# Angular for CF Developers

**Josh Kutz-Flamenbaum**

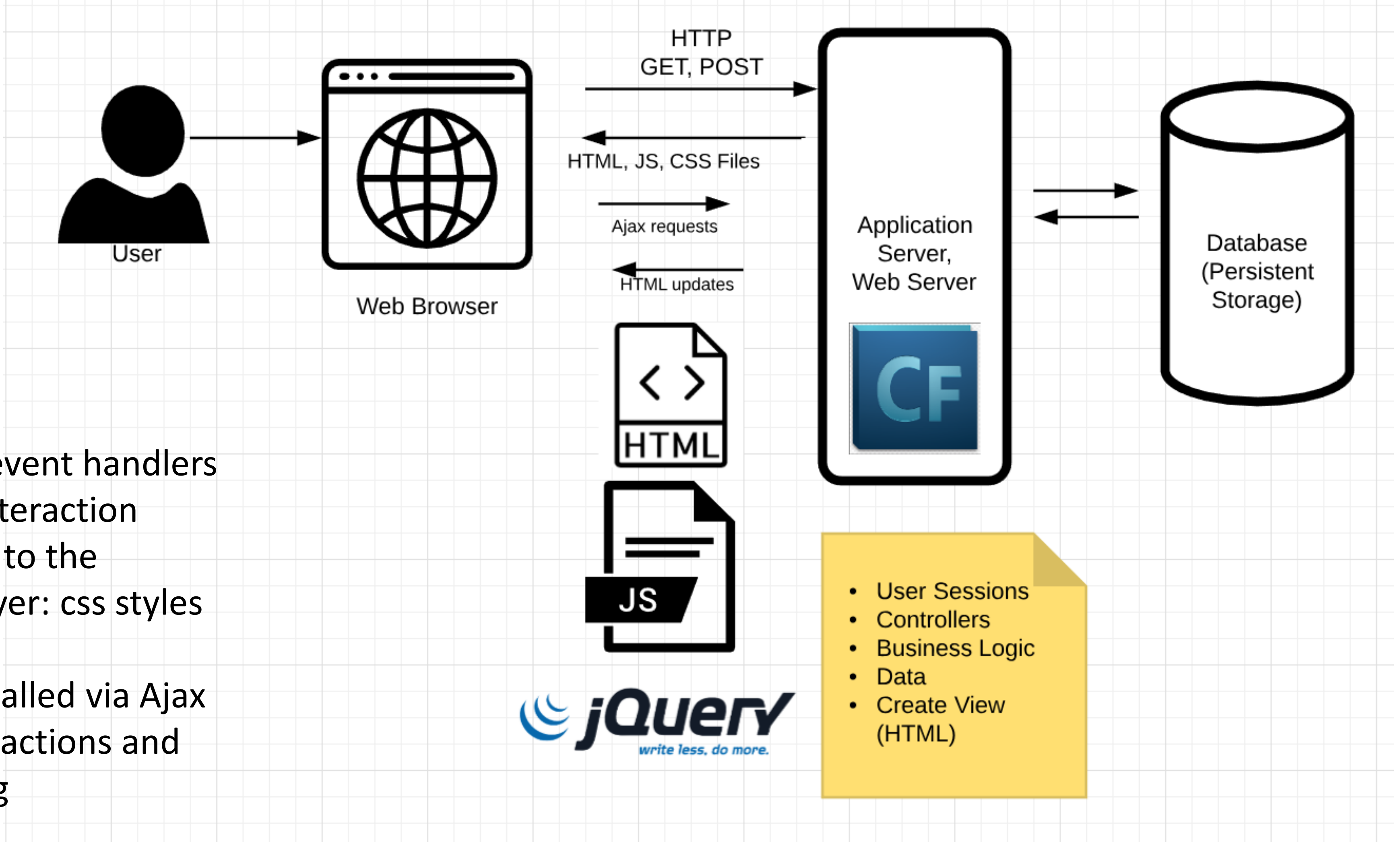
M.S. Information Systems
Bumble and Bumble. Senior Software Architect
jkutzflamenbaum@gmail.com / @jkutzfla
https://github.com/jkutzfla/cf2018demo

Adobe ColdFusion Summit 2018

+

# Traditional Web Dev

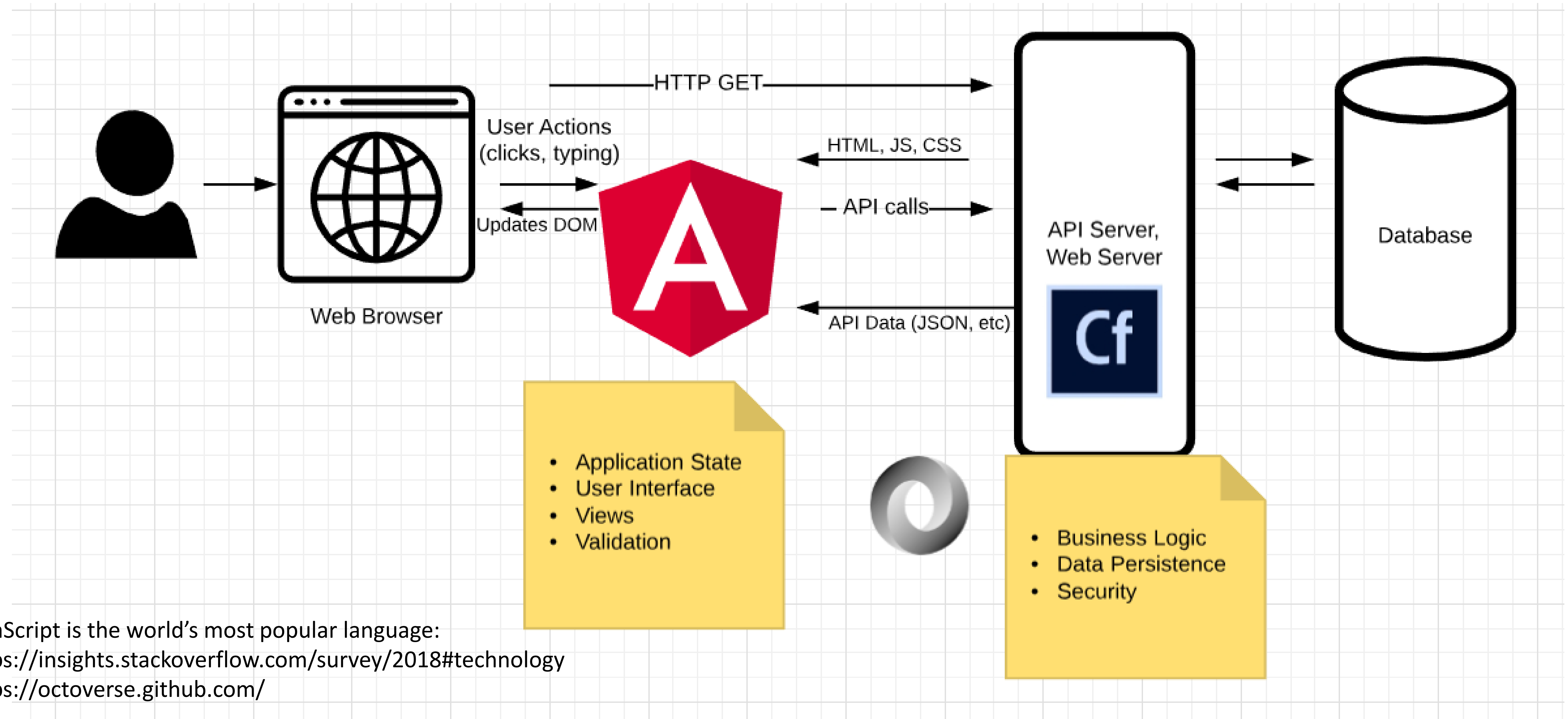
- Old way of creating front-end functionality is limiting and frustrating.



Issues:

- jQuery nested event handlers for each new interaction
- Tightly coupled to the presentation layer: css styles and ids
- The .cfm page called via Ajax does both data actions and HTML rendering

## New Architecture



JavaScript is the world's most popular language:
https://insights.stackoverflow.com/survey/2018#technology
https://octoverse.github.com/

## Technical Features & Advantages of Angular

- Web component architecture
- Two-way Data Binding & forms
- Dependency Injection
- Asynchronous HTTP Support
- TypeScript in Angular 6

- Also animation support but I ran out of time for this.

