
Develop
Intelligence
Learning Solutions to Attract, Retain,
and Grow your top technical talent.
http://bit.ly/kamrenz-spitfire
https://github.com/kamrenz/spitfire
©COPYRIGHT DEVELOPINTELLIGENCE LLC

Who am I?!?

Kamren Zorgdrager
Director of Product Development @ DevelopIntelligence
kamren@developintelligence.com
@kamrenz

About DI

Develop Intelligence

Company

- Highly-customized, role-based learning solutions
- ***Managed Learning Solutions*** include training design and development, and learning program delivery and management
- ***Technical learning offerings*** in software development, open source technologies, and technology leadership development
- Software development and engineering organizations benefit most from our Managed Learning Solution

Background

- Founded in 2003
- Headquartered in Boulder, Colorado
- Founder and CEO, Kelby Zorgdrager, has 20 years' experience in Technical Learning and Development industry
- More than **40,000 developers** trained in 30 countries since 2003
- Managed and delivered learning solutions to more than 4,000 developers globally in 2015

Course Offerings

Develop Intelligence

Overview

- Over ***200 different course*** offerings available in Java development, Mobile Application development, and more.
- Courses delivered by an ***expert practitioner*** – our instructors are passionate about doing and teaching.
- Consistently achieve ***98%*** or higher ***satisfaction*** rating from students.

Technologies we cover

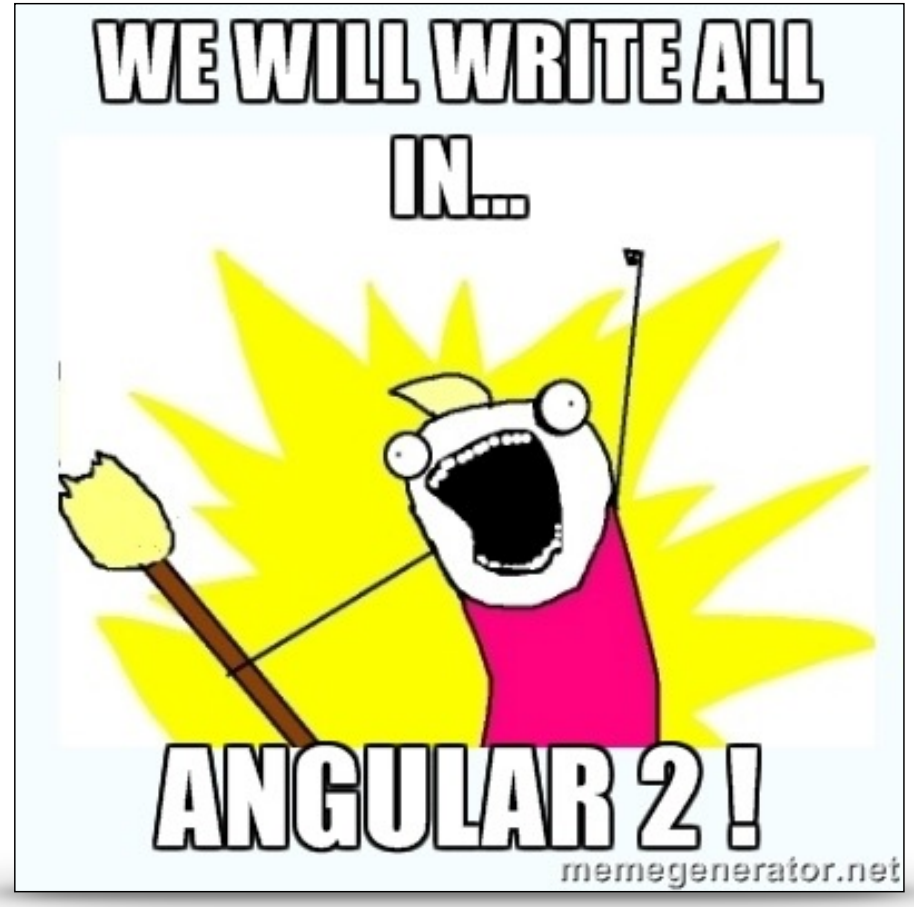
Clients

Develop Intelligence



Angular 2

The component mental mind shift



"Today, at a special meetup at Google HQ, we announced the final release version of Angular 2, the full-platform successor to Angular 1."

- Jules Kremer @ Angular 2 development blog
September 14 2016

Is Google using it?

- AdWords
- GreenTea: Google's internal CRM
- Google Fiber

- Others besides Google
  - https://www.madewithangular.com/#/

What is it?

Develop Intelligence

"Angular 2 is the next generation web application framework from Google. It aims to fix some Angular 1 shortcomings and provide a foundation for JavaScript application development over the next few years."

- Kamren Zorgdrager
Today

Today's Outline

- Part 1
  - How'd we get here?
  - Componentization
  - New ES6 language foundation
  - TypeScript foundation
- Part 2
  - Angular component
  - Angular-CLI
  - Data bindings, Directives, Inputs & Outputs
  - Ajax & Services
  - Where to go from here

Today's Goals

- Understand the need for Angular 2
- Make a mental mind shift to components
- Learn the foundations of Angular 2
- Have reference material for more in-depth learning

