Thinking In React

FACEBOOK'S GUIDE FOR DESIGNING REACT APPS

Objectives

- To understand how to develop a React application
- To be able to identify components
- To be able to identify where state should live
- To be able to implement inverse data flow

Thinking In React

- React makes developers think about the applications that are being built as they build them
- Facebook's recommendation is to follow these steps when building apps using React:

  - Start with a mock

  1. Break the UI into a component hierarchy
  2. Build a static version in React
  3. Identify the minimal (but complete) representation of UI state
  4. Identify where your state should live
  5. Add inverse data flow

3

Start with a mock

- All UIs to be built should have a mock up or a wireframe
- Data should be available in a suitable format too
- Facebook's guide shows a Product Search page and some JSON for products

```
[
  {category: "Sporting Goods", price: "£49.99", stocked: true, name: "Football"},
  {category: "Sporting Goods", price: "£9.99", stocked: true, name: "Baseball"},
  {category: "Sporting Goods", price: "£29.99", stocked: false, name: "Basketball"},
  {category: "Electronics", price: "£99.99", stocked: true, name: "iPod Touch"},
  {category: "Electronics", price: "£399.99", stocked: false, name: "iPhone 5"},
  {category: "Electronics", price: "£199.99", stocked: true, name: "Nexus 7"}
];
```

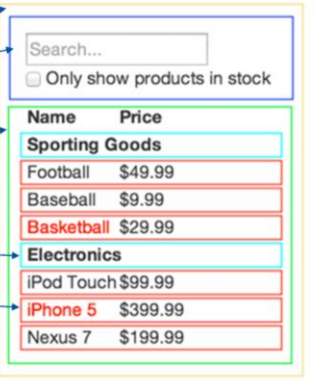
4

1. Break the UI into a Component Hierarchy

- Draw boxes around every component and subcomponent in the mock and give them names
  - Image layers may end up being names of components
- To help identify, components should only do one thing
- Data model and data should map nicely

```
FilterableProductTable
        SearchBar
        ProductTable
                ProductCategoryRow
                ProductRow
```

- Component Hierarchy already defined
- Not the only way to implement...



5

2. Build a static version in React

- Build version that takes data model and renders UI without interactivity
  - Decoupling view and interactivity good as static versions are lots of typing and little thinking and vice versa for interactive versions
  - Best practice is to build components that reuse other components and pass data using *props*
    - *state* not used as it is for interactivity
- Top-down building as acceptable as bottom-up
  - Bottom-up more common in TDD environments
- Helps establish library of reusable components to render each data model
  - Each only has render method

6

3. Identify minimal (complete) representation of UI state

- Require ability to trigger changes
  - Made easy through use of *state*
- Best practice to first think of minimal set of mutable state app needs
  - Everything else computed on demand
  - In example, these are:
    - Original list of products
    - Search text user enters
    - Value of checkbox
    - Filtered list of products
- *props is data passed in as an HTML attribute when the component is created*
- To figure out state, ask:
  1. Is it passed in from a parent via props? Probably isn't state
  2. Does it remain unchanged over time? Probably isn't state
  3. Can you compute it based on any other state or props in the component? Isn't state

7

3. Identify minimal (complete) representation of UI state

- In example:
  - *Original list of products* – passed in as props – not state
  - *Search text user enters* – state
  - *Value of checkbox* – state
  - *Filtered list of products* – can be computed by combining original list of object, search text and checkbox value – not state

8

4. Identify where state should live

- Involves identifying which component mutates or owns the state
  - React all about one-way data flow down component hierarchy
  - May not be immediately clear which component should own state
- To work out where state should live:
  1. Identify every component that renders something based on state
  2. Find common owner component
  3. Either common component or component even higher up should own state
  4. If no component makes sense, create new component to hold state and add it into the hierarchy above the common owner component
- For the example:
  - **`ProductTable`** needs to filter product list based on state and **`SearchBar`** needs to display search text and checked state
  - Common owner component is **`FilterableProductTable`**
  - Conceptually makes sense for filter text and checked value to live in **`FilterableProductTable`**

9

5. Add inverse data flow

- App renders correctly as function of *props* and *state* flowing down hierarchy
- To support data flowing other way, form components deep in hierarchy need to update state in `FilterableProductTable`
- Want to make sure whenever user changes form, state is updated
  - **`FilterableProductTable`** needs to pass callback to **`SearchBar`**
    - Fires whenever state should be updated
  - Use **`onChange`** event on inputs
    - Callback will call **`setState`** and update the app

10

Objectives

- To understand how to develop a React application
- To be able to identify components
- To be able to identify where state should live
- To be able to implement inverse data flow

Exercise Time

- Primers for each stage of this process will be done at appropriate parts of other exercises.

12