## How to do it– connect to Adobe ColdFusion

- Demo – a shopping cart of products
- Demo code is also available: https://github.com/jkutzfla/cf2018demo
- Same initial functionality in both AngularJS and Angular v6
  - But the angular v6 has more
- Use Browser DevTools (F12) to show the api data

## Technical Features

- **Web Component Architecture**
- Two-way Data Binding & forms
- Dependency Injection
- Asynchronous HTTP Support
- TypeScript in Angular 6

## Introducing Web Components

- A new* way to assemble a web application from discrete pieces

```html
<html>
  <body>
    <h1>Title</h1>
    <my-component my-parameter="Josh"></my-component>
  </body>
</html>
```

Angular was created to “make HTML what it always wanted to be” -- Misko Hevery

* AngularJS 1.5.0 released February 2016
W3C standard

## Web Component Template Language

```
<!-- Angular.js Template Language: -->
<div ng-init="$ctrl.showContentFlag = false">
  <button ng-click="$ctrl.showContentFlag = ! $ctrl.showContentFlag">
     Click to toggle</button>
  <div ng-if="$ctrl.showContentFlag">Content appears when true. </div>
</div>
```

# Web Component Controller Syntax

```
// my-component.component.js
// add this to angular.module('app')
.component('myComponent', {
  ...
controller: function() {
    this.showContentFlag = false;
    this.data = ['Model','Data','Here'];
    this.doSomething = function() {

    }
  }
});
```

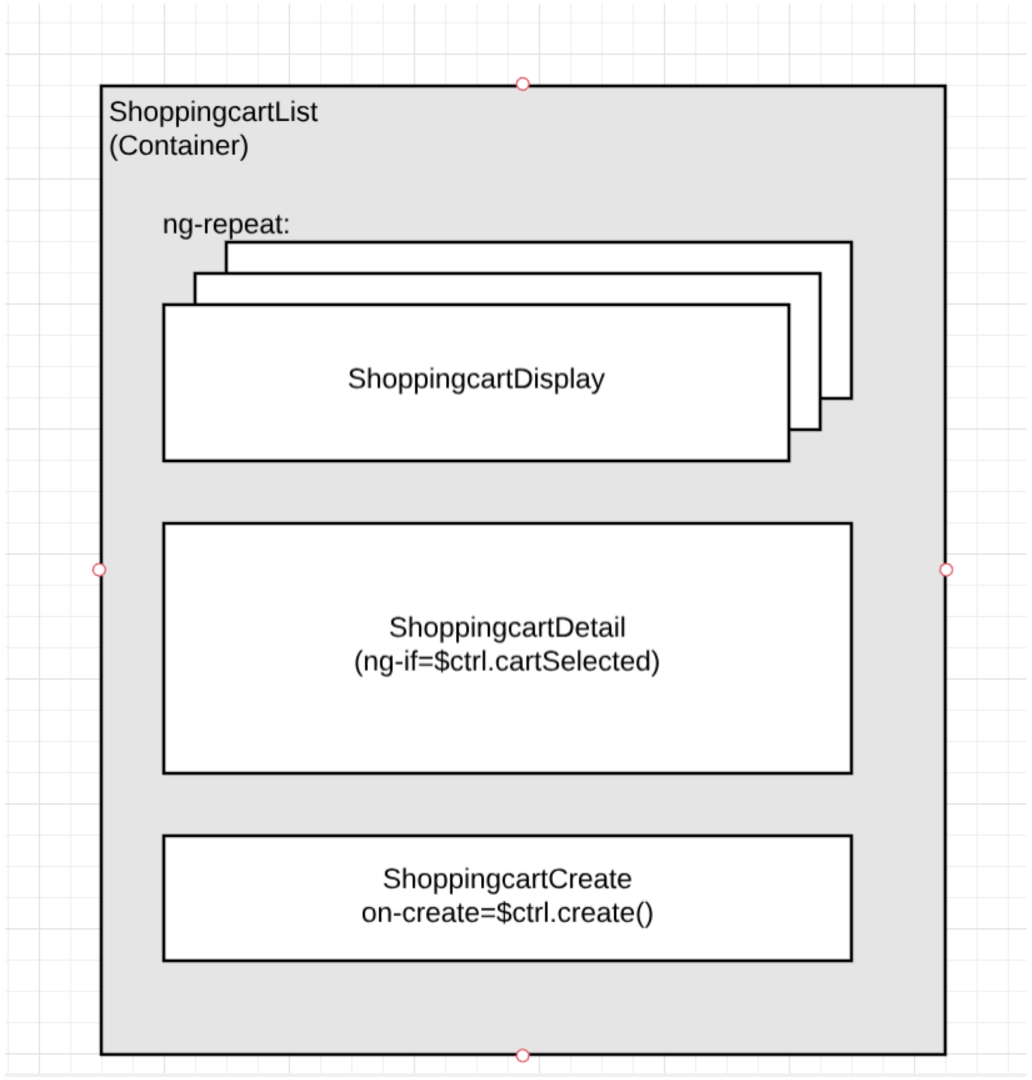
Lifecycle hooks:
**$onInit()**
**$onChanges(changeObj)**
$doCheck()
**$onDestroy()**
$postLink()

## Web Component Bindings Syntax

```
bindings: {
  variableIn: '<',
  twoWayData: '=',
  actionOut: '&',
  stringIn: '@'
}
```

These will be used to connect the components within your application.
Send data “down”, receive actions “up”.

## Web Component Architecture



- Think of the application as a Tree of Components
- Smart container pass data “down” to dumb presentation/view components
- Events/Actions are passed “up” to the container
- Container manages state and service calls
- Benefits are:
  * Better code organization
  * Reuse
  * Explicit lifecycle hooks

## Define ShoppingcartListComponent

```
// shoppingcart-list.component.js
// The App container
angular.module('app').component('shoppingcartList', {
  bindings: {},

  templateUrl: 'containers/shoppingcart-list/shoppingcart-list.html’,

  controller: ['Shoppingcart', 'Product', function(Shoppingcart, Product) {
    //functionality goes here
  }]
});
```

```
<!-- index.html -->
<html><head><!-- include angular.min.js, app.js here -->
<script src="…path_to/shoppingcart-list.component.js"></script></head>
<body ng-app="app">
  <shoppingcart-list>Loading...</shoppingcart-list>
</body>
</html>
```

# ShoppingcartList Component - onInit()

- $onInit() – explicit lifecycle method on the controller
- Get data from a service and store it

```
// inside shoppingcart-list.component.js controller
// this.cartlist is a variable on the controller.
// In the template it will be accessible as $ctrl.cartlist.
    this.cartlist = [];

    this.$onInit = function() {
      this.isLoading = true;
      var self = this;
      //Shoppingcart was injected
      Shoppingcart.getList().then( function() {
        self.cartlist = Shoppingcart.cartlist;
        Product.getList().then(function() {
          self.isLoading = false;
        });
      });
```

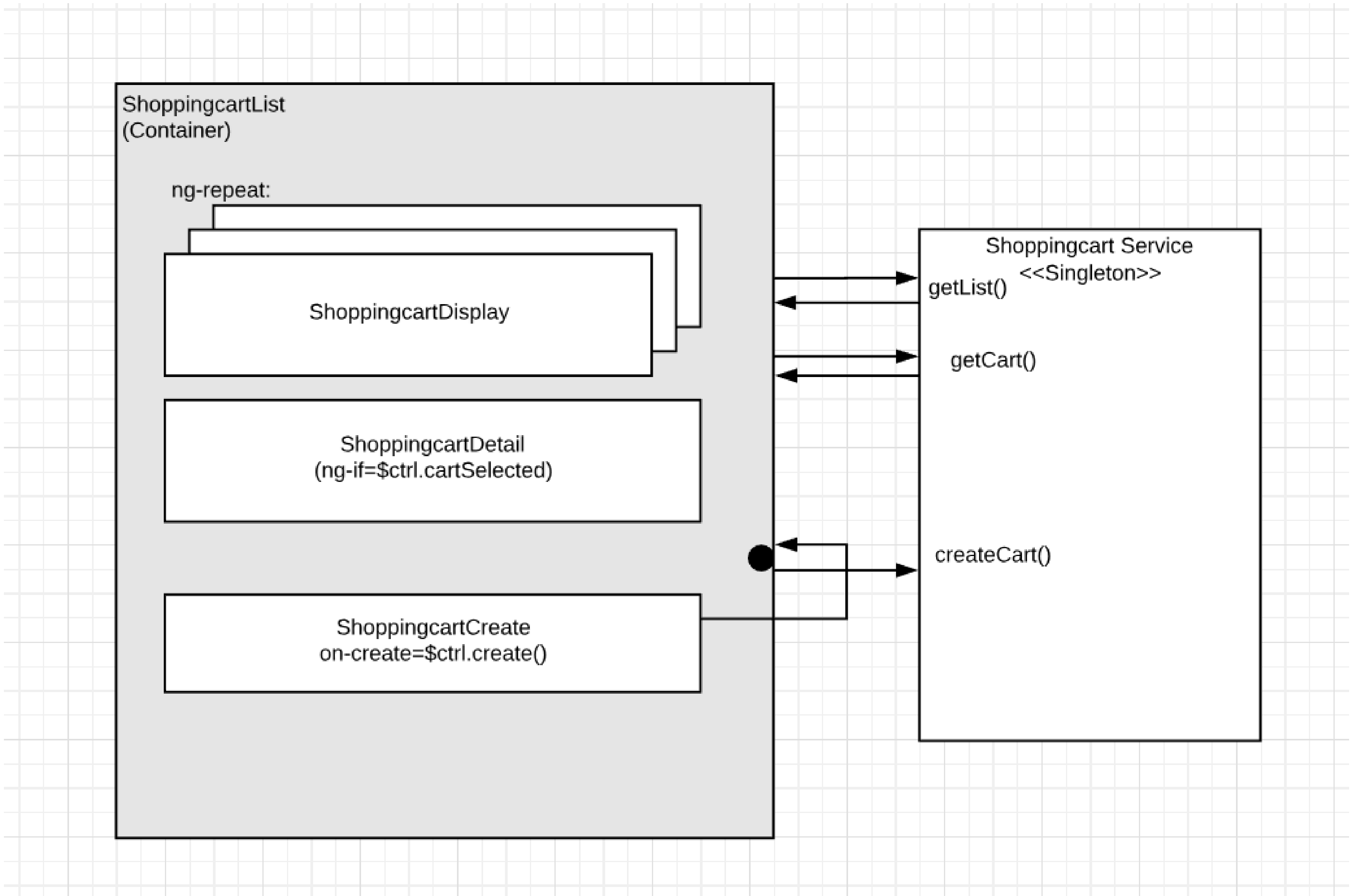
## ShoppingcartList Passes Data Down

```
<!-- File: shoppingcart-list.html
     shoppingcart-list loops the carts and passes each to the display component: -->
<ul>
  <li ng-repeat="c in $ctrl.cartlist" style="display: block;">
    <shoppingcart-display cart="c"></shoppingcart-display>
  </li>
</ul>
```

```
// shoppingcart-display.component.js
angular.module('app').component('shoppingcartDisplay', {
  bindings: {
    cart: '<' //input
  },
  templateUrl: 'components/shoppingcart-display/shoppingcart-display.html',
  controller: function() {}
});
```

```
<!– File: shoppingcart-display.html -->
<span>Id: {{$ctrl.cart.id}} - name: {{$ctrl.cart.name}}</span>
```

