CSE 331
Software Design & Implementation

Kevin Zatloukal

Spring 2020

Modern Web UIs

Dynamic Web Content

- Earlier example had a fixed set of components.
  - same for iPhone / Android apps

- More realistic apps need to change the set of components displayed on the screen dynamically
  - consider Gmail as an example
  - need the components to come from code

Example 1

register-js/index.js

Problems

These tools can be used to write Gmail

But it has a number of problems…

1. Lack of tool support
   - no checking of types, tags, etc.
2. No support for modularity
   - all the code and UI in a single file
3. More boilerplate
   - minimized JS file would change function names
   - need to call `btn.addEventListener` by hand

JS Modules

- EcmaScript6 (ES6) added support for modules.
- Each file is a separate unit ("namespace")
- Only exported names are visible outside:

```
export function average(x, y) { …
```

- Others can import using:

```
import { average } from './filename';
```

Example 2

register-js2/…

JS Classes

- ES6 added new syntax for classes:

```
class Foo {
  constructor(val) {
    this.secretVal = val;
  }

  secretMethod(val) {
    return val + this.secretVal;
  }
}
```

More from Example 2

register-js2/…

JS vs Java Classes

- JS method signatures are just the name
  - JS objects are just HashMaps
  - field names are the keys
- Java methods signatures are name + arg types
  - e.g., `average(int, int)`
- JS has only one method with a given name
  - language allows different numbers of arguments
    - Missing arguments are undefined
  - can strengthen a spec by accepting a wider set of possible input types

Problems

These tools can be used to write Gmail

But it has a number of problems…

1. Lack of tool support
   - no checking of types, tags, etc.
2. No support for modularity
   - all the code and UI in a single file
3. More boilerplate
   - minimized JS file would change function names
   - need to call `btn.addEventListener` by hand

TypeScript

- Adds type constraints to the code:
  - arguments and variables

```
let x : number = 0;
```

  - fields of classes

```
quarter: string;
```

- **`tsc`** performs type checking

- Creates version has type annotations removed

TypeScript Types

- Basics from JavaScript:
  number, string, boolean, string[], Object

- But also
  - specific classes Foo
  - tuples: `[string, int]`
  - enums (as in Java)
  - allows null to be included or excluded (unlike Java)
  - `any` type allows any value
  - …

Example 3

register-ts/…

TypeScript

- Type system is unsound
  - can't promise to find prevent all errors
  - can be turned off at any point with any types
    - `x as Foo` is an unchecked cast to `Foo`
    - `x!` casts to non-null version of the type (useful!)

- Full description of the language at `typescriptlang.org`

Problems

This is better, but it still has problems…

1. Still no checking of HTML (opaque strings)

2. Modularity is still poor
   - need to join strings into one big string

3. More boilerplate
   - minimized JS file would change function names
   - need to call `btn.addEventListener` by hand

JSX

- Fix the first problem by adding HTML as a JS type
- This is supported in `.jsx` files:

```
let x = <p>Hi, {name}.</p>;
```

- Compiler can now check that this is valid HTML
- {…} replaced with string value of expression

JSX Gotchas

- Put (..) around HTML if it spans multiple lines

- Cannot use `class="btn"` in your HTML
  - `class`, `for`, etc. are reserved words in JS
  - use `className`, `htmlFor`, etc.

- Must have a single top-level tag:
  - not: `return <p>one</p><p>two</p>;`
  - usually fixed by wrapping those parts in a `div`

Problems

This is even better, but it still has problems…

1. Modularity is still poor
   - need to join strings into one big string
2. More boilerplate
   - minimized JS file would change function names
   - need to call `btn.addEventListener` by hand

React

- Regain modularity by allowing custom tags

```
let app = (
    <div>
      <TitleBar name="My App"/>
      <EditPane rows="80" />
    </div>);
```

- `TitleBar` and `EditPane` can be separate modules
  - their HTML gets substituted in these positions

React

- Custom tags implemented using classes

```
class TitleBar extends React.Component {
```

- Attributes (`name="My App"`) passed in `props` arg
- Method `render` produces the HTML for component
- Framework joins all the HTML into one blob
  - can update in a single call to `innerHTML = …`

