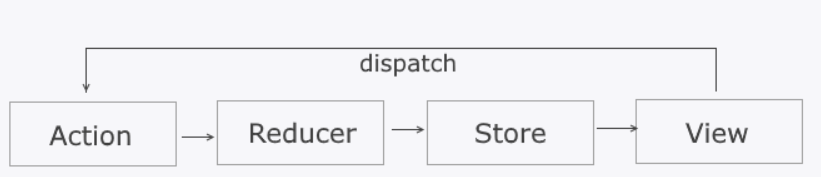
Redux представляет собой контейнер для управления состоянием приложения.

Ключевые моменты Redux:

* Хранилище (store): хранит состояние приложения
* Действия (actions): некоторый набор информации, который исходит от приложения к хранилищу и который указывает, что именно нужно сделать. Для передачи этой информации у хранилища вызывается метод dispatch().
* Создатели действий (action creators): функции, которые создают действия
* Reducer: функция (или несколько функций), которая получает действие и в соответствии с этим действием изменяет состояние хранилища

Общую схему взаимодействия элементов архитектуры Redux можно выразить следующим образом:



По лабе: показать actions, что там происходит

*let* nextTodoId : *number* = 0;

Эта переменная будет использоваться для генерации уникальных идентификаторов для каждой задачи.

Затем определены три интерфейса: **AddTodoAction**, **SetVisibilityFilterAction** и **ToggleTodoAction**. Эти интерфейсы представляют различные типы действий, которые могут быть выполнены в приложении.

Функция **addTodo** является action creator'ом (создателем действия) для добавления новой задачи.

Функция **setVisibilityFilter** также является action creator'ом и используется для установки фильтра видимости задач.

Функция **toggleTodo** представляет action creator для переключения состояния задачи.

Кроме того, экспортируется константа **VisibilityFilters**, которая содержит различные фильтры для видимости задач. Они определены как объект со следующими свойствами: **SHOW\_ALL**, **SHOW\_COMPLETED**, **SHOW\_ACTIVE** и **SHOW\_NONE**. Эти значения представляют различные варианты фильтрации задач в списке.

Потом про редьюсеры

**Редьюсеры** (Reducers) определяют, как состояние приложения изменяется в ответ на экшены, отправленные в стор. Помните, что экшены только описывают, что произошло, но не описывают, как изменяется состояние приложения.

Index –

В представленном коде происходит определение и экспорт корневого редьюсера Redux. Давайте рассмотрим основные моменты этого кода:

1. Импорты:
   * **combineReducers** импортируется из библиотеки Redux. Эта функция используется для объединения нескольких редьюсеров в один.
   * **todos** импортируется из файла **'./todos'**. Этот файл содержит редьюсер и типы данных для управления состоянием задач (todos).
   * **visibilityFilter** импортируется из файла **'./visibilityFilter'**. Этот файл содержит редьюсер для управления фильтром видимости задач.
2. Определение интерфейса:
   * **RootState** определяет интерфейс состояния корневого редьюсера. В данном случае, состояние состоит из массива задач (todos) и строки для фильтра видимости.
3. Комбинирование редьюсеров:
   * **rootReducer** объединяет редьюсеры **todos** и **visibilityFilter** с помощью функции **combineReducers**. Редьюсер **todos** будет отвечать за обработку действий, связанных с задачами, а **visibilityFilter** - за обработку действий, связанных с фильтром видимости.
4. Экспорт:
   * **export default rootReducer** экспортирует корневой редьюсер, который будет использоваться при создании Redux-хранилища. Этот редьюсер будет управлять всем состоянием приложения, объединяя состояния, управляемые редьюсерами **todos** и **visibilityFilter**.

Таким образом, данный код определяет и экспортирует корневой редьюсер, который объединяет редьюсеры для управления состоянием задач и фильтра видимости в Redux-приложении.

