# Client side choices

## TypeScript & AngularJS 2

Using single page applications to your advantage

**Doug Hoff – email@doughoff.com**

CENTRIQ TRAINING TechSmartKC

# Web Architecture

CENTRIQ TRAINING

TechSmartKC

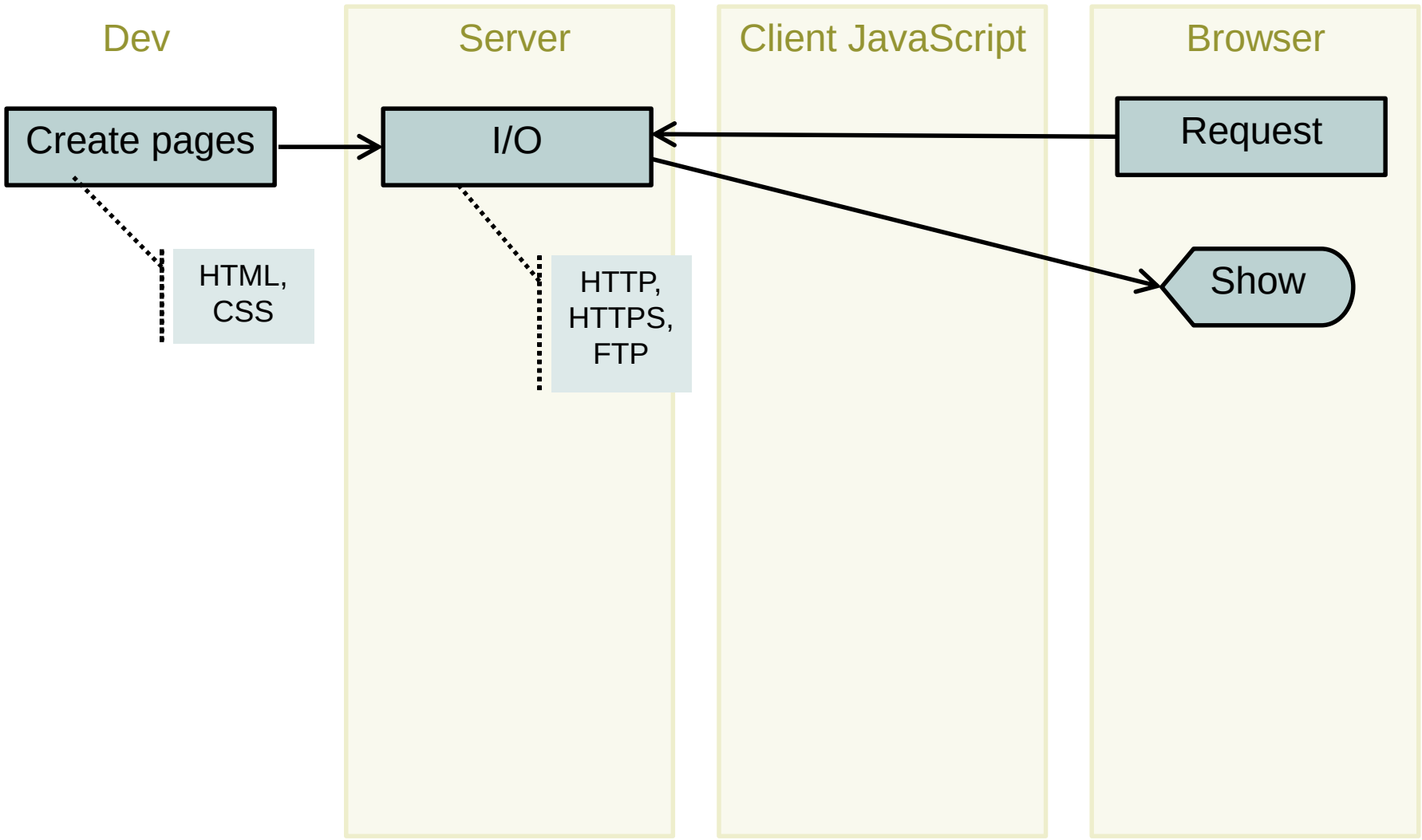
# Browser client operations

- make http requests
- bundle up form data
- execute JavaScript
  - process user interactions, update DOM
- parse and render html & CSS
- allow plugins to extend features
- cache/retrieve local files
- render XML, RSS, SVG, GZIP…
- launch apps for other files