# Data down, Actions up



The ShoppingcartCreate create() action
is passed "up" to the ShoppingcartList

## ShoppingcartCreate passes the ‘onCreate’ Action up

```
// shoppingcart-create.component.js binding to pass an event up
angular.module('app').component('shoppingcartCreate', {
  bindings: {
    onCreate: '&' },

    // when the create button is clicked:
    this.create = function() {
      console.log('create() in shoppingcart-create');
      this.isLoading = true;
      var self = this;
      this.onCreate({name: this.name}).then(function() {
        self.isLoading = false;
        self.name = "";
      });
    }
```

## onCreate Event is bound by the ShoppingcartList container

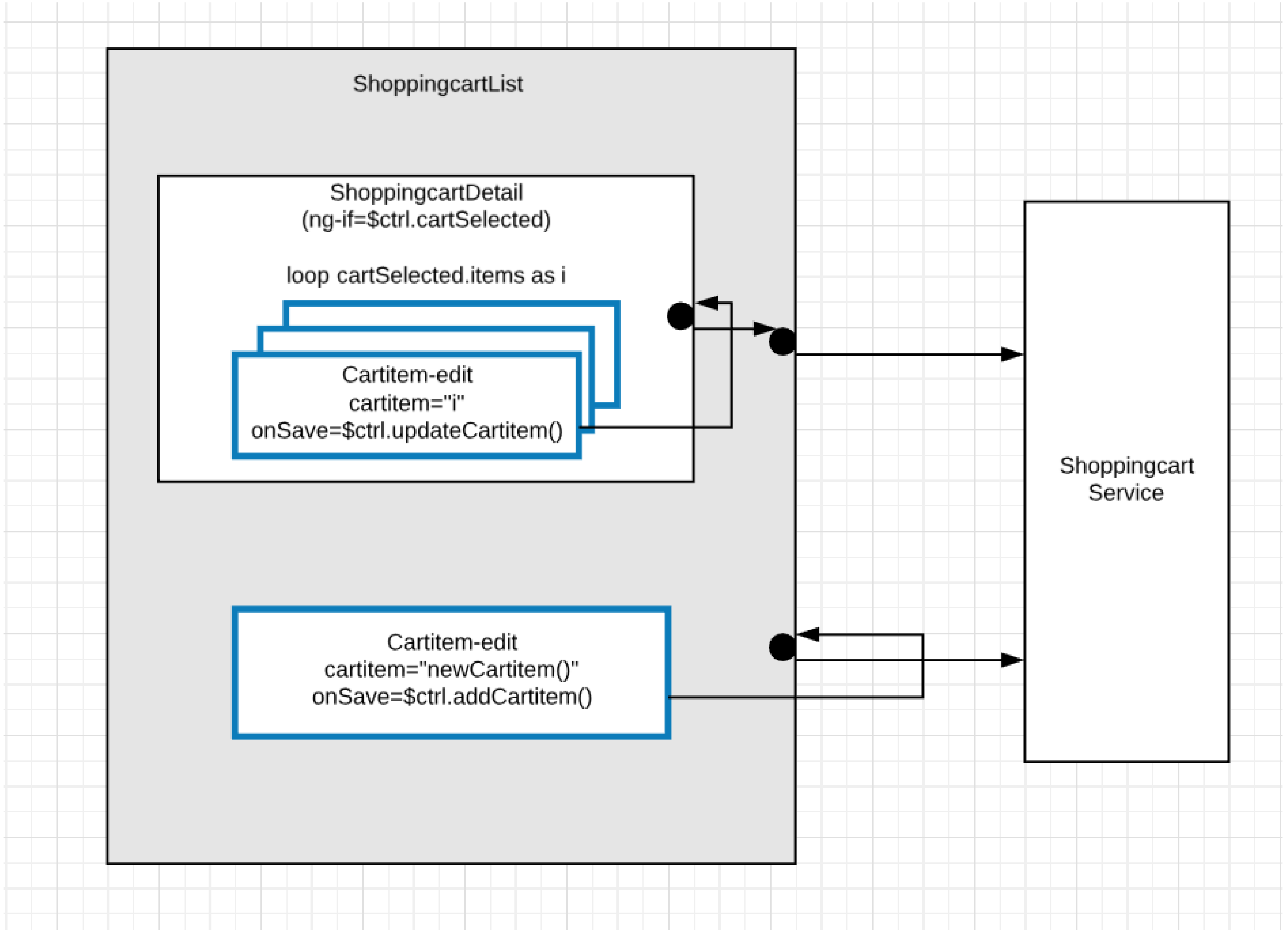
```
// Back in shoppingcart-list.component
//
// we have bound to this method:
// <shoppingcart-create on-create="$ctrl.createCart(name)">

    this.createCart = function(name) {
      console.log('in create, name=', name);
      this.isLoading = true;
      var self = this;
      return Shoppingcart.create(name).then( function() {
        self.cartlist = Shoppingcart.cartlist;
        self.isLoading = false;
      });
    }
```

# Component Reuse Achieved!

- cartitem-edit component is used twice:
  - add a new item
  - edit existing item

- The same component is bound to two different container methods

- Benefit:
  the populateProduct()
  and cartWorkingTotal()
  logic
  inside cartitem-edit
  is not duplicated.



# AngularJS Web Components & ColdFusion

- Can call them within a .cfm
- 4 steps:
  - Include angular
  - Include your javascript source code
  - Set the ng-app (must match your angular.module name)
  - Call the web component

```
<!--- e.g. in Bumble wholesale.cfm --->
<cfset upc="123">
<oos-notice upc="#upc#"></oos-notice>
```

In a pinch, you can also pass JSON as an input

## Demo of JSON sent from CF to ng

- /www/samples/in_a_pinch.cfm

- (JSON will be discussed more in the http section)

## Technical Features

- Web component architecture
- **Two-way Data Binding & forms**
- Dependency injection
- Asynchronous HTTP Support
- TypeScript

# Two-way Data Binding

```
<!-- cartitem-edit.html -->
<br>Working total: {{$ctrl.getCartTotal() | currency}}

<form>
  <select  … ></select>

  <input type="number" placeholder="Quantity"
     ng-model="$ctrl.item.quantity">
  <input type="number" placeholder="Price"
     ng-model="$ctrl.item.priceDollar">
</form>

<div>
  <button class="btn btn-large" ng-click="$ctrl.addItem()">addItem</button>
</div>
```

