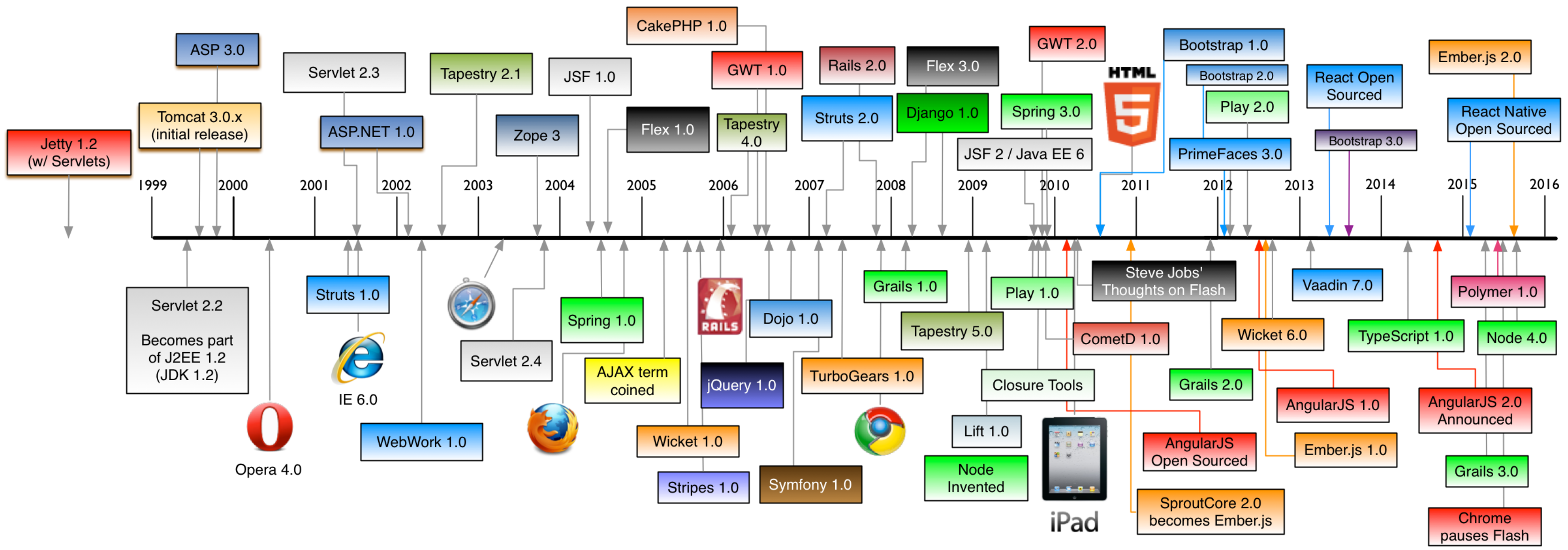
How to build

# Fat Web UI

**Without JS framework !**

# No Framework? – Oh really?

## Are you desperate of switching newer UI FW?



Forget about a framework

**Follow Web standards!**

## What we need !

- **Simplicity**
  - Shallow learning curve => no framework, just **dev stack**
- **Rich functionality**
  - No framework restrictions, we can do anything! Yes we can!
- **Re-usability**
  - Building reusable **components** – functionality encapsulation
- **Modularity**
  - Splitting application into modules – fast parallel development
- **Maintainability**
  - No framework – no dependences – no FW upgrades

## Components

- Web UI Component
  - **Reusability**, **testability**, **encapsulation** of functionality
  - Class in programming language perspective
  - Consist of:
    - **Custom app logic** – no need general interface
    - **HTML / DOM rendering** – need general interface
- Use TypeScript/TSX instead JavaScript/JSX
- Component – simple interface:

```
interface Component {
  render():HTMLElement;
}
```

## Component example – basic

```
class Item {
  constructor(public text:string, public count:number) {
  }
}

class MyComponent implements Component {
  private _items:Item[] = [];

  constructor(items:Item[]) {
    this._items = items;
   }

  render():HTMLElement {
    return <ol>
      { self._items.map((item) => {
        return <li>
          <span class="label">{ item.text }</span>{ ' ' }
          <small class="count">[{ item.count }]</small>
        </li>;
      })
      }
    </ol>;
  }
}
```

