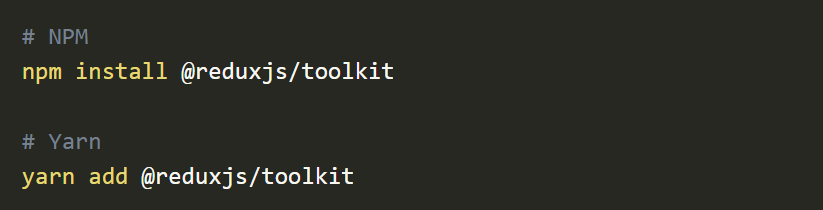
Таким образом, получив объект в аргументе функции, мы будем изменять оригинальный объект

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

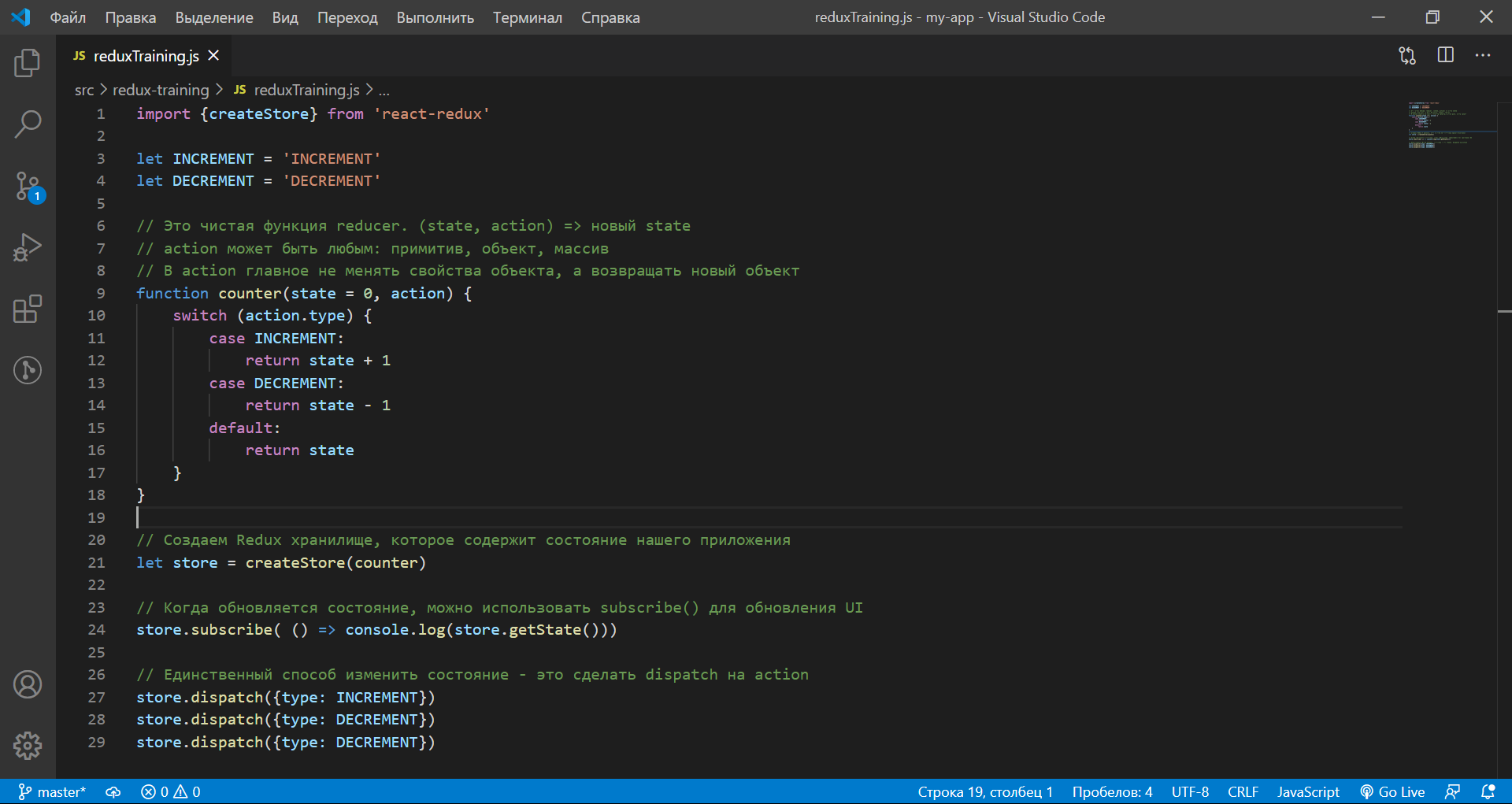
Для решения проблем управления состоянием JS использует функциональное программирование:  
- чистые функции   
- неизменность данных