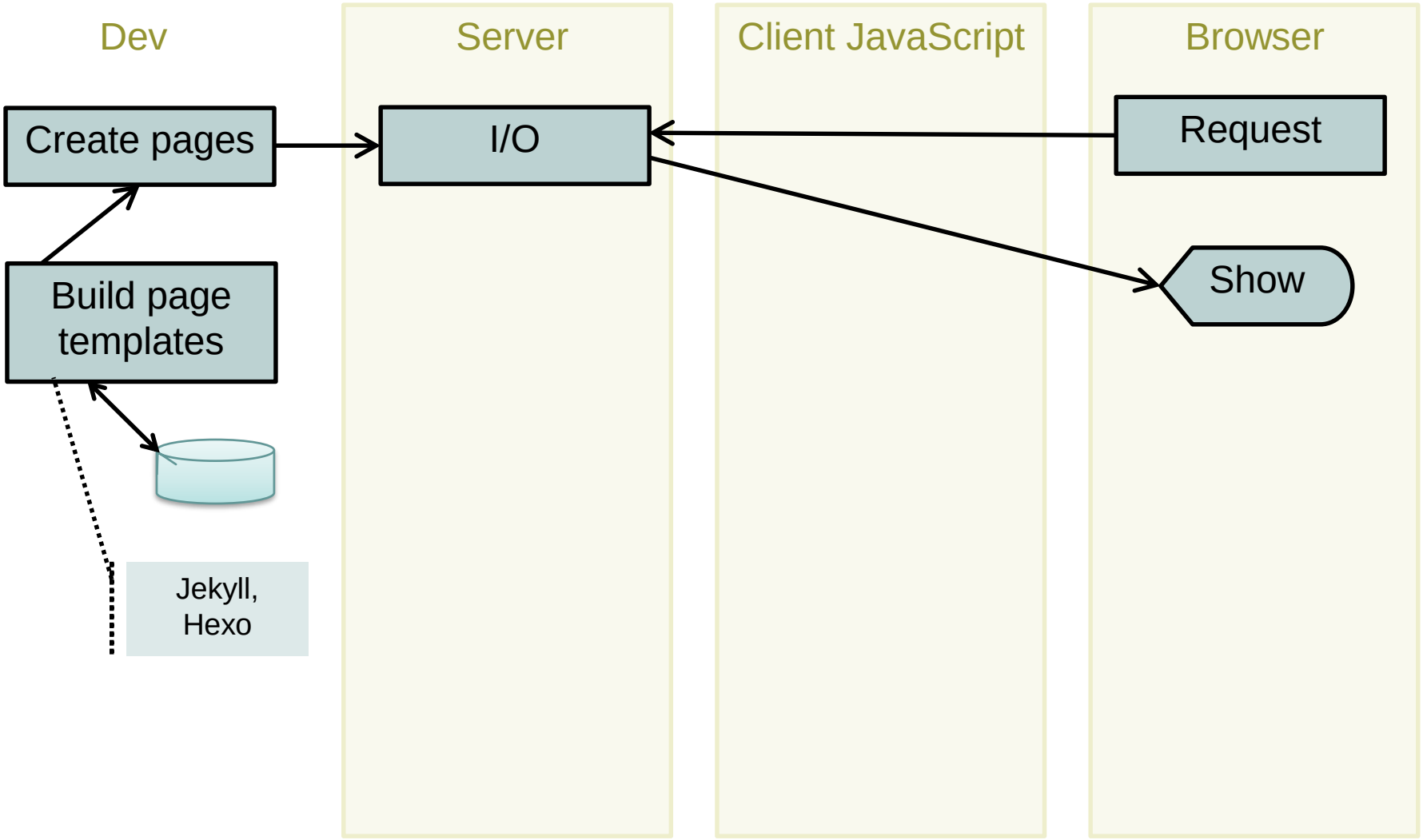
# Web server operations

- listen/respond to http request
- create a header
- get and send files
- handle email, ftp
- execute a language –build web pages
  - process routing
  - process page templates
  - process authentication rules
  - talk to DBMS
- manage multiple web sites
- store session data

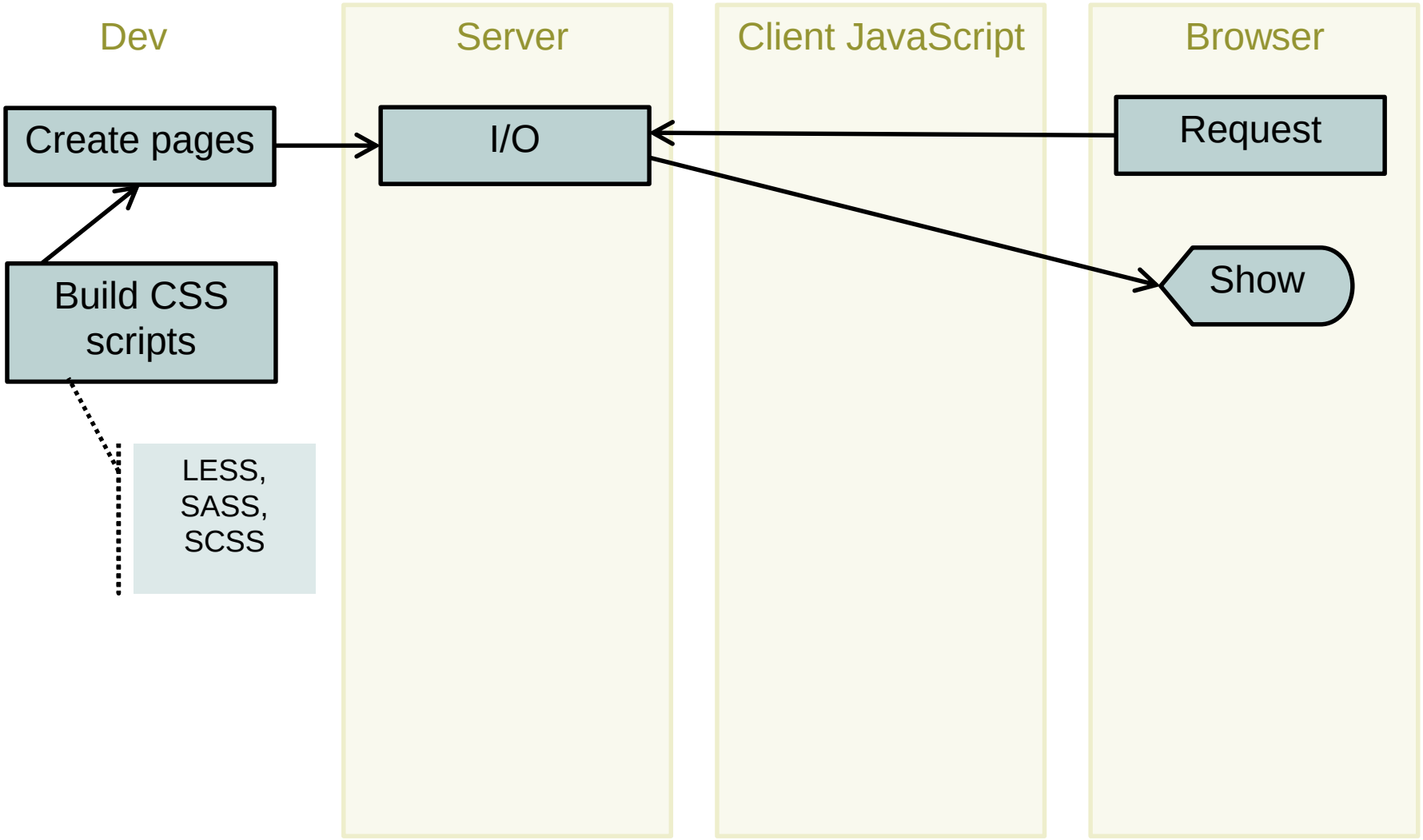
# Web 1.0


Dev
Server
Client JavaScript
Browser
Create pages
I/O
Request
Show
HTML, CSS
HTTP, HTTPS, FTP