Todos –

В представленном коде определен редьюсер **todos**, который отвечает за обработку действий, связанных с задачами. Давайте рассмотрим его основные моменты:

1. Импорты:
   * Отсутствуют импорты в этом коде.
2. Определение типов:
   * **Todo** - интерфейс, который определяет структуру объекта задачи. Каждая задача имеет поля **id** (числовой идентификатор), **text** (текст задачи) и **completed** (флаг, указывающий, завершена ли задача).
   * **Action** - объединение типов действий, которые могут быть выполнены над задачами. В данном случае, допустимы два типа действий: **ADD\_TODO** (добавление новой задачи) и **TOGGLE\_TODO** (переключение статуса задачи).
3. Редьюсер **todos**:
   * **todos** - это функция-редьюсер, которая принимает два аргумента: **state** (состояние, по умолчанию пустой массив задач) и **action** (действие, которое нужно обработать).
   * Внутри функции используется оператор **switch** для обработки различных типов действий.
   * При действии типа **ADD\_TODO**, новая задача добавляется в состояние с помощью распыления оператора (**...state**). Новая задача создается объектом с полями **id**, **text** и **completed** (установленным в **false**).
   * При действии типа **TOGGLE\_TODO**, состояние обновляется с помощью метода **map**. Если **id** задачи соответствует **id** из действия, меняется значение поля **completed** на противоположное. В противном случае, возвращается исходная задача без изменений.
   * Если тип действия не совпадает ни с одним из указанных в **switch**, возвращается текущее состояние без изменений.
4. Экспорт:
   * **export default todos** экспортирует редьюсер **todos**. Это позволяет другим модулям импортировать и использовать этот редьюсер при создании Redux-хранилища.

Таким образом, данный код определяет редьюсер **todos**, который обрабатывает действия, связанные с задачами, и возвращает обновленное состояние на основе этих действий.

visibilityFilter –

В представленном коде определен редьюсер **visibilityFilter**, который отвечает за обработку действий, связанных с фильтром видимости задач. Давайте рассмотрим его основные моменты:

1. Импорты:
   * **VisibilityFilters** импортируется из **'../actions'**. Этот файл (который не представлен в данном фрагменте кода) содержит определение констант для фильтров видимости задач.
2. Определение типов:
   * **ActionFilter** - тип действия, которое может быть выполнено над фильтром видимости. В данном случае, это действие **SET\_VISIBILITY\_FILTER**, которое устанавливает новое значение фильтра.
3. Редьюсер **visibilityFilter**:
   * **visibilityFilter** - это функция-редьюсер, которая принимает два аргумента: **state** (состояние, по умолчанию установлено в **VisibilityFilters.SHOW\_ALL**) и **action** (действие, которое нужно обработать).
   * Внутри функции используется оператор **switch** для обработки типа действия.
   * При действии типа **SET\_VISIBILITY\_FILTER**, состояние обновляется и возвращается новое значение фильтра **action.filter**.
   * Если тип действия не совпадает ни с одним из указанных в **switch**, возвращается текущее состояние без изменений.
4. Экспорт:
   * **export default visibilityFilter** экспортирует редьюсер **visibilityFilter**. Это позволяет другим модулям импортировать и использовать этот редьюсер при создании Redux-хранилища.

Таким образом, данный код определяет редьюсер **visibilityFilter**, который обрабатывает действия, связанные с фильтром видимости задач, и возвращает обновленное состояние на основе этих действий. Значение фильтра устанавливается в соответствии с переданным действием типа **SET\_VISIBILITY\_FILTER**.

Начать говорить про компоненты, какие бывают (функциональные и классовые)

App –

В представленном коде определен компонент **App**, который является корневым компонентом React-приложения. Давайте рассмотрим его основные моменты:

1. Импорты:
   * **React** импортируется из библиотеки 'react'. Этот модуль предоставляет необходимые функции и классы для работы с React.
   * **Footer** импортируется из файла './Footer'. Этот файл содержит компонент, отображающий футер приложения.
   * **AddTodo** и **VisibleTodoList** импортируются из контейнеров '../containers/AddTodo' и '../containers/VisibleTodoList' соответственно. Контейнеры являются компонентами, связанными с Redux, и предоставляют доступ к состоянию и действиям через Redux-стор.
2. Компонент **App**:
   * **App** - это функциональный компонент React, определенный с помощью **React.FC** (тип-параметр, указывающий типы свойств компонента).
   * Внутри компонента **App** определена разметка JSX, которая содержит следующие компоненты:
     + **<AddTodo />** - отображает компонент **AddTodo**, который отвечает за добавление новых задач.
     + **<VisibleTodoList />** - отображает компонент **VisibleTodoList**, который отображает список задач, учитывая текущий фильтр видимости.
     + **<Footer />** - отображает компонент **Footer**, который отображает информацию о приложении внизу страницы.
3. Экспорт:
   * **export default App** экспортирует компонент **App**, чтобы он мог быть использован в других модулях.