  
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

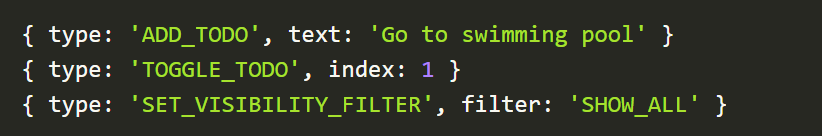
# **08.08.2020 2 часа Отчет: Redux – Introduction: Motivation**

**Motivation**С развитием сложных требований для SPA наш код должен управлять state больше, чем раньше  
State может содержать в себе разные данные: ответы от сервера, локальные данные

Управлять таким постоянно-изменяющим state становится сложно. Особенно когда одна модель обновляет другую модель и зависит еще от одной – мы теряем понимание, как работает наше приложение, теряем контроль над state

Для того, чтобы справиться с такими сложностями, придумали Redux

**Core concepts**Представим, что у нас есть такой state. В нем нет сеттеров. Это делается для того, чтобы различные части кода не могли произвольно менять состояние, вызывая ошибки

To change the state we need to dispatch an action. Action – обычный объект, который описывает, что произошло  


При обеспечении того, чтобы каждое изменение описывалось как действие, мы имеем четкое представление о том, что происходит в нашем приложении

Для того, чтобы связать state и action, мы пишем функцию reducer. Данная функция принимает state и action в качестве аргументов, возвращает новый state. Трудно написать функцию reducer для всего приложения, поэтому, чтобы управлять состоянием, мы пишем мелкие функции, которые управляют частями состояния

**10.08.2020  
4 часа Redux + 2 часа установка проекта  
Отчет: Redux Introduction, Redux Essential Part 1 – Part 2**

**3 принципа Redux:  
Единственный источник истины**Глобальный state приложения хранится в дереве объектов в пределах одного store  
Это позволяет легко создавать универсальные приложения, так как состояние с сервера может быть передано в клиент без дополнительного кода. Также дерево cостояния позволяет с легкостью инспектировать и дебажить приложение

**State только для чтения**Единственный способ изменить state – запустить action (объект, описывающий что произошло)  
Это гарантирует, что UI и колбэки никогда не с могут менять state. Вместо этого они выражают намерение преобразовать state. Action – простые объекты, которые можно регистрировать, хранить, воспроизводить для отладки или тестирования

**Изменения в state делаются только с помощью чистых функций**Чтобы указать, как дерево state может преобразовываться, мы пишем чистые reducer  
Reducer – это чистая функция, которая принимает предыдущий state и action, возвращает новый state. Изменять state можно начать с помощью одного reducer, но по мере роста приложения можно разбить его на более мелкие reducer, управляющие определенными частями дерева state

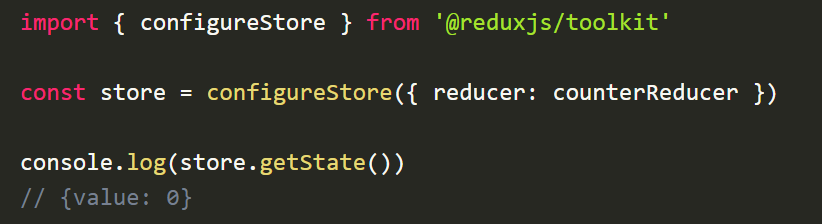
**Prior Art**Redux имеет смешанное наследие. Он похож на некоторые модели и технологии, но также отличается от них в важных отношениях