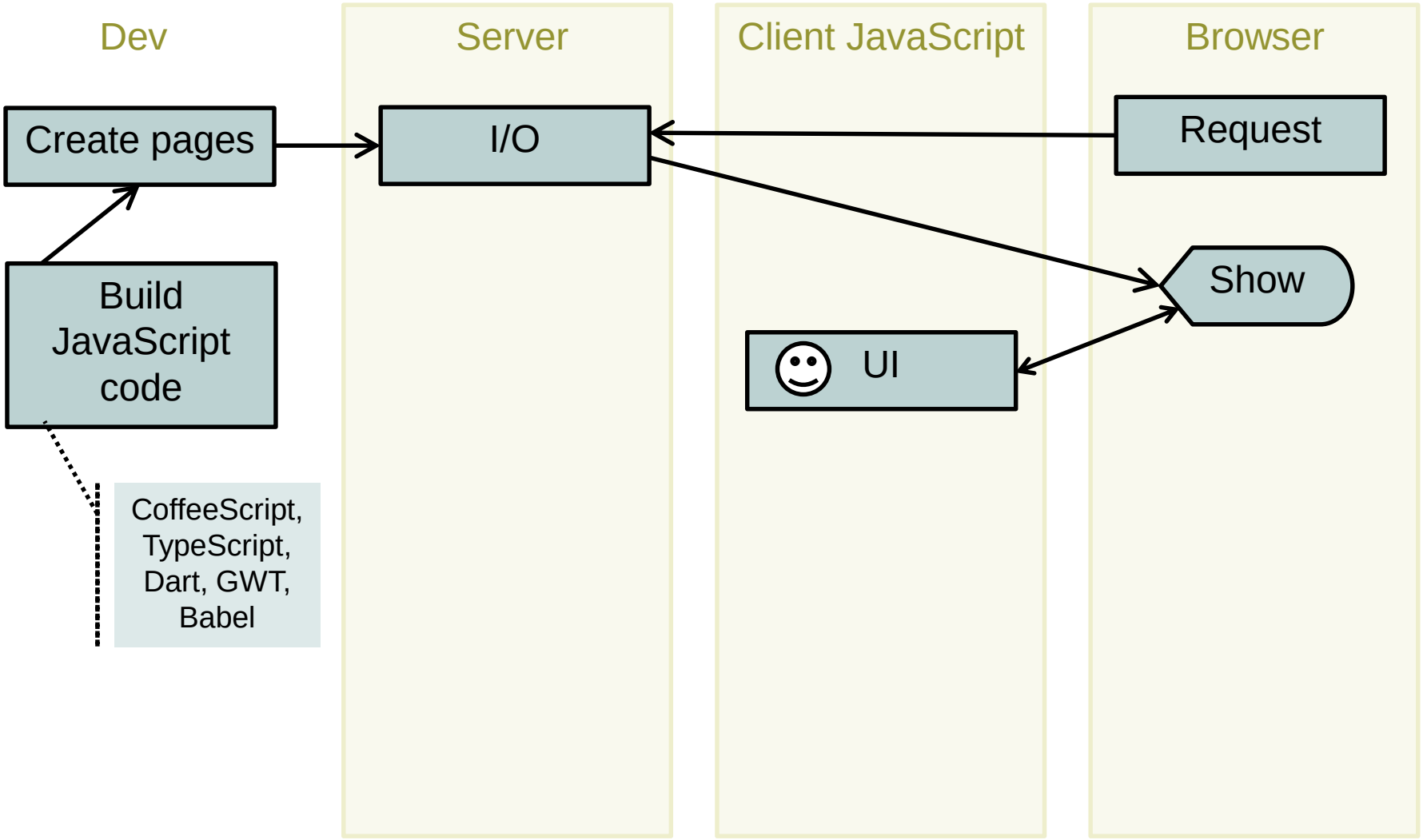
# Web 1+ – static site generators



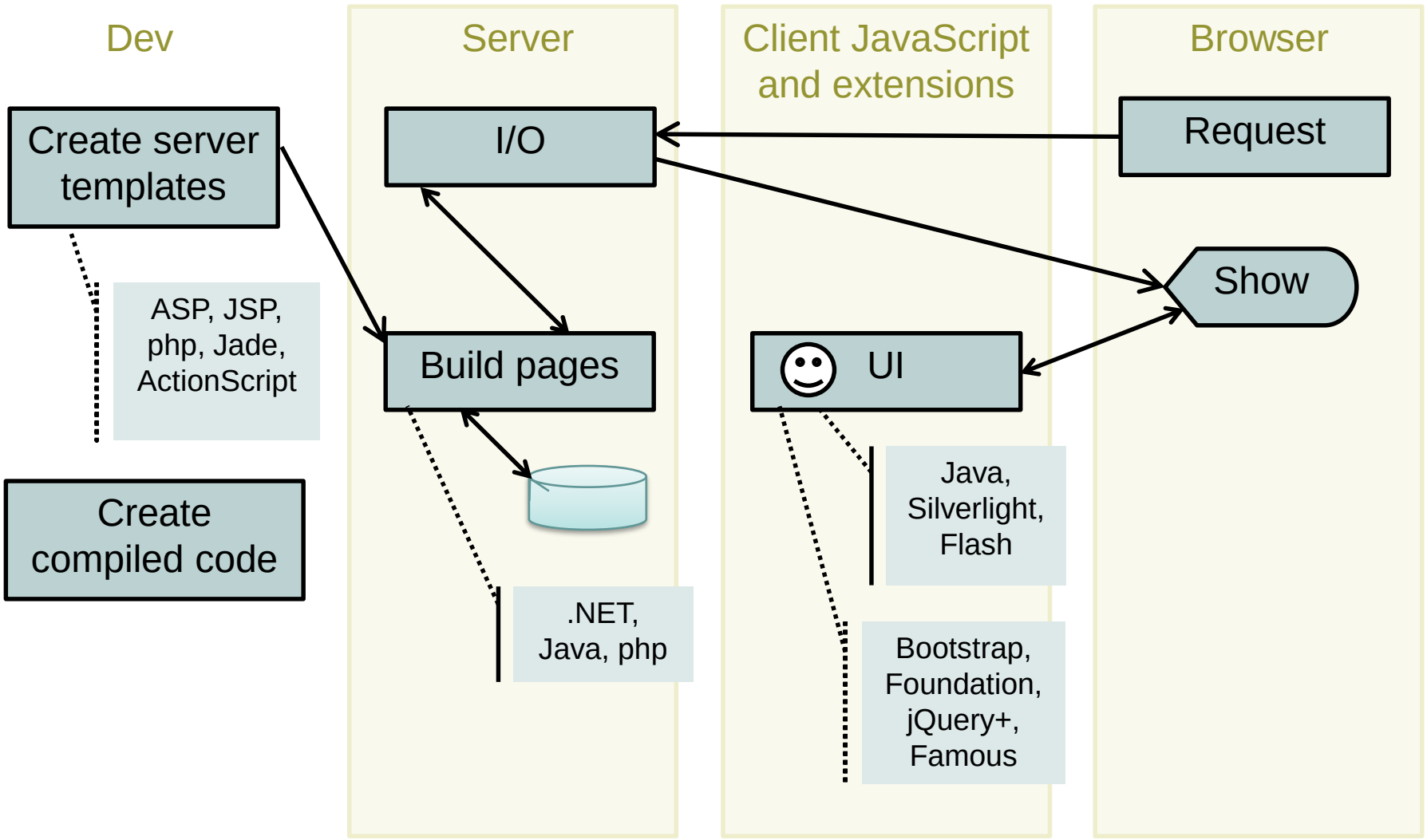
# Web 1+ – CSS generators



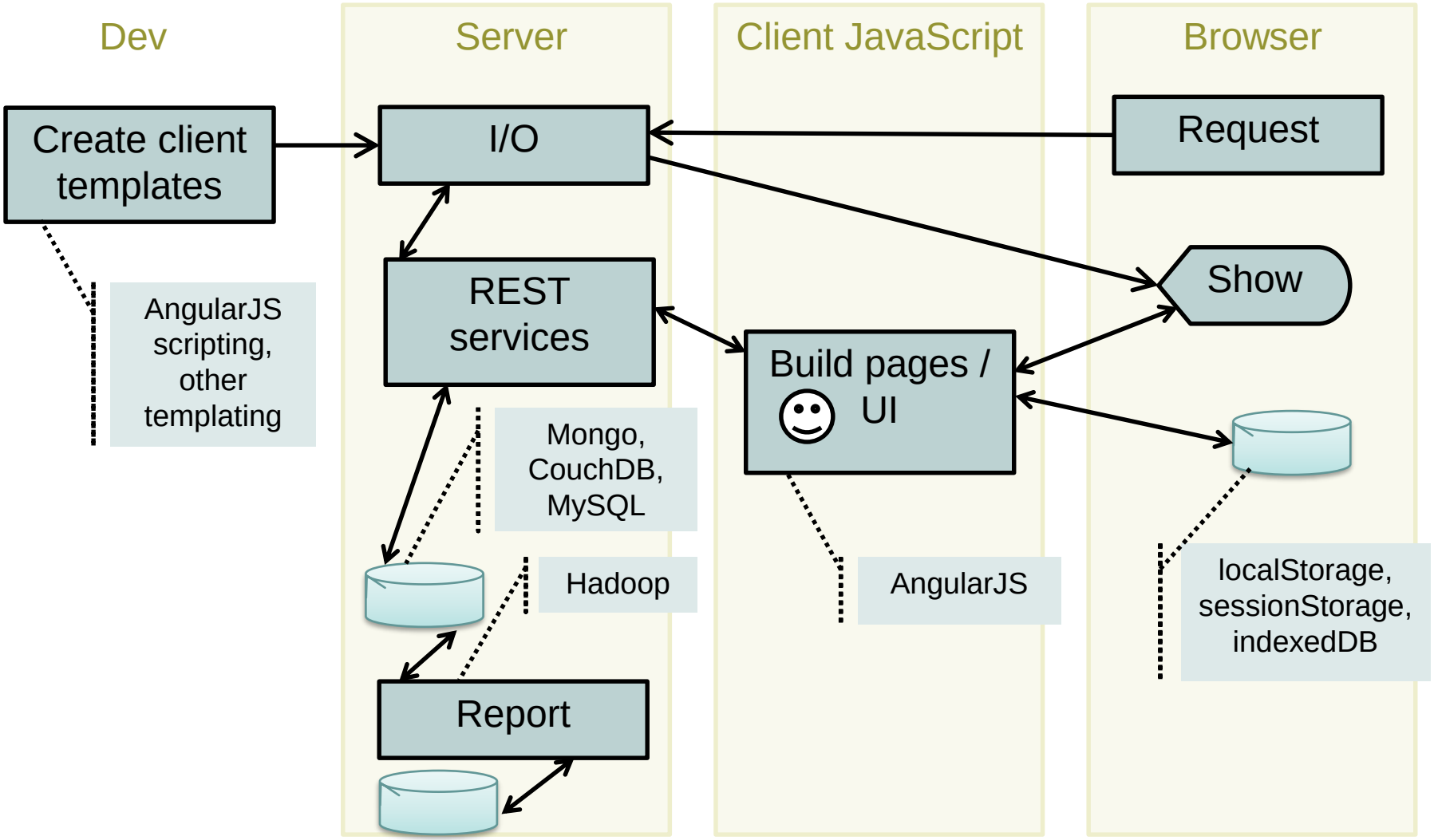
# Web 1+ – JavaScript generators



# Web 2.0 – dynamic web sites



# Web 3 – apps


Dev
Server
Client JavaScript
Browser
Create client templates
I/O
Request
AngularJS scripting, other templating
REST services
Show
Build pages / UI
Mongo, CouchDB, MySQL
Hadoop
AngularJS
localStorage, sessionStorage, indexedDB
Report

