# Contents



# 1 Talk on Javascript Framework using HyperApp

## 1.1 What is HyperApp?

- Simple JavaScript framework with state and pure functional components
- Less boilerplate, more convention
- 10k javascript source code, 1k zipped

## 1.2 Main difference with React

- no action objects
- no local state by default

- built in asynchronous actions

### 1.3 Virtual Dom

Simple idea:

- instead of manipulating actual dom nodes (slow)
- we use a light-weight js representation
  - node name (h1)
  - attributes (class=header)
  - children (The Title)

### 1.4 Virtual dom rendering

First render:

- create a dom node (element) for each vdom node

Further renders:

- compare old vdom and new vdom
- if different, update matching dom node (element)
- 'matching'...

### 1.5 h aka createElement

- Creates a virtual dom node

```
{
  name: "div",
  props: {
    class: "main"
  },
  children: ["Hello World"]
}
```

- Example:

```
const node = h("div", { class: "main" }, "Hello World")
```

- To support JSX, the virtual node builder needs to deal with:
  - Functional components
  - Variadic children arguments
  - Nested children arrays
  - Conditional rendering

### 1.6 JSX details: Nested children arrays

```
h("div", { "class": "superb"} , "hello pivotal", ["it is nice to see you", "today"])

RESULT
{ nodeName: 'div',
  attributes: { class: 'superb' },
  children: [ 'hello pivotal', 'it is nice to see you', 'today' ],
  key: undefined }
```

### 1.7 JSX details: a function instead of a name

```
h((attributes, children) =>  h("span", attributes, children),
  { "class": "superb"} ,
  ["it is nice to see you", "today"] )

RESULT:
{ nodeName: 'span',
  attributes: { class: 'superb' },
  children: [ 'it is nice to see you', 'today' ],
  key: undefined }
```

### 1.8 JSX Details: A function (2)

Why do we want this? Because of components!

```
const Button = props => <button class="main">{props.title}</button>
const node = <Button title="Hello World" />
```

Babel will compile this to:

```
const Button = props => h("button", { class: "main" }, props.title)
const node = h(Button, { title: "Hello World" })
```

### 1.9 JSX Details: A function (3)

If you add the following to your '.babelrc':

```
"plugins": [
   ["@babel/transform-react-jsx", { "pragma": "h" }],
]
```

You can use inline jsx in your 'babel-node' live command line:

```
> const Button = props => <button class="main">{props.title}</button>
> Button
[Function: Button]
> Button.toString()
'function Button(props) {
  return h("button", {
    class: "main"
  }, props.title);
}'
>
```

### 1.10 App: oldNode and container replacement

The view that HyperApp is going to produce will have to be attached to the real dom. Here we create a virtual dom equivalent of that current dom node, recursively copying any 'text' elements that current dom node may have.

We use 'recycleElement' to do the recursive copying. It is used only here.

From README.md:

> Hyperapp will also attempt to reuse existing elements inside the container enabling SEO optimization and improving your sites time-to-interactive. The process consists of serving a fully rendered page together with your application. Then instead of throwing away the existing content, we'll turn your DOM nodes into an interactive application out of the box.

### 1.11 scheduleRender

Run the renderer at the earlier opportunity, in a separate thread. Uses 'skipRender' to avoid scheduling the renderer more than once:

- Once we schedule a render, set skipRender to true
- If skipRender is truthy, scheduleRender will be a no-op.

### 1.12 render

- Toggle 'skipRender'
- Materialize our view by executing all nodes that are functions and removing null nodes (function 'resolveNode')
- patch the designated root element by adding our view to it
- set 'isRecyling' to false: Recycling=true for the very pass of the renderer, when we try to re-use existing dom elements.
- run all lifecycle hooks

### 1.13 Lifecycle hooks

- every time we run 'updateElement', this function will add itself back to the lifecycle methods
- if the user adds an 'oncreate' attribute to a new node, it will be added to the lifecycle hook array
- Available lifecycle hooks: oncreate, onupdate, onremove, ondestroy

### 1.14 Wire State to Actions

- Store = (global) state
- This function will make your actions work on the store.
- Actions can be namespaced, as in the example
- Actions can return a value, in which case the store gets updated
- Or they can return a function with arguments (store, actions), in which case the result of that function is the new value for store.

### 1.15 Example App - Walkthrough

- See 'traced' directory
- Working setup with jsx/babel/webpack
- Commented hyperapp.js file with some console.logs in their
- Very simple app: Counter example, but with components.
- [Switch between demos by commenting in/out]

### 1.16 Demo 1

- Simple app using jsx
- No clickable elements, no actions
- Simple state and one state insertion
- Inserted in dom after '#target'

### 1.17 Demo 2

- Same thing after JSX conversion
- Draw vdom node tree
- Let's look at hyperapp's events in console log
  - Patch: to insert dom nonde
  - create DOM elements for each vdom node

### 1.18 Demo 3

- Using a counter, with simple up/down actions
- Fancy 1: Counter is a component, so we can have two
- Fancy 2: Counter is named, see ugly state update
- Fancy 3: Counters don't go below 0
- [Comment out the 'minutes' counter now]

### 1.19 Hyper app tracing

- App startup console logs: app is created and attached to DOM
- Click button: See that only counter is updated
- Patch does a two diff between old and new vdom, updates DOM elements accordingly
- Actions have been wired to run render ('wireStateToActions')

### 1.20 Thank you

### 1.21 Resources

- How to use local state in hyperapp (make a nested app). `https://zaceno.github.io/hypercraft/post/stateful-components/`
- Reasonable looking starter (did not try) `https://github.com/selfup/hyperapp-one`
- What go me started: "I abandonded React in favor of HyperApp, here's why' `https://hackernoon.com/i-abandonded-react-in-favor-of-hyperapp-heres-why-df6`