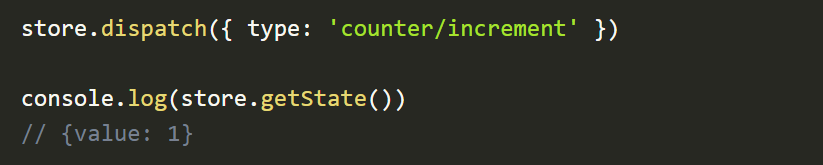
**Flux**   
Redux был создан на основе логики Flux. Redux, как и Flux, предписывает использовать логику обновления модели на определенном уровне приложения (stores в Flux = reducers в Redux)  
Вместо того, чтобы напрямую изменять состояние, Flux и Redux призывают описывать каждое возможное изменение в объекте Action

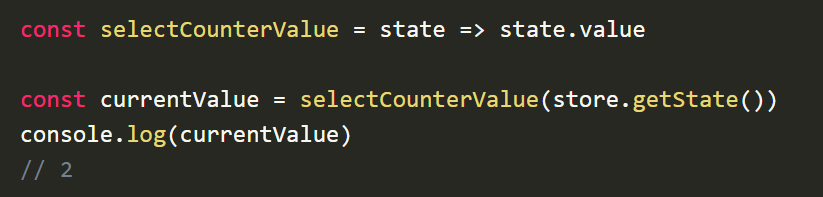
В отличие от Flux у Redux нет такого понятия, как Dispatcher. Это происходит, потому что Redux работает с помощью чистых функций вместо эмиттеров событий. Чистые функции легко создавать, они не нуждаются в дополнительной сущности. Для того, чтобы изменить состояние в Flux нужно придерживаться такой же схемы, как в Redux: (state, action) => state

Еще одно важное отличие от Flux заключается в том, что Redux предполагает, что вы никогда не мутируете свои данные. Вы можете использовать простые объекты и массивы для вашего состояния просто отлично, но мутировать их внутри редукторов категорически не рекомендуется. Вы всегда должны возвращать новый объект, что легко сделать с предложением оператора распространения объекта или с библиотекой типа Immutable

**Redux essentials  
Часть 1: Обзор Redux и основные концепции  
*Почему функции в Redux назвали reducers?***В JS существует метод массива reducer(), который принимает в себя два параметра, возвращает новый параметр на основании этих двух

**Store**State, который мы используем в Redux, живет в объекте store  
Store создается с помощью передачи в reducer, имеет метод getState, который возвращает state  


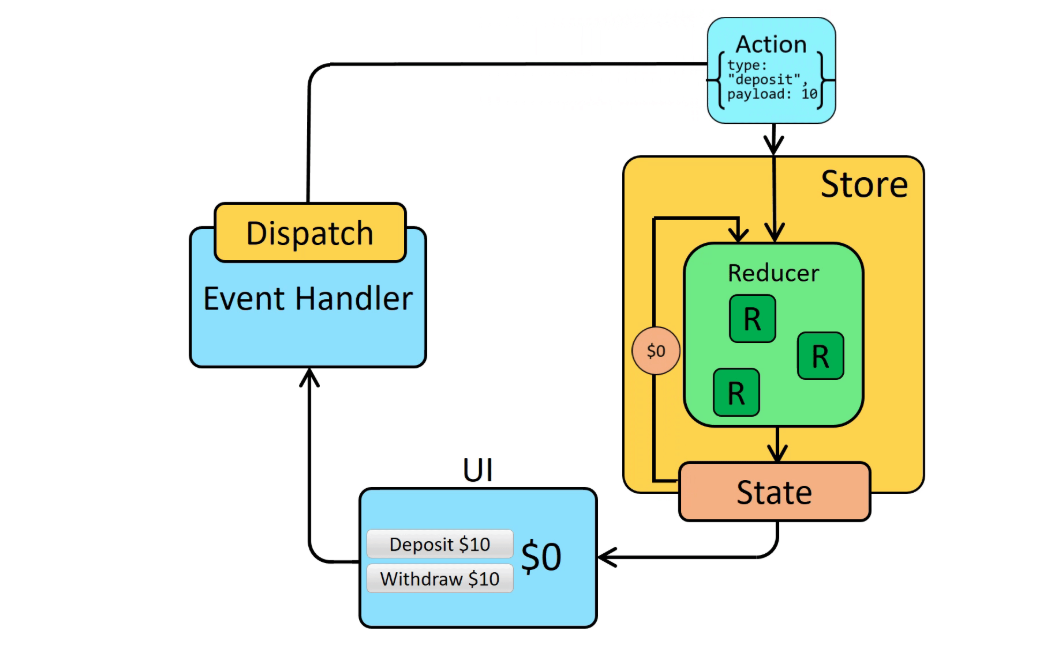
**Dispatch**У store есть метод, который называется dispatch. Единственный способ обновить состояние – вызвать store.dispatch и передать в action. Хранилище запустит свою функцию reducer и сохранит новое состояние внутри. Для того, чтобы получить обновленное состояние вызовем getState()