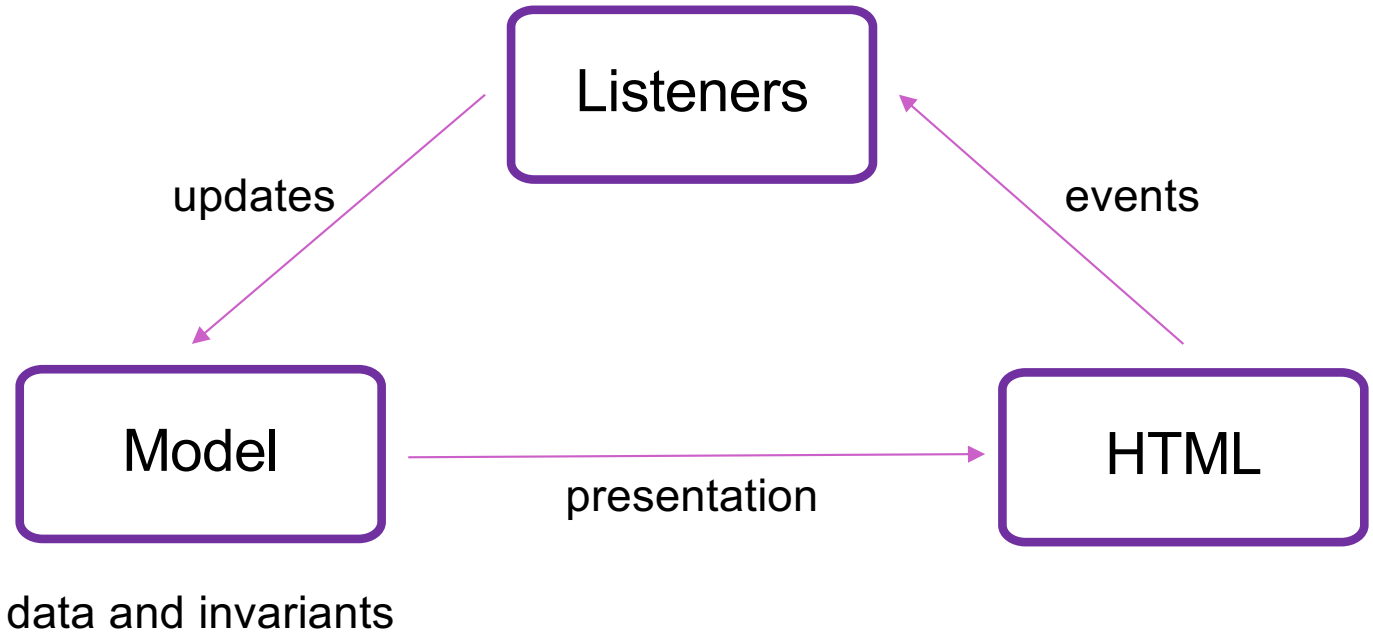
Example 4

register-react/…

React State

- Last example was not dynamic!
  - there was no model
  - (why have classes then?)

Structure of a React Application



React State

- Last example was not dynamic!
  - there was no model
  - (why have classes then?)

- Components become dynamic by maintaining state
  - stored in fields of `this.state`
  - call `this.setState({field: value})` to update

- React will respond by calling `render` again
  - will automatically update the HTML to match the HTML produced by this call

Example 5

register-react2/…

React State

- Custom tag also has its own events

- Updating data in a parent:
  - sends parent component new data via event
  - parent updates state with `setState`
  - React calls parent's `render` to get new HTML
    - result can include new children
    - result can include changes to child props

- State should exist in the lowest common parent of all the components that need it

