# Работа с Redux

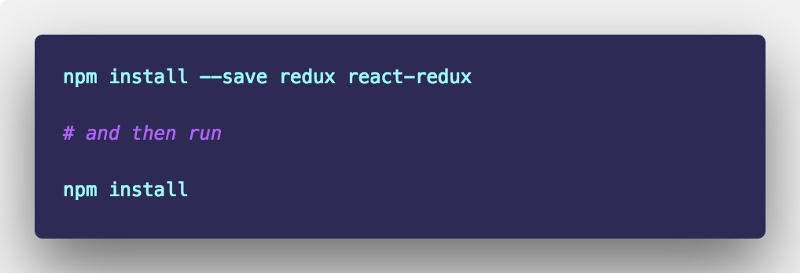
### Цель работы

Интегрировать в App редюсер (reducer) — чистую функцию, принимающую предыдущее состояние и экшен (state и action) и возвращающую следующее состояние (новую версию предыдущего).

### Задания для выполнения

1. Реализуйте стандартный секундомер.
2. Создайте компонент Reducer, который будет вычислять следующее состояние на основе начального или предыдущего состояния.

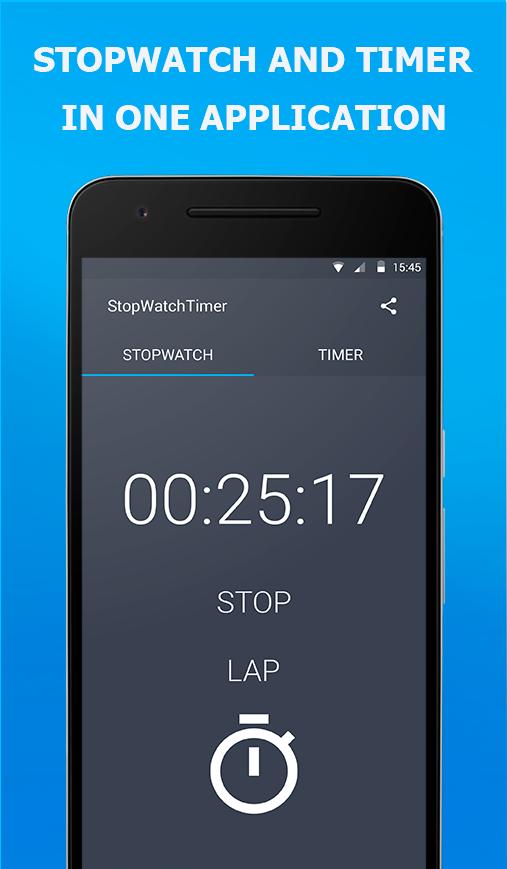
Команды для установки библиотеки среды:



Воспользуйтесь следующим кодом:



1. Добавьте стили для секундомера:

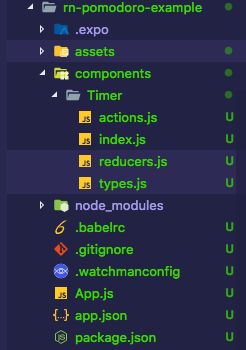


1. Поделитесь ссылкой на проект в Expo. Загрузить созданное приложение на GitHub в репозиторий Student, используя формат в названии Фамилия (латинскими буквами)\_22.