How’d we get here

An evolutionary tale

Develop Intelligence

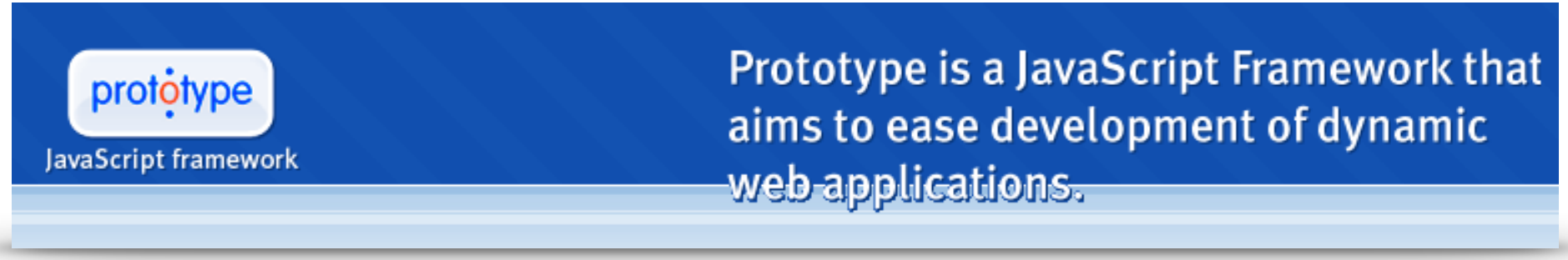
Not a revolution

An evolution

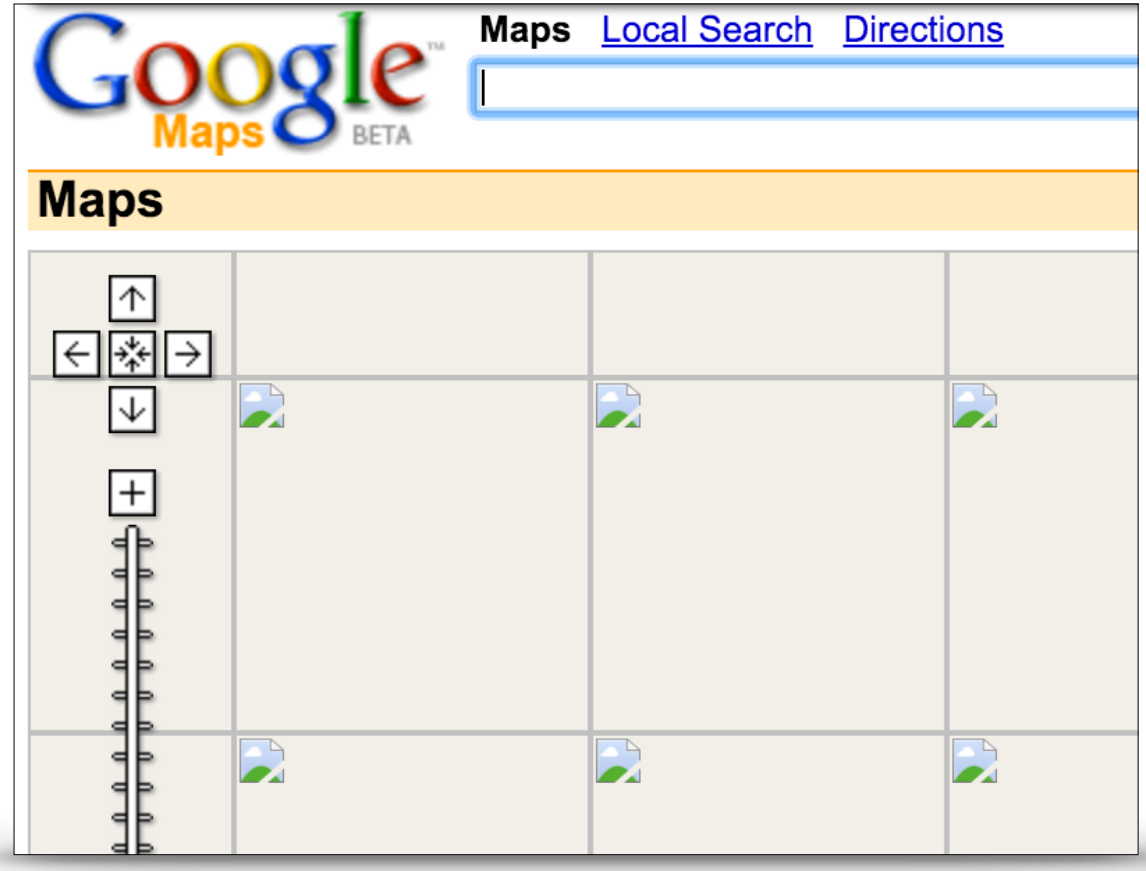
Ajax hit the masses: 2005-2006

August 26, 2006

February 2005



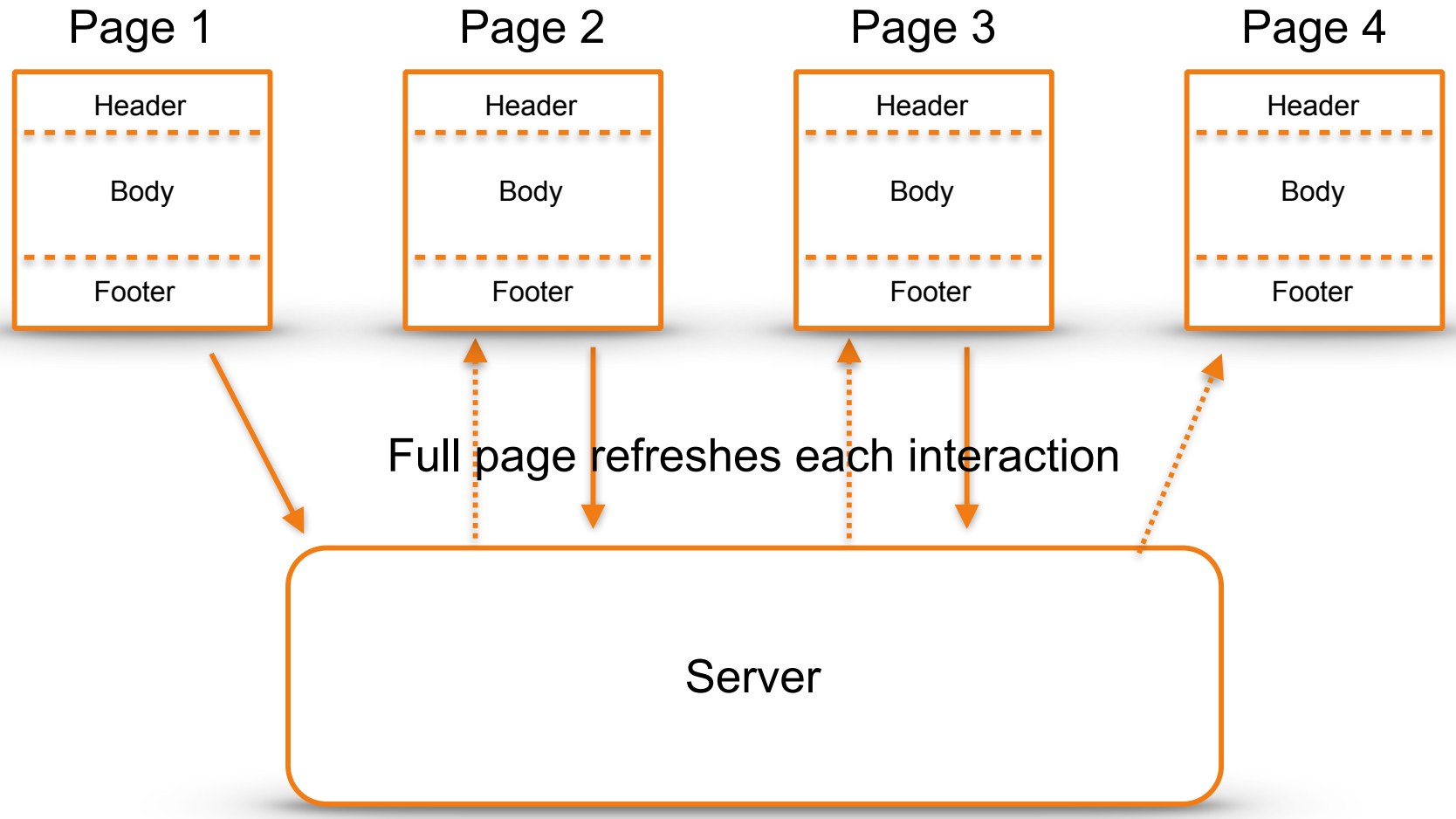
Ajax ups the game



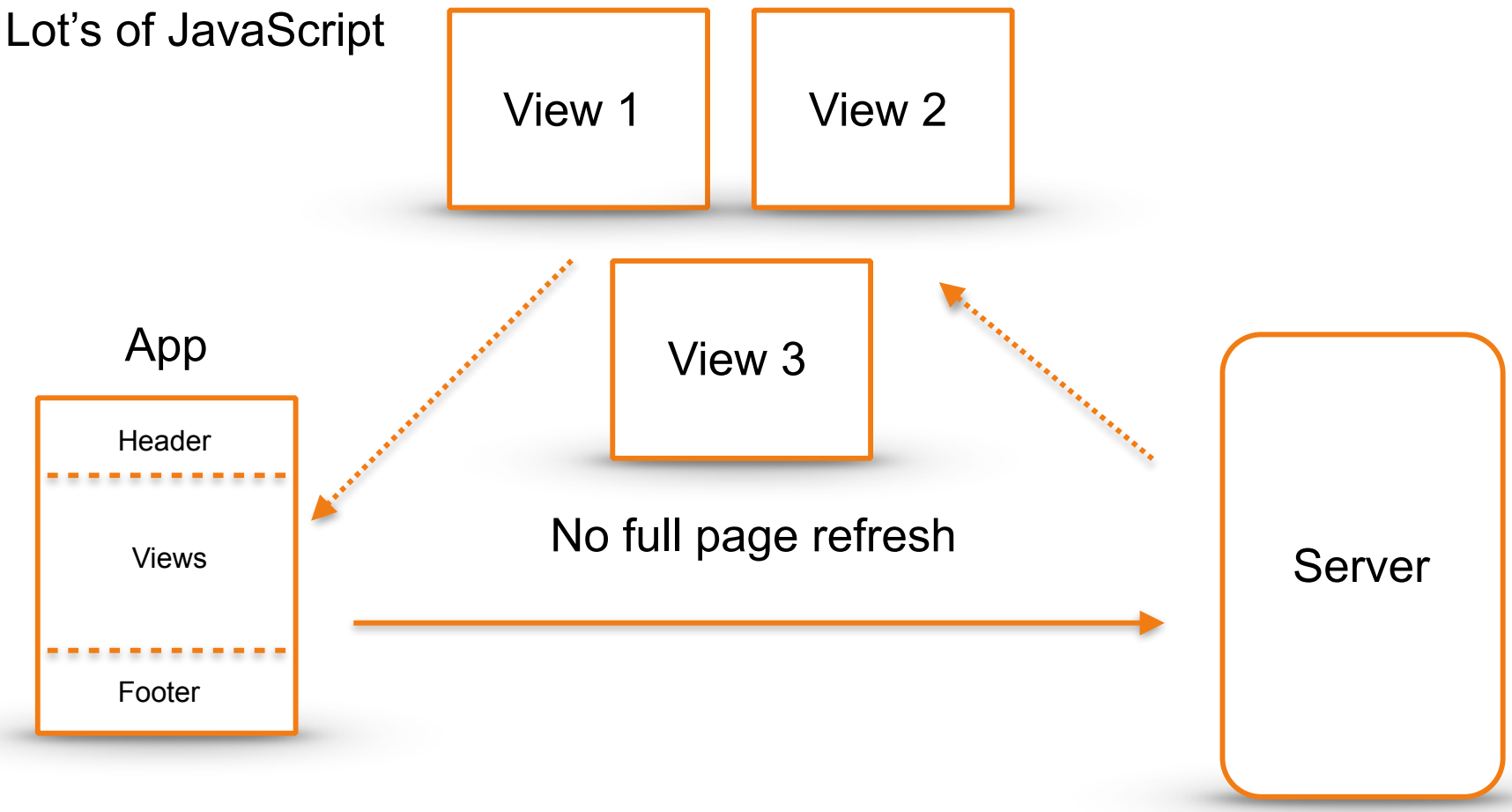
Problems

- Users wanted awesome experiences
- Users wanted web applications

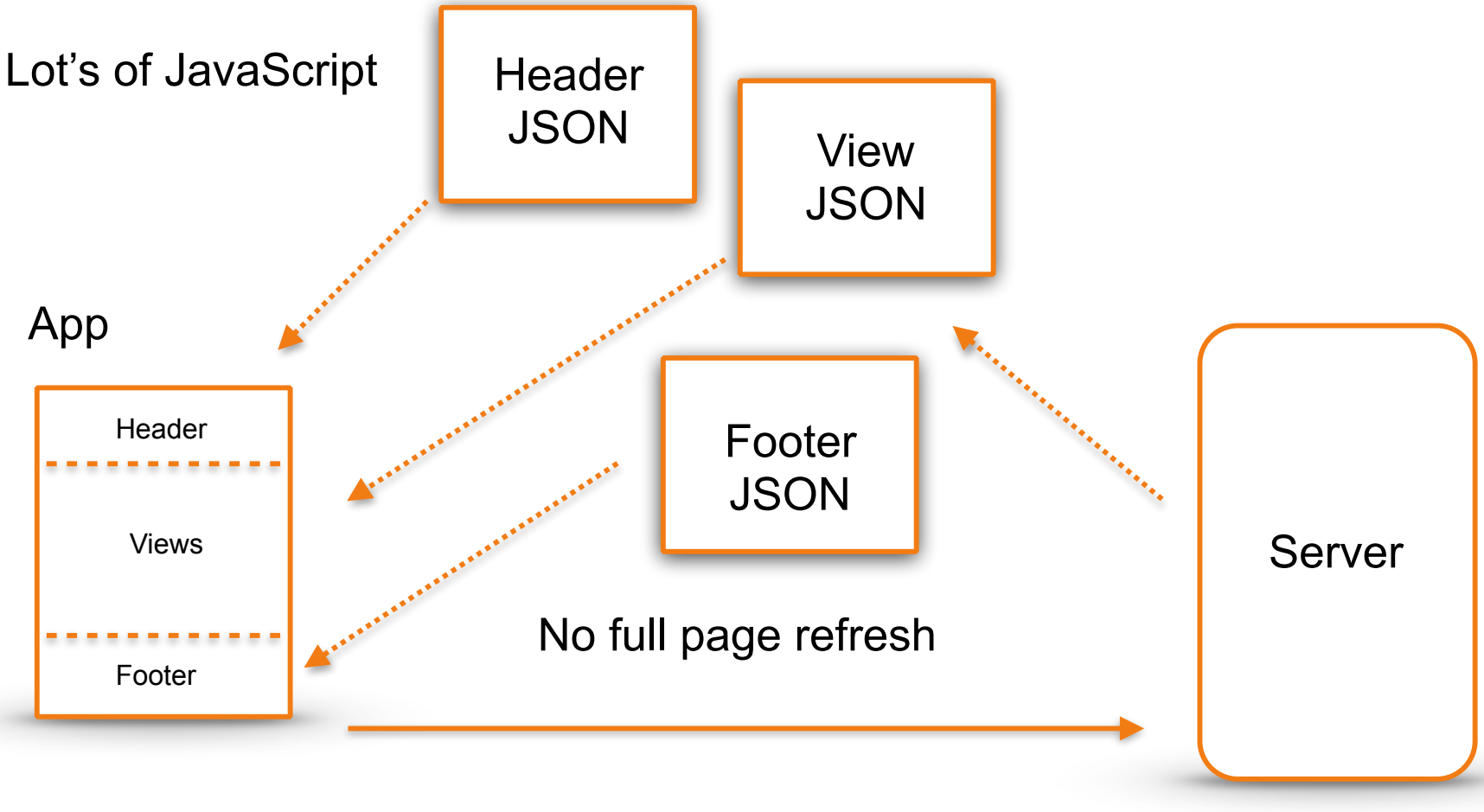
Complete page reloads



Swap view content



Swap data



Can you lift it?

Rollin' your own MVC

- Company after company began rolling home-brewed MVC concoctions
- Some companies swam
- Others began to sink

To the rescue

2010

2009

Angular 1 strengths

Develop Intelligence

- Dependency Injection
- Testing & Mocks
- Controllers, Services & Views for architecture
- Routing
- Directives with reusability
- 2-way Data Binding

Angular 1 limitations

- A stepping stone beyond jQuery
- Poor rendering performance on complex nested loops
  - Struggle with complex change management
- Steep learning curve
- Nesting directives was overly complicated

The web has evolved

- ES6 / ES7
- HTML5

Did I hear you say web components?

The pitch...

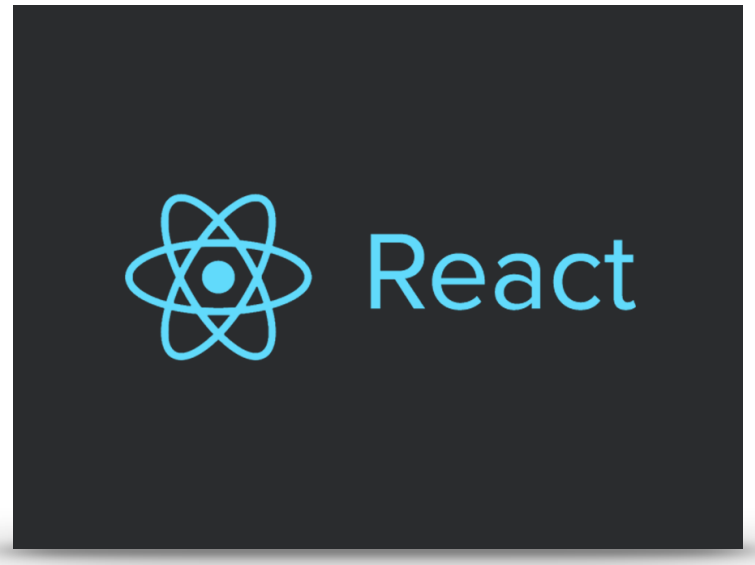
Develop Intelligence

"What if HTML was expressive enough to allow us to extend HTML so we can fill in the gaps in functionality with our own tags? Well, Web Components enable that."

- Zeno Rocha & Addy Osmani @ WebComponents.org
April 16, 2015

The move to components

React



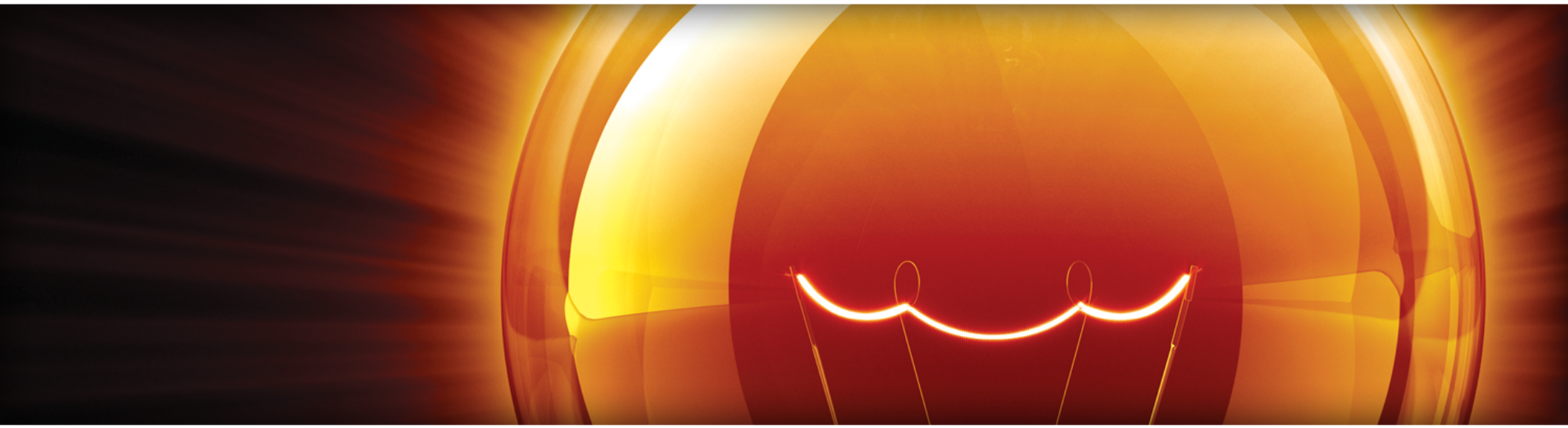
Ember 2

Angular 2

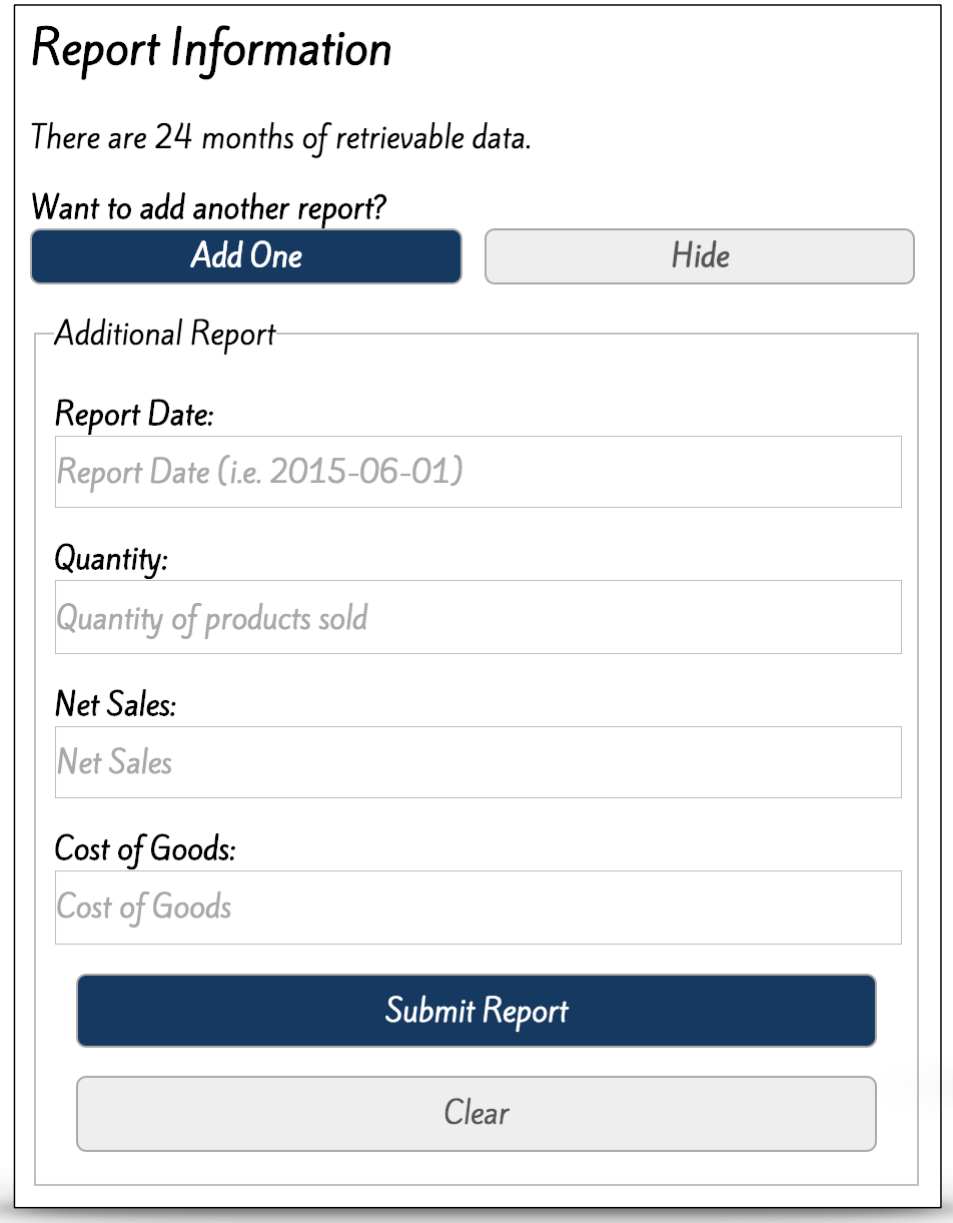
A component world view

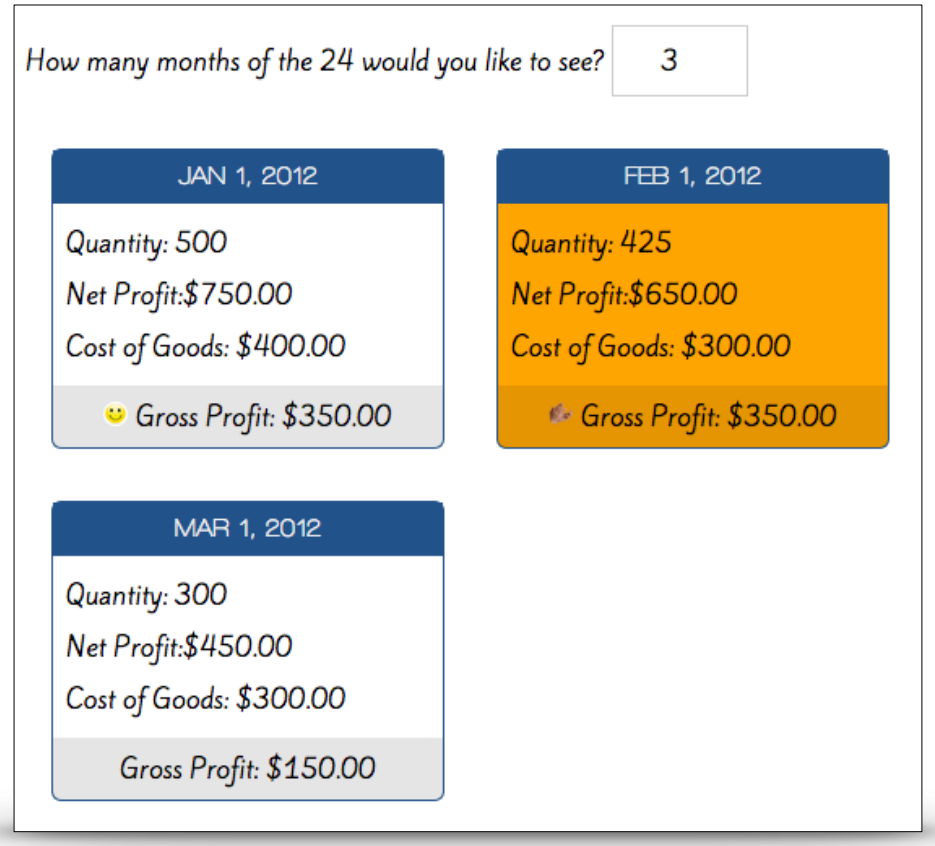
- Create application-specific HTML tags
- Encapsulated, User-Interface building blocks

Componentization

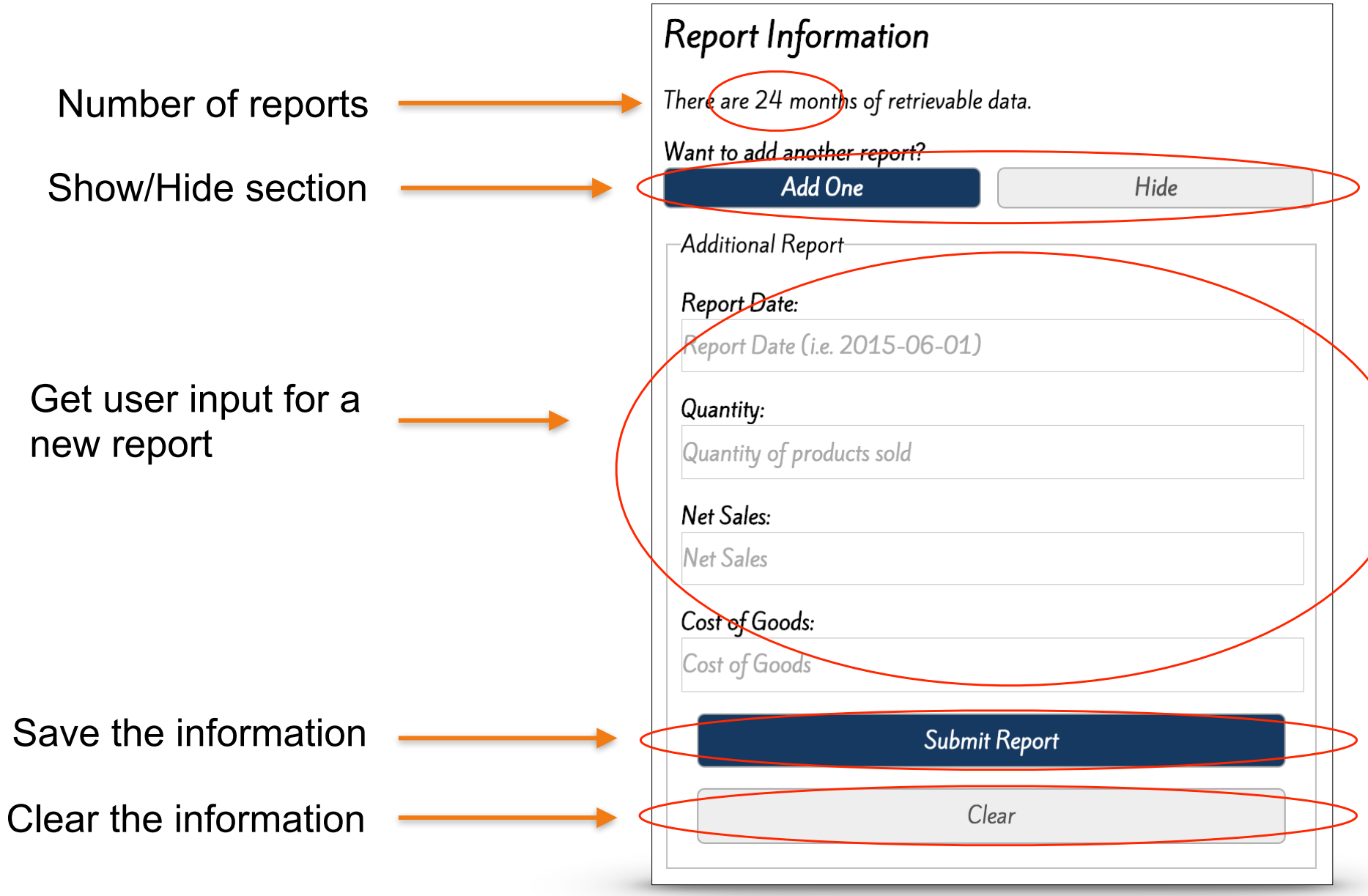


An application breakdown





An application breakdown [cont.]