# Component example – advanced

```
class MyComponent implements Component {
  private _items:Item[] = [];
  private _onSelect:(item)=>void;

  constructor(items:Item[]) {
    this._items = items;
  }

  onSelect(callback:(item:Item)=>void) {
    this._onSelect = callback;
  }

  render():HTMLElement {
    var self = this;
    return <ol>
      { self._items.map((item) => {
        return <li onclick={
            (e) => {
              e.stopPropagation();
              self._onSelect(item);
            }
          }>
          <span class="label">{ item.text }</span>{ ' ' }
          <small class="count">[{ item.count }]</small>
        </li>;
      })
      }
    </ol>;
  }
}
```

# Component example – usage

```html
<div id="container"></div>
<p>
  Seleced: <span id="selected"></span>
</p>
```

```typescript
var myComponent = new MyComponent([
    new Item('text 1', 1),
    new Item('text 2', 2),
    new Item('text 3', 3)
  ]);

myComponent.onSelect(
  (item) => {
    console.log('selected:', item);
    var selected = document.getElementById('selected') as HTMLSpanElement;
    selected.innerHTML = JSON.stringify(item);
  });

var myComponentElement = myComponent.render();

var container = document.getElementById('container') as HTMLDivElement;
container.appendChild(myComponentElement);
```

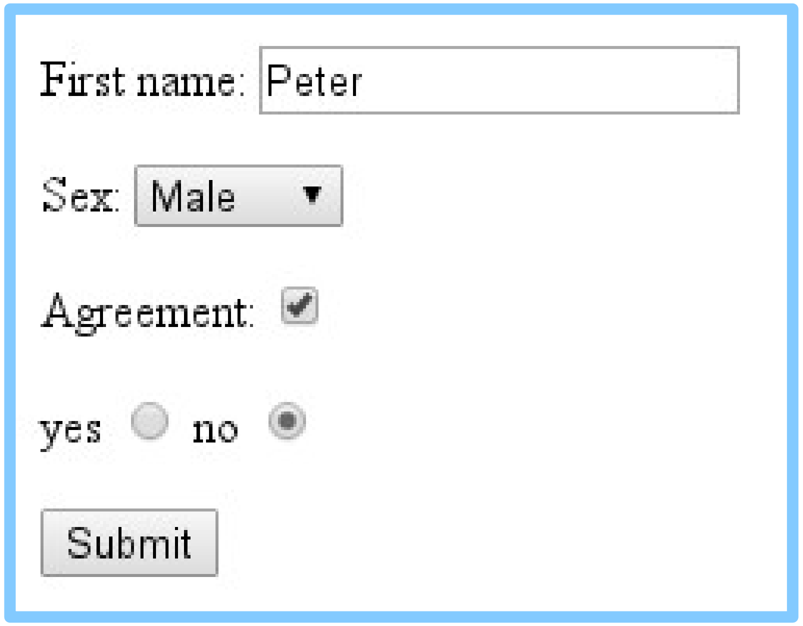
## Forms, Inputs and validation

- **Common interface** for different inputs
  - text, select, checkbox, radio
- **Input validation** based on locale

```
interface Entry {
  getName():string;
  getValue():string;
  setValue(value:string):Entry;
  validate(locale?:string):Object;
  setValidator(validator:(value:string, locale?:string)=>string):Entry;
  onChange(callback:(value)=>void):Entry;
}

 var emptyValueValidator = (value:string, locale:string) => {
   switch (locale) {
     case 'sk': return value ? '' : 'Prázdna hodnota';
     default: return value ? '' : 'Empty value';
   }
 };
```

```html
<form id="form" action="">
  <p>
    <label for="name">First name:</label>
    <input id="name" type="text" name="name" value="">
    <span id="name-err"></span>
  </p>
  <p>
    <label for="sex">Sex:</label>
    <select id="sex" name="sex">
      <option value=""></option>
      <option value="M">Male</option>
      <option value="F">Female</option>
    </select>
    <span id="sex-err"></span>
  </p>
  <p>
    <label for="agree">Agreement:</label>
    <input id="agree" type="checkbox" name="agree">
    <span id="agree-err"></span>
  </p>
  <p>
    <label for="yes-no-y">yes</label>
    <input id="yes-no-y" type="radio" name="yes-no" value="y">
    <label for="yes-no-n">no</label>
    <input id="yes-no-n" type="radio" name="yes-no" value="n">
    <span id="yes-no-err"></span>
  </p>
  <p>
    <input type="submit" value="Submit">
  </p>
</form>
```