**Selectors**Селекторы – функции, которые знают, как извлечь определенные фрагменты информации из значения состояния хранилища. По мере роста приложения это может помочь избежать повторения логики

**Redux Application Data Flow­**Ранее мы говорили об «одностороннем» потоке данных, который имеет эту последовательность:  
- State описывает состояние приложения в определенный момент времени  
- UI рендерится на основании этого state  
- Когда что-то происходит (пр: нажатие на кнопку), состояние обновляется на основе того, что произошло  
- UI повторно рендерится на основе нового состояния

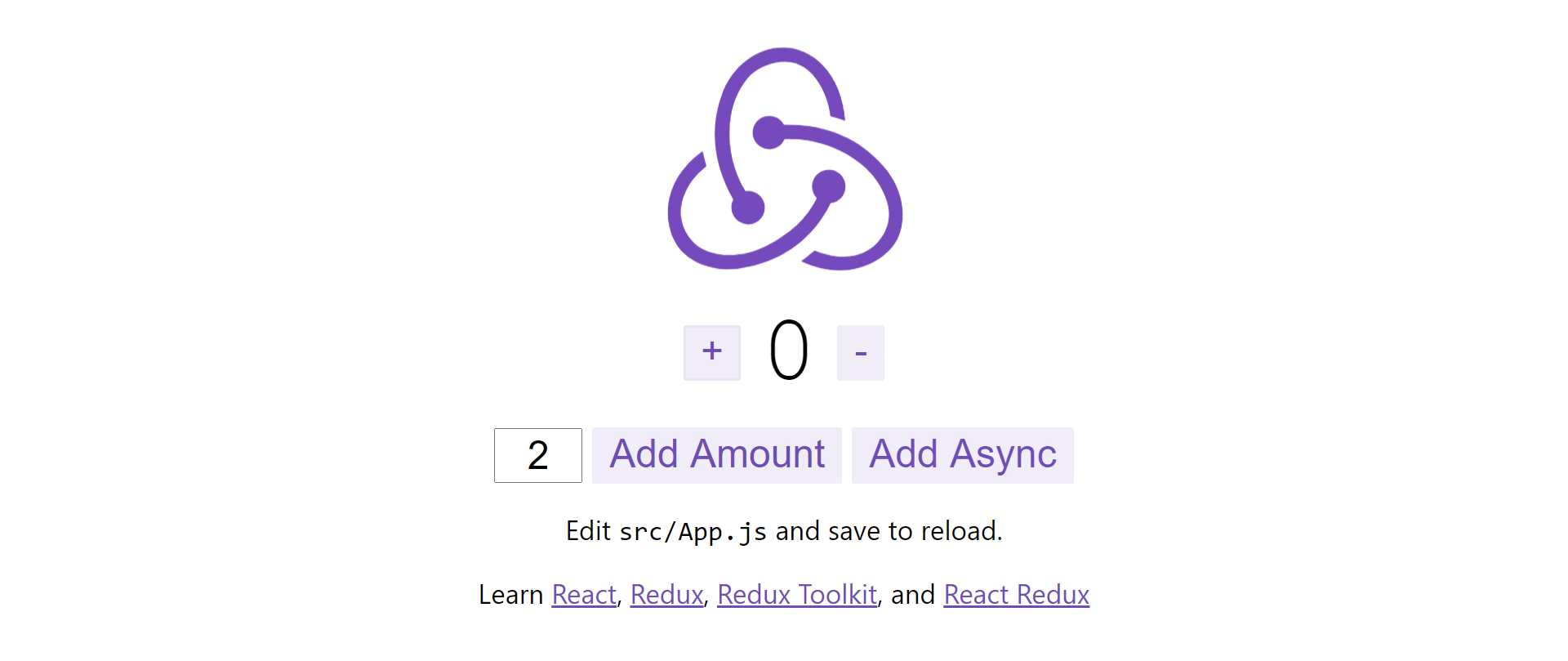
**Конкретно для Redux мы можем рабить эту последовательность более детально:  
Первоначальная настройка:**  
- Хранилище Redux создается с помощью функции reducer  
- Хранилище вызывает reducer один раз и сохраняет возвращаемое значение в исходном state  
- Когда UI впервые отрендерился, UI-компоненты получают доступ к state хранилища Redux. Они также подписываются на любые будущие обновления store, чтобы знать о изменениях state