An application breakdown [cont.]

Develop Intelligence

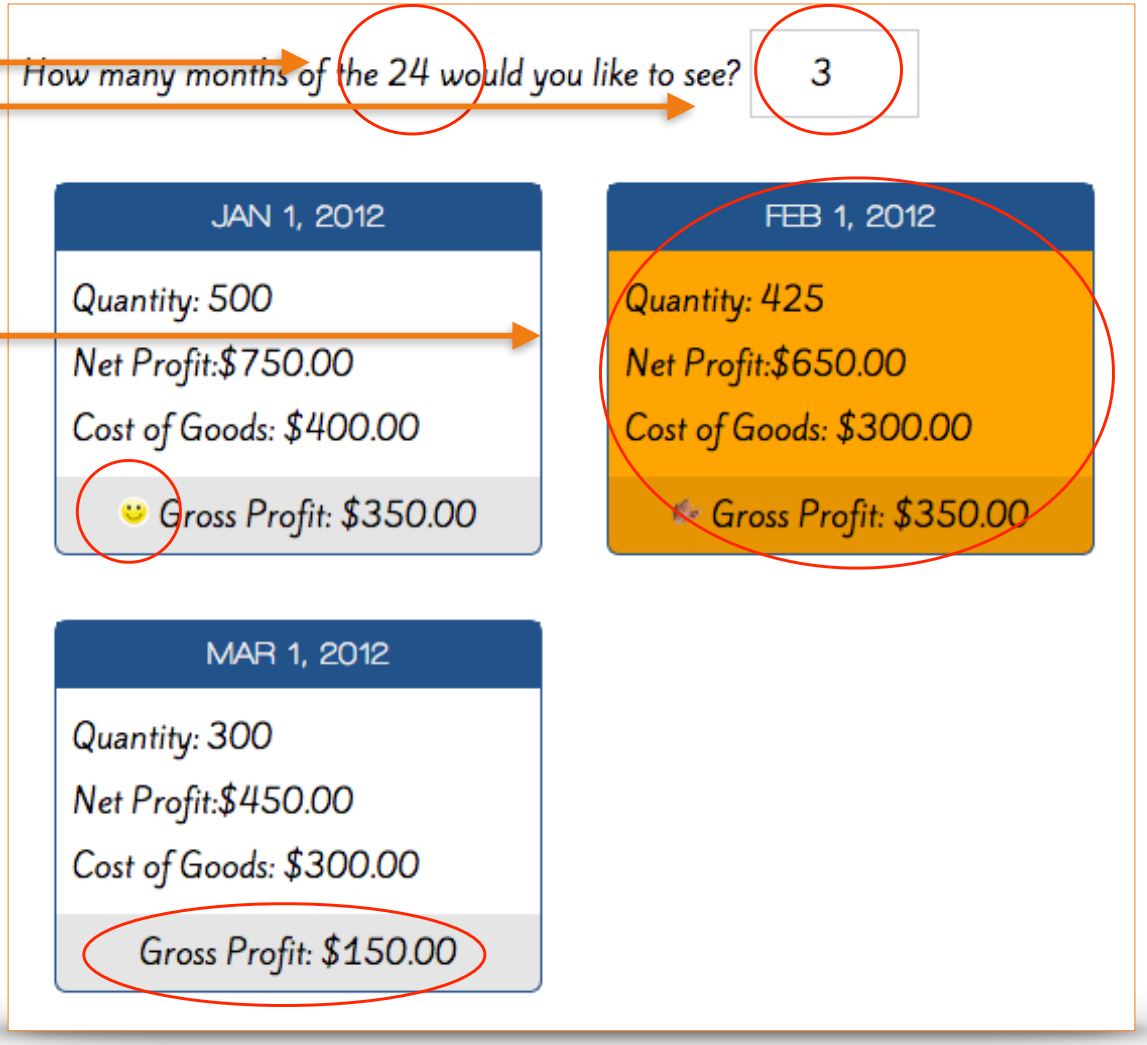
Number of reports

Filtered number of reports

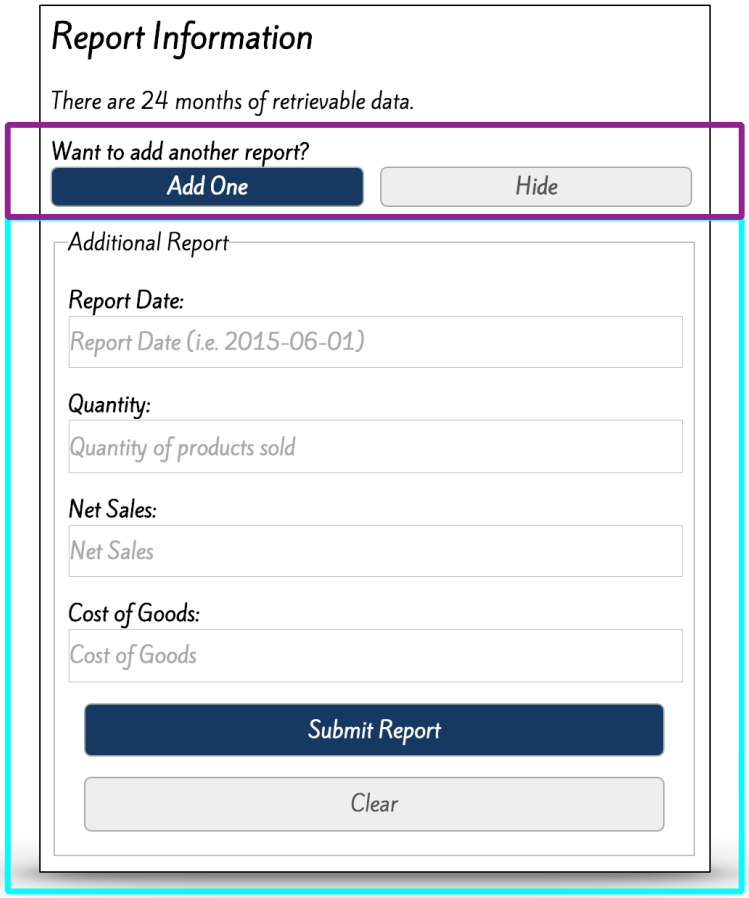
Individual report

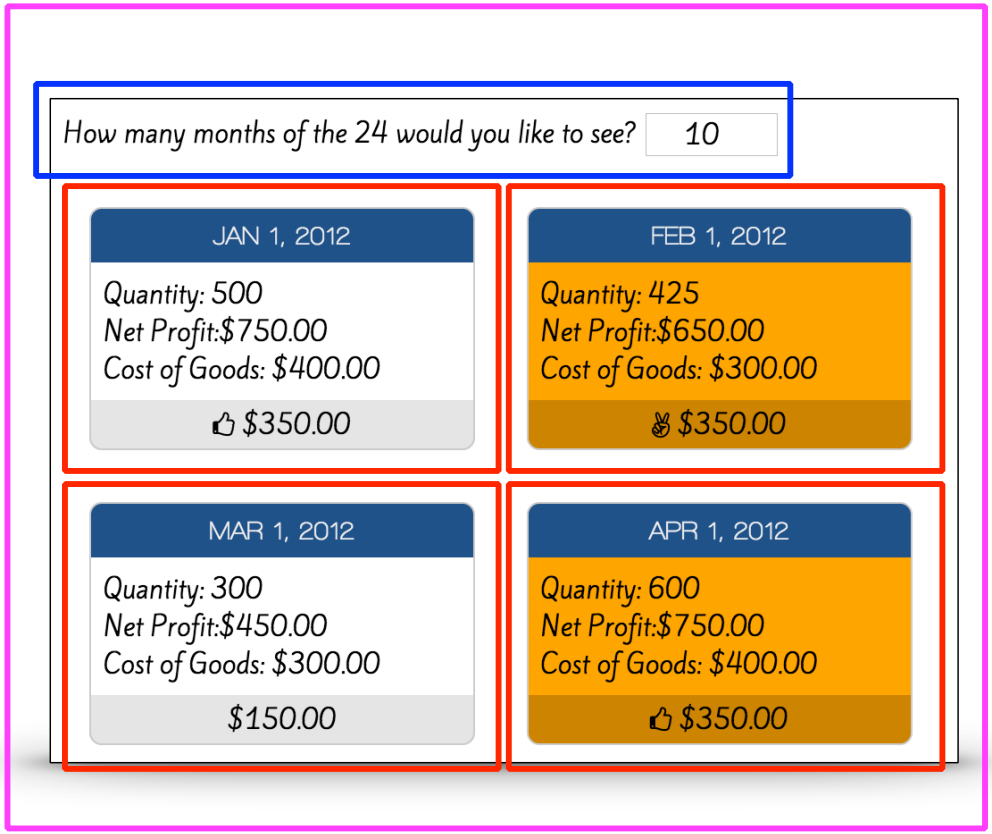
Change images when filtered number changes

Calculate gross profit and add a picture if it is over $200.00



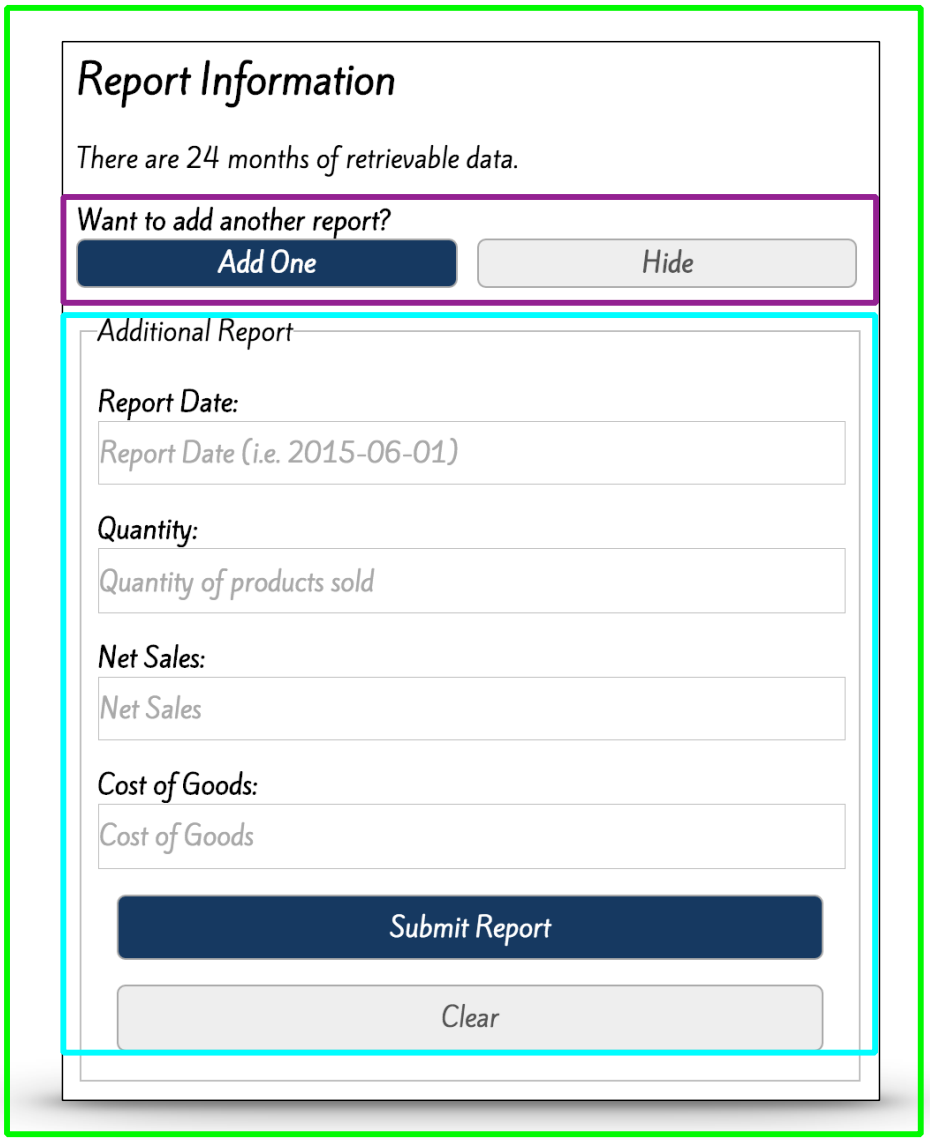
Composing components





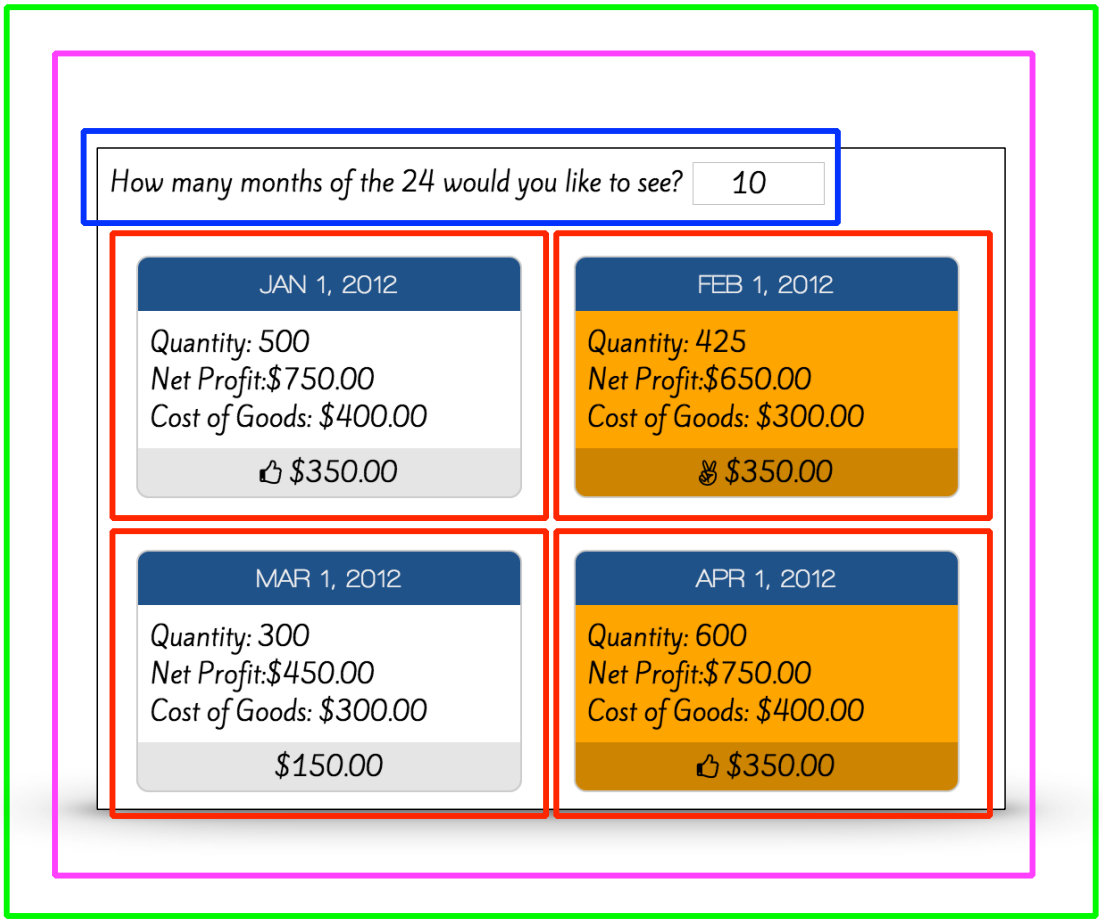
Composing components [cont.]

- Report application component
- Report form control component
- Report form component



Composing components [cont.]