# Ng-model with Form Validation

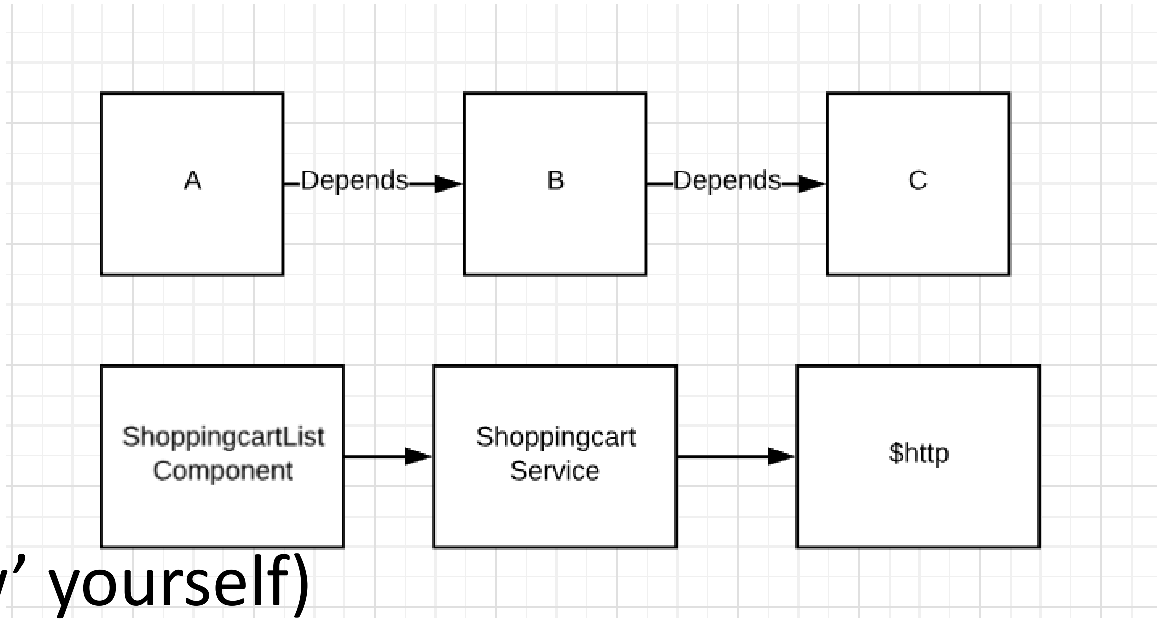
```
<form #promocodeForm="ngForm">
 <div class="form-group">
    <label for="message">Message:</label>
    <input type="text" class="form-control" id="message"
      required
     [(ngModel)]="promocode.message" name="message"
     #message="ngModel">
     <div [hidden]="message.valid || message.pristine" class="alert alert-danger">
       Message is required
     </div>
  </div>

  <div>
    <button (click)="save()" class="btn btn-success"
    [disabled]="!promocodeForm.form.valid || isSaving">
    Save</button>
    <span *ngIf="isSaving">Saving...</span>
  </div>
  </form>
```

## Technical Features

- Web component architecture
- Two-way Data Binding & forms
- **Dependency Injection**
- Asynchronous HTTP Support
- TypeScript in Angular 6

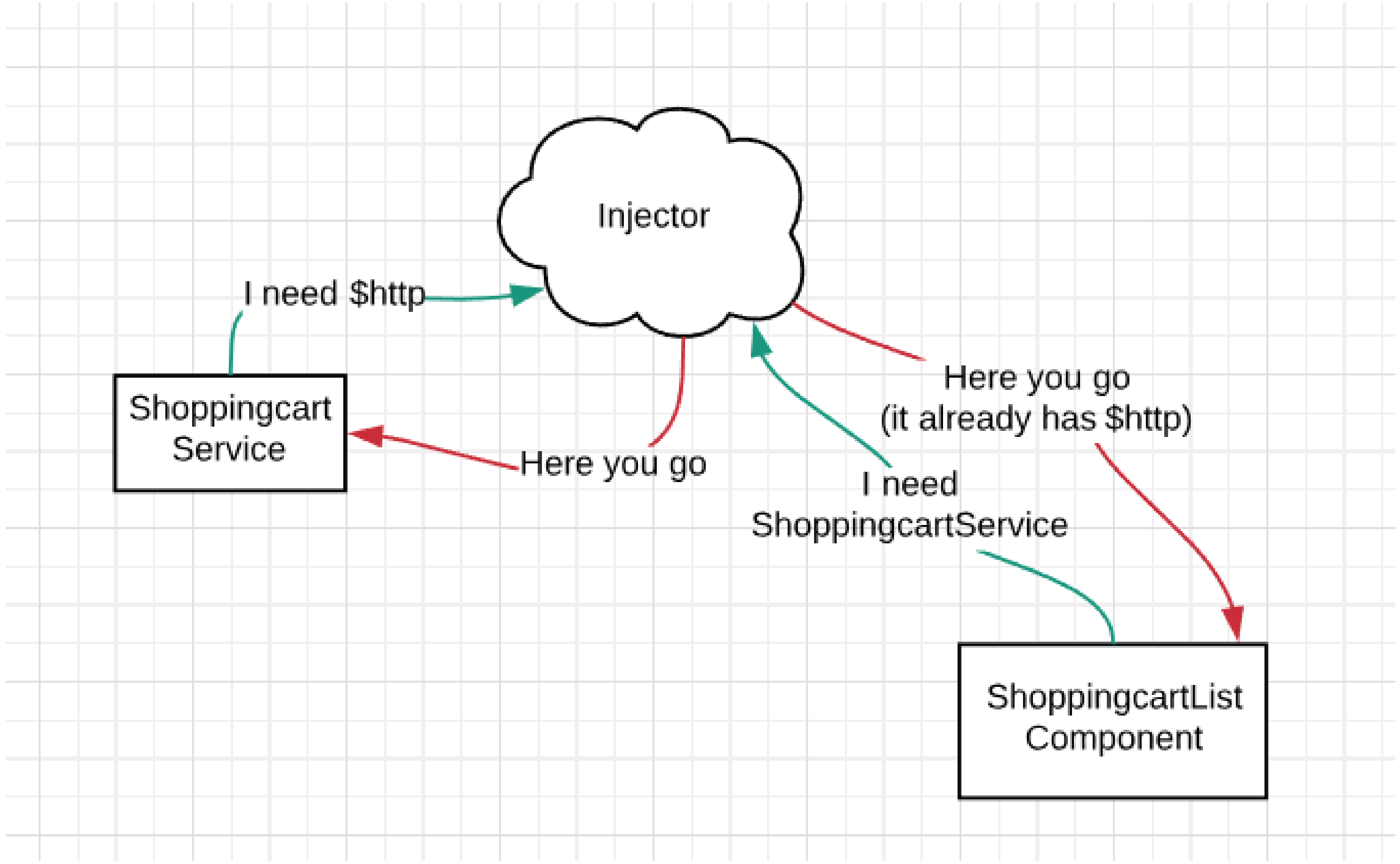
# Dependency Injection

- Software is assembled from small pieces to large
- One piece depends on others
- But sometimes the connections can be hard to manage
- Instead of direct dependencies, use DI libraries (avoid calling ‘new’ yourself)
- Why? More modular code, testability, avoid spaghetti code and avoid Global scope



https://martinfowler.com/articles/injection.html
[https://wirebox.ortusbooks.com/getting-started/overview](https://wirebox.ortusbooks.com/getting-started/overview)

# Dependency Injection in the Demo


Injector
I need $http
Shoppingcart Service
Here you go
Here you go (it already has $http)
I need ShoppingcartService
ShoppingcartList Component

# DI Eliminates complex object creation

- You should not call ‘new’ yourself.

Avoid:

```
ShoppingcartListComponent:
New ShoppingcartService();
New ProductService();

ShoppingcartService:
New $http();
```

- Why? If the services are constructed for you, they can be swapped out behind the scenes if you want to write tests for your http services.
- And it becomes easier to combine discrete services if you are not worried how to create them

## Dependency Injection examples from Demo

- Declare the dependency when creating the service or component
- Use the injected object (as a Singleton)

```
// shoppingcart.service.js
// create the Shoppingcart service
angular.module('app’)
  .factory('Shoppingcart’, ['$http', function($http) {
  //later:
  $http.get(url);
}
```

```
// in shoppingcart-list.component.js:
// inject the Shoppingcart service
controller: ['Shoppingcart', function(Shoppingcart) {
    this.$onInit = function() {
      //Shoppingcart was injected
      Shoppingcart.getList();
    }
```

N.B. This syntax will fail if code is minified.
See ng-annotate to support minification.