# Architecture - SS framework

**Server – ASP.NET or MVC**

- Routing
- Controller logic
- Page generation
  - Data binding
  - Templates
- Security
- Services for client
  - data extraction

**Client – jQuery, Bootstrap, etc.**

- user triggered CSS
  - click / touch / hover
- user triggered server process
- browser triggered CSS
  - screen width

# Architecture – SPA framework

**Server – static files**

- Services for client

**Client – browser with**

- **Routing**
- **Controller logic**
- **Page generation**
  - **Data binding**
  - **Templates**
- **Security**
- user triggered CSS
  - click / touch / hover
- user triggered server process
- browser triggered CSS
  - screen width

# Angular features

- Page generation
  - Data binding
  - Templates
- Controller logic
- Routing
- Reusable components

# Integration with server side frameworks?

- Duplication of page generation
- Server side
  - Focus on data base APIs & security - Firebase, etc.
  - Simplify with static page generation - https://www.staticgen.com/
- Client side
  - Component library
  - Few dependencies allow mobile use


modern JavaScript

CENTRIQ
TRAINING

# Creation

- Anders Hejlsberg (1960 - )
  - Borland - Turbo Pascal, Delphi
  - 1996 Microsoft – J++, 2000 C# lead architect
- C# / Java friendly
- Better large-scale development
  - static typed
    - more testable, better IDE support
  - outputs ES 3 or 5

# Versions

- 0.8 – Oct 2012
- 1.0 – April 2014
- 1.8 – Jan/Feb 2016
  - 1.8.9 – Mar
- 1.9 – Apr 2016 dev

# Use cases

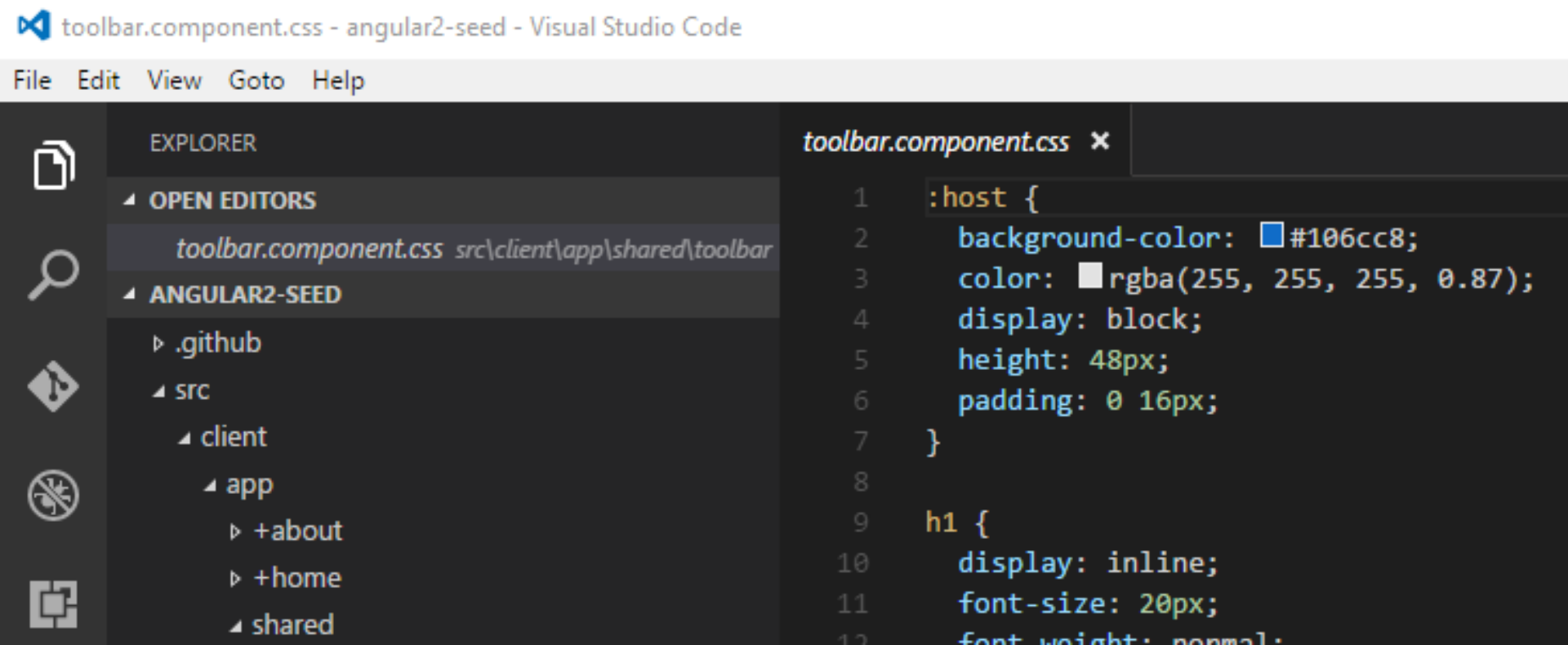
- Local development
  - install transpiler locally, convert JS code and deploy
- Remote development
  - use remote transpiler to convert JS code and deploy
  - download transpiler code, load web app, browser manages transpile

# Install TypeScript

- http://www.typescriptlang.org/
- Install node.js to get npm
  - Download from https://nodejs.org
- Install typescript with npm
  - `npm install -g typescript`
- Compile a file from the command line
  - `tsc helloworld`
- Compile and watch for changes then recompile
  - `tsc helloworld --watch`