- Report application Component
- Report list component
- Report filter card component
- Report card component



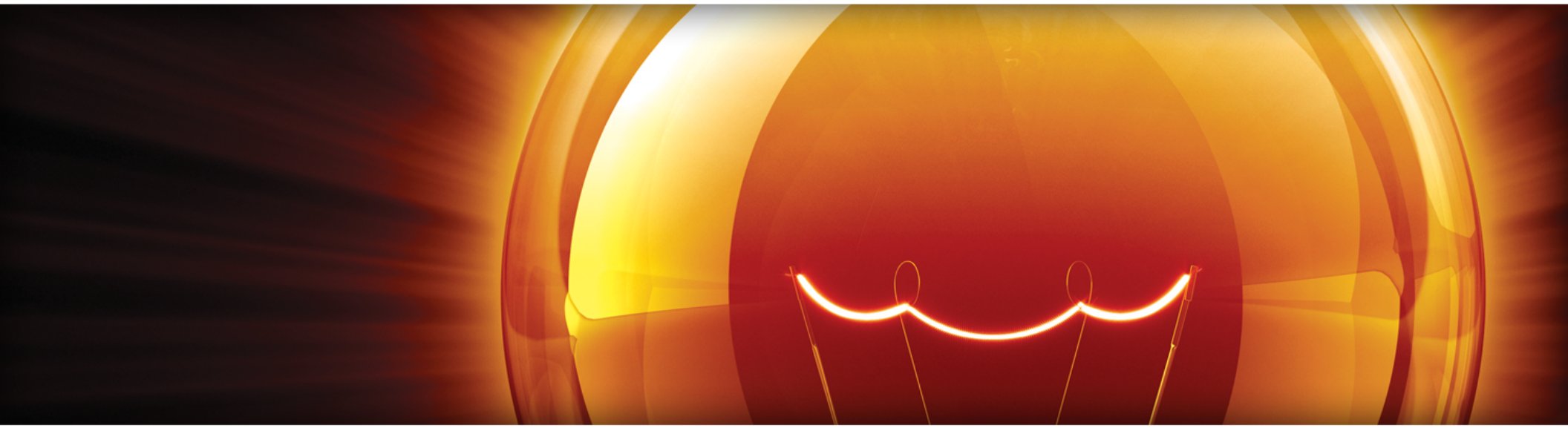
A first component

- https://angular.io/
- Let’s explore the Angular 2 QuickStart on Plunker
- What do you observe about this basic application?

Lab 1

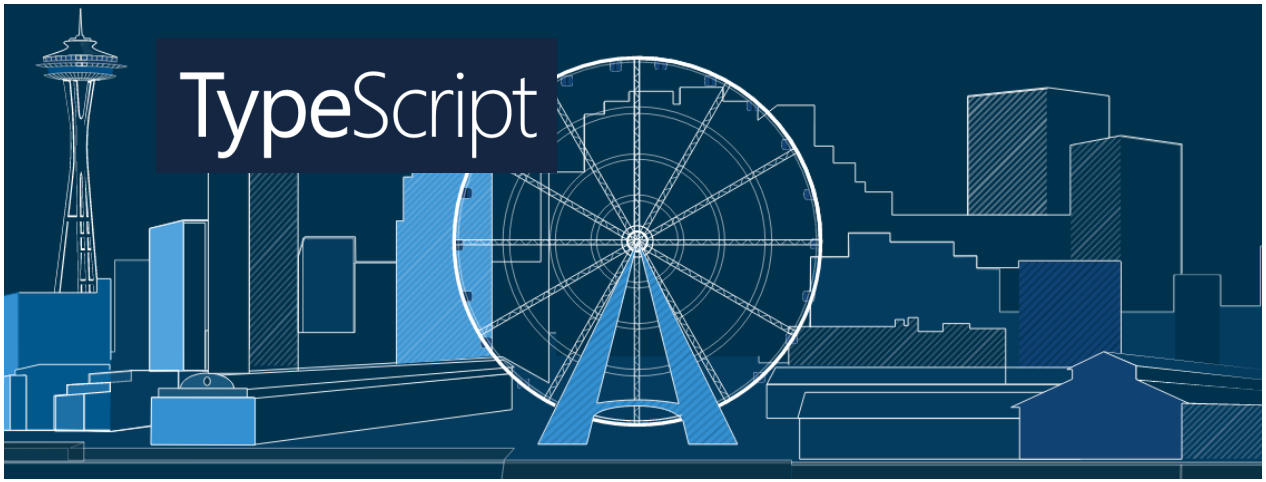
- Make the app say “Hello Angular”
- Change from initially saying “Loading…” to “Launching…”
- Change the HTML tag to <di-app> from <my-app>

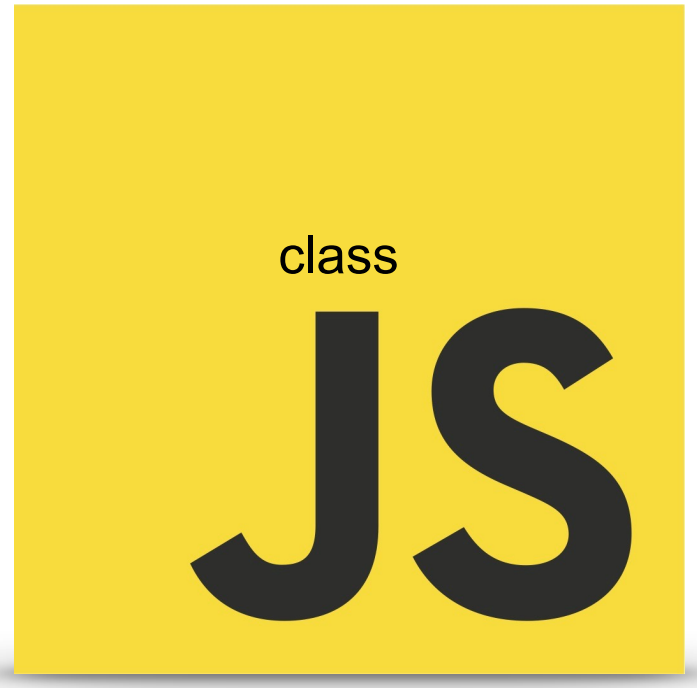
Language Foundations



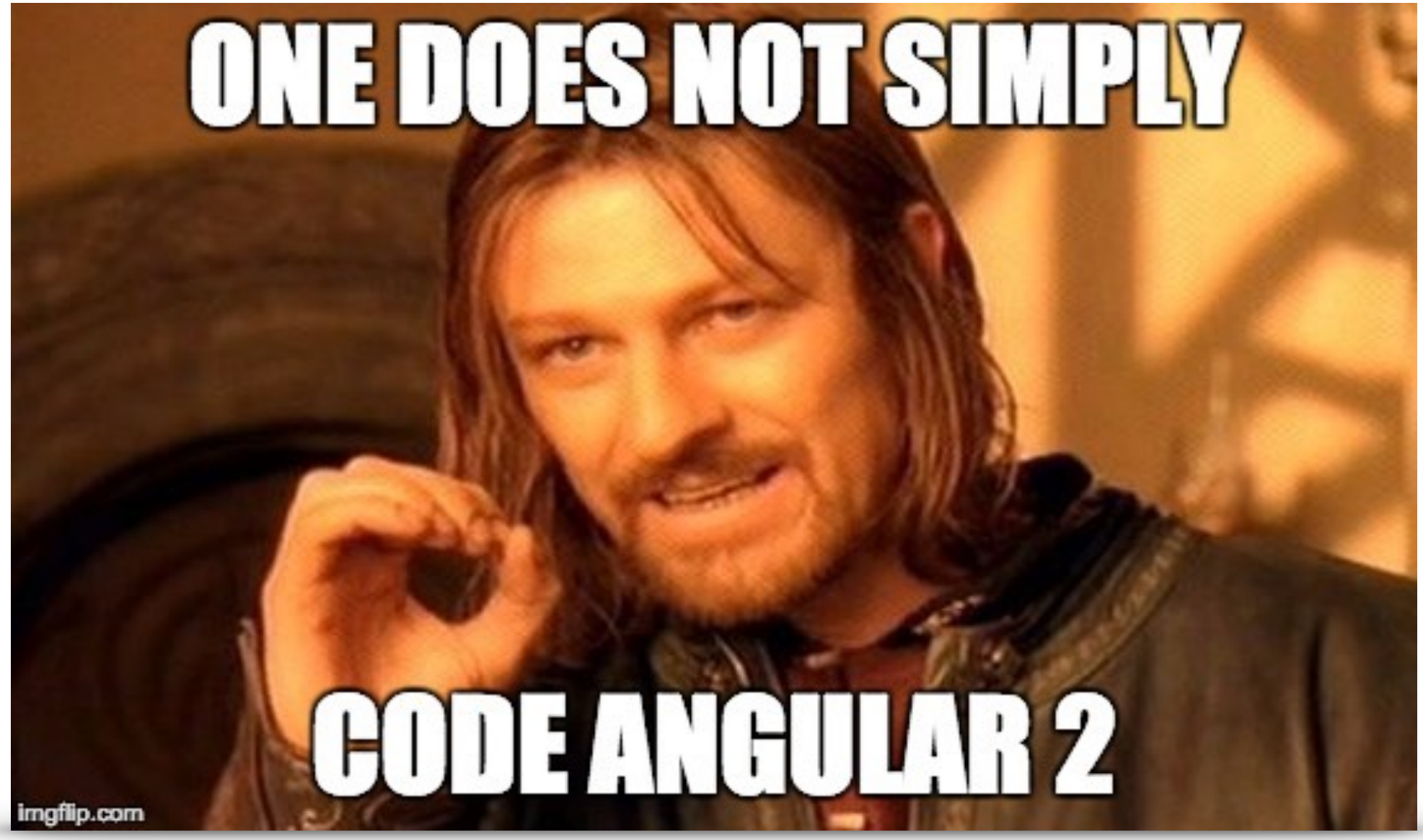
Component alchemy

- Component annotation
- Classes
- Modules

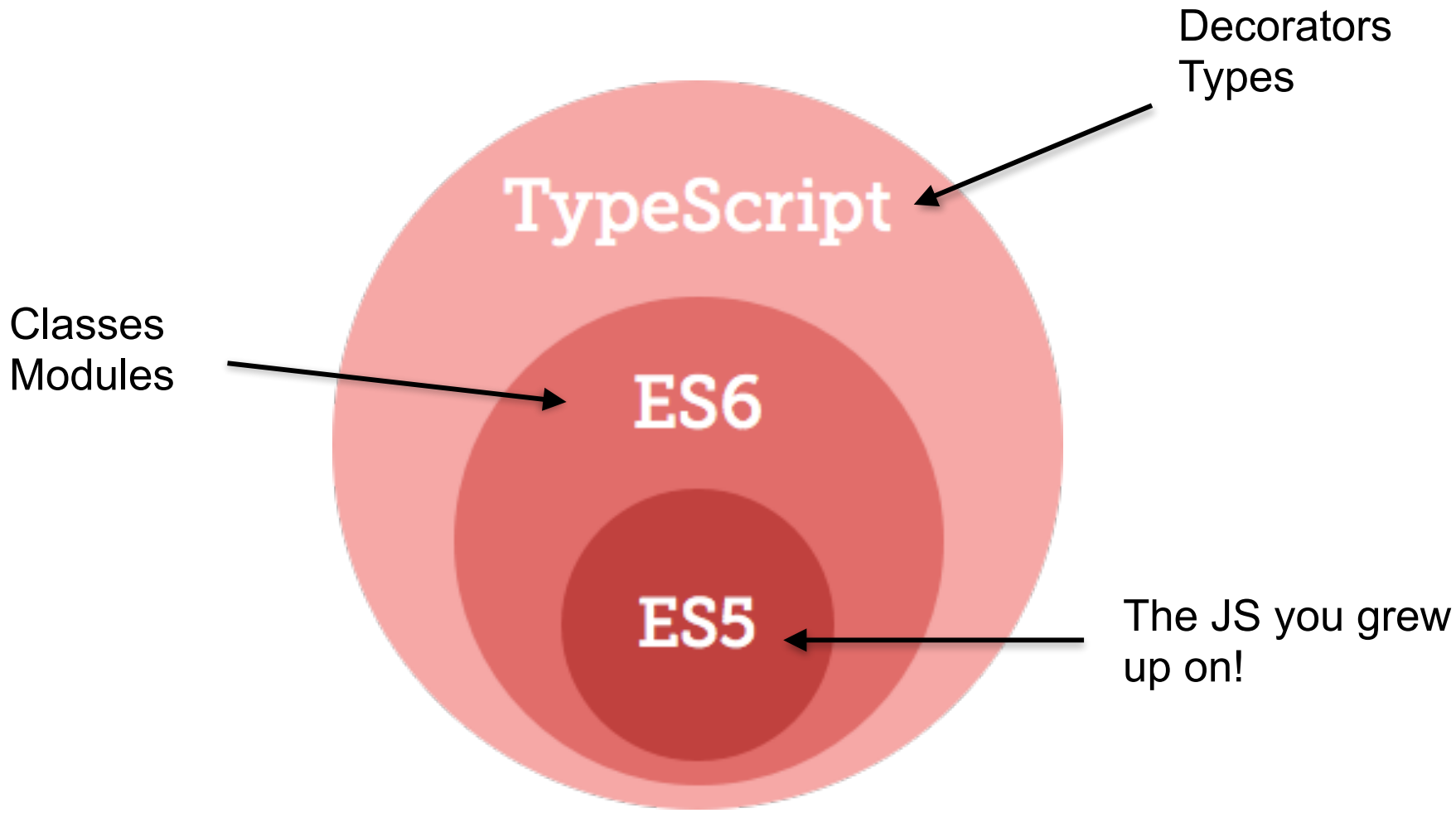




Ummm... Really?!?



Component alchemy



- The new standard in JavaScript development

- Why?
  - Because not much has changed in 6 years
  - ES5 was released in 2009 for IE9+ browsers

ES6 primer topics

- Scope
- Arrow Functions
- Back Ticks
- Classes

- More functionality
  - https://kangax.github.io/compat-table/es6/

Scope: ES5

- JS has function based scope
- No other structures in JS have scope

```
//var i is function scoped
for (var i=0; i<10; i++) {
  console.log("hello");
}

//var aNumber is function scoped
if (aVariable) {
  var aNumber = 42;
}
```

Scope: ES6

- ES6 gives block-level scoping
  - New keywords **let** and **const**

- let

```
//var i is function scoped
var aBoolean = true;

if (aBoolean) {
  let aNumber = 42;
  console.log('Inside block:' + aNumber);
}

console.log('Outside block:' + aNumber);
```

https://jsbin.com/hesuleluri/edit?html,js,console

const

- Identify a variable as a constant
  - Variable itself isn't a constant
  - Reference to the variable is a constant (i.e. the assignment)

```
//Constants are great for identifiers
const PRODUCT_KEY = '42TY889RR';

//Constants help us get rid of magic numbers
//const APR = 0.06;
```

https://jsbin.com/filure/5/edit?html,js,console

- Object properties can change with **const**

```
//Constants are great for identifiers
const config = {
  PRODUCT_KEY: '42TY889RR',
  API: '/data/product/42'
};

//Change reference: Doable
config.PRODUCT_KEY = 'hat';

//Change reference to constant object: Error thrown
config = 'something';
```

When to use what?

- **const**
  - Use const as much as possible
- **let**
  - Use when an actual variable is needed
- **var**
  - Use when all else won't do

Arrow functions

- Originated from CoffeeScript
  - Brings a more concise syntax to anonymous functions

```
// ES5 syntax
var multiply5 = function(x, y) {
  return x * y;
};

// ES6 syntax
var multiply6 = (x, y) => { return x * y; };

multiply5(3, 4);
multiply6(3, 4);
```

http://jsbin.com/meveda/3/edit?html,js,console

- Syntax for 1 parameter

```
// ES5 syntax
var splitIt5 = function(phrase) {
  return phrase.split(' ');
};

// ES6 syntax
var splitIt6 = phrase => { return phrase.split(' '); };
var splitIt6Alt = (phrase) => { return phrase.split(' '); };

splitIt5('Hello World');
splitIt6('Hello World');
splitIt6Alt('Hello World');
```

http://jsbin.com/mugavu/3/edit?html,js,console

- Syntax for 0 parameters

```
// ES5 syntax
var noArgs5 = function() {
  console.log('Hello 5');
};

// ES6 syntax
var noArgs6 = () => { console.log('Hello 6'); };
var noArgs6Alt = _ => { console.log('Hello 6 alt'); };

noArgs5('Hello World');
noArgs6('Hello World');
noArgs6Alt('Hello World');
```

http://jsbin.com/foyuke/2/edit?html,js,console

- Returning an Object Literal
  - Body needs to be parenthesized

```
// ES5 syntax
var setItem5 = function(id, name) {
  return { id: id, name: name };
};

// ES6 syntax
var setItem6 = (id, name) => ({
  id:id, name:name
});

setItem5(42, 'Bill');
setItem6(31, 'Jim');
```

http://jsbin.com/loboro/1/edit?html,js,console

- Simplifying Array functions

```
var lengths = [1, 3, 5];

// ES5 syntax
var areas5 = lengths.map(function areaOfASquare(length) {
  return length * length;
});

//ES6 syntax
var areas6 = lengths.map(lengths => length * length);

console.log("Areas ES5 " + areas5);
console.log(“Areas ES6 “ + areas6);
```

http://jsbin.com/fidobexaci/2/edit?html,js,console

Back Ticks

- Typically creating multi-line strings can be a pain
  - The **+** operator is cumbersome
  - Having a new string on each line is hard to read

```
var NEGATION = 'not';
var element = $('<section><p>This is ' + NEGATION +
  ' such a nice thing</p></section>');
```

Back Ticks [cont.]