# DI – real world example

```
// example from Bumble University Registration System
angular.module('core')
  .factory('Attendee', ['BbUClass','AttendeeExperienceRequired',
'ContactEduhistoryCheck', 'Account',
  function (BbUClass, AttendeeExperienceRequired, ContactEduhistoryCheck, Account) {
```

```
// later in the Attendee service,
// validity requires checking multiple domain objects:
// the BbUClass, the EduHistory, the Account contact info
if (this.contactType != 'tbd' && ContactEduhistoryCheck.isCheckNeeded()) {
  if (this.contactId != '') {
    var contact = Account.findContact(this.contactId);
    var check_edu = ContactEduhistoryCheck.check(contact);
    if (!check_edu) {
      this.errorMessages.push('Bb.U Prereqs are not satisfied.')
    }
  }
}
```

## ngModule

```
// app.js
// step 1: define your module and specify required modules
angular.module('app', ['shared']);

// step 2-n, add stuff to the module
angular.module('app').component('name', {});
```

- Like the DI system, angular modules allow for code to be divided into independent files and re-combined.
- The only global object is ‘angular’!
- Everything is added to the angular.module(‘modulename’)
  - .component() .factory() .directive() .pipe()

## Technical Features

- Web component architecture
- Two-way Data Binding & forms
- Dependency Injection
- **Asynchronous HTTP Support**
- TypeScript in Angular 6

## Asynchronous = “Out of time”

- In a Web Browser, all resources are loaded across the network
- This action cannot block the main application thread or the application would freeze; terrible UX
- So the network requests occur in a separate thread, asynchronously, and the programmer does not know when they will complete.
- But programming logic must be written in an imperative manner:

  if this, then that.
- So Promises are added to AngularJS to give you control

JavaScript Promises are closures and will affect scope, see:
http://javascriptissexy.com/understand-javascript-closures-with-ease/

## AngularJS Promise API:

```
// angularJS Promise API:
function then(successCallback, [errorCallback], [notifyCallback])
function catch(errorCallback)
function finally(callback, notifyCallback)
```

```
//used like:
var self = this;
$http.get('url').then(function(response) {
  console.log('received response', response);
  self.userid = response.data.userid;
})
.catch( function(error) {
  console.log('received error: ', error);
})
.finally( function() {
  console.log('finally runs regardless');
});
```

## Services return the Promise

```
// in shoppingcart.service.js
// will return a promise from the $http.get()
getList: function() {
  var url = "/api/cart/list";
  //$http was injected:
  return $http.get(url).then(function (response) {
    // store the result in the object:
    service.cartlist = response.data;
  });
},
```

## Chaining Promises

- By returning the Promise object, you can chain .then() after .then() and write code that will only run after the required network requests are done.

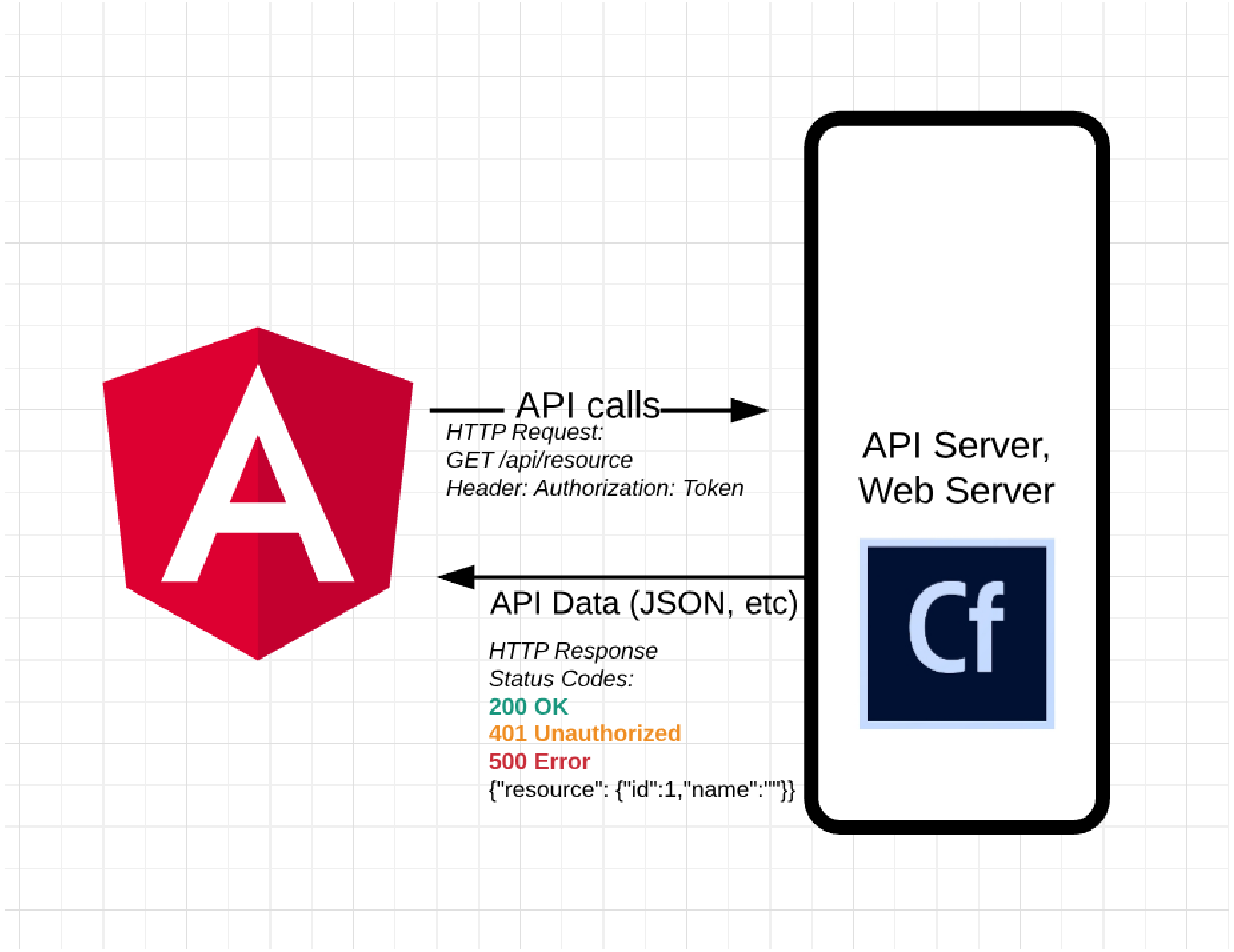
```
// onInit in shoppingcart-list.js
this.$onInit = function() {
  this.isLoading = true;
  var self = this;
  //Shoppingcart object was injected
  Shoppingcart.getList().then( function() {
    // inside the promise returned by Shoppingcart.getList()
    self.cartlist = Shoppingcart.cartlist;
    Product.getList().then(function() {
      // inside the promise from Product.getList(), now template is ok to render:
      self.isLoading = false;
    });
  });
}
```

# API Authentication

Use HTTP Interceptors within the Angular app:

- Add the api auth token to outgoing requests
- Check incoming responses for auth errors and forward to login

This is a design choice, it could also be handled by an API Service class in Angular that all calls are sent thru.



## ColdFusion receive JSON

```
// in a .cfm:
// how to get the HTTP body from Angular:
dataIn = deserializeJSON(getHTTPRequestData().content);
```

```
// FW/1 config:

variables.framework = {
  decodeRequestBody = true, // will convert the JSON from the HTTP body
  preflightOptions=true     // respond to OPTIONS request
```

# ColdFusion send JSON

```
<!--- in a .cfm, build up one object and output --->
id = regSrv.getRegistrationId();
data = {"id"=id};

<cfheader name="Content-Type" value="application/json">

<cfoutput>
    #serializeJSON(data)#
</cfoutput>
```

Also:
<cffunction access="remote" returnFormat="json">

```
/**
* FW/1 API for Josh Kutz-Flamenbaum Angular Demo
*/
component accessors="true" {
  property framework;
  property CartService;

  function list(struct rc) {
    var carts = variables.CartService.list();
    variables.framework.renderData().data(carts).type("json");
  }
```