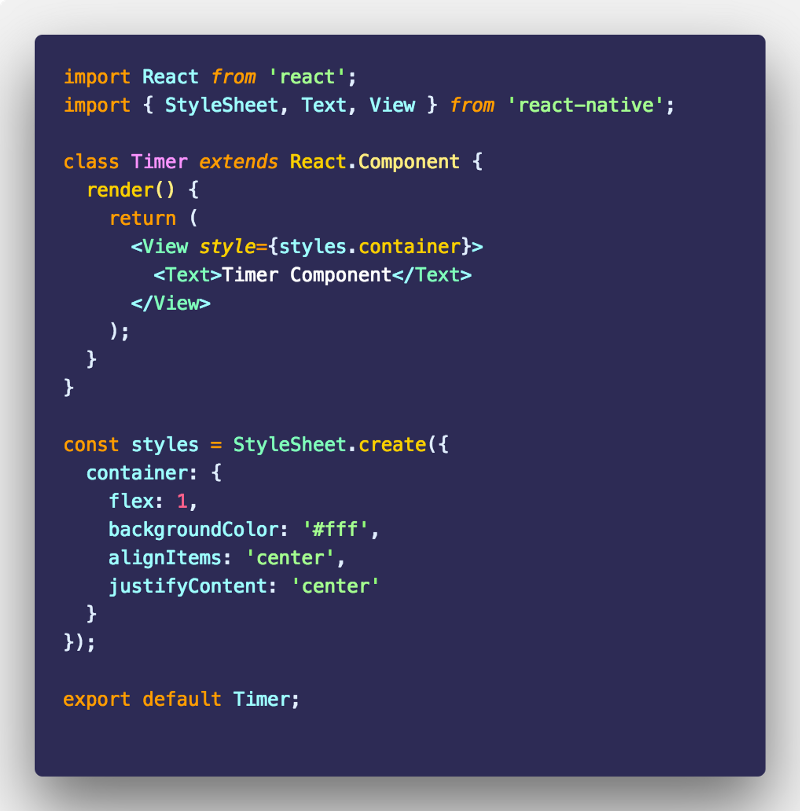
**Методические рекомендации**

Redux является важной частью экосистемы React Native. Если ваш мир вращается вокруг JavaScript, вы, вероятно, слышали о Redux. Прежде чем читать остальную часть руководства и идти дальше, просто постарайтесь вспомнить, что вы изучаете Redux только потому, что это упростит вам задачу, а не усложнит ее. Теперь давайте узнаем, зачем вам нужен Redux в вашем приложении.

Создайте следующие файлы и папки внутри components каталога.