- They simplify multi-line strings
  - They allow for templating dynamic content into strings

```
var NEGATION = 'not';
var elementContents = `
   <section>
     <p>This is ${NEGATION} such a nice thing</p>
   </section>`;
var element = $(elementContents);
```

http://jsbin.com/pikayu/1/edit?html,js,console

Classes

- The class keyword has been finally utilized within ES6
- Don’t be fooled we are still dealing with objects and prototypal inheritance
  - Constructor
  - Instance properties/methods
  - Prototype properties/methods

- Basic JavaScript constructor

```
function City() {
  //Stuff in here to construct
}

var lansing = new City();

console.log("instanceof: " + (lansing instanceof City));
console.log("constructor check: " +
  (lansing.constructor === City));
```

- ES6 constructor

```
class City {
  //Stuff in here to construct
  constructor() {
    //Set any instance variables/methods here
  }
}

var lansing = new City();

console.log("instanceof: " + (lansing instanceof City));
console.log("constructor check: " +
  (lansing.constructor === City));
```

- Basic JavaScript instance properties / methods

```
//We need to create a new City to use instance properties
function City(numOfPeople) {
  //Instance Property
  //  Every City object instance will have this property
  this.numOfPeople = numOfPeople;
}
```

- ES6 instance properties / methods

```
class City {
  //Stuff in here to construct
  constructor(numOfPeople) {
    //Instance Property
    //  Every City object instance will have this property
    this.numOfPeople = numOfPeople;
  }
}
```

- Basic JavaScript static properties / methods

```
//We could just say var City = {} if we are doing utilities
//  instead of a Constructor function
function City(numOfPeople) {
  this.numOfPeople = numOfPeople;
}

//Class/Utility method
//  This method exists only on the City object not on the
instances
City.moreHouseHolds = function (cityA, cityB) {
  if (cityA.numOfPeople > cityB.numOfPeople) {
    return cityA;
  } else {
    return cityB;
  }
}
```

- ES6 static properties / methods

```
class City {
  constructor(numOfPeople) {
    this.numOfPeople = numOfPeople;
  }

  //Class/Utility/Static method called off of Object
  static moreHouseHolds(cityA, cityB) {
    if (cityA.numOfPeople > cityB.numOfPeople) {
      return cityA;
    } else {
      return cityB;
    }
  }
}
```

- Basic JavaScript prototype methods

```
function City(numOfPeople) {
  this.numOfPeople = numOfPeople;
}

//Method on the shared Object Prototype
//  Only 1 copy exists on City.prototype property
City.prototype = {
  getNumOfPeople: function () { return this.numOfPeople; },
}

var boulder = new City(101808);
console.log('Population:' + boulder.population());
```

- ES6 prototype methods

```
class City {
  constructor(numOfPeople) {
    this.numOfPeople = numOfPeople;
  }

  //Method on the shared Object Prototype
  getNumOfPeople() {
    return this.numOfPeople;
  }
}
var lansing = new City(110000);
console.log("Number of people: " + lansing.getNumOfPeople());
```

Loosing our reference

- ES5 uses **bind()** to save “this” reference

```
function City(numOfPeople) {
  //var that = this
  this.numOfPeople = numOfPeople;
  var divisor = 3;

  var calcHouseHolds = function() {
    //return that.numOfPeople / divisor;
    return this.numOfPeople / divisor;
  }.bind(this);

  this.getHouseHolds = function () {
    return calcHouseHolds();
  };
}

var lansing = new City(110000);
console.log(lansing.getHouseHolds());
```

https://jsbin.com/ceyoce/edit?js,console

Loosing our reference [cont.]

- ES6 uses Fat Arrow to save “this” reference

```
function City(numOfPeople) {
  this.numOfPeople = numOfPeople;
  var divisor = 3;

  //Captures the “this” value of the enclosing context
  var calcHouseHolds = () => {
      return this.numOfPeople / divisor;
  };

  this.getHouseHolds = function () {
    return calcHouseHolds();
  };
}

var lansing = new City(110000);
console.log(lansing.getHouseHolds());
```

https://jsbin.com/lucasi/edit?js,console

Lab 2

- Change code from ES5 to ES6

- Start with the code @ JSBin
  - http://jsbin.com/bupehaz/edit?js,console

- Class
- Fat Arrow
- Back ticks
- let / const

Why Modules?

- JavaScript applications grow in complexity
- Need reusable functionality
- Want encapsulation

Module types

- IIFE
- Module/Revealing Module Pattern
- AMD & CommonJS
- ES6

- Pattern used to build modules

```
//Douglas Crockford's suggestion
(function () {
  var speak = 'yelp';
  console.log(speak);
}());

//This works fine also
(function () {
  var speak = 'yelp';
  console.log(speak);
})();
```

```
//Syntax error: a declaration
function () {
  console.log("yelp");
}();

//You meant: an expression
var aFunction = function () {
  console.log("yelp");
}();
```

Revealing Module Pattern

Develop Intelligence

```
var aModule = (function () {
  var time = 8;
  var doWork = function doWork() {
    console.log("working " + time + " hours");
  };
  var doMoreWork = function doMoreWork() {
      time += 2;
      console.log("working more " + time + " hours");
  };
  return {
    doWork: doWork,
    doMoreWork: doMoreWork
  };
}());

console.log("time: " + aModule.time);  //Will this work?
aModule.doWork();
aModule.doMoreWork();
aModule.doMoreWork();
```

http://jsbin.com/gaxinuc/edit?html,js,console

AMD and CommonJS

- Module syntax
- Need a module loader
  - RequireJS, Browserify, SystemJS or Webpack

- AMD
  - **define**, **require**
- CommonJS
  - **require, exports**

ES6 Modules

- Module syntax built into JavaScript

- Currently no browsers support these :(
- We need a transpiler & module loader

ES6 Modules [cont.]

- export
  - Defines the API of application module
- import
  - Imports dependencies (i.e. required modules)
  - Every time we need a module we will require it

import

- Requiring modules
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

```
//lemonade.js file
//  Importing the 'insert' method from the 'date' module
import {insert} from './date';
//import * as date from './date';

//Main functionality
//  Use the imported 'insert' method
//    without the 'date' namespace
insert('header h1');
//date.insert('header h1');
```

- Creating modules
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export

```
//Date module: date.js file

//Including the NPM package jquery for this Date module
import $ from 'jquery';

export function insert(elementSelector) {
    //DO stuff
}
```

- Why?
  - Developing large scale applications in JavaScript can be a mess
  - Static typing aides in IDE support and cleaner language syntax

- No browser support
  - Need Typescript compiler

TypeScript primer topics

- Typing
- Decorators

- Other Typescript functionality
  - Interfaces, mixins, generics, namespaces, enum

- API
  - https://www.typescriptlang.org/docs/handbook/basic-types.html
- Playground
  - https://www.typescriptlang.org/play/

Typing

```
//TypeScript
let aDecimal: number = 6;
//JS
//  var aDecimal = 6;

//TS
let isFormVisible: boolean = true;
//JS
//  var isFormVisible = true;

//TS
let aName: string = “Kamren";
//JS
//  var aName = “Kamren";

//TS
let ages: number[] = [3, 8, 10];
//JS
//  var ages = [3, 8, 10];
```

Typing [cont.]


```
//TypeScript
let notSure: any = 4;
notSure = "maybe a string instead”;
//JS
//  var notSure = 4;
//  notSure = "maybe a string instead”;

//TS
function warnUser(): void {
    alert("This is my warning message");
}
//JS
function warnUser() {
    alert("This is my warning message");
}
```

Decorators

- Decorators allow for annotations
  - https://www.typescriptlang.org/docs/handbook/decorators.html
  - Currently in the process of being adopted by JavaScript:

- Angular 2 uses annotations to decorate classes as components

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: '<h1>My First Angular 2 App</h1>'
})
export class AppComponent { }
```

SystemJS

- Module loader
  - https://github.com/systemjs/systemjs
  - Allows for browsers to be able to consume modules
  - Similar to RequireJS

- How to install?
  - **npm install systemjs --save-dev**

- Install
  - **npm install -g typescript@rc**
  - check version with **tsc -v**

- Install Angular 2 TypeScript Snippets for VS Code
  - Angular 2 TypeScript Snippets from John Pappa
  - type **ext install Angular2** into the Command Palette
  - Don’t forget to enable it

Using Typescript & ES6 in the Browser

- Install Typescript and SystemJS plugin
  - npm install plugin-typescript@4.0.16 --save-dev
  - Automatically installs Typescript for app use
  - Allow the transpiring to happen real-time with SystemJS

- Install typings
  - https://github.com/typings/typings
  - **npm install typings -g**
- Install jQuery typing globally
  - **typings install dt~jquery --global —save**
  - Do this is within the folder

Lab 3

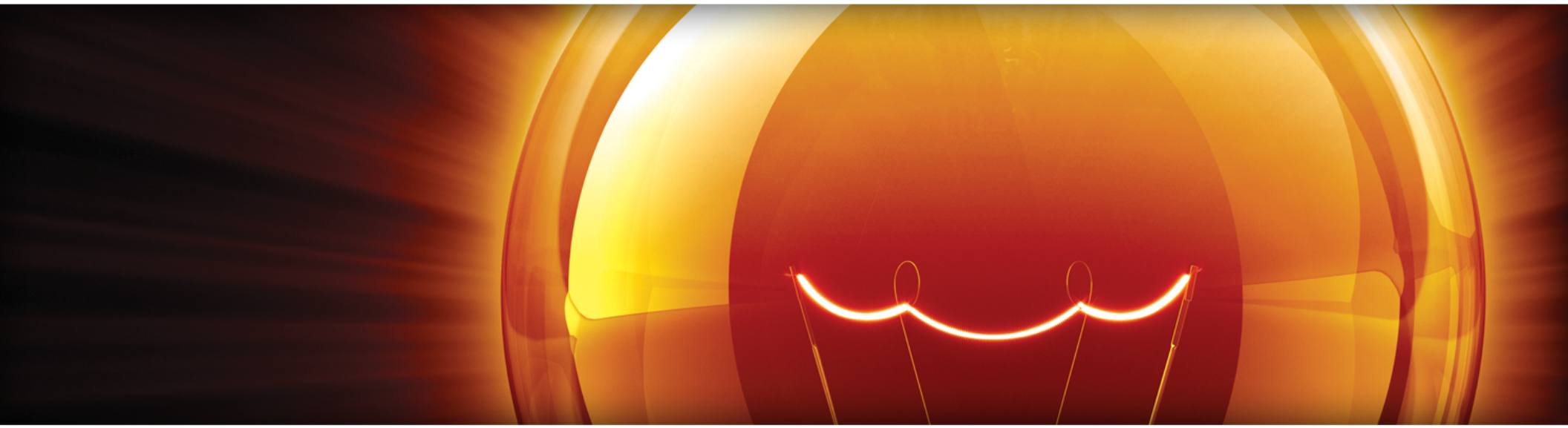
- Create ES6 modules w/ Typescript & SystemJS
  - Start with Lab 3 starter
  - Code into ES6 modules
  - Use **npm install** to get your dependencies

- Get index.html to load the proper JavaScript files
- Create a main.ts file
- Create a productKey.ts

Lab 3b

- Add typing to ES6 class
  - Go to previous JSBin and make the changes

Angular 2 componentization



- API
  - https://angular.io/docs/ts/latest/api/
- Browser Support
  - https://angular.io/docs/ts/latest/guide/browser-support.html
- Polyfills
  - Angular provides suggestions about what polyfills are needed to get IE9+ working

A component world view

- Simple … cohesive … composable … components

Demo

- Remember our Hello Angular on Plunker
  - http://plnkr.co/edit/6sP5NfiXWejhY414fKQR

A cohesive component

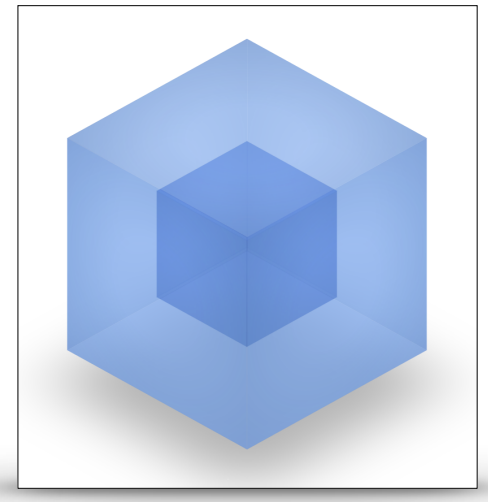
TypeScript
Component annotation

```
import { Component } from '@angular/core';

@Component({
  selector: 'di-app',
  template: '<h1>Hello Angular</h1>'
})
export class AppComponent { }
```

ES6 class definition

Angular CLI

- We could build all the configuration for Angular 2… buy why?
- A command line interface for Angular
  - Built off of Ember's CLI
  - Now uses Webpack rather than SystemJS



- Install
  - **npm install angular-cli@latest -g**

Our 2nd Angular App

- Create a project
  - **ng new demo_spitfire**

- Make sure it works
  - **ng serve**
  - Go to http://localhost:4200/

Using model data

- How do we get dynamic data into our application?

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works!';
}
```

```
<h1>
  {{title}}
</h1>
```

Model data

Generating a component

- Use Angular CLI to generate a component
  - **ng generate component hello-world**

- What the CLI gives us
  - **hello-world.component.css**
  - **hello-world.component.html**
  - **hello-world.component.ts**
  - **hello-world.component.spec.ts**

More on model data

- How do we change to “Hello World,” with **World** being model data?

hello-world.component.html

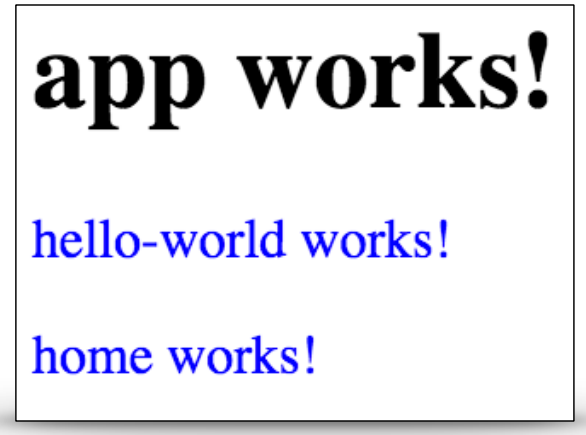
```
<p>
  hello-world works
</p>
```

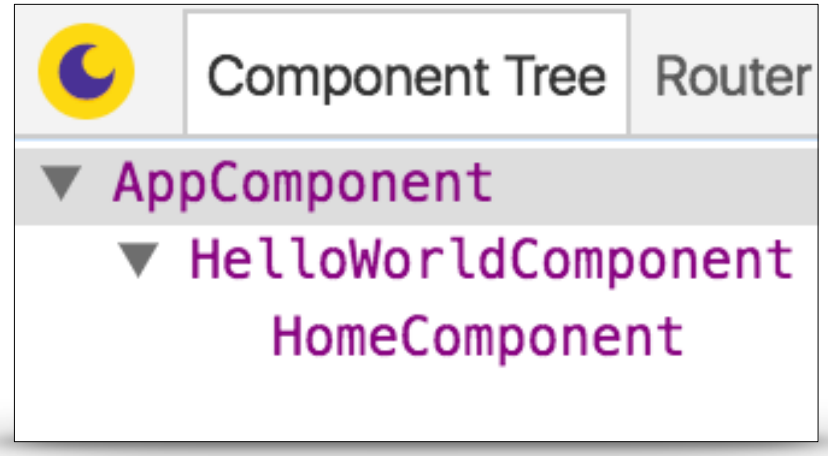
hello-world.component.ts

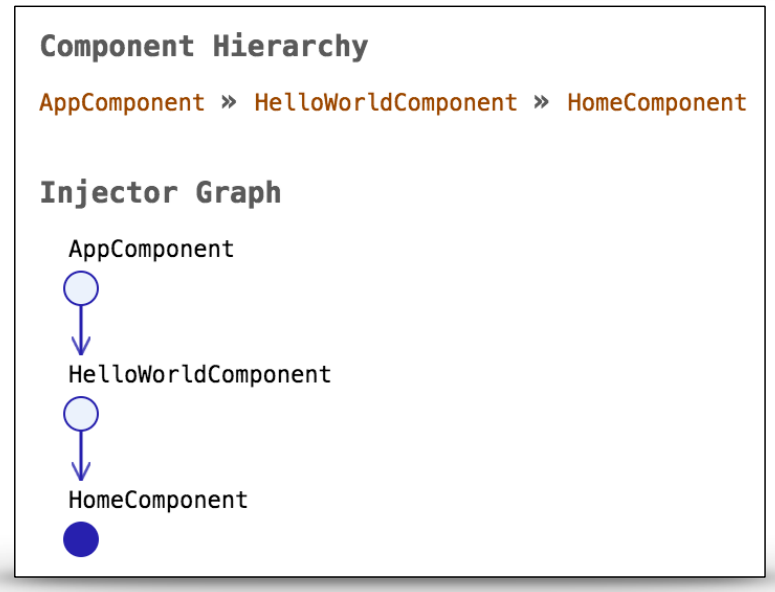
```
export class HelloWorldComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

- Angular 2 Chrome Dev Extension
  - https://augury.angular.io/

- Give visibility into the component tree







Inspect the browser

- Where does the CSS get placed from the build?
- What JavaScript files are created?
- Where can we actually see the code we wrote?

The module

- What do you observe

app.module.ts

Decorator for a module

Components within module

Modules needed for this module

Root component