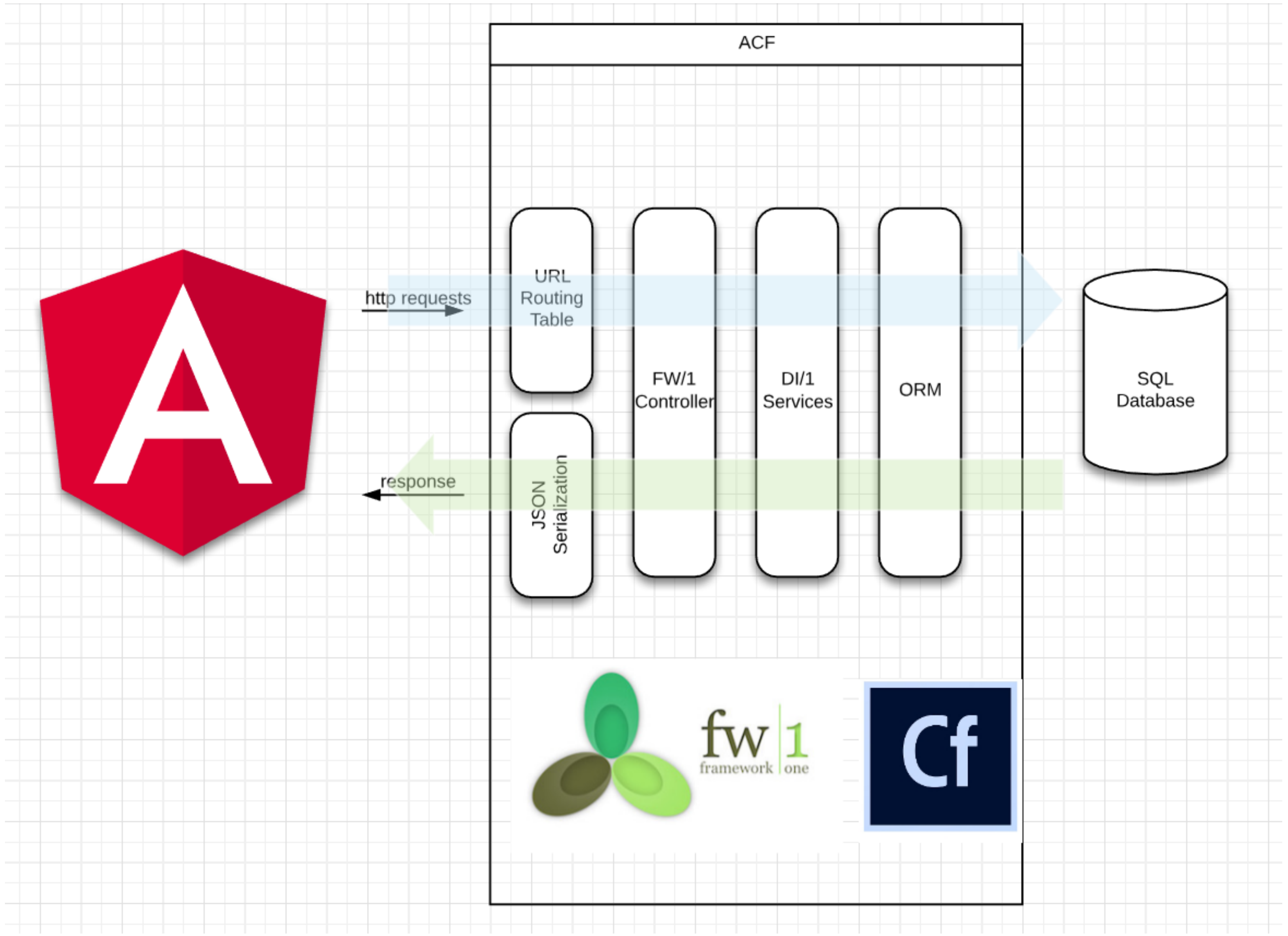
## ColdFusion JSON Serialization

- CF11 – will preserve the case of the struct key
  - Set this.serialize in Application.cfc
  - Otherwise always set with {“keyName”=value}
- CF2016 update 2 – able to enforce data types with setMetadata()
- Timezones?
- When in doubt, write some tests

```
// Application.cfc - CF2018 Demo
component {
  this.name = "CF2018Demo";
  // the default is false, meaning that struct keys are force to UPPPERCASE.
  this.serialization.preserveCaseForStructKey = true;
}
```

https://helpx.adobe.com/coldfusion/cfml-reference/coldfusion-functions/functions-s/serializejson.html
https://cfdocs.org/serializejson

# ColdFusion API Architecture



## AngularJS vs Angular 6

**AngularJS**

- Version 1.7
  July 2018
  LTS
- 3 years of security and browser compatibility fixes

**Angular #itsjustangular**

- Version 6
  May 2018
  6 months active support
  12 months LTS
  (LTS starts November 2018)
- Version 7 expected Fall 2018
- Version 8 expected Spring 2019

Source: https://docs.angularjs.org/misc/version-support-status#
Source: https://angular.io/guide/releases

## Technical Features

- Web component architecture
- Two-way Data Binding & forms
- Dependency Injection
- Asynchronous HTTP Support
- **TypeScript in Angular 6**

## Angular version 6

- TypeScript
  - Benefits: type safety on code, libraries & JSON
    - JSON data casted to a TS Interface in one line!
    - Static checks while transpiling to js
  - Developer Productivity benefits: Intellisense in typescript-aware editor
    - Very useful to see your controller variables while in HTML view.

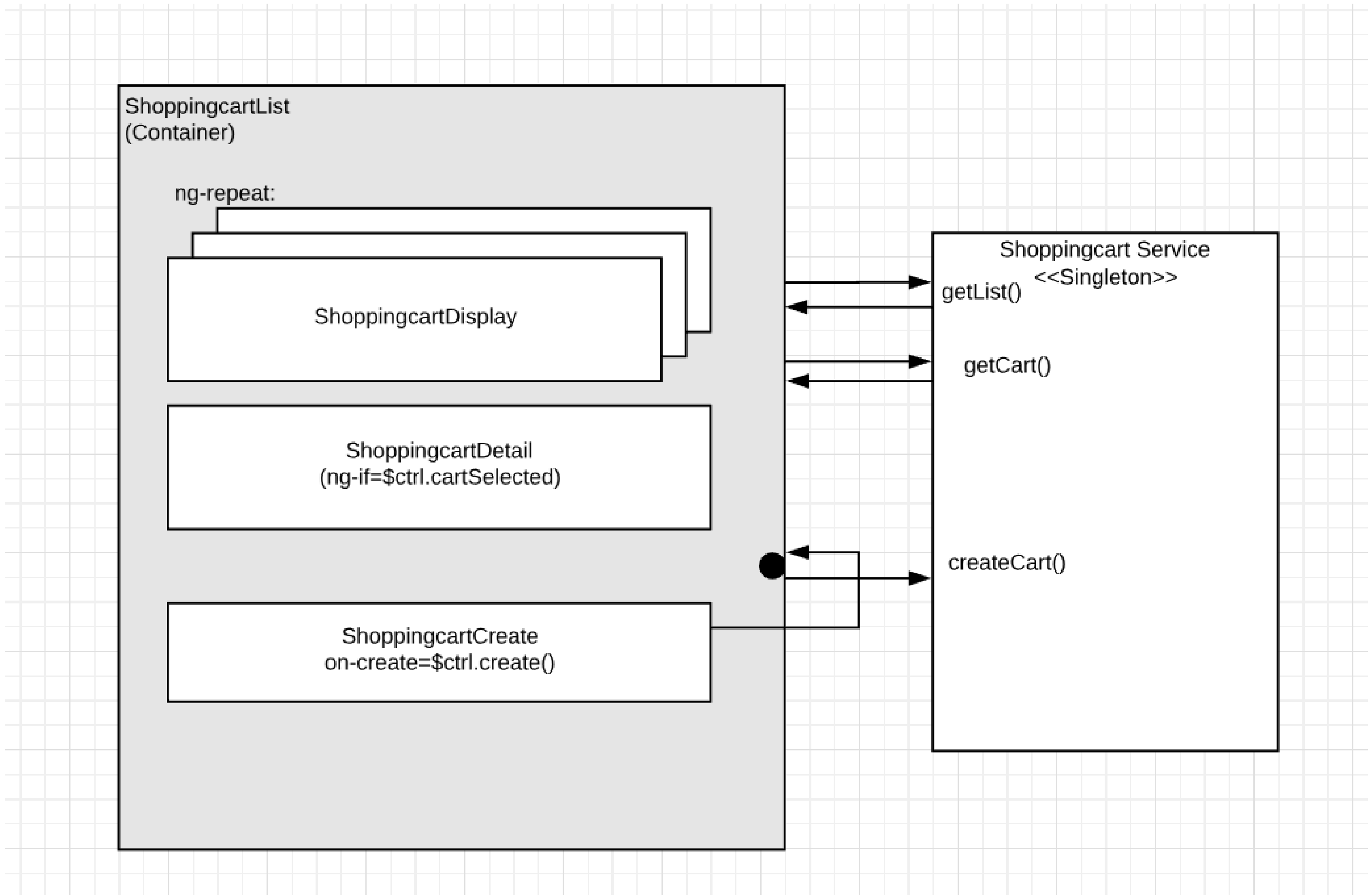
- JavaScript ES6
  - Benefits:
    - import/export is simple compared to ES5 add ons (CommonJS, RequireJS)
    - Decorators are a clean way to add meta info to source code.
  - New language syntax: ‘let’ and ‘const’ variables
  - Fat arrow => makes closure code easier because ‘this’ is left alone
  - Requires polyfills

- Observables
  - Benefits: react to browser events with more control

- ng CLI – scaffold code & build/optimization tool

* ECMAScript 5 (2009) compared to ES6 ECMAScript 2015

# Web Component Architecture stays the same in Angular 6



## Same Web Component Concepts, New Syntax

```
// shoppingcart-detail.component.ts
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
import { Shoppingcart, ShoppingcartItem } from '../../core/shoppingcart.interface';

@Component({
  selector: 'app-shoppingcart-detail',
  templateUrl: './shoppingcart-detail.component.html' })
export class ShoppingcartDetailComponent implements OnInit {
  @Input() cart: Shoppingcart;
  @Output() updatedItem = new EventEmitter<ShoppingcartItem>();

  selectedCartitem = 0;
  constructor() { }

  updateItem(item: ShoppingcartItem) {
    console.log('updateItem() in sc-detail.comp');
    this.updatedItem.emit(item);
  }
}
```

