FLUX INTRODUCTION

Nils Hartmann (nils@nilshartmann)
WHY FLUX?

Some background first...
SINGLE COMPONENT HIERARCHIE

Communication
- Down via properties
- Up via callbacks

Top-level component
- Works on single model
- (whatever a 'model' is)
Isolated Models with independent component trees

- Each component (tree) works on its own model
- Peaceful co-existence
**Shared Model with independent component trees**

- Each component (tree) works on the **same** model
- Can be confusing: who’s taking responsibility? Can lead to back loops
FLUX
"Application architecture for building user interfaces"

- http://facebook.github.io/flux/
- Design pattern, not a framework
- Various implementations and interpretations
- Not tied to React

Introduces unidirectional data flow
Component triggers an Action using an ActionCreator

```jsx
<Button onClick={() => {
    UserActionCreators.addUser(...);
}}> </Button>
```
**ActionCreator** sends **Action** to Central Dispatcher

```javascript
addUser(name) {
    Dispatcher.dispatch({
        type: 'USER_ADD_ACTION',
        payload: { user: name }
    });
}
```

(UserActionsCreator)
Dispatcher forwards the Action to Store
Store processes **Action** and emits a change event

```javascript
handleAction(action){
  if (action.type==='USER_ADD_ACTION') {
    this.users.push(action.payload.user);
    emitStoreChangeEvent();
  }
}
```
Component receives event and updates its state

```javascript
onUserStoreChange() {
    this.setState(
        {users: UserStore.getAllUsers();}
    );
}
```
Note: different implementations interpret differently
COMPONENT

„Regular“ React component
• Triggers an action
  • e.g. on user interaction
• Listens to one or more Stores
• Derives its state from Store
  • Update children from render() method

Controller Component
• Top-level component should be the only component interacting with Flux
ACTION and ACTIONCREATOR

**Action**
- Represents *semantic* event happened in the app
- Has an identifying „type“
- Contains information what happened (payload)
- Triggered by Components

**ActionCreator**
- Factory for Actions
- Ensures integrity of action object,
- Semantic methods
- Pass Action to Dispatcher
• Pure technical component, no business logic
• Only one Dispatcher in your application (singleton)
• Forwards Actions to all registered Stores
• Works synchronous
• Contains the **business logic** for a specific domain
• Connect to the central Dispatcher via callback method
• Process only Actions they are interested in (e.g., filter by type)
• Emit events after changing model
UNI-DIRECTIONAL DATA FLOW - EXAMPLES

Very flexible scenarios...
...Action can be dispatched to multiple stores...

...A component can listen to multiple stores...
Multiple components can listen to the same store, working on the same domain "model"...
...“external events“ (e.g. response from server) can trigger actions
• Flow is still the same
SUMMARY

Useful when working on same model

Defined flow of data
• Easy to track
• Recordable

Lots of new vocabular
• eg: what is the shape of a Store?

Many implemenations
• No „standard“ implementation
THANK YOU!

Slides:
tbd