# Microsoft Visual Studio Code

- an Electron project – not like VS
- supports JS, TS, C#, git, …



# Typing

- var counter; // unknown (any) type
- var notSure**: any** = 4;

- var counter = 0; // number (inferred)
- var counter **: num**ber; // number
- var counter **: number** = 0; // number, initialized
- var list: **number[ ]** = [1, 2, 3];
- var isDone: **boolean** = false;
- var name: **string** = "bob";

# Scope – var vs let

- function / **lexical** scope
  - **var** mynum : number = 1;
- **block** scope
  - **let** mynum : number = 1;
  - **const** mynum : number = 1;

# Function return types

- var getAOne  = function( )  { return 1; }
- var getAOne  = function( ) : **any** { return 1; }
- var getAOne  = function( ) : **number** { return 1; }
- function getNothing() : **void** {
-  return;
- }

# Function declaration types

- function declaration with name
  - **function getOne( ) : number** { return 1; }
- function declaration assigned to variable
  - var getADigit = **function getOne( ) : number** { return 1; }
- anonymous function declaration assigned to variable
  - var getAOne = **function ( ) : number** { return 1; }
- arrow function declaration assigned to variable
  - var getOneDigit = **( ) : number =>** { return 1; };

# Arrow / lambdas

- Arrow aka lambda function
  - a variation on the anonymous function

- function (radius) { return Math.PI * Math.pow(radius, 2); };
  - can be written
- (radius) => { Math.PI * Math.pow(radius, 2); }
- radius  => Math.PI * Math.pow(radius, 2);

# Parameters

- ? allows a nulllable type, aka optional parameter

- function sendNumString (firstArg : number, secondArg?: string) : void {
- return;
- }
- sendNumString(1,'a');
- sendNumString(1);

# Parameters

- Parameter with default value
  - function order(meal: string, **drink : string = 'water'**) : void {
  - console.log('You ordered', meal, drink);
  - }
  - order('a hamburger');
  - order('a cheeseburger', 'pepsi');
- Optional parameters with default value not allowed. (but will run as JS)

# Template strings

- backticks delimit a string with ${ } variables
- function order (**meal**: string) : string {
- return **`Ordered a ${meal}`**;
- }

# Class declaration, instance vars

- class Dog {
- name: string;
- age: number;
- }
- var fido: Dog = new Dog();
- console.log(fido);

# Accessors

- requires ES5
- class Gift {
- private _contents : string;
- **get contents( )**: string { return this._contents; }
- **set contents(incoming: string)** { this._contents = incoming; }
- }
- var xmasPresent: Gift = new Gift();
- xmasPresent.**contents** = "pair of socks";
- console.log(xmasPresent.**contents**);

# Static properties

- one value for all class objects
- class MarshallsGift {
- static storeName: string = 'Marshall\'s';
- }

- console.log(MarshallsGift.storeName);

# Constructor

```
class Dog {
        private name: string;
        private age: number;
        constructor(name? : string, age?: number) {
                this.name = name || 'Rover' ;
                this.age = age || 5;
        }
}
var fido: Dog = new Dog();
console.log(fido);
```

# Interfaces

```
interface HasBooleanCheck {
        result: boolean;
        isTrue(), isFalse(): boolean;
}
class Box implements HasBooleanCheck {
        private result: boolean;
        constructor() {
                this.result = false;  }
        isTrue(): boolean { return this.result;}
        isFalse(): boolean { return this.result;}
}
```

# Modules

- modules = external modules (TS 1.5 – ES6)
  - declared dependencies
  - better code reuse, stronger isolation, better tooling support
  - recommended
  - any file containing a top-level import or export is considered a module (ES 2015)
  - **export** from module, **import** into code = require( )
- Compile requires target of module loader
  - ES6 modules are not supported in node 4 yet.

# AngularJS by Google

Building the next version of the web with browser applications

CENTRIQ TRAINING

# History

- 2009 – team started with Brad Green, manager
- Sep 2012 - 1.0.2
- March 2015 - Angular 2 announced
  - Dec – Angular 2 beta
  - May 2016 – RC
    - June – RC4

# Resources - official

- Site: https://angular.io/
- Code: https://github.com/angular
- Docs: https://angular.io/docs/
  - Cheatsheet - https://angular.io/docs/ts/latest/guide/cheatsheet.html
- Blog: http://angularjs.blogspot.com/

# Angular Universal

- Parses your app's JavaScript by pre-rendering the first view on the server-side.
  - Full Stack Angular 2, Jeff Whelpley and Patrick Stapleton - https://www.youtube.com/watch?v=MtoHFDfi8FM
- Using server-side rendering in IIS via nodeServices (not yet available)
  - Steve Sanderson (ng2 + ASP.NET5 / MVC6 Music Store, React – no TypeScript) Nov 2015 - Channel9 video
  - https://github.com/aspnet/NodeServices/tree/master/samples

# Angular CLI

- https://cli.angular.io/
- Scaffolding tool
- Based on Ember's CLI
- Automates basic tasks for setup and boilerplate code

# Other

- Animations
  - https://angular.io/docs/ts/latest/guide/animations.html
- Testing with Jasmine, Karma, Augury (Chrome extension
  - https://angular.io/docs/ts/latest/guide/testing.html
  - https://augury.angular.io/
- RxJS ("Reactive Extensions")
  - a 3rd party library, endorsed by Angular, that implements the *asynchronous observable* pattern.