Таким образом, данный код определяет компонент **App**, который является корневым компонентом React-приложения. Внутри него определена разметка JSX, которая включает компоненты **AddTodo**, **VisibleTodoList** и **Footer**, используемые для отображения функциональности добавления задач, списка задач и футера приложения соответственно.

Link –

В представленном коде определен компонент **Link**, который представляет собой кнопку-ссылку в React-приложении. Давайте рассмотрим его основные моменты:

1. Импорты:
   * **React** импортируется из библиотеки 'react'. Этот модуль предоставляет необходимые функции и классы для работы с React.
2. Типы:
   * **Props** - тип, который определяет свойства компонента **Link**. В данном случае, свойства включают **active** (логическое значение, указывающее, активна ли ссылка), **children** (дочерние элементы, которые будут отображаться внутри кнопки-ссылки) и **onClick** (функция обратного вызова, которая будет вызываться при клике на кнопку-ссылку).
3. Компонент **Link**:
   * **Link** - это функциональный компонент React, определенный с помощью **React.FC** (тип-параметр, указывающий типы свойств компонента).
   * Внутри компонента **Link** определена разметка JSX, которая представляет кнопку-ссылку.
   * Внутри кнопки-ссылки:
     + **onClick={onClick}** - устанавливает функцию обратного вызова **onClick** в качестве обработчика события клика на кнопку.
     + **disabled={active}** - устанавливает состояние **disabled** кнопки в зависимости от значения **active**. Если **active** равно **true**, кнопка будет отключена.
     + **style={{ marginLeft: '4px' }}** - устанавливает встроенный стиль для кнопки, задавая отступ слева равным '4px'.
   * Внутри кнопки-ссылки отображаются дочерние элементы (**children**), которые были переданы компоненту.
4. Экспорт:
   * **export default Link** экспортирует компонент **Link**, чтобы он мог быть использован в других модулях.

Таким образом, данный код определяет компонент **Link**, который представляет собой кнопку-ссылку в React-приложении. Компонент принимает свойства **active**, **children** и **onClick** и отображает кнопку с соответствующими значениями и обработчиком события клика.

Todo –

В представленном коде определен компонент **Todo**, который представляет отдельную задачу в списке задач в React-приложении. Давайте рассмотрим его основные моменты:

1. Импорты:
   * **React** и **FC** (Functional Component) импортируются из библиотеки 'react'. **React** предоставляет необходимые функции и классы для работы с React, а **FC** представляет тип функционального компонента.
2. Интерфейс:
   * **Props** - это интерфейс, который определяет свойства компонента **Todo**. В данном случае, свойства включают **onClick** (функция обратного вызова, которая будет вызываться при клике на задачу), **completed** (логическое значение, указывающее, выполнена ли задача) и **text** (текст задачи).
3. Компонент **Todo**:
   * **Todo** - это функциональный компонент React, определенный с помощью **FC<Props>** (тип-параметр, указывающий типы свойств компонента).
   * Внутри компонента **Todo** определена разметка JSX, которая представляет задачу в виде элемента списка (**<li>**).
   * Внутри элемента списка:
     + **onClick={onClick}** - устанавливает функцию обратного вызова **onClick** в качестве обработчика события клика на задачу.
     + **style={{ textDecoration: completed ? 'line-through' : 'none' }}** - устанавливает стиль текста задачи в зависимости от значения **completed**. Если **completed** равно **true**, текст будет зачеркнут.
   * Внутри элемента списка отображается текст задачи (**text**), который был передан компоненту.
4. Экспорт:
   * **export default Todo** экспортирует компонент **Todo**, чтобы он мог быть использован в других модулях.

Таким образом, данный код определяет компонент **Todo**, который представляет отдельную задачу в списке задач в React-приложении. Компонент принимает свойства **onClick**, **completed** и **text** и отображает задачу с соответствующими значениями и обработчиком события клика. В зависимости от значения **completed**, текст задачи может быть зачеркнут или не иметь стиля декорации.

