**Flux-архітектура**

Flux-архітектура — архітектурний підхід до розробки додатків, що ґрунтується на однонаправленому потоці даних.

ДІЯ

ДИСПЕЧЕР

СХОВИЩЕ

ПРЕДСТАВЛЕННЯ

ДІЯ

Основні складові

|  |  |
| --- | --- |
|  |  |
| Actions (дії) | Действия (англ. actions) — выражение событий (часто для действий используются просто имена — строки, содержащие некоторый «глагол»). Диспетчеры передают действия нижележащим компонентам (хранилищам) по одному. Новое действие не передаётся пока предыдущее полностью не обработано компонентами. Действия из-за работы источника действия, например, пользователя, поступают асинхронно, но их диспетчеризация явлется синхронным процессом[5][6]. Кроме имени (англ. name), действия могут иметь полезную нагрузку (англ. payload), содержащую относящиеся к действию данные[7]. |
| Stores (сховище) | Хранилище (англ. store) является местом, где сосредоточено состояние (англ. state) приложения. Остальные компоненты, согласно Flux, не имеют значимого (с точки зрения архитектуры) состояния. Изменение состояния хранилища происходит строго на основе данных действия и старого состояния хранилища(см. чистая функция). |
| Views (представлення) | Представление (англ. view) — компонент, обычно отвечающий за выдачу информации пользователю. Во Flux-архитектуре, которая может технически не касаться внутреннего устройства представлений вообще, это — конечная точка потоков данных. Для информационной архитектуры важно только, что данные попадают в систему (то есть, обратно в хранилища) только через действия. |
| Dispatcher (диспечер) | Диспетчер (англ. dispatcher) предназначен для передачи действий хранилищам. В упрощённом варианте диспетчер может вообще не выделяться, как единственный на всё приложение. В диспетчере хранилища регистрируют свои функции обратного вызова (callback) и зависимости между хранилищами[8]. |

|  |  |  |
| --- | --- | --- |
|  | Пояснення | Приклад |
| Actions (дії) | Як правило це об’єкти, у яких обов’язково є поле type, яке містить інформацію про зміст події та можуть міститися додаткові дані (наприклад, при додаванні у сховище ного елемента об’єкт події буде містити інформацію про цей об’єкт)  {  type: “назва\_дії”, 🡨 Що потрібно зробити    . . . Додаткові дані . . .  } | const addAction={  type: “ADD\_PRODUCT”, **🡨 Що потрібно зробити**    payload: { 🡨Додаткові дані для виконання дії  pruductID: 12,  title: “Bread”,  price: 25  }  }  const delAction={  type: “DELETE\_PRODUCT”, 🡨 Що потрібно зробити  payload:{productID: 5} 🡨Додаткові дані  } |
| Stores (сховище) | Сховище -- це об’єкт, де зберігається стан (state). Зміна стану відбувається тільки на основі дій (actions).  Як правило сховище є об’єктом емітера і може надсилати до презентаційних елементів повідомлення про зміну сховища. | var ProductListStore = {  // Колекція продуктів  products : [],  . . . . . . . . .  } |
| Views (представлення) | За представлення відповідають презентаційні компоненти. При цьому компоненти:   * дані одержують зі сховища; * як правило, не містять власного стану; * за необхідності внести зміни надсилаємо дії до сховища за допомогою диспечера. | Компонент для змін у сховищі здійснює передачу дії через диспечер    AppDispatcher.dispatch( //🡨 Диспечер  { //🡨 Дія  type: “ADD\_PRODUCT”,  payload: {  pruductID: 12,  title: “Bread”,  price: 25  }  }  ); |
| Dispatcher (диспечер) | Диспечер – єдиний компонент, який приймає дії і передає їх сховищу (чи сховищам). | var AppDispatcher = new Dispatcher();  AppDispatcher.register( function( action ) {  switch( action.type ) {  case ' ADD\_PRODUCT ':  ProductListStore.products.push(action.payload);  . . . . . . . .  break;  . . . . . . . .  }  return true;  }); |