# Hybrid apps using Angular

- Ionic2
  - http://ionic.io/2
    - alpha – 11/2015
  - Build native apps from JS/TS APIs
- Telerik's NativeScript
  - http://www.telerik.com/nativescript
  - Build native apps with XML custom language

# Setup choices – code only

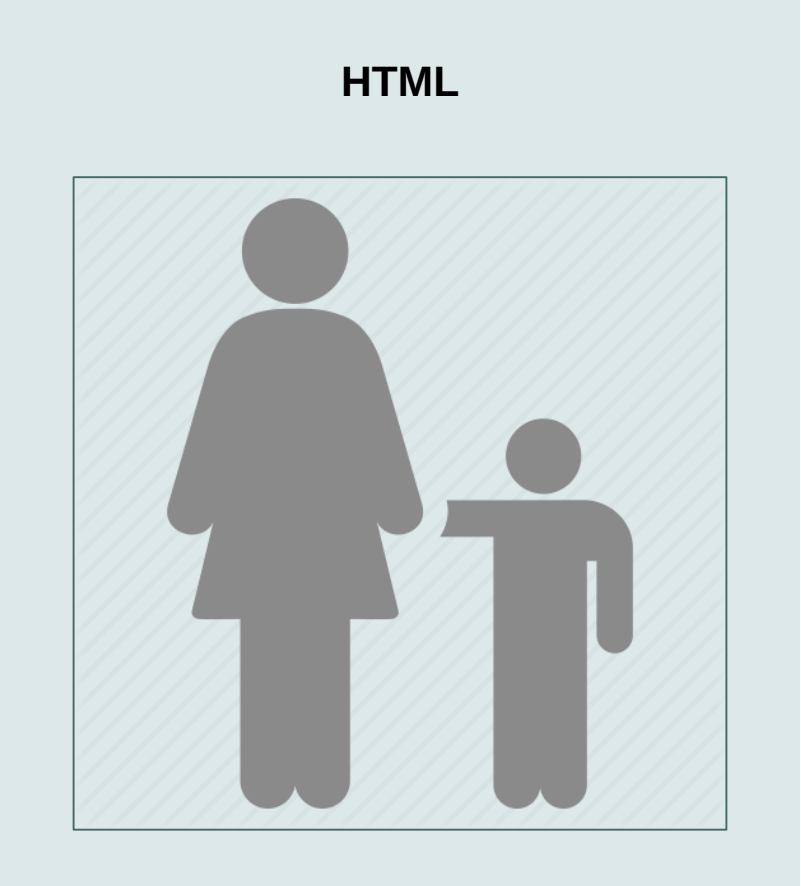
- npm angular2
  - https://www.npmjs.com/package/angular2
  - 2.0.0-beta.17 – 7/19/2016
- GitHub angular-master
  - https://github.com/angular/angular
- Quickstart
  - https://github.com/angular/quickstart
  - Pre-built ES5 version of Angular 2 alpha-14

# Setup choices – code + scaffold

- Angular CLI
  - https://cli.angular.io/
- Minko Gechev's Angular Seed project
  - https://github.com/mgechev/angular2-seed

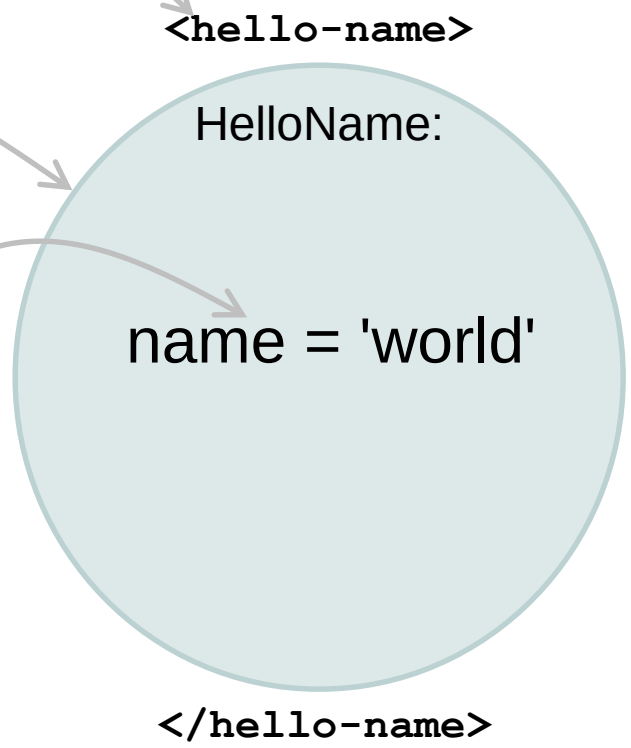
# App structure

- HTML page
  - HTML
  - bootstrap code
  - root app selector
    - component code
    - template
    - styles
    - dependencies
    - child components selectors
      - child component code…



# Component to DOM – app.ts

- import { Component } from "angular2/core";
- @Component({
- selector: 'hello-name'
- })
- export class HelloName
  private name: string = 'world';
  …



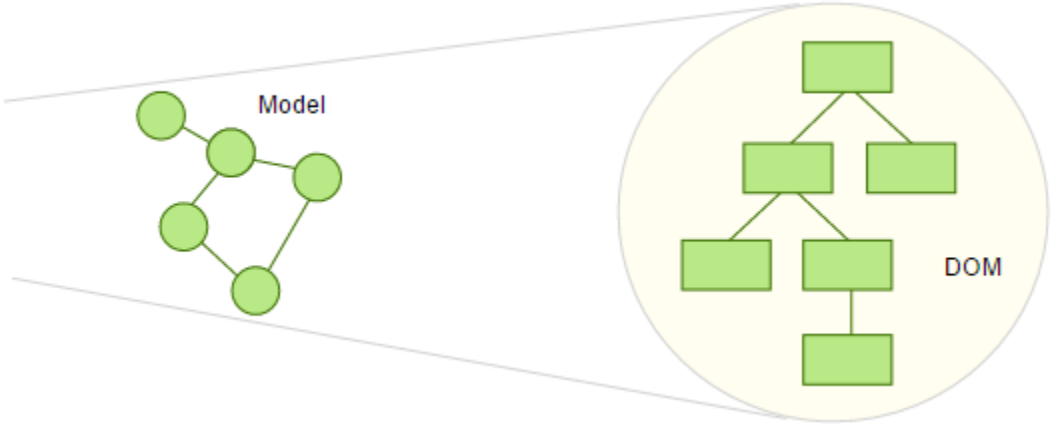
# Data binding – one way values

- from class (data source) to template in DOM (view target)
- most often a @Component class property
- template:`
  - `<input type="text" value="{{name}}" />`
  - `<div>Hello, {{name}}!</div>`
- export class BuiltIn {
- private **name**: string = 'John Smith';

