

# CURSO DESARROLLO WEB CON MEAN



# M E A N

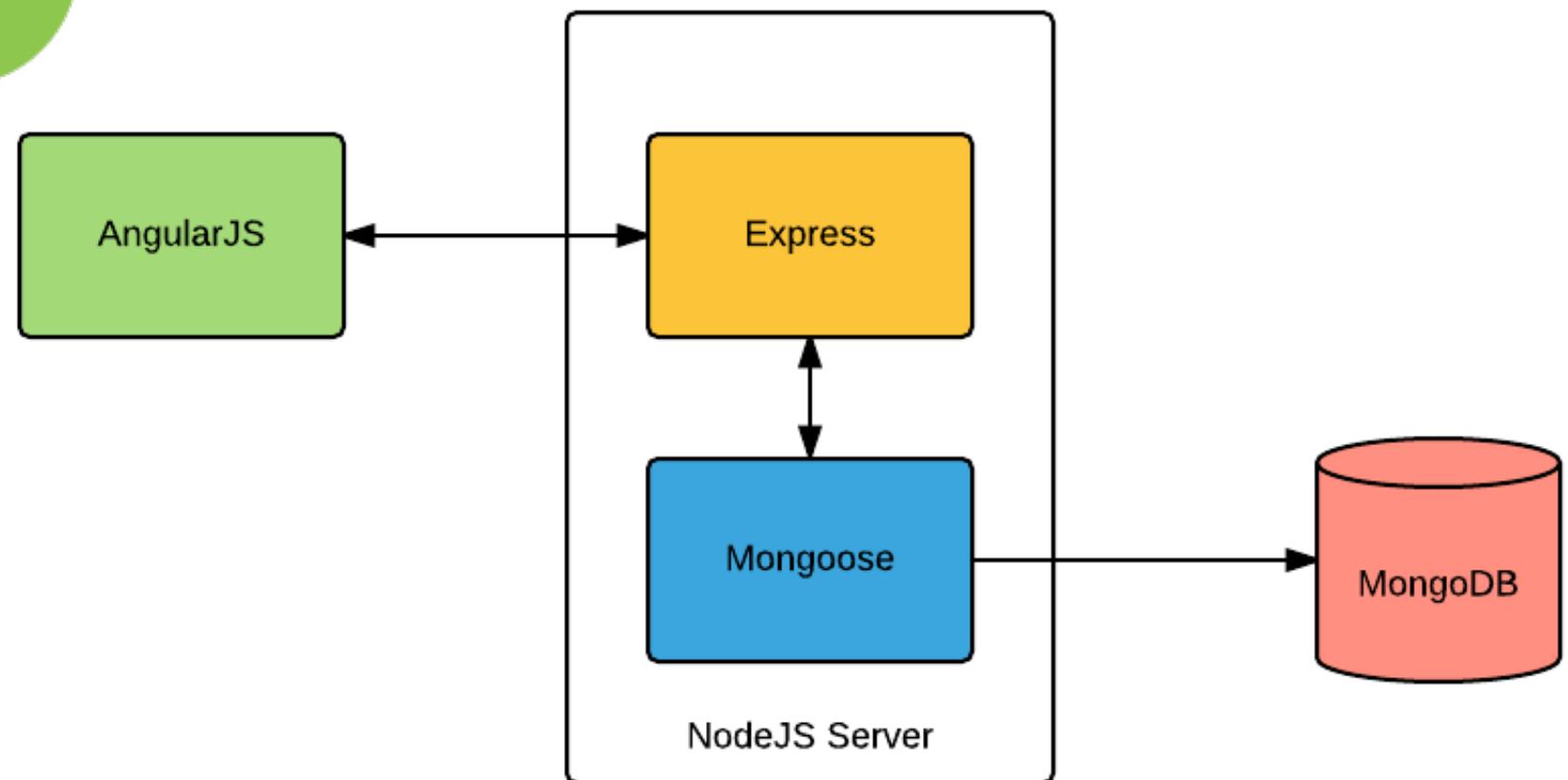
## WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez  
[germanux@gmail.com](mailto:germanux@gmail.com)





# Arquitectura MEAN



# ÍNDICE

- 1) NodeJS
- 2) NPM
- 3) Express
- 4) Mongoose
- 5) BB.DD. Relacionales VS No-SQL
- 6) Arquitectura basada en componentes
- 7) WebComponents
- 8) Angular

# NodeJS



# NodeJS

## ¿Qué es NodeJS?

- Entorno en tiempo de ejecución multiplataforma de código abierto para la capa del servidor (pero no limitándose a ello)
- Basado en el lenguaje de programación ECMAScript
- Asíncrono
- Con I/O de datos en una arquitectura orientada a eventos
- Basado en el **motor V8** de Google.