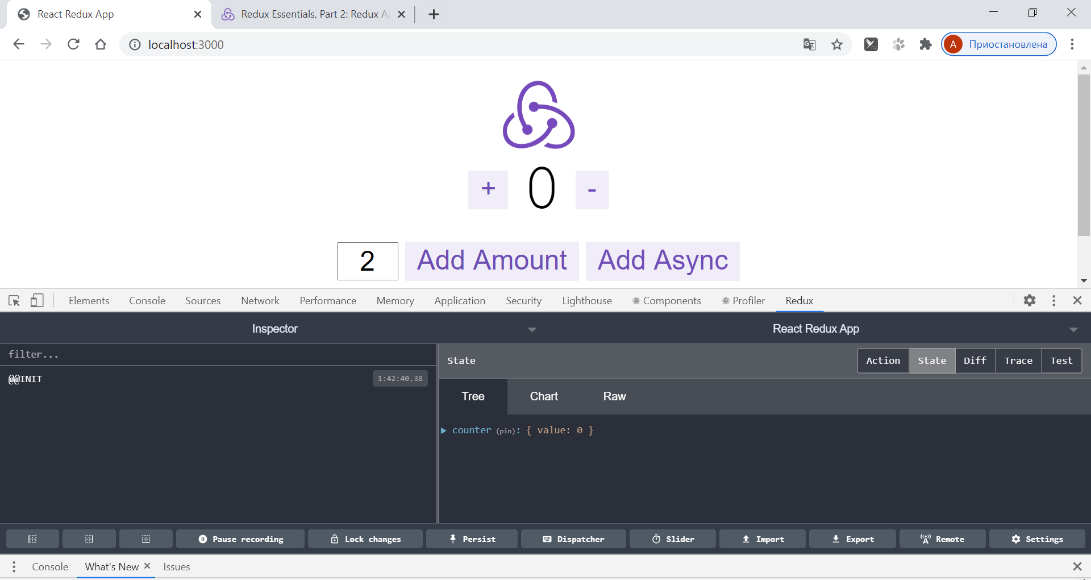
**Обновления:**  
- Что-то происходит в приложении, например пользователь нажимает на кнопку  
- Код приложения диспатчит объект action в хранилище Redux  
- Хранилище запускает функцию reducer(state, action) => action  
- Хранилище оповещает все части UI об обновлении store, которые были подписаны на store  
- Каждый UI-компонент, который нуждается в данных из store, проверяет не изменились ли необходимые ему части состояния  
- Каждый компонент, который видит, что его данные изменились, заставляет повторно отрендериться с новыми данными