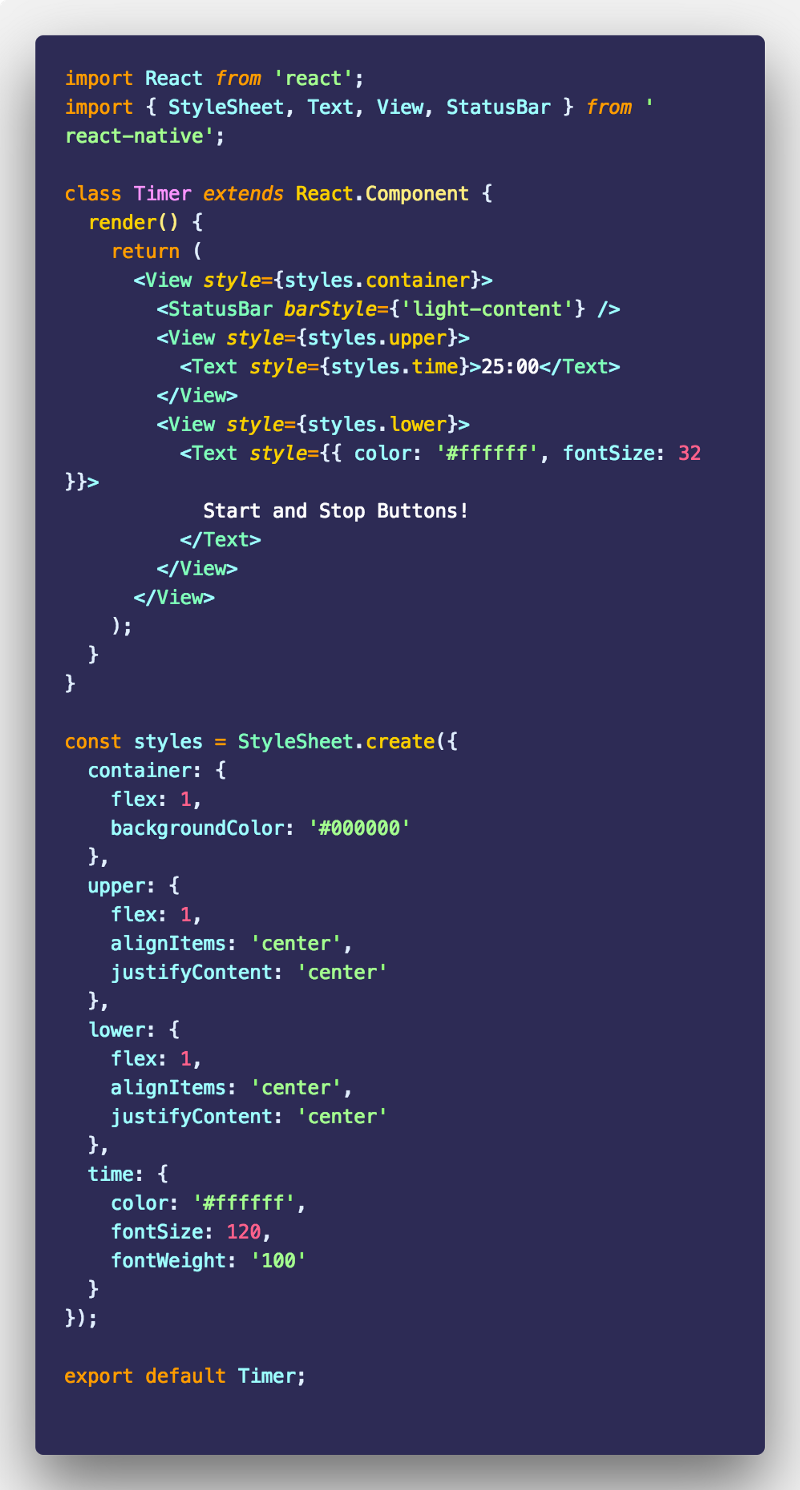
Во-первых, мы создадим немой компонент Таймера и подключим его к App.jsнему. Добавьте следующий код вTimer/index.js:



Затем измените App.js файл:



Теперь мы создадим компонент статического таймера, чтобы посмотреть, как все вписывается. Мы начнем с изменения StatusBar. Затем мы определяем два Textэлемента из react-nativeбиблиотеки, чтобы указать, где будет отображаться фактический таймер и где будут отображаться кнопки для запуска и остановки таймера. На данный момент оба являются текстовыми полями.



Теперь мы собираемся заменить отображаемый раздел Start and Stop Buttons настоящими кнопками. Мы будем использовать TouchableOpactiy, чтобы заставить это работать. TouchableOpacity компонент действует как оболочка для того, чтобы представления правильно реагировали на прикосновения. Непрозрачность обернутого представления (или кнопки в нашем случае) уменьшается всякий раз, когда пользователь прикасается к ней.

Мы создаем повторно используемый компонент, так как нам нужны две кнопки: Start и Stop.