# NodeJS

## ¿Qué es NodeJS?

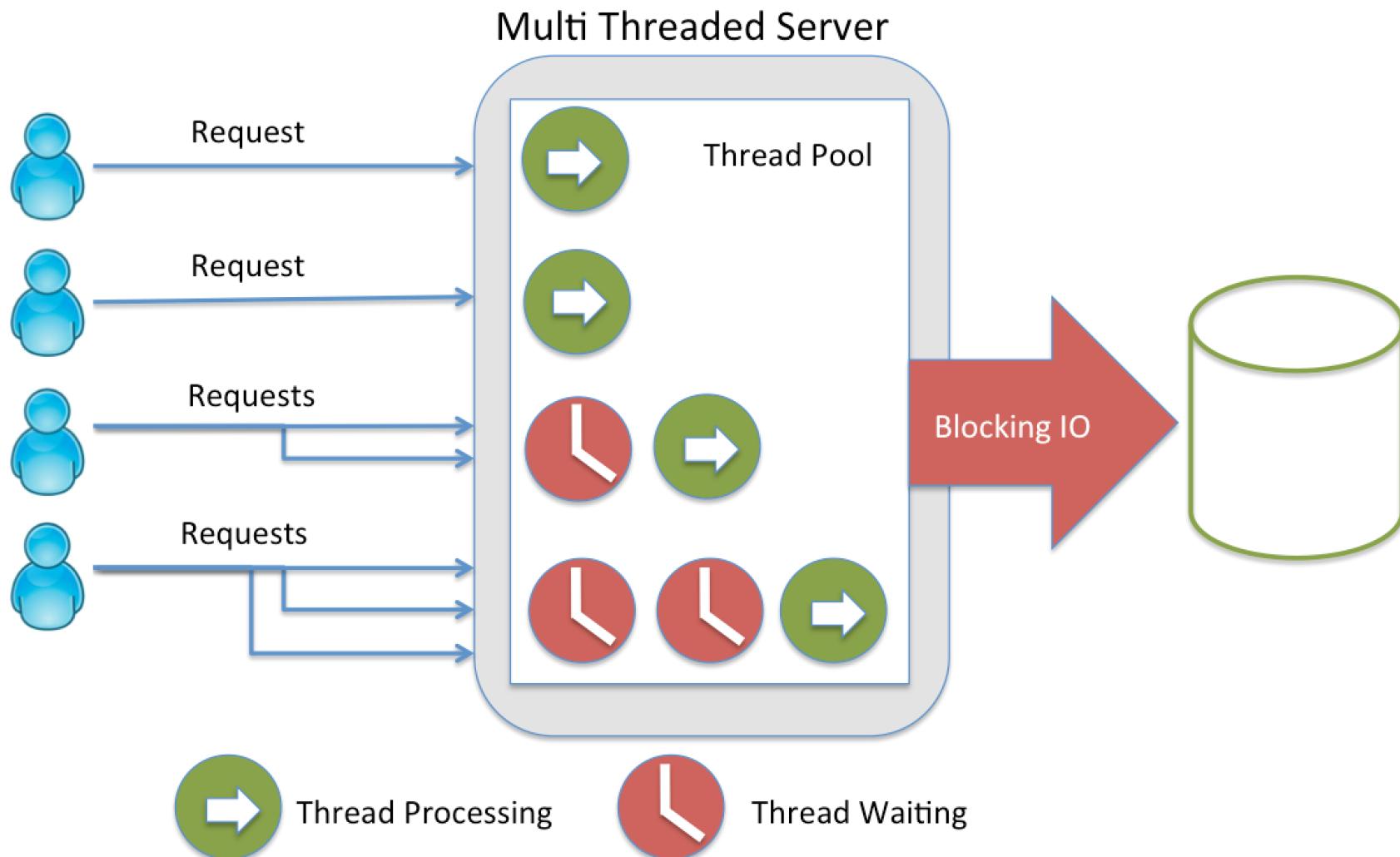
- Con Node.js es posible hacer en el servidor, todo lo que necesitas:
  - Acceso a ficheros
  - A bases de datos
  - Conexiones de clientes...
  - Ejecutar programas JS para:
    - Crear servidores web
    - Crear programas en línea de comandos
    - Etc.