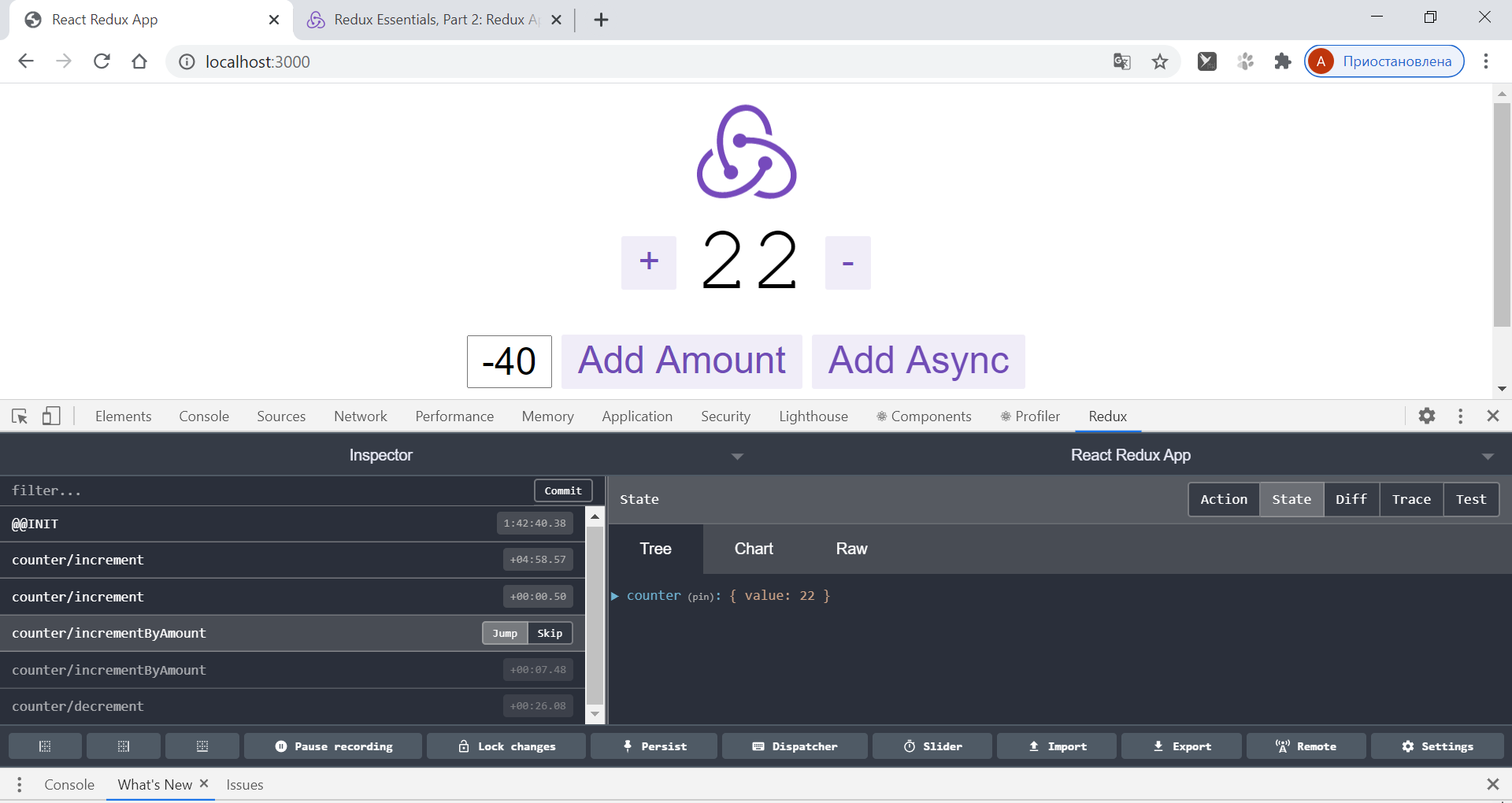
Выглядит это примерно так: 