```
@NgModule({
  declarations: [
    AppComponent,
    HelloWorldComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The module

Develop Intelligence

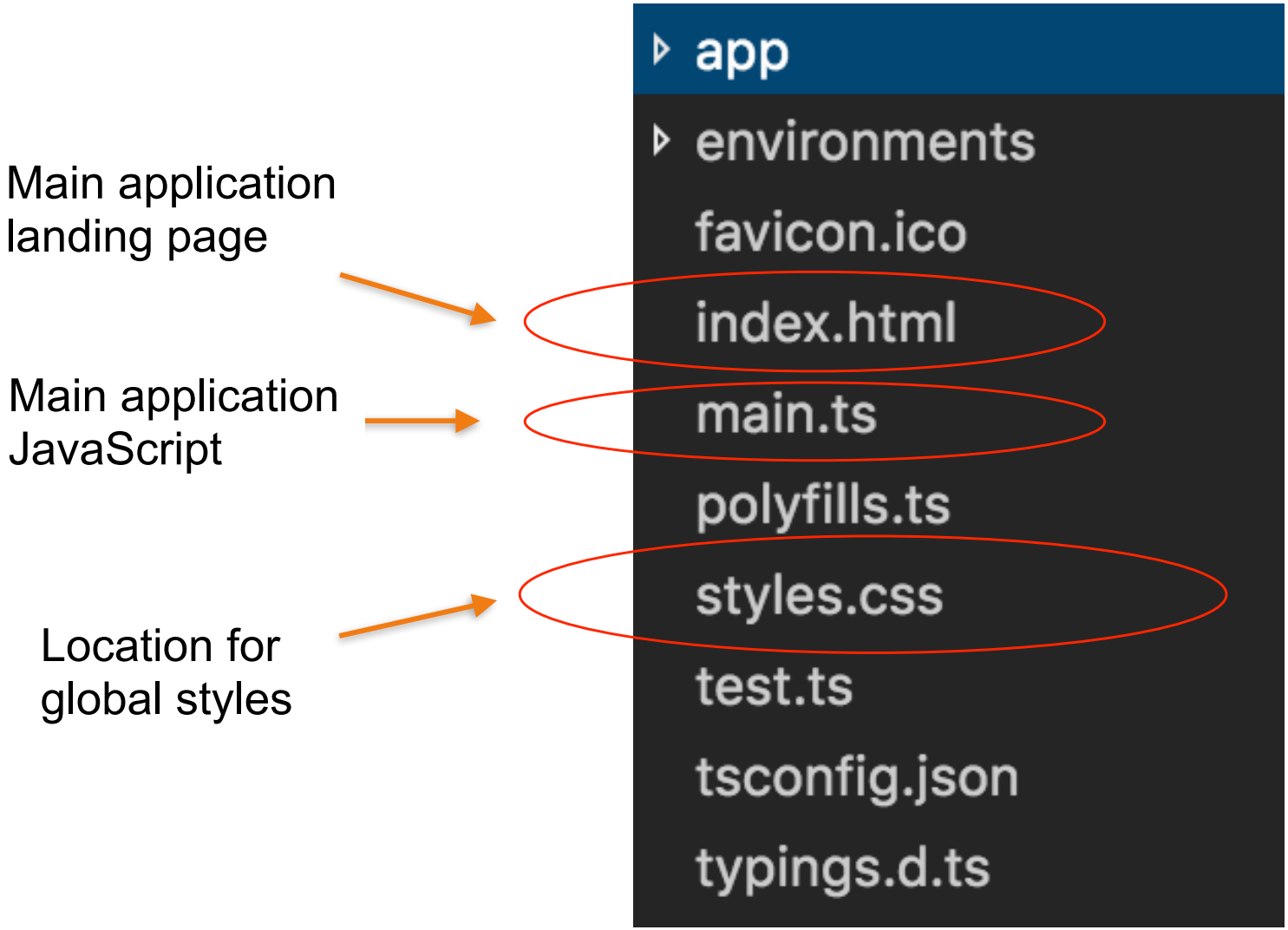
Manual import of modules

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import { HelloWorldComponent } from './hello-world/hello-world.component';
import { HomeComponent } from './home/home.component';
```

Manual import of components

Additional bootstrapping



The main

Develop Intelligence

main.ts

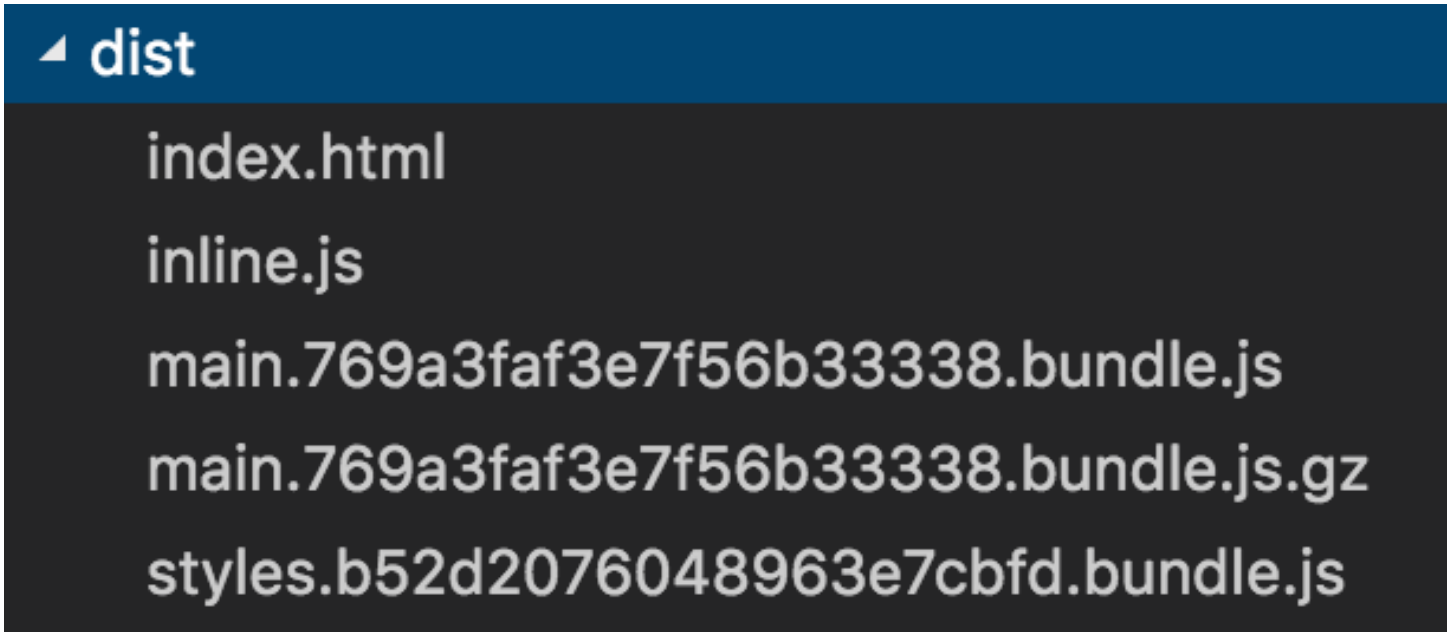
```
import './polyfills.ts';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { enableProdMode } from '@angular/core';
import { environment } from './environments/environment';
import { AppModule } from './app/';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Kicking off the application

Quick production build

- Build for production
  - **ng serve --prod**

- Uglify code
- Tree shaking

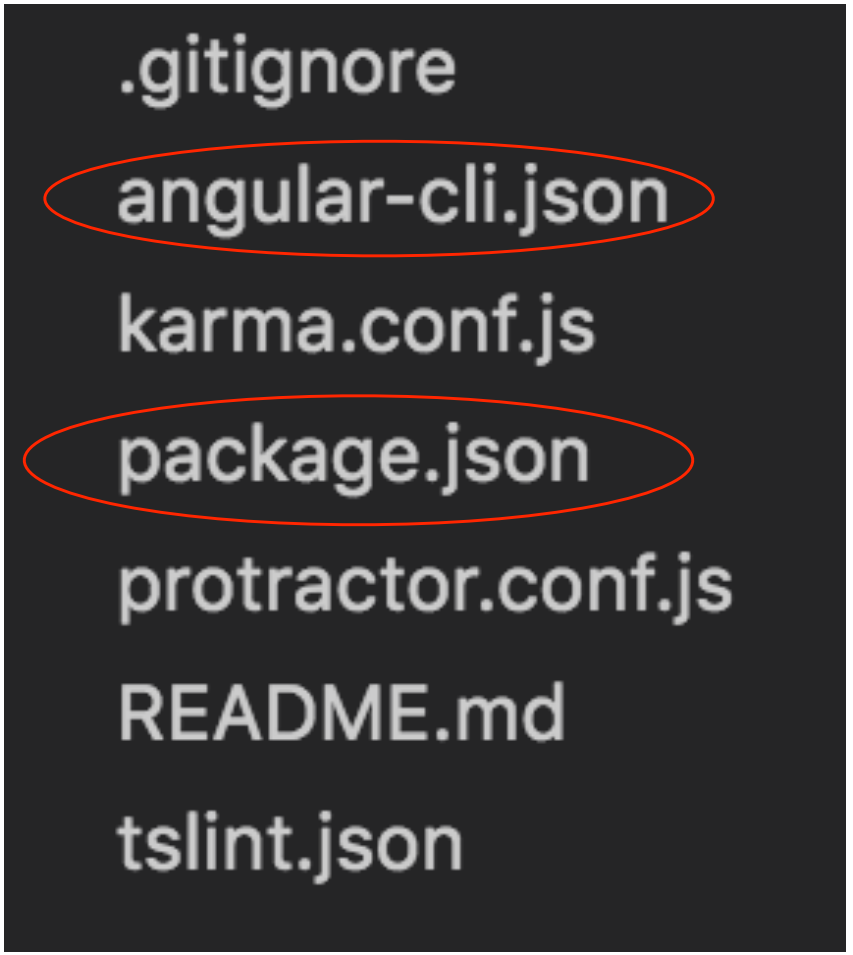


More configuration

Develop Intelligence

Configuration for Angular CLI

Configuration for Node



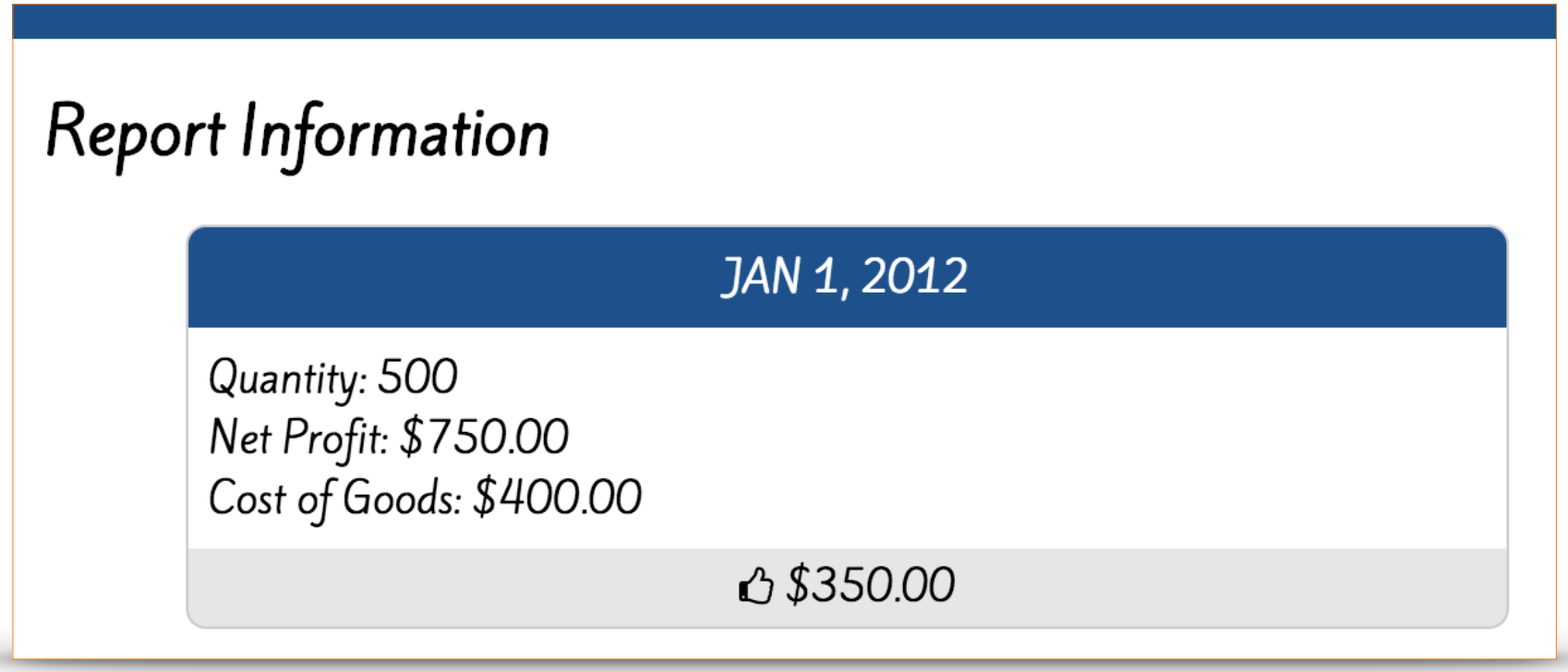
Lab 4

- Start building a report

- Create a header component
- Utilize component specific CSS

- Use the provided CSS as a start
  - Specific the **.title** classes
  - General CSS: @font-face, #content, body

- Add a component



- Use the provided CSS as a start
  - Specifically the **.report** classes
  - Use font-awesome for the graphic:
    - http://fontawesome.io/icons/

Pipes

- Pipes are filters from Angular 1
- Allow model data to have a different look in the view

- A few different built-in pipes
  - Date
  - LowerCase
  - UpperCase

- Pipes
  - https://angular.io/docs/ts/latest/api/
  - https://angular.io/docs/ts/latest/guide/pipes.html

Pipes [cont.]

- How could we remove the **W** in World?

- How could we only show **orl** in World?

Data Binding

- We have seen data binding…

- Other data binding?

Data Binding [cont.]

- Binding an attribute
  - [target]="expression"

home.component.html

```
<p [style.background]="'orange'">
  home works!
</p>
```

- View to model
  - (target) = “statement”

hello-world.component.html

```
<p>
  Hello World
  <button
    (click)="removeMe()"
  >Remove</button>
<p>
```

Directives

- Angular 1 had lots of directives
- Angular 2 removes most of those with its data binding

- Directive categories
  - Component
  - Structural
  - Attribute

Structural Directive

- ngIf … remove / add DOM elements

```
<p>
  Hello World
  <button
    (click)="removeMe()"
  >Remove</button>
  <app-home
    *ngIf="isRemoved"
  ></app-home>
</p>
```

Structural Directive [cont.]

- ngFor … the ng-repeat alternative

```
<ul>
  <li *ngFor="let city of cities">{{city.name}} </li>
</ul>
```

```
export class HelloWorldComponent implements OnInit {
  cities: any[];

  constructor() {
    this.cities = [
      { name: 'Boulder', population: 103000 },
      { name: 'Colorado Springs', population: 439886 },
      { name: 'Denver', population: 649495 },
      { name: 'Fort Collins', population: 152061}
    ];
  }

}
```

Attribute Directive

- ngStyle … adding styles to an element
- How do we set one style?