}

# Data binding – projecting data

- Data model in code → DOM
- mapping, projecting, no change
- updates require mapping/binding
- must track state (the model/DOM data)



# Templates – inline vs file

- **template**:`<div>Hello, {{name}}</div>`
  using ES6 template strings
  or
- **templateUrl**: '/templates/hello_name.html'
  - path from root, not file

# Child pages

- In order to use directives within a directive

```
import {Component} from 'angular2/core';
import {SpyComponent} from './spy.component';

@Component({
    selector: 'app',
    template: `<div><spy></spy></div>`,
    directives: [SpyParentComponent]
})
```

# Selectors

- selector: 'custom-box, .custom-box, [custom-box]', :not( )
  - **<custom-box>Matching tags</custom-box >**
    - < custom-box > is not valid – must use a closing tag, not empty
  - <span **class= custom-box '**>a class
  - <span **custom-box** >an attribute
- selector: '.custom-box:not(h1)'

# Styles

- Defined in the @Component decorator
- Written to a style element in the rendered page.
- Styles are bounded by the element of the selector (view encapsulation)
  - Emulated View Encapsulation - default
- **styles:** [ '.primary {color: red}', '…' ]
  - an array of rule-sets, not a multi-line string for all!
- **styleUrls:** [ 'my-component.css' ]

# Syntax - literals

- Mixed with ASP.NET server data bindings
  - {{ '<%= DateTime.Now %>' }}
  - server code executes first, then renders client side literal
- Mixed with attribute value text
  - <a href="img/{{ username }}.jpg">

# Property binding

- set the default value of an input from the class
- template:`
- <input **[value]** = "**defaultName**"> `
- })
- export class BuiltIn {
- private **defaultName** : string = 'John Smith';

}

  - value does not work with ngControl="…"

# Property binding

- <button [disabled]="isUnchanged"> Save </button>
- Not the attribute of button!
- The property of the DOM element
  - or Component or Directive
- Attributes initialize DOM properties – final
  - watch the DevTools when you update a text field
- Property bindings are not final
  - button disabled="false" does not work
  - button [disabled]="isInvalid()" does work

# Local variables

- uses pound sign before a scoped variable name for DOM element
- also called a resolve
- <div **#newDiv** />
  - almost like **id='newDiv'** for cross element access
- variable is now accessible from this element or in any descendant
- alternative syntax
  - <div **var-newDiv** />

# Other bindings

- Style
  - <button [**style.color**] = "isSpecial ? 'red' : 'green'">
- Class
  - <div **[class.myClassName]**="isTruthy">
- Attribute
  - <div **[attr.**role**]** = "myAriaRole">

# for - syntax

- collection to iterate over in Component class
  - private names: string[ ] = ["Alfred", "Bill", "Charles"];
- template
  - <li *ngFor="let **item** **of** names">Hello {{ **item** }}</li>
  - <div *ngFor="let city **of** cities; **let i = index**">
    {{city}}
- let <var>– declare local variable

# if - syntax

- @Component({
- selector: 'built-in',
- template:`
- <div ***ngIf** ="x > y"> x bigger than y
- <div ***ngIf** ="x <= y"> x less than or = to  y `
- })
- export class BuiltIn {
- private x : number = 300;
  private y : number = 200;
- }

# switch - syntax

- value
  - <div [ngSwitch]="x>y">
  - <template **[ngSwitchWhen]**="true">x > y</template>
  - <template [ngSwitchWhen]="false">y > x</template>
  - <template **ngSwitchDefault**>default text</template>
  - 
- literal string
  - <div [ngSwitch]="stringVar">
  - <template **ngSwitchWhen**="a">aaaa</template>
  - <template ngSwitchWhen="b">bbbb</template>
  - <template ngSwitchDefault>not a or b</template>
  - 

# Pipes - common and custom

- Common
  - The hero's birthday is {{ birthday | date:’ ‘ }}
- Custom
  - Card No.: {{ cardNumber | myCreditCardNumberFormatter }}

# Event binding

- embed event in element like it would be in JS
  - JavaScript
    - <button **onclick**='showMessage( )' >Show Message</button>
  - ng2 – event only in parentheses
    - <button **(click)**='showMessage( )' >Show Message</button>

# Form submit process

- Input (model)
- Create form merging model data
  - state pristine
- User enters data
  - state dirty
- Validate data by field
  - state valid/invalid, show/clear error messages
- User submits data
- Output (ngForm)

# Form styles

- manual binding
  - inputs are bound to local variables
- template-driven
  - inputs are 2-way bound to ngModel
  - build forms with very little to none application code required
- model-driven
  - ngFormModel
  - testability without a DOM being required
- model-driven with FormBuilder

# Validation – built-in

- Angular
  - required (HTML5 also)
  - minLength(#)
  - maxLength(#)
  - nullValidator
  - pattern('regex string')
- https://coryrylan.com/blog/angular-2-form-builder-and-validation-management - using validations

# Service provider

- any value, function or feature that our application needs
- just a class with a narrow, well-defined purpose
  - logging service
  - data service
  - message bus
  - tax calculator
  - application configuration

# Router use cases

- Create manageable paths for same page/app
  - http://<domain name>/person // search
  - http://<domain name>/person/all // show all
  - http://<domain name>/person/345 // details
  - http://<domain name>/person/create
  - http://<domain name>/person/edit/345
  - http://<domain name>/person/delete/345

# Blogs

- Victor Savkin – http://victorsavkin.com
- Thoughtram - http://blog.thoughtram.io/
- Scotch.io - https://scotch.io/tag/angular-js