A PAIRSHAPED TUTORIAL

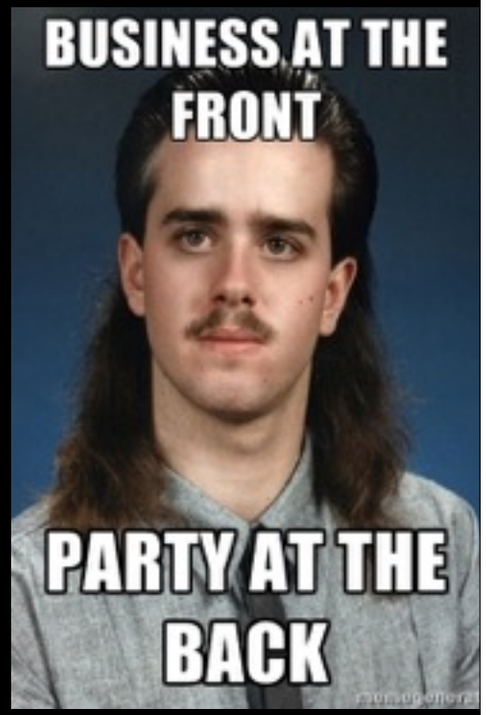
INTRODUCING FLUX

WHAT IS FLUX?

- Not quite MVC
- What does flux give us?
- Flux segregates your business logic from your views
- Flux uses unidirectional data flow
- Flux makes your application predictable

SEPARATE BUSINESS AND VIEW LOGIC

- By building Stores and providing Actions, we can better identify:
  - where data originates,
  - where it is changed,
  - who is listening to it.
- We can accomplish similar behaviour using a React mixin, but this invalidate the above statements.



WHAT DOES FLUX GIVE US?

- Dispatcher
  - Typically a singleton.
  - A place where actions are processed and emitted.
- Stores
  - A single place to access and store our data.
  - Try and keep a store to one piece or type of information.
  - We also have an opportunity to modify or process data before it is emitted.
- Action
  - Moves data from the store to a listener in a single direction.
  - Actions should always be thought in terms of tell, don’t ask.
- Action Listeners
  - Callbacks where we can access processed data from a store.

FLUX ACTIONS

- An action is a callable function that acts like an event emitter.
- They provide one way data flows
- Actions can be asynchronous
- A single action can be listened to on multiple points.
- Actions usually fall into one of these groups:
  - Setter (component to store)
  - Response (store to component)]
  - Getters should be avoided, since it breaks the “tell, don’t ask” rule, but these are occasionally unavoidable.

A getter may be used in cases where data is cached in the store, but should not mutate after it has been retrieved. As an example, when an API requires the user to create a session or bring in a configuration before accessing the rest of the API.

FLUX STORE

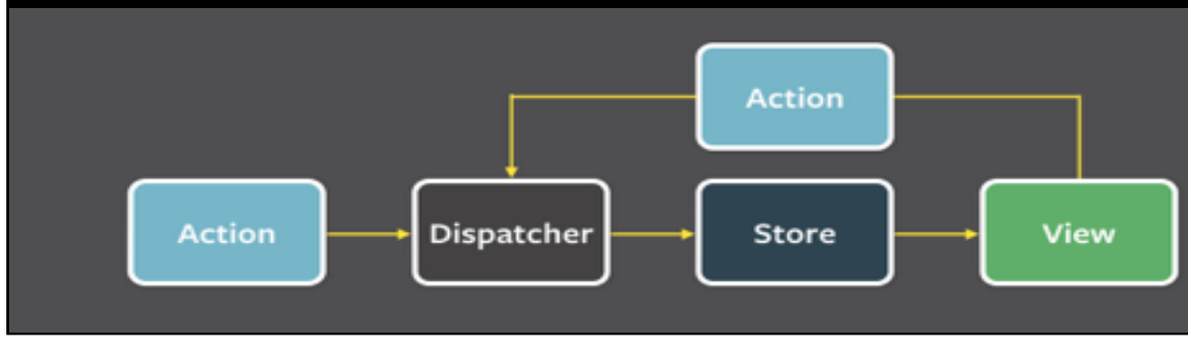
- A store is where we interface with our data.
- Data can be anything, including SQL, an API, cookies, or local storage.
- Stores should listen to a set of actions, and be prepared to send out responses after dealing with an action.

FLUX DISPATCHER

- The dispatcher is a singleton object that implements a function to handle and route actions.
- This is usually done with an awful and large switch statement.
- The dispatcher will, by default, back onto the NodeJS EventEmitter, but can also hook into Promises (bluebird), or custom event implementations.
- Some Flux alternatives do not use a singleton dispatcher.

UNIDIRECTIONAL DATA FLOW

- Flux typically requires an action to kick off a series of other actions.
- Actions only flow one direction, through:
  - Dispatcher, Store, then Component



FLUX WITH REACT

- React components should handle view logic, and bindings between business and view logic, but should avoid performing much, if any, business logic.
- It should always be safe to store data sent from an action into a components state because:
  - Unidirectional dataflow should mean that the data will be in the correct format at the time of arrival
  - Any modifications that happen to the data will happen at the store level, and then propagate to components after mutations.
- If you think the data will be out of sync or problematic to store in the state of a component, it means you are not following unidirectional data flow!

SUMMARY BEFORE CODE

- Flux is built to move business logic out of the view logic.
- Flux flows from an action to the dispatcher, store, then component.
- Unidirectional flow should guarantee integrity of data.

Any questions before we hit some code?

RESOURCES

- This one made Flux make the most sense from any tutorial I have read: http://www.toptal.com/front-end/simple-data-flow-in-react-applications-using-flux-and-backbone
- Dispatcher docs: https://facebook.github.io/flux/docs/dispatcher.html
- Todo MVC Example: https://github.com/facebook/flux/tree/master/examples/flux-todomvc