TodoList –

В представленном коде определен компонент **TodoList**, который представляет список задач в React-приложении. Давайте рассмотрим его основные моменты:

1. Импорты:
   * **React** и **FC** (Functional Component) импортируются из библиотеки 'react'. **React** предоставляет необходимые функции и классы для работы с React, а **FC** представляет тип функционального компонента.
   * **PropTypes** импортируется из библиотеки 'prop-types'. **PropTypes** используется для проверки типов свойств компонента.
2. Интерфейс:
   * **TodoItem** - это интерфейс, который определяет структуру отдельной задачи. Он включает **id** (идентификатор задачи), **completed** (логическое значение, указывающее, выполнена ли задача) и **text** (текст задачи).
   * **Props** - это интерфейс, который определяет свойства компонента **TodoList**. В данном случае, свойства включают **todos** (массив задач) и **toggleTodo** (функция обратного вызова для переключения состояния задачи).
3. Компонент **TodoList**:
   * **TodoList** - это функциональный компонент React, определенный с помощью **FC<Props>** (тип-параметр, указывающий типы свойств компонента).
   * Внутри компонента **TodoList** определена разметка JSX, которая представляет список задач в виде неупорядоченного списка (**<ul>**).
   * Внутри списка задач используется метод **.map()** для отображения каждой задачи из массива **todos**.
   * Для каждой задачи создается компонент **Todo**, передавая ему свойства **key**, **...todo** (распаковка всех свойств задачи) и **onClick**, который вызывает функцию **toggleTodo** при клике на задачу.
4. Проверка типов с помощью PropTypes:
   * **TodoList.propTypes** определяет проверку типов для свойств компонента **TodoList** с использованием **PropTypes**.
   * **todos** должен быть массивом объектов с определенной структурой, описанной в **PropTypes.shape()**.
   * **toggleTodo** должен быть функцией.
5. Экспорт:
   * **export default TodoList** экспортирует компонент **TodoList**, чтобы он мог быть использован в других модулях.

Таким образом, данный код определяет компонент **TodoList**, который представляет список задач в React-приложении. Компонент принимает свойства **todos** и **toggleTodo** для отображения списка задач и выполнения действий по переключению состояния задачи. Происходит также проверка типов свойств с помощью **PropTypes**.

Рассказать про контейнеры

AddTodo –

В данном коде реализован компонент React с использованием библиотеки Redux для управления состоянием приложения. Давайте разберем, что происходит по шагам:

1. Импортированы необходимые зависимости:
   * **React** и **useRef** из пакета 'react': для создания функционального компонента и использования ссылки на DOM-элемент.
   * **connect** и **ConnectedProps** из пакета 'react-redux': для связи компонента с Redux-хранилищем.
   * **RootState** из '../reducers': для типизации корневого состояния Redux-хранилища.
   * **addTodo** из '../actions': для импорта функции-создателя действия, которая будет диспетчеризовать событие добавления задачи.
2. Определены интерфейсы типов данных:
   * **OwnProps**: пустой интерфейс для определения собственных свойств компонента (в данном случае нет дополнительных свойств).
   * **DispatchProps**: определение типа для свойства **addTodo**, которое должно быть функцией, принимающей строковый аргумент.
   * **Props**: типизация всех свойств компонента, включая собственные свойства (**OwnProps**), подключенные свойства (**ConnectedProps<typeof connector>**) и свойства диспетчера (**DispatchProps**).
3. Определен функциональный компонент **AddTodo**:
   * Принимает деструктурированный объект **addTodo** из свойств (**Props**) компонента в качестве аргумента.
   * Использует хук **useRef** для создания ссылки **inputRef**, связанной с элементом **input** в DOM.
   * Определен обработчик **handleSubmit**, который будет вызываться при отправке формы:
     + Предотвращает стандартное поведение отправки формы, чтобы страница не перезагружалась.
     + Проверяет, если **inputRef.current** существует и его значение (**value.trim()**) не пустое после удаления начальных и конечных пробелов.
     + Вызывает функцию **addTodo** с переданным значением из **inputRef.current.value**.
     + Очищает поле ввода задачи (**inputRef.current.value = ''**).
   * Возвращает разметку компонента:
     + Форму с обработчиком **handleSubmit**.
     + Поле ввода с привязкой к **inputRef**.
     + Кнопку "Add Todo" для отправки формы.
4. Определены функции **mapStateToProps** и **mapDispatchToProps**:
   * **mapStateToProps**: пустая функция, которая возвращает пустой объект. В данном случае, компонент не нуждается в получении состояния из Redux-хранилища.
   * **mapDispatchToProps**: объект, содержащий функцию **addTodo**. Она будет доступна в свойствах компонента (**Props**) и позволит вызывать действие **addTodo** при добавлении новой задачи.
5. Создана переменная **connector**, использующая функцию **connect** для связывания компонента с Redux-хранилищем. В качестве аргументов передаются **mapStateToProps** и **mapDispatchToProps**.
6. Компонент **AddTodo** экспортируется, обернутый вызовом **connector(AddTodo)**. Это связывает компонент с Redux-хранилищем и делает его доступным для использования в приложении.