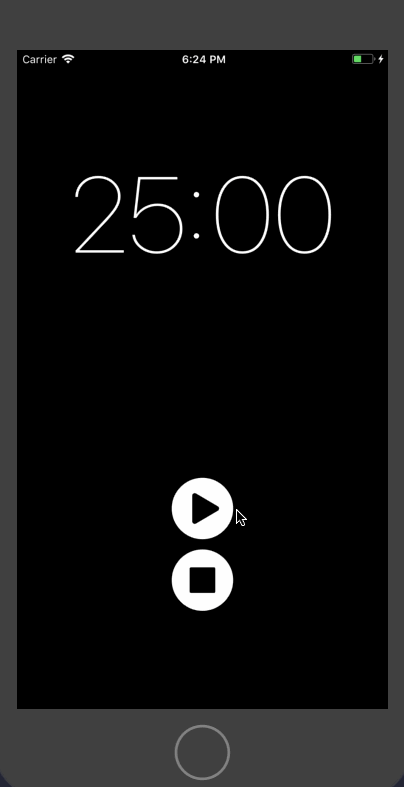


Это компонент без состояния, поэтому у него нет класса — он нужен нам только для представления кнопки в пользовательском интерфейсе нашего приложения. Мы также импортируем иконки FontAwesome из @expo/vector-icons, который является ответвлением от react-native-vector-icons и поставляется непосредственно с expo SDK. Нет необходимости устанавливать его как отдельную зависимость. Чтобы отобразить значок, нам нужно определить его size.

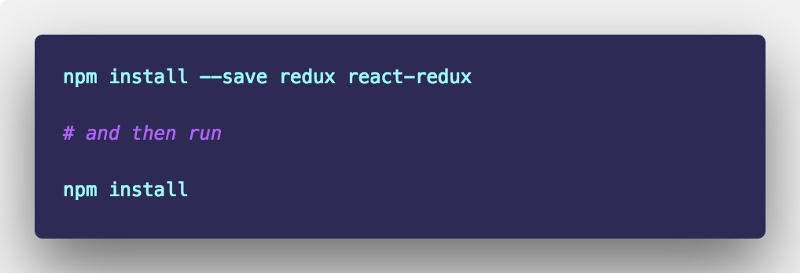
Наконец, в приведенном выше компоненте без состояния мы определяем propTypes. Я буду обсуждать, как и почему мы должны использовать PropTypes в приложении React Native в другой статье.

В мобильном приложении события запускаются прикосновением. Для обработки этих событий мы собираемся использовать onPress. Здесь у нас будет только два события: Start и Stop. Обе кнопки в нашем приложении будут использовать onPressOutто, что отличается от onPress. Вызывается onPressOutвсякий раз, когда пользователь отпускает касание (когда пользователь перестает нажимать кнопку). Он вызывается раньше onPressи является более точным в ситуации, подобной нашей, когда нам нужно запустить или остановить таймер нажатием кнопки, как только пользователь закончит.

Теперь нам потребуется этот Buttonкомпонент в нашем компоненте Timer.



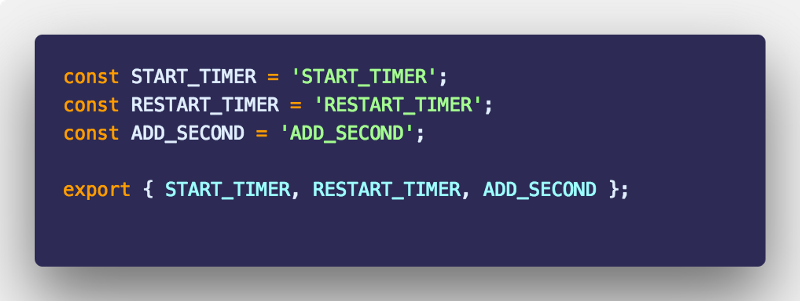
Пока что наше приложение Timer не делает ничего, кроме отображения минимального пользовательского интерфейса. Чтобы заставить его работать, мы начинаем с добавления некоторых необходимых зависимостей Redux.



Теперь давайте начнем интегрировать Redux в наше приложение.

В Redux состояние всего приложения представлено одним объектом JavaScript. Считайте этот объект доступным только для чтения, поскольку мы не можем напрямую вносить изменения в это состояние (которое представлено в виде дерева). Нам нужно actionsэто сделать.

Действия подобны событиям в Redux. Они могут запускаться в виде щелчков мыши, нажатий клавиш, таймеров или сетевых запросов. Природа каждого упомянутого события изменчива. Действие - это объект JavaScript. Чтобы определить действие, существует одно требование: каждое действие должно иметь свое собственное свойство type. Мы определяем эти типы в файле под названиемtypes.js:



Пока нашему приложению требуется всего три действия. Тип любого действия является строковым значением и определяется как константа.