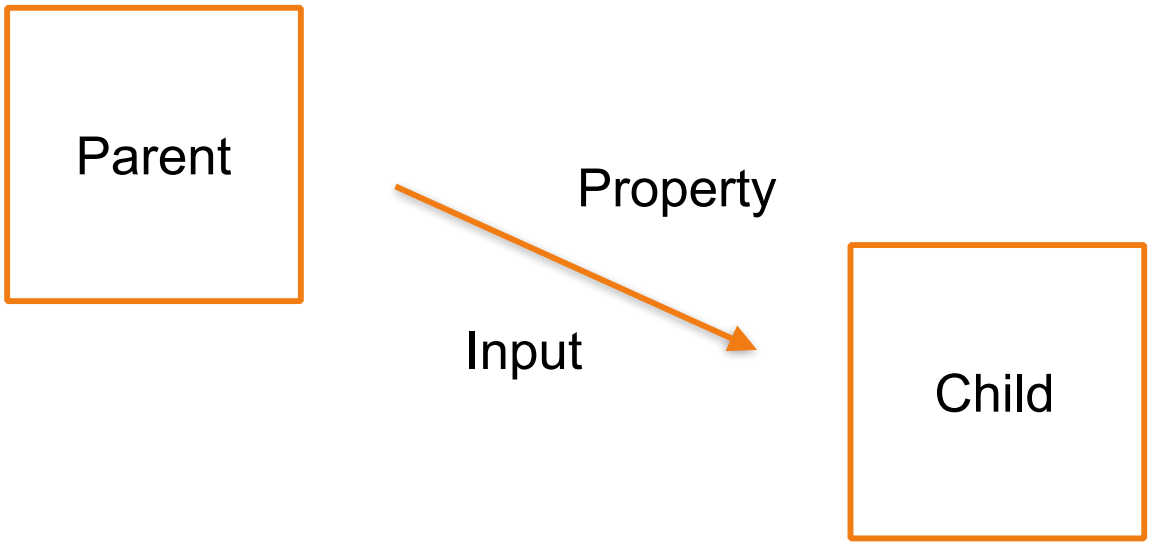
- **ngStyle**s let's us set multiple styles at once

```
<mark [ngStyle]="setStyles()">Read only :(</mark>
```

```
setStyles() {
  return {
    'font-style': this.isReadOnly ? 'bold' : 'normal',
    'color': this.isReadOnly ? 'gray' : 'black'
  };
}
```

Inputs

- We are going to need to pass data from parent to child
- Think of it as parent to child communication
- A one-way data flow



- Parent has a property the child needs

```
export class HelloWorldComponent implements OnInit {

  meaningOfLife: number;

  constructor() {
    this.meaningOfLife = 42;
  }

}
```

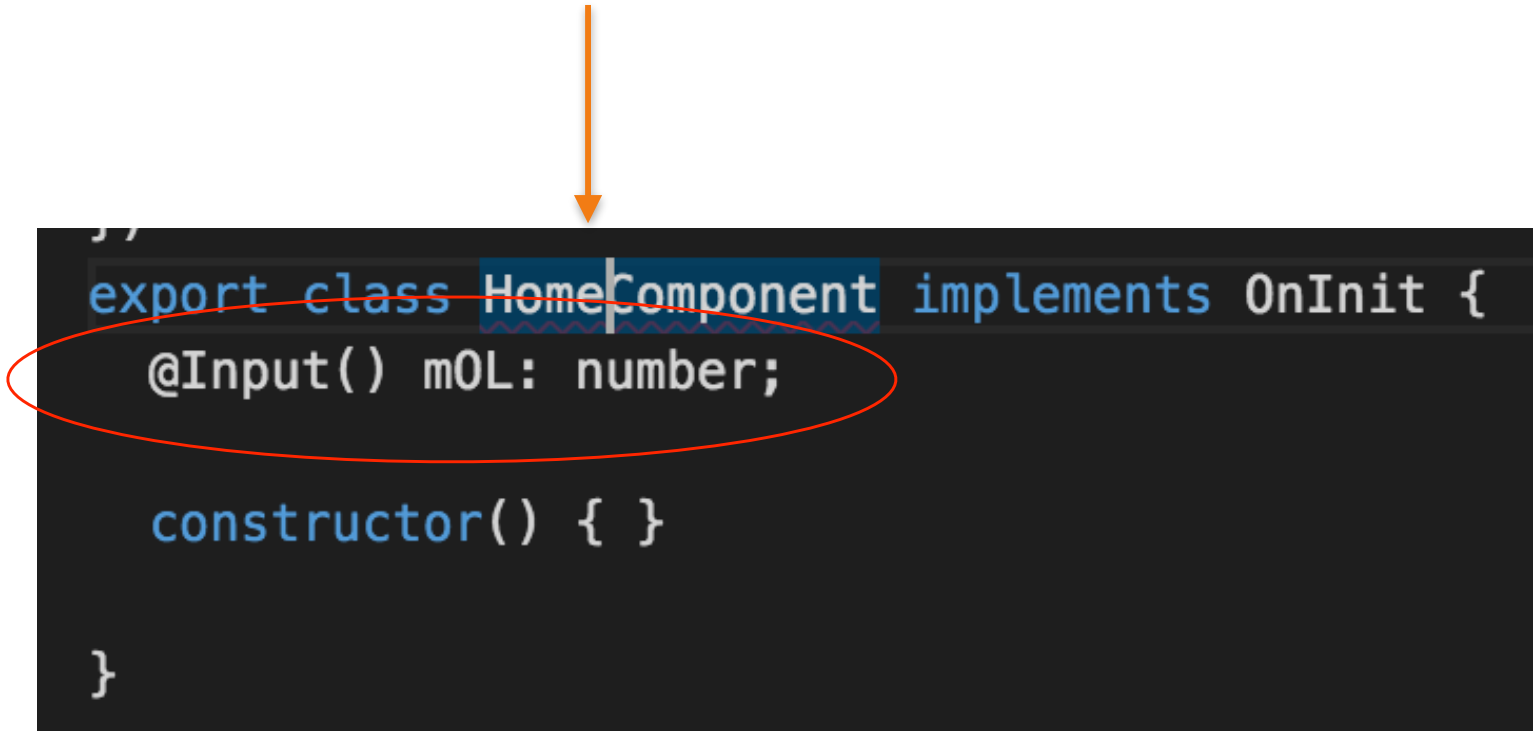
Passing **meaningOfLife** from parent to child

```
<p>
  Hello World
  <app-home
    [mOL] = "meaningOfLife"
  ></app-home>
</p>
```

Develop Intelligence

- Parent has a property the child needs

Specifying expected inputs into the component



- It’s now time to build multiple reports

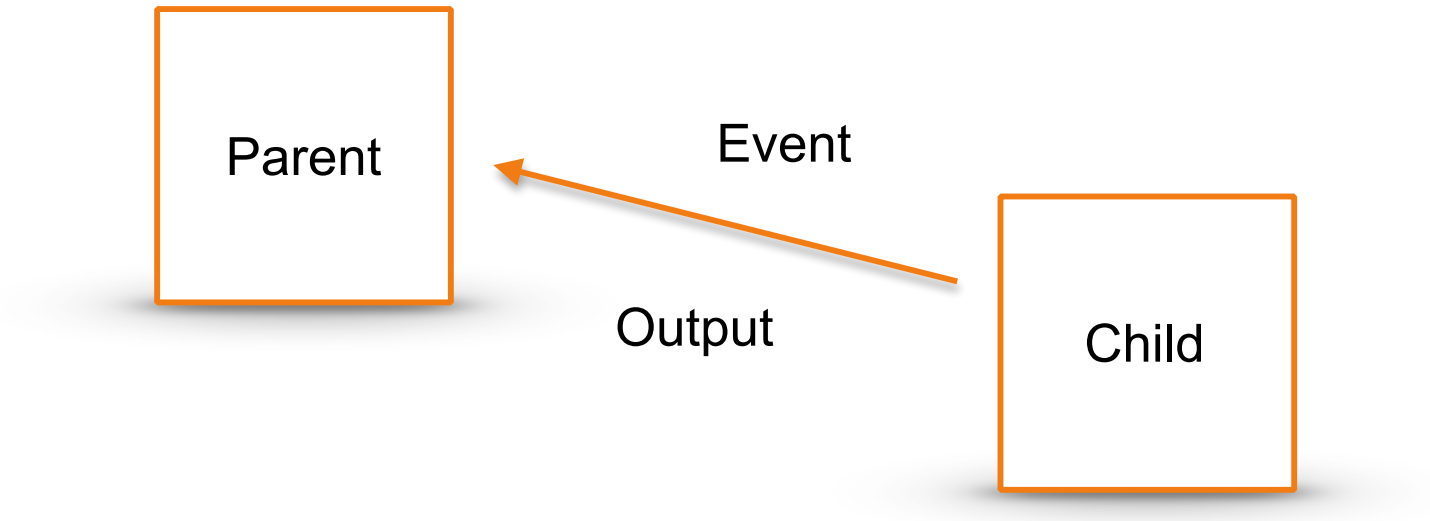
Lab 5b

- Grab report data and make a list of cards from:
  - http://lemon-aide-api.azurewebsites.net/data/reports
  - Create a card list component

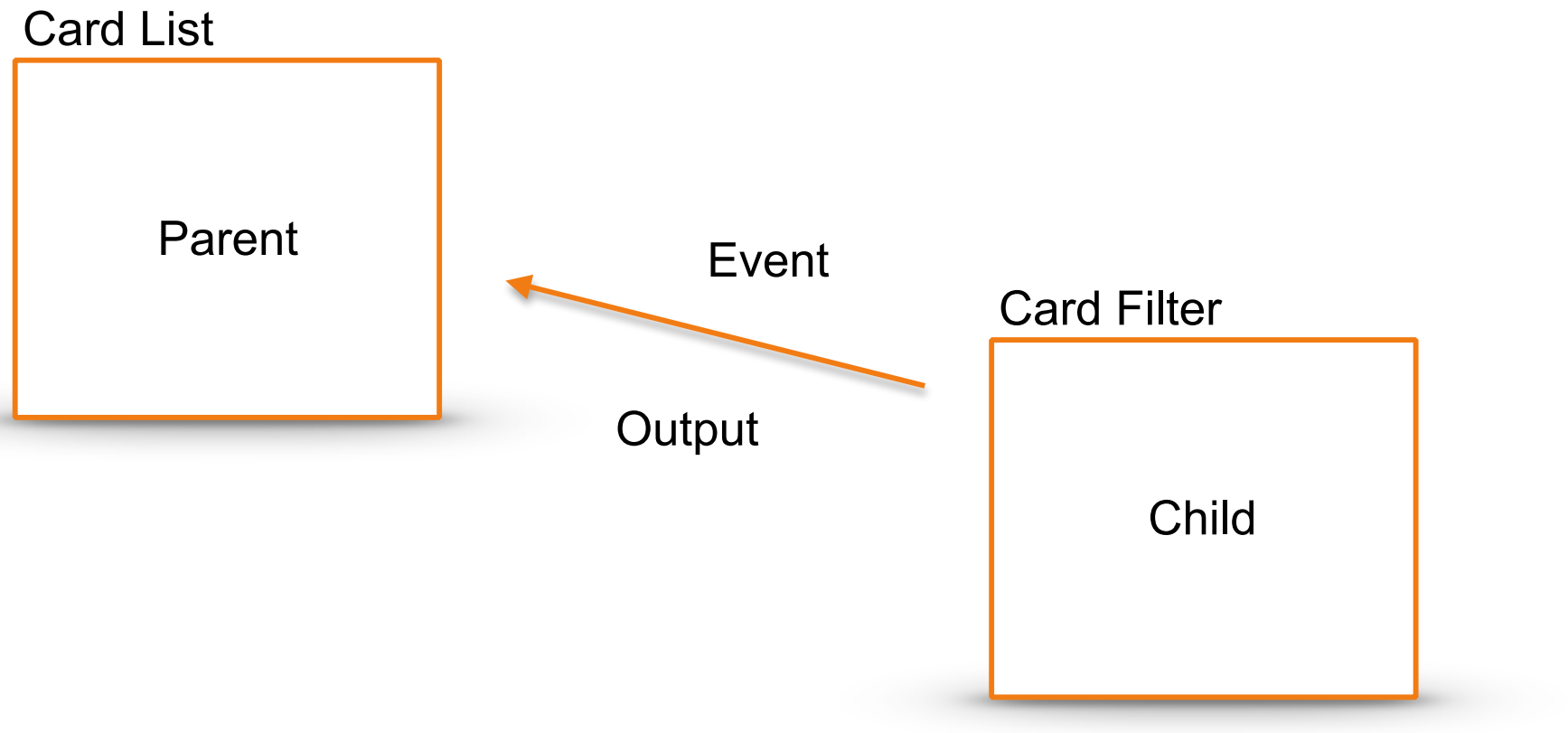
- Get that report object from earlier
  - Create a new report object for each chard
  - Use pipes to filter the strings into a nice look
  - Only show an image if report is profitable (i.e. Gross Profit over $200)
  - Show odd cards with an different color background

Outputs

- We are going to need to send data from child to parent
- We want a one-way data flow so we will use events to send data up from child to parent
  - Properties flow down
  - Events bring data up



Outputs [cont.]



Child to parent communication

- Report application Component
- Report list component
- Report filter card component
- Report card component

Outputs [cont.]

- Let's look at this in our report list?

card-filter.component.html

```
<section class="report-filter">
  <button (click)="showCards(2)">Show 2 months</button>
</section>
```

Non-semantic child event

card-list.component.html

```
<section>
    <app-card-filter
        (onShowCards)="onShowCards($event)"
        ></app-card-filter>
    <ul class="report-listing">
        <app-card
            *ngFor="let report of reports, let i=index"
            [report] = "report"
            [index] = "i">
        </app-card>
    </ul>
</section>
```

Semantic custom event output

Develop Intelligence

- Coding the event emitter

card-filter.component.ts

```
import { Component, EventEmitter, OnInit, Output } from '@angular/core';

@Component({
  selector: 'app-card-filter',
  templateUrl: './card-filter.component.html',
  styleUrls: ['./card-filter.component.css']
})
export class CardFilterComponent implements OnInit {

  @Output() onShowCards = new EventEmitter<number>();

  constructor() { }

  showCards(amount: number): void {
    this.onShowCards.emit(amount);
  }

}
```