# Angular 6 Template Syntax

```
<!-- shoppingcart-detail.component.html -->
<h3>Cart: {{cart.name}} ({{cart.id}})</h3>
<div *ngIf="!cart.items || cart.items.length == 0">No items</div>

<ul>
  <li *ngFor="let i of cart.items">
    {{i.productName}} {{i.quantity}} @ {{i.priceDollar | currency}}

    <span *ngIf="i.id == selectedCartitem">
      <button (click)="selectedCartitem=0">Close</button>
    </span>

    <span *ngIf="i.id != selectedCartitem">
      <button (click)="selectedCartitem=i.id">Edit</button>
    </span>

    <div *ngIf="selectedCartitem==i.id">
     <app-cartitem-edit [cart]="cart" [cartitem]="i"
(savedShoppingcartItem)="updateItem($event)"></app-cartitem-edit>
```

No $ctrl. anymore

# DI Angular 6

```
import { Component, OnInit } from '@angular/core';
import { ProductService } from '../../core/product.service';
import { ShoppingcartService } from '../../core/shoppingcart.service';
import { Shoppingcart, ShoppingcartItem } from '../../core/shoppingcart.interface';
@Component({
  selector: 'app-shoppingcart-list',
  templateUrl: './shoppingcart-list.component.html’
})
export class ShoppingcartListComponent implements
  isLoading: boolean;
  cartList: Shoppingcart[];
  cartSelected: Shoppingcart;
  constructor(
    private productService: ProductService,
    private shoppingCartService: ShoppingcartService) { }

  ngOnInit() {
    this.isLoading = true;
    this.cartList = [];
    this.getProducts();
```

# Angular 6 Create Service

```
// @Injectable decorator creates a Service.
// providedIn: 'root' ensures Singleton.
@Injectable({ providedIn: 'root' })
export class ShoppingcartService {
  private apiUrl = '/api/cart';

  constructor(
    private http: HttpClient ) {}

 // service methods here
}
```

# Observables replace Promises

```
// in shoppingcart.service.ts
// create an Observable of an array of Shoppingcart
// to be subscribe()'ed in a component.
getCartlist(): Observable<Shoppingcart[]> {
  const apiUrl = this.apiUrl + '/list';
  return this.http.get<Shoppingcart[]>(apiUrl);
}
```

```
// shoppingcart-list.component.ts
// called from ngOnInit or to update the list of carts.
getCarts(): void {
  // subscribe to the Observable,
  // save the data in the component and flip the isLoading flag.
  this.shoppingCartService.getCartlist()
    .subscribe(carts => {
      this.cartList = carts;
      this.isLoading = false;});
```

The intro to Reactive Programming by André Staltz:
https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

# forkJoin Multiple Observables

RxJS

```
ngOnInit() {
  this.isLoading = true;
  this.cartList = [];
  const products = this.productService.getProductsCached();
  const carts = this.shoppingCartService.getCartlist();
  // when starting up, dont show the page until
  // all required network resources are loaded.
  // forkJoin is the Promise.all for rxjs.
  forkJoin( [products, carts] ).subscribe( results => {
    // results[0] is products,
    // results[1] is carts.
    this.cartList = results[1];
    this.isLoading = false;
  });
```

Like chaining the Promise.then()'s in AngularJS, this uses a Boolean flag to delay the page render until both the Product and Shoppingcart services have returned.  This example loads in parallel.

## TypeScript your JSON

- Turn this:

```
{ "cart": {
   "totalDollar": 145.94,
   "dateModified": "September, 25 2018 05:23:55",
   "totalPoint": 0,
   "items": [
    {
     "priceDollar": 14.99,
     "quantity": 3,
     "dateModified": "September, 24 2018 18:06:50",
     "pricePoint": 0,
     "id": 28,
     "productName": "Drug Store Conditioner",
     "dateCreated": "September, 24 2018 18:06:50"
    },
    {
     "priceDollar": 28.00,
     "quantity": 2,
     "dateModified": "September, 25 2018 05:23:32",
     "pricePoint": 0,
     "id": 30,
     "productName": "Prestige Conditioner",
     "dateCreated": "September, 25 2018 05:23:32"
    },
    {
     "priceDollar": 14.99,
     "quantity": 3,
     "dateModified": "September, 25 2018 05:23:55",
     "pricePoint": 0,
     "id": 31,
     "productName": "Drug Store Shampoo",
```

## TypeScript your JSON

- Into this:

```typescript
export interface Shoppingcart {
  id: number;
  name: String;
  totalDollar?: number;
  totalPoint?: number;
  items?: ShoppingcartItem[];
  dateCreated: Date;
  dateModified: Date;
}

export interface ShoppingcartItem {
  id: number;
  quantity: number;
  priceDollar: number;
  pricePoint?: number;
  productName: string;
  productId?: number;
  dateCreated: Date;
  dateModified: Date;
}
```

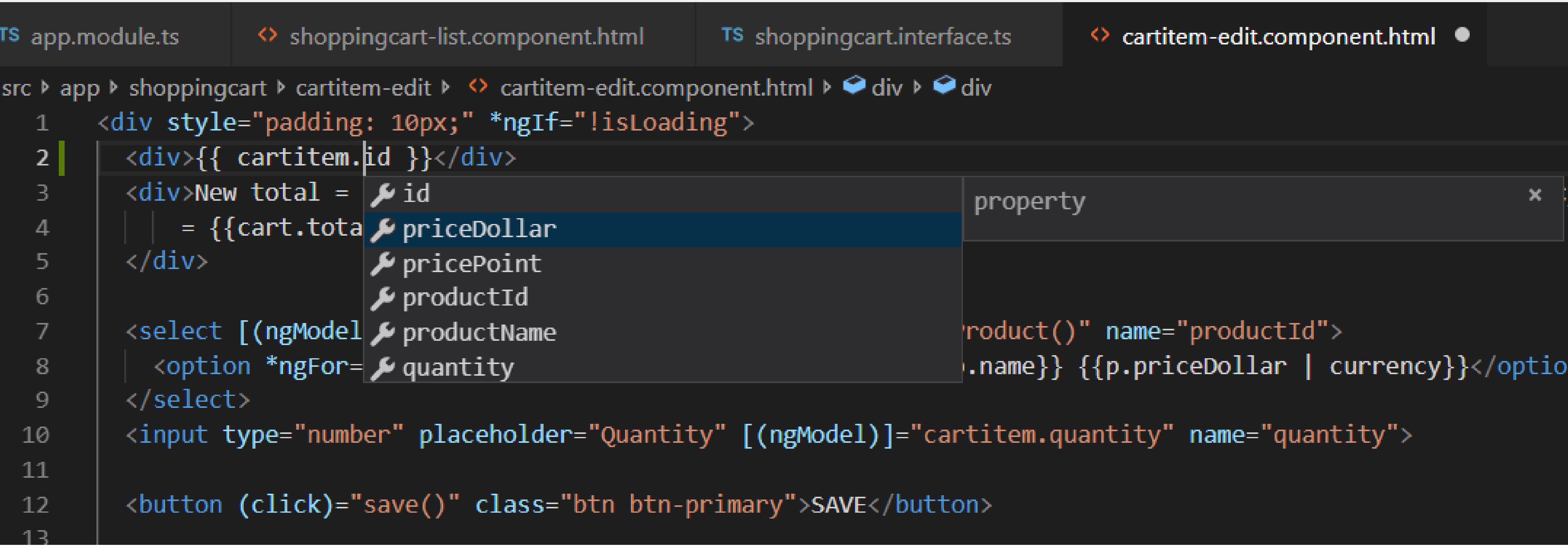
## TypeScript your JSON

- By doing this:

```
// in shoppingcart.service.ts
// do HTTP GET /api/cart/{cartId}
getCart(cartid: Number): Observable<Shoppingcart> {
  return this.http.get<any>(this.apiUrl + '/' + cartid)
    .pipe(
      map(response => response.cart as Shoppingcart),
        tap( … )
      );
  }
```