В файле actions.jsмы потребуем, чтобы эти типы определяли создателей действий. Создатели действий - это функции, которые создают действия.



Получатель действия известен как редуктор. Всякий раз, когда запускается действие, состояние приложения меняется. Обработка состояния приложения осуществляется редукторами.

Редуктор - это чистая функция, которая вычисляет следующее состояние на основе начального или предыдущего состояния. Он всегда выдает один и тот же результат, если состояние не изменилось. Для этого требуется два ввода, а состояние и действие должны возвращать состояние по умолчанию.



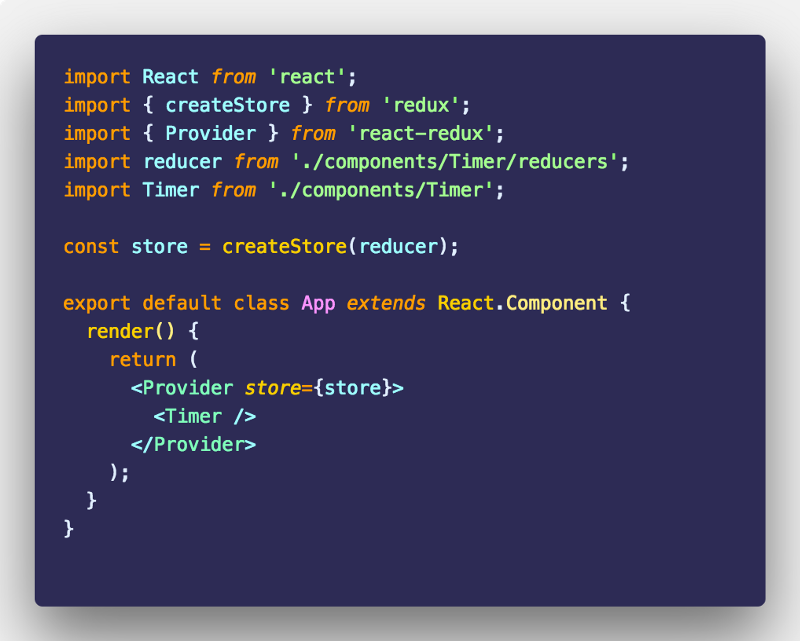
В нашем начальном состоянии мы определяем три атрибута: isPlaying, elapsedTimeи timerDuration. В настоящее время таймер имеет значение по умолчанию 6 (секунды) для целей тестирования, но фактическое значение, которое мы собираемся изменить позже, равно 25(или 1500 секунд).

Затем есть три вспомогательные функции:

* applyStartTimer запустится таймер
* applyRestartTimer остановит функцию таймера и установит все по умолчанию
* и, наконец, applyAddSecondпроверит, не меньше ли пройденное время, чем общая продолжительность таймера. Если это так, то это добавит еще одну секунду, чтобы увеличить его ценность. Если нет, то он вернет состояние по умолчанию и остановит выполнение функции таймера.

После этого мы определяем нашу функцию редуктора и экспортируем ту же функцию. Обратите внимание, как организована функция редуктора. Этому образцу следуют большинство членов сообщества, которых я видел в Интернете.

С помощью редуктора и начального состояния мы можем создать объект хранилища.