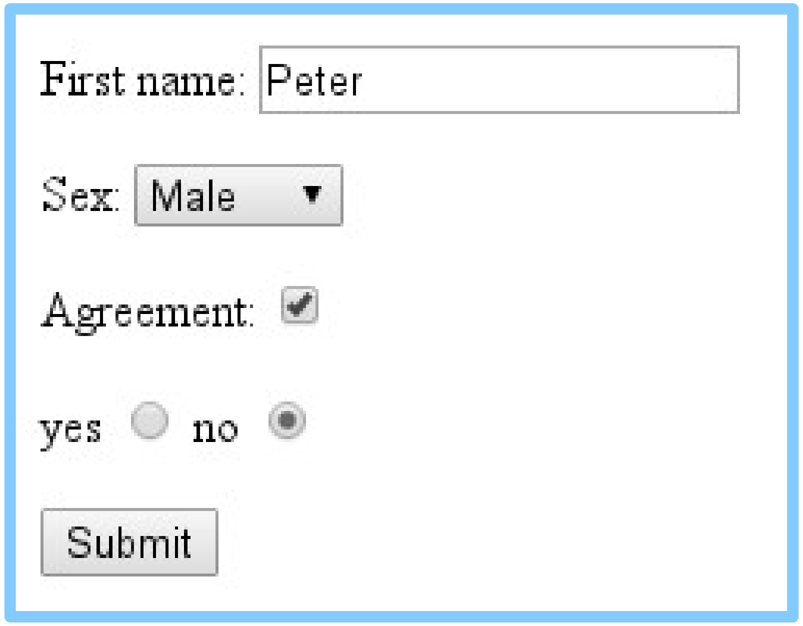

First name: Peter

Sex: Male

Agreement: ✓

yes ○ [✓ no]

```
var f = new Form('form')
  .addEntry(new InputEntry('name')
    .setValue('Peter')
    .setValidator(emptyValueValidator)
    .onChange(showChange))
  .addEntry(new SelectEntry('sex')
    .setValue('M')
    .setValidator(emptyValueValidator)
    .onChange(showChange))
  .addEntry(new CheckboxEntry('agree')
    .setValue(true.toString())
    .setValidator(emptyValueValidator)
    .onChange(showChange))
  .addEntry(new RadioEntry(['yes-no-y', 'yes-no-n'])
    .setValue('n')
    .setValidator(emptyValueValidator)
    .onChange(showChange))
  .onSubmit(() => {
    var errors = f.validate('sk');
    for (var error in errors) {
      document.getElementById(error + '-err').innerHTML = errors[error];
    }
    if (f.isValid(errors)) {
      document.getElementById('values').innerHTML = JSON.stringify(f.getValues());
    } else {
      document.getElementById('values').innerHTML = '';
    }
  });
```



First name: Peter

Sex: Male

Agreement: ✓

yes [✓ no]

## My recommendations

- Build your set of reusable Web UI components
  - Application specific components
  - Company specific components
- Do not build complex Web IU application as single page app => split application into set of single page applications, modules
- Assemble your application strictly from components for modularity and encapsulation purpose
- If you miss a specific component, create new one – **it is easy!**

# Still confused?

- **Are you unexperienced developer?**
  Go for some well known framework like React, Angular, Polymer, …
- **Are you working on small startup?**
  Go for framework which gives you functionality you need out of box.
- **Are you company working on long term projects or you need scale and speed up development?**
  Go the way of building components I described

## Still confused?

**Peter Rybar**

- **IT consulting**
- **Project leadership**
- **IT solutions and architecture**
- **Technology leadership**
- **Application, system and data integration**
- **Software development – Agile, Scrum**
- **HR consulting**

http://prest-tech.appspot.com/peter-rybar

https://sk.linkedin.com/in/peter-rybar-9861996