React Event Listeners

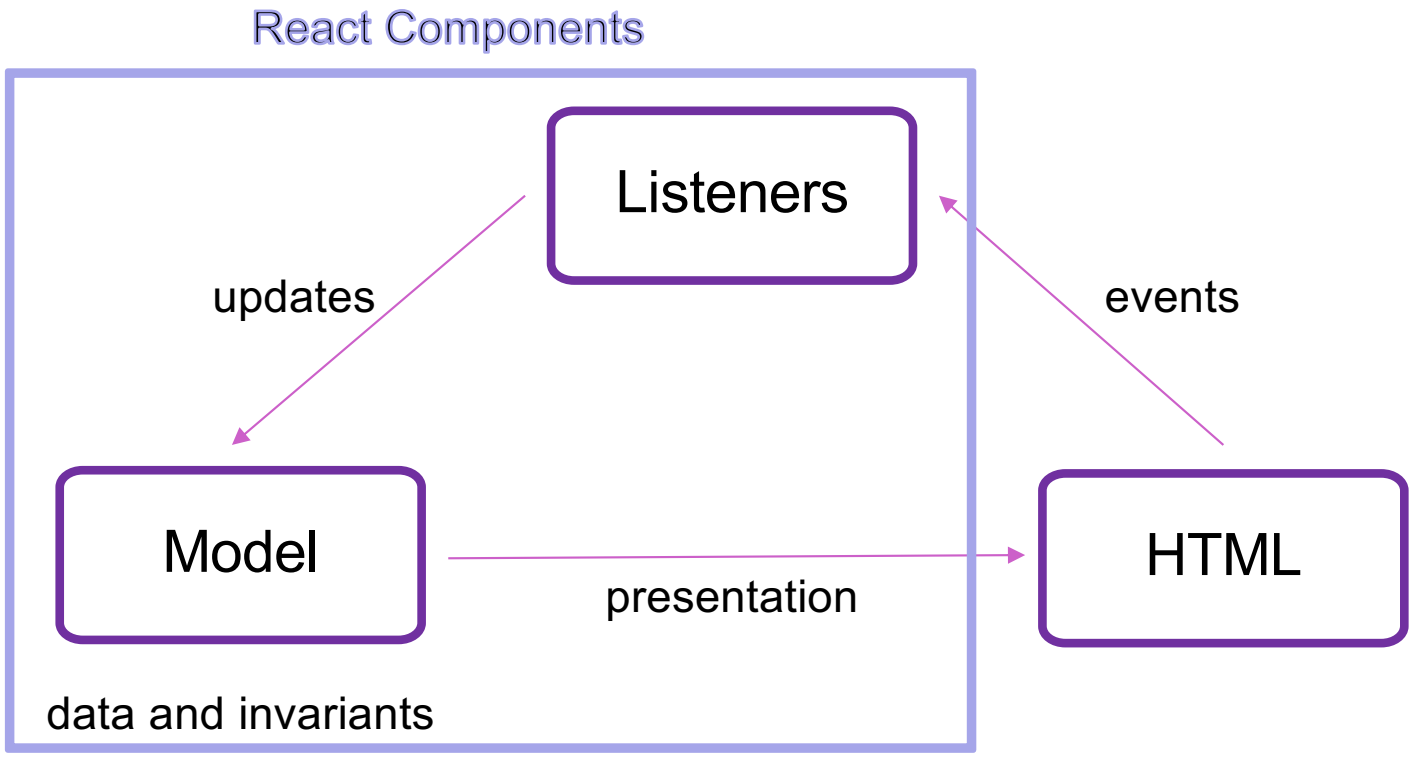
- Solves the problems of poor modularity
- Also removes an ugly hack in the earlier code

```
<button onClick="PickQuarter(…)">
window["PickQuarter"] = PickQuarter
```

- Event listeners can be added in the natural way:

```
<button onClick={this.onClick.bind(this)}>
<button onClick={evt => this.onClick(evt)}>
```

Structure of a React Application



Structure of a React Application

- At any moment, want model to store all data necessary to generate the exact UI on the screen

- Any time react updates the HTML, it should match up to what is currently

React setState

- `setState` does not update state instantly:

```
// this.state.x is 2
this.setState({x: 3});
console.log(this.state.x); // still 2!
```

- Update occurs after the event finishes processing
  - `setState` adds a new event to the queue
  - work is performed when that event is processed

- React can batch together multiple updates

React Performance

- React re-computes the tree of HTML on state change
  - can compute a “diff” vs last version to get changes

- Surprisingly, this is not slow!
  - slow part is calls into browser methods
  - pure-JS parts are very fast in modern browsers
  - processing HTML strings is also incredibly fast

React Tools

- Use of compilers etc. means new tool set

- `npm` does much of the work for us
  - installs third-party libraries
  - runs the compiler(s)