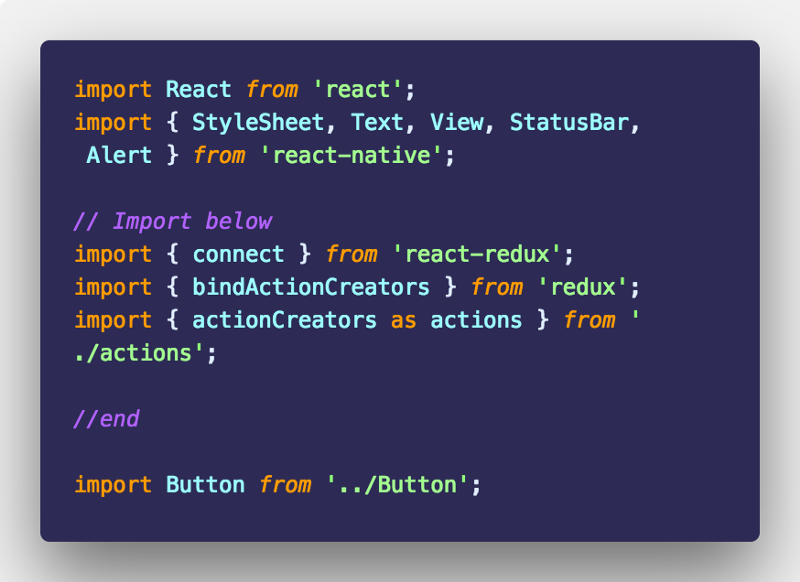


Хранилище - это объект, который объединяет действия и редукторы. Он обеспечивает и сохраняет состояние на уровне приложения, а не отдельных компонентов. Redux - это не самоуверенная библиотека с точки зрения того, какой фреймворк или библиотека должны ее использовать или нет.

Чтобы связать приложение React или React Native с Redux, вы делаете это с react-reduxпомощью module . Это делается с помощью компонента high ordered Provider. Он в основном передает хранилище остальной части приложения.

Нам нужно связать создателей действий с нашей функцией таймера, чтобы сделать ее полностью функциональной (чтобы она реагировала на осязаемые события или запуск или перезапуск таймера). Мы сделаем это в Timer/index.jsфункции.

Сначала мы импортируем необходимые зависимости для привязки создателей действий.

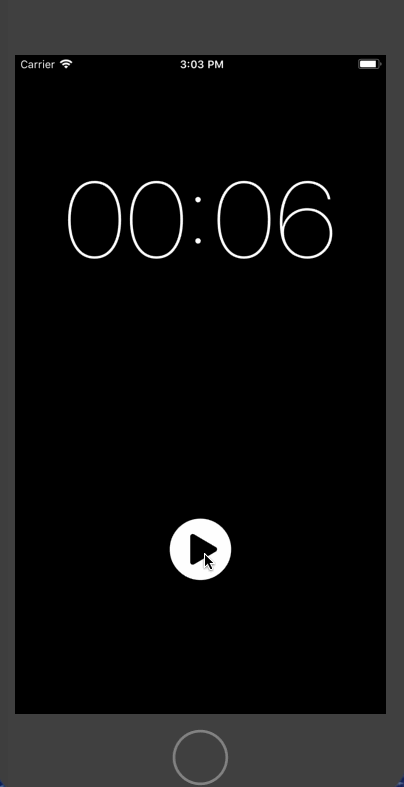




mapStateToProps это объект, который находится в хранилище, ключи которого передаются компоненту в качестве реквизита. Ниже приведен полный код компонента Timer.



Я создал пользовательскую функцию, вызываемую formatTimeдля отображения времени в правильном формате, но вы можете использовать любую библиотеку таймеров. Затем, чтобы увеличить значение времени, я использую метод жизненного цикла реакции componentWillReceiveProps. Я знаю, что скоро он устареет, но пока он работает. Смотрите наше мини-приложение в действии ниже:



Для краткости и в этой демонстрации я использую только секунды для отображения таймера. Вы можете увеличить значение таймера, отредактировав значение константы TIMER\_DURATIONв reducers.js.

**Полезные ссылки**

<https://www.freecodecamp.org/news/how-to-integrate-redux-into-your-application-with-react-native-and-expo-ec37c9ca6033/>

<https://enappd.com/blog/redux-in-react-native-app/92/>