## Intellisense (in a template)

Code



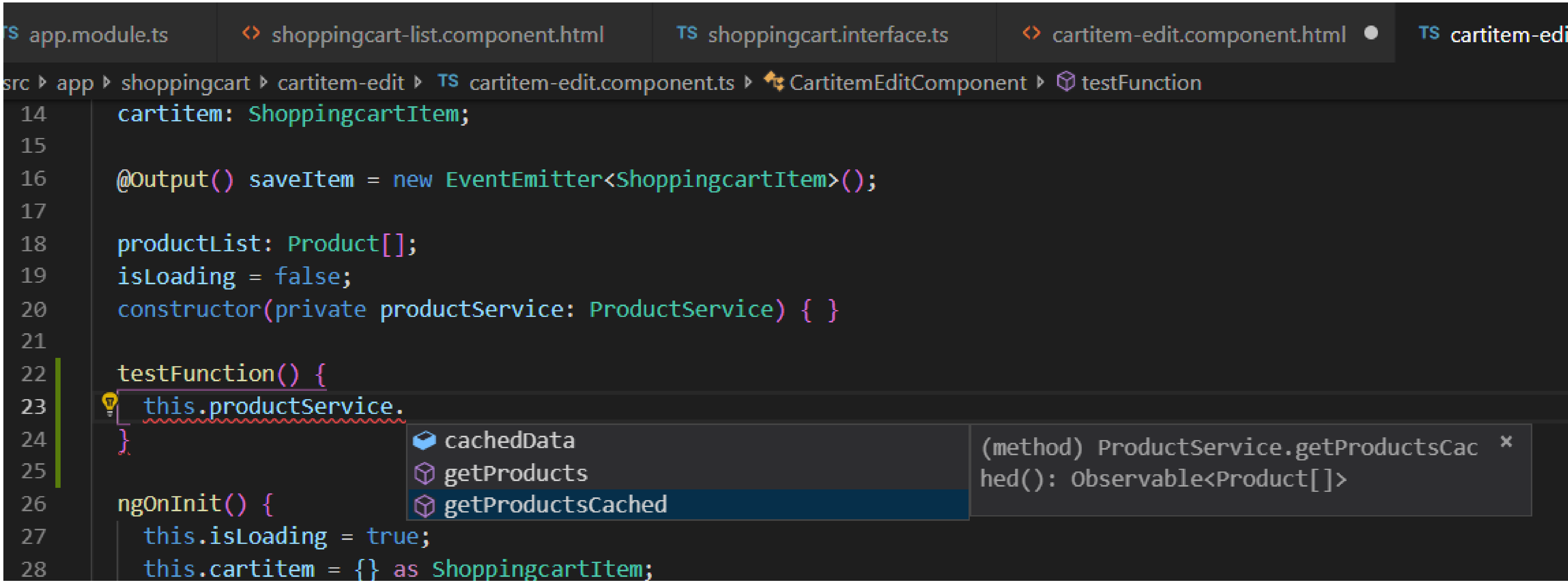
app.module.ts | shoppingcart-list.component.html | shoppingcart.interface.ts | cartitem-edit.component.html

src › app › shoppingcart › cartitem-edit › cartitem-edit.component.html › div › div

```
<div style="padding: 10px;" *ngIf="!isLoading">
  <div>{{ cartitem.id }}</div>
  <div>New total =
    = {{cart.tota
  </div>

  <select [(ngModel                                              roduct()" name="productId">
    <option *ngFor=                                              .name}} {{p.priceDollar | currency}}</optio
  </select>
  <input type="number" placeholder="Quantity" [(ngModel)]="cartitem.quantity" name="quantity">

  <button (click)="save()" class="btn btn-primary">SAVE</button>
```

VS Code w/ Angular Language Service Extension & TypeScript support

# Intellisense (in .ts files)



app.module.ts | shoppingcart-list.component.html | shoppingcart.interface.ts | cartitem-edit.component.html | cartitem-edi

src › app › shoppingcart › cartitem-edit › cartitem-edit.component.ts › CartitemEditComponent › testFunction

```
  cartitem: ShoppingcartItem;

  @Output() saveItem = new EventEmitter<ShoppingcartItem>();

  productList: Product[];
  isLoading = false;
  constructor(private productService: ProductService) { }

  testFunction() {
    this.productService.
  }

  ngOnInit() {
    this.isLoading = true;
    this.cartitem = {} as ShoppingcartItem;
```

```
cachedData
getProducts
getProductsCached
```

```
(method) ProductService.getProductsCached(): Observable<Product[]>
```

N.B. VS Code + CFML Extension by KamasamaK + CFLint.jar will intellisense var scope bugs in a <cfcomponent>!

## Conclusion

By adopting Angular, you will gain:

- Best of both worlds:
  - Pure js frontend
  - Pure cf backend
- Usability, Agility, Quality
- Code Reuse
- Performance
- Developer Productivity & Engagement

## Q & A

- Do you think you will try Angular?
- Version 1.7 or 6?

# P.S.

Don’t forget:

- Error reporting
  - Need a reporting endpoint and an Exception Override
  - Not perfect
- Browser compatibility
  - Need Polyfills for some js functionality (find, map, …)
- There will be some duplication between app and API
  - **Validation**
- SEO & first time page render
- Security
- Case-insensitivity

## Exception Override

- Catch errors and forward to a logger endpoint

```
// File misc/exceptionOverride.js
var mod = angular.module('exceptionOverride', []);
mod.config(function ($provide) {
  $provide.decorator("$exceptionHandler",
    ['$delegate', '$injector', function ($delegate, $injector) {
    return function (exception, cause) {
      var $http = $injector.get("$http");
      var $log = $injector.get("$log");
      var registration = $injector.get("Registration");
      registration.account.contacts = ['Snipped']; //dont need these filling the error log.
      console.log("Registration", registration);
      //also, exception.message, .fileName, .lineNumber
      var data = { exception: exception.toString(), message : exception.msg,
        stack: (exception.stack) ? exception.stack : 'No Stacktrace Found’, … };
      //want to only log non-http errors.
      var exception_text = exception.toString();
      if (exception_text.indexOf("$http") == -1) {
        $http.post('error.cfm', data);
```

## Resources

- AngularJS docs
  - https://docs.angularjs.org/tutorial
  - https://docs.angularjs.org/guide
- Angular docs - https://angular.io/docs
- Ng-book - https://www.ng-book.com/
- UpgradingAngularJS.com
  - Paid Course on migration from AngularJS to modern
  - Videos that show all the steps
- John Papa style guide
  - https://github.com/johnpapa/angular-styleguide

# Cheatsheet

| | CF | AngularJS | Angular |
|---|---|---|---|
| String output | #variable# | {{ variable}} | {{ variable }} |
| Combine files | <cfinclude> | <webComponent my-param=“”> | <webComponent [param]=“”> |
| Iterator | <cfloop> | <li ng-repeat=“c in $ctrl.cartlist”> | <li *ngFor=“let cart of cartlist”> |
| Dump | <cfdump var=“#variable#”> | {{ variable \| json}} | {{ variable \| json }} |
| DI | component accessors=“true” {<br>Property MyService; | Angular.module(‘app’)<br> .factory(‘MyService’) | @Injectable()<br>export class MyService {} |
| Code packages | mappings | Angular.module(‘app’) | @NgModule() |

## Debugging

- {{ var | json }}
- Chrome Devtools (F12) is your friend.
- AngularJS: angular.element(document.body).injector();
- Angular v6 – Augury Extension - $$el.componentInstance
- Dev vs. Prod builds
- CommandBox server log –follow
    - Rewrites too

## Angular v6 How-to

- New development workflow: ng-cli & a build process
  - Watch out for virus scanning software, vpn security, etc.
- Will need to learn npm – the Node Package Manager

```
"scripts": {
  "start": "ng serve --proxy-config proxy.config.json --progress --open",
```

```
{
  "/api": {
    "target": "http://127.0.0.1:8070/api/",
    "secure":false,
    "pathRewrite": {
      "^/api" : ""
    },
    "logLevel":"debug"
```

## Gotchas

- Case-sensitive
- && instead of AND
- Arrays start at 0! (Array syntax in general, push vs append, .length vs .len())
- Structure key syntax uses :, not =
- HTML comments use two dashes, CFML uses three

## ngRouter

- Use the URL to maintain application state

## Observables: Everything is a stream

- RxJS
- Hot vs Cold
- HTTP requests are a cold stream that emits one value
- Nothing happens without the .subscribe()
- Rxmarbles.com

## Testing

This is the pay-off
from DI.

- See the *.spec.ts files in the Angular 6 Demo
- AngularJS:

```
// test the service against known JSON:
describe("http data test", function() {
  it("matches known json", inject(function($injector) {
    var $httpBackend = $injector.get('$httpBackend');

    var acc = $injector.get('Account');

    $httpBackend.when('GET', 'json.cfm?method=getAccount&id=125489')
      .respond({account:{}, points:{educ:100}, contacts:{}, cards:{}, comps:[]});
    var p = acc.get('125489');
    $httpBackend.flush();

    expect(acc.isLoaded).toBe(true);
    expect(acc.points.educ).toBe(100);
  }))
```

# Future Directions

- Angular Universal (on the server) & Performance Improvements
- Redux State Machine
- Angular Material Component Library
- NativeScript
- WebSocket
- Service Workers