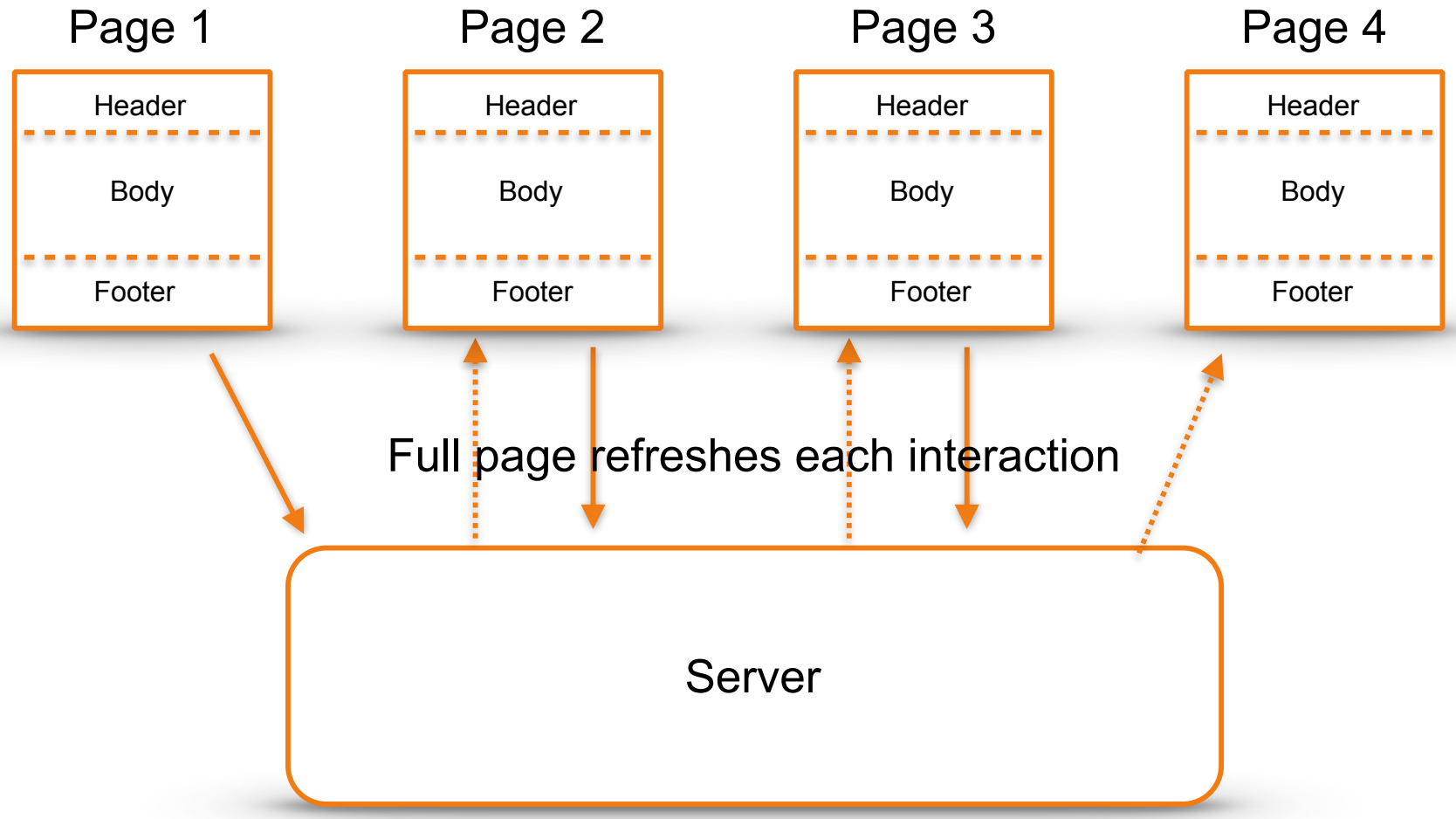
Develop Intelligence

- Receiving the event and using the data

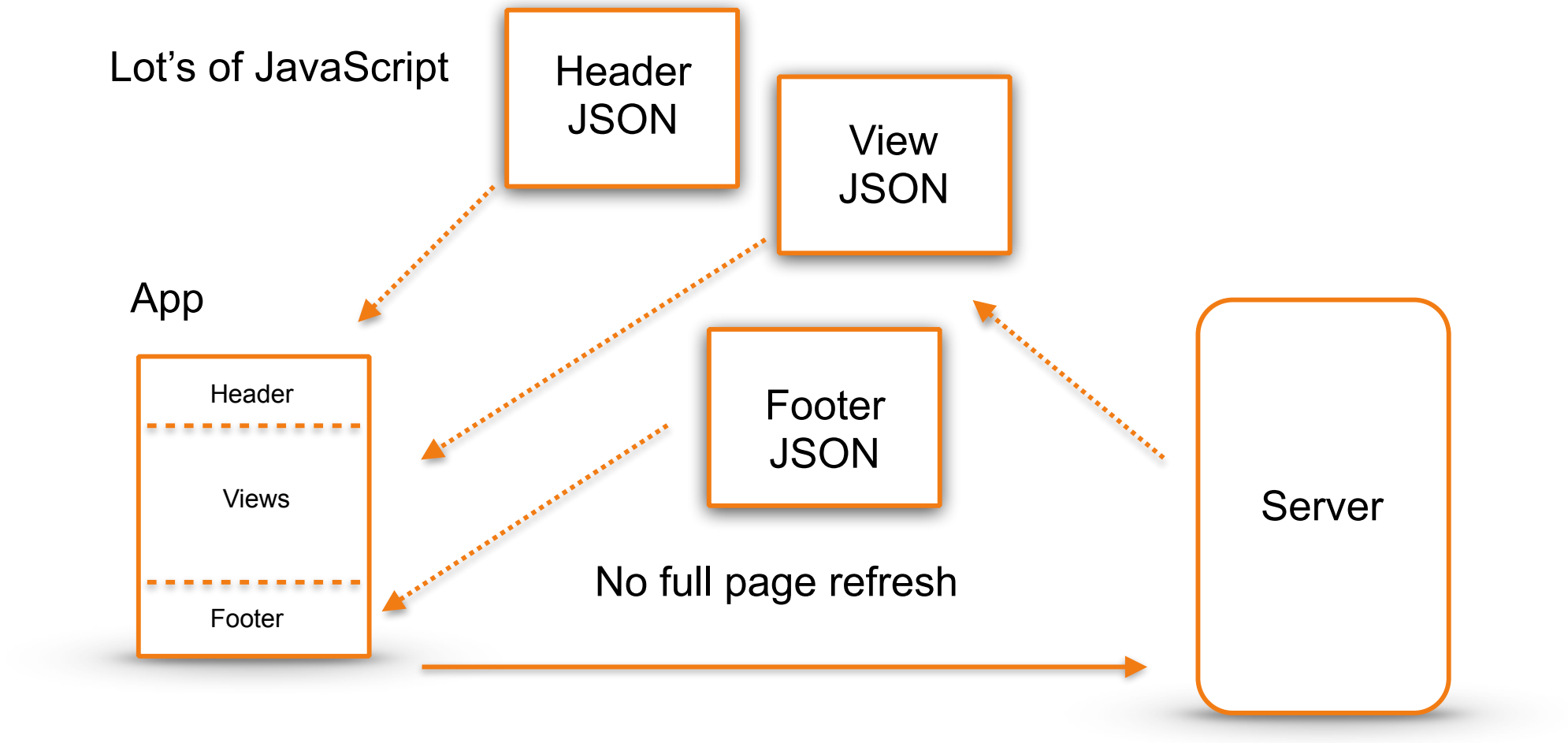
card-list.component.ts

```
export class CardListComponent implements OnInit {
  reports: Report[];

  constructor() {
    this.reports = [
      new Report('0', '1-1-2012', '500', '750', '400'
      new Report('1', '2-1-2012', '425', '650', '300'
      new Report('2', '3-1-2012', '300', '450', '300'
      new Report('3', '4-1-2012', '300', '450', '300'
    ]
  }

  onShowCards(reportsShown: number): void {
    console.log('on show parent');
    this.reports.length = reportsShown;
  }
```

Remember no complete page reloads



Swap data via Ajax



Moving to Ajax

- We want to move from hard coded data to fetching it from a server

- REST
  - POST – Create
  - **GET – Read**
  - PUT – Update
  - DELETE – Delete

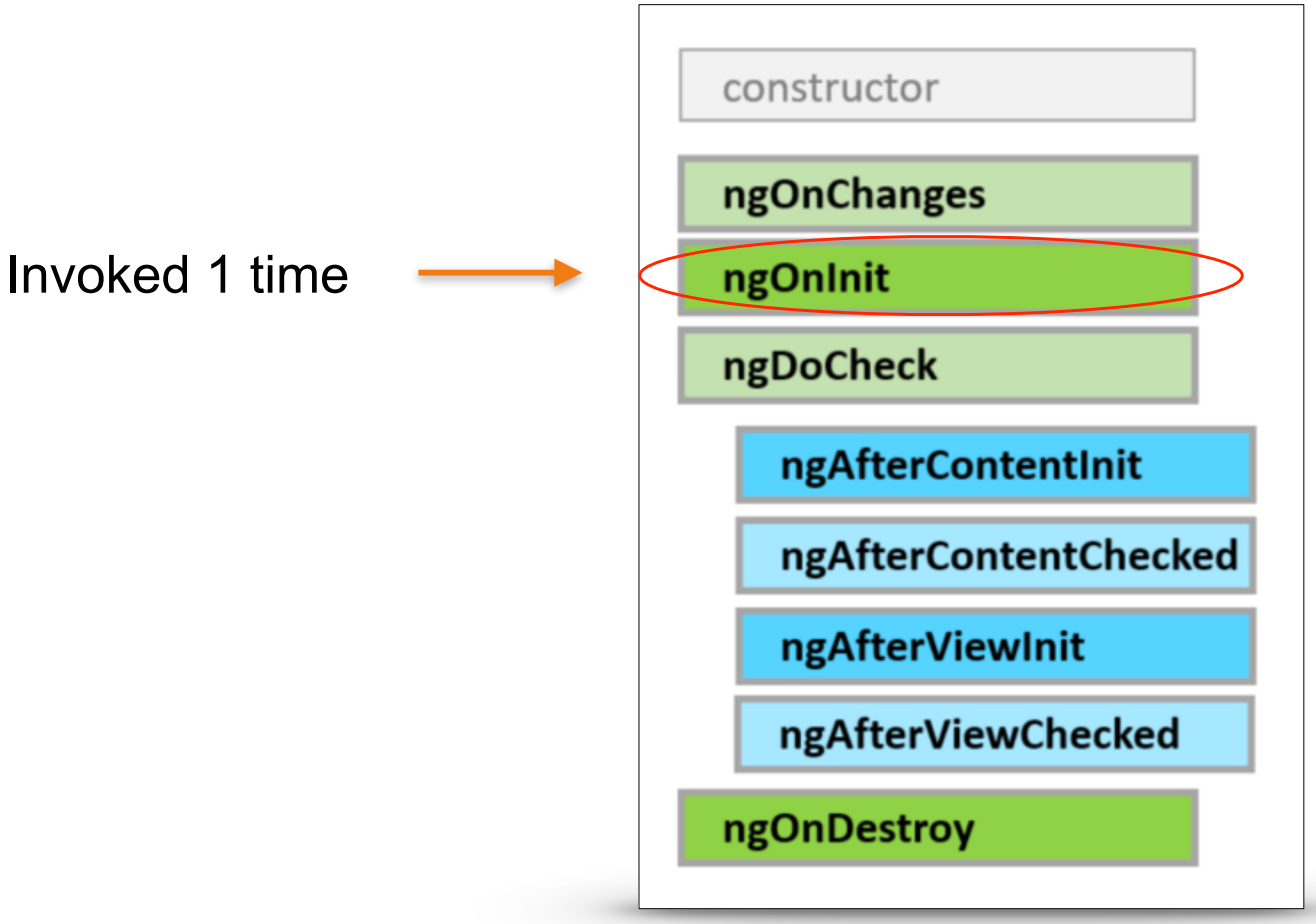
Http Module

- Angular 2 gives an HTTP module that is not part of the **core**
  - Import and Register in ngModule
  - Import in component
    - import { Http } from '@angular/http';

- We want our report data to load before the page is draw. How can we do that?

Component life-cycle events

- Angular provides hooks during creation of a component
  - https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html



Use Http Module

- To use the Http Module it needs to be dependency injected into our code base

card-list.component.ts

```
})
export class CardListComponent implements OnInit {
  reports: Report[];

  constructor(private http: Http) {
    this.reports = [];
  }
```

Use Http Module [cont.]

- The Http module has a **get** method to retrieve data

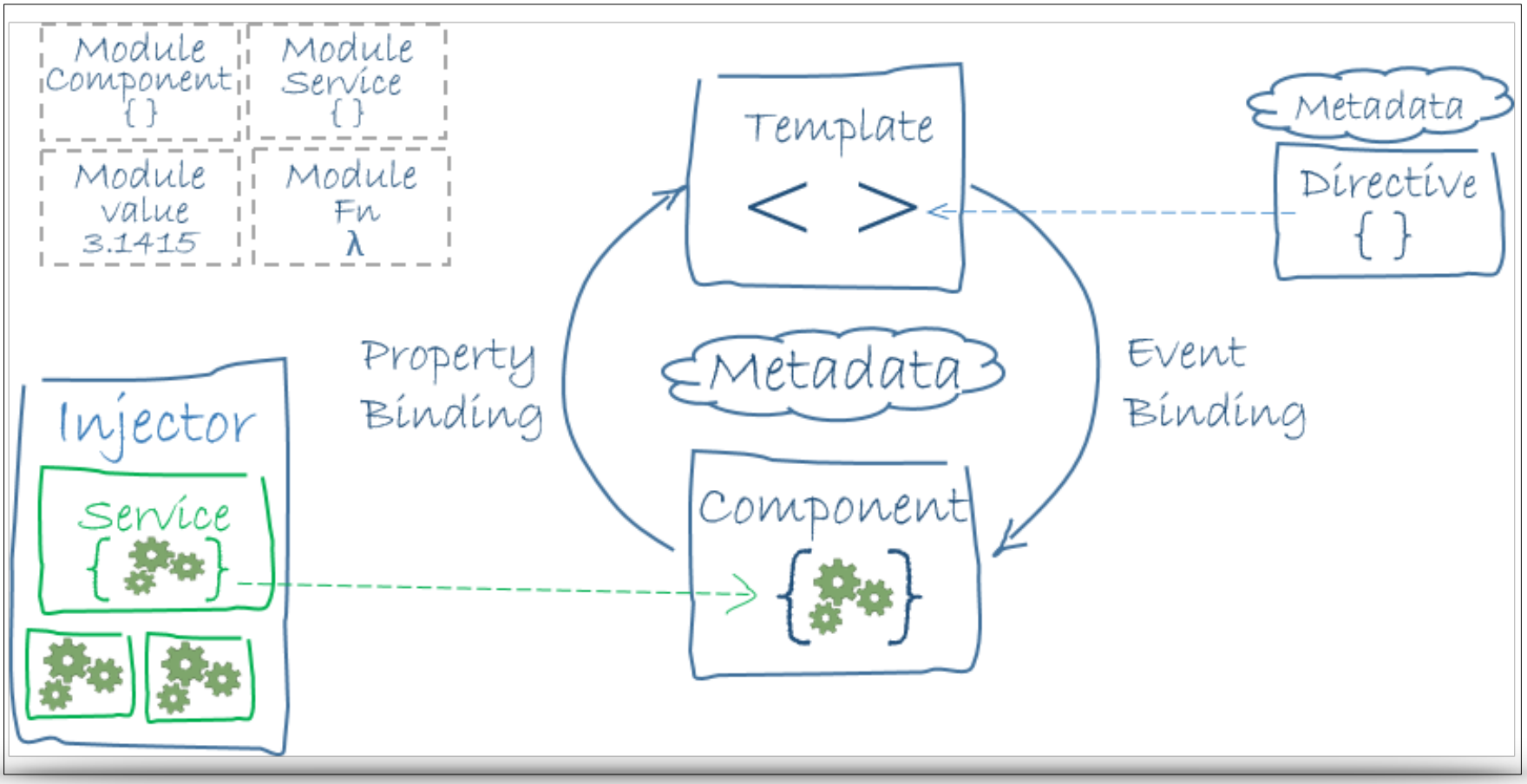
card-list.component.ts

```
ngOnInit() {
  const URL = 'http://lemon-aide-api.azurewebsites.net/data/reports';
  //Observervable based implementation
  this.http.get(URL)
    .subscribe((response) => {
      console.log('made it');
      let data = response.json();
      this.reports = data.map((report: any) => {
        return new Report(report.id,
          report.date,
          report.quantity,
          report.netSale,
          report.costOfGoods);
      });
      console.log(data);
    });
```

Services

- Services are reusable pieces of code
- The could be used to provide a communication avenue between grandparents and children
- They can be used to abstract Ajax calls

- Angular's ideas of how services fit in



Services [cont.]

- Let’s create a service to handle our call for report data

- **ng generate service card-list/card-list**

Generated code

Allows for
Dependency Injection

card-list.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class CardListService {

  constructor() { }

}
```

Injecting the service

Develop Intelligence

Import the Service

Specify in Providers List

Specify in Providers List

card-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { NgFor } from '@angular/common';
import { Report } from '../shared/report';
import { CardListService } from "./card-list.service";

@Component({
  selector: 'app-card-list',
  templateUrl: './card-list.component.html',
  styleUrls: ['./card-list.component.css'],
  providers: [CardListService],
})
export class CardListComponent implements OnInit {
  reports: Report[];

  constructor(private cardListService: CardListService) {
    this.reports = [];
  }

  onShowCards(reportsShown: number): void {
    this.reports.length = reportsShown;
  }
```

Abstract Http to Service

Develop Intelligence

Configure the data

card-list.service.ts

```
init() {
  const URL = 'http://lemon-aide-api.azurewebsites.net/data/reports';
  return this.http.get(URL)
    .map((response) => {
      let data = response.json();
      this.reports = data.map((report: any) => {
        return new Report(report.id,
            report.date,
            report.quantity,
            report.netSale,
            report.costOfGoods);
      });
      console.log(data);
    });

}
```

Using the service

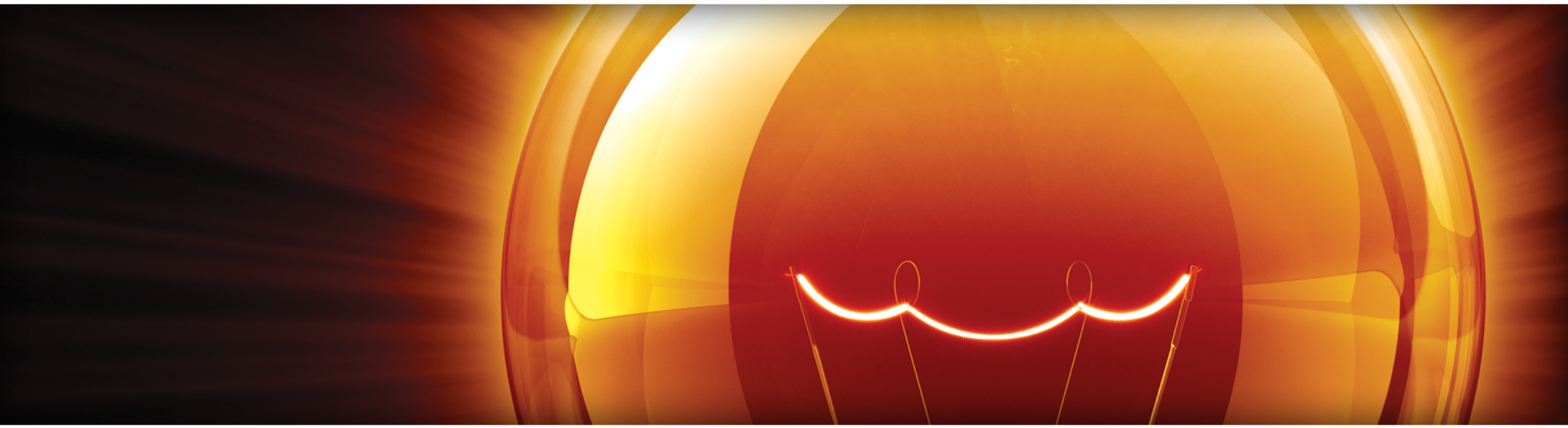
Develop Intelligence

Configure the data

card-list.service.ts

```
ngOnInit() {
  //Observervable based implementation
  this.cardListService.init()
    .subscribe(()=>{
        this.reports = this.cardListService.getReports();
    });
}
```

Where to go from here



Areas to research…

- Ionic 2: Mobile apps for Angular 2
- Angular Universal: Isomorphic Angular 2 support
- Angular Material 2: Material Design for Angular 2
- Migrating to Angular 2
- Style Guide
- System JS: Module loader
- RxJS: Reactive programming library in JS
- Zone.js: No need for $scope.$apply()
- Angular Router
- Protractor: Testing End-to-End

Today's Goals

- Understand the need for Angular 2?
- Make a mental mind shift to components
- Learn the foundations of Angular 2
- Have reference material for more in-depth learning

Develop Intelligence

Prepared by Kamren Zorgdrager

kamren@developintelligence.com
September, 2016

Confidential