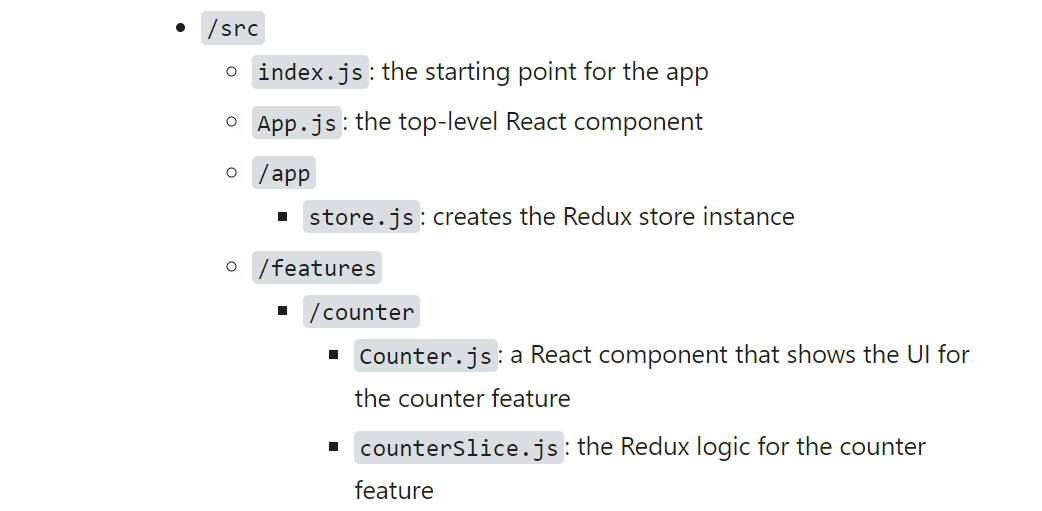
**Часть 2. Структура приложения Redux**  
В этой части мы пройдем:  
- Структуру типичного приложения React + Redux  
- Как посмотреть изменения состояния Redux в инструментах разработчика

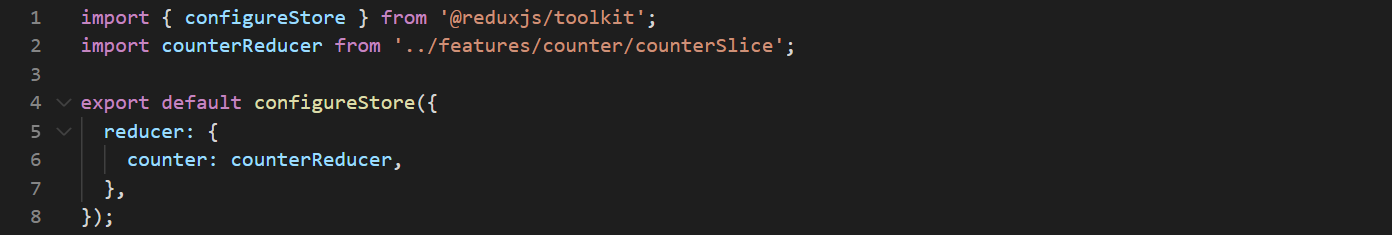
**Создание счетчика на React + Redux:**Воспользуемся официальным шаблоном для React + Redux приложений с помощью команды:  
npx create-react-app redux-essentials-example --template redux

Запускаем приложение с помощью команды: npm start, и получаем:  


Далее, откроем панель разработчика, перейдем во вкладку Redux:

Тестируя приложения, замечаем, что состояние все время меняется, причем история всех наших действий видна в левом нижнем столбце. Нажимая на них, мы «путешествуем во времени»  


На рисунке ниже можно увидеть, как устроено данное приложение по файлам: 

Теперь рассмотрим, как устроено хранилище Redux в данном приложении:  
**Создание хранилища Redux:**  


Хранилище Redux было создано с помощью функции configureStore из ReduxToolkit  
configureStore требует, чтобы мы передали аргумент в reducer

Наше приложение может состоять из множества различных функций, и каждая из этих функций может иметь свою собственную функцию reducer. Когда мы вызываем configureStore, мы можем передавать все различные reducer в объекте. Имена ключей в объекте будут определять ключи в нашем конечном значении

У нас есть файл который имеет адрес features/counter/counterSlice, который экспортирует функцию reducer для счета. Мы можем импортировать эту функцию сюда и включить ее при создании логики нашего приложения

Когда мы передаем объект типа {counter: counterReducer}, это говорит о том, что мы хотим иметь секцию state.counter нашего объекта Redux state, и что мы хотим, чтобы функция counterReducer отвечала за принятие решения о том, следует ли и как обновить состояние.встречный раздел всякий раз, когда отправляется действие