Таким образом, данный компонент **AddTodo** представляет форму для добавления новой задачи. При отправке формы вызывается действие **addTodo**, которое обновит состояние Redux-хранилища с новой задачей.

FilterLinks –

Данный код описывает процесс подключения компонента **Link** к Redux-стору с помощью функции **connect** из библиотеки **react-redux**. Здесь применяется паттерн контейнерной компоненты (Container Component) для управления состоянием компонента **Link** с помощью Redux.

* **mapStateToProps** - функция, которая принимает текущее состояние Redux-стора (**state**) и пропсы, переданные в **Link** (**ownProps**), и возвращает объект **StateProps**. В данном случае, **StateProps** содержит свойство **active**, которое определяет, является ли текущий фильтр (**ownProps.filter**) активным фильтром в Redux-сторе (**state.visibilityFilter**).
* **mapDispatchToProps** - функция, которая принимает диспетчер Redux (**dispatch**) и пропсы, переданные в **Link** (**ownProps**), и возвращает объект **DispatchProps**. В данном случае, **DispatchProps** содержит метод **onClick**, который при вызове диспатчит действие **setVisibilityFilter** с переданным фильтром (**ownProps.filter**).

Затем используется **connect**, чтобы связать компонент **Link** с Redux-стором, передавая **mapStateToProps** и **mapDispatchToProps** в качестве аргументов. Это создает контейнерную компоненту, которая будет обновляться автоматически при изменении состояния Redux-стора и/или при вызове действий, связанных с **setVisibilityFilter**.

Наконец, экспортируется результат вызова **connect**, то есть контейнерная версия компонента **Link**, готовая к использованию в приложении.

visibleTodoList –

В данном коде описывается подключение компонента **TodoList** к Redux-стору с помощью функции **connect** из библиотеки **react-redux**.

* **mapStateToProps** - функция, которая принимает текущее состояние Redux-стора (**state**) и пропсы, переданные в **TodoList** (**ownProps**), и возвращает объект **StateProps**. Здесь **StateProps** содержит свойство **todos**, которое содержит список задач (**todos**), полученных с помощью функции **getVisibleTodos**. **getVisibleTodos** принимает список всех задач из стейта (**state.todos**) и фильтр видимости (**state.visibilityFilter**), и возвращает отфильтрованный список задач в зависимости от выбранного фильтра.
* **mapDispatchToProps** - объект, который определяет действия, связанные с компонентом. В данном случае, он содержит одну функцию **toggleTodo**, которая принимает идентификатор задачи (**id**) и диспатчит действие **toggleTodo**, передавая этот идентификатор.

Затем используется **connect**, чтобы связать компонент **TodoList** с Redux-стором, передавая **mapStateToProps** и **mapDispatchToProps** в качестве аргументов. Это создает контейнерную компоненту, которая будет обновляться автоматически при изменении состояния Redux-стора и/или при вызове действий, связанных с **toggleTodo**.

Наконец, экспортируется результат вызова **connect**, то есть контейнерная версия компонента **TodoList**, готовая к использованию в приложении. Этот компонент будет получать список отфильтрованных задач (**todos**) из Redux-стора и иметь возможность диспатчить действие **toggleTodo** для изменения состояния задачи.