# NodeJS

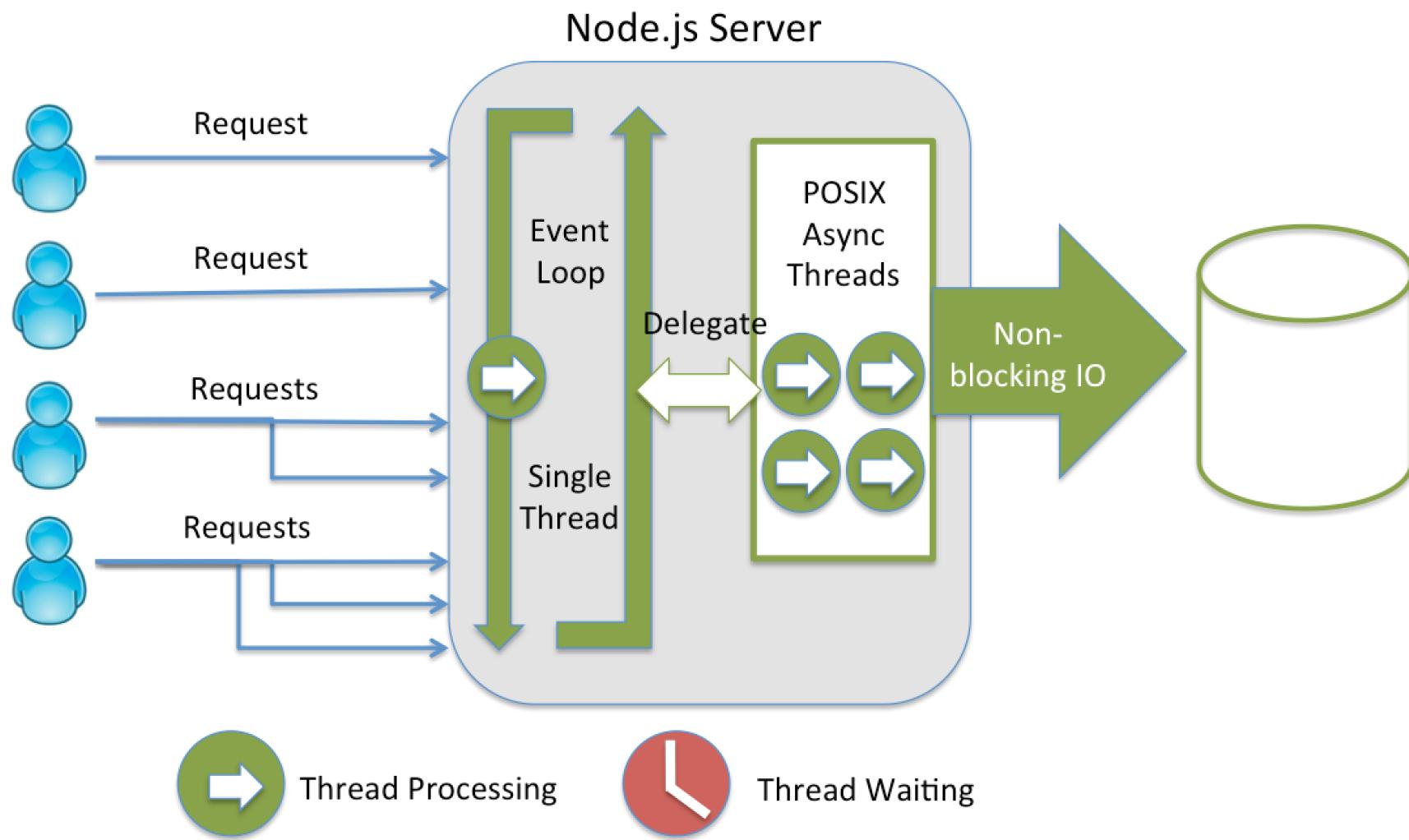
¿Qué es NodeJS?

- Node.js trabaja con un **único hilo de ejecución** que es el encargado de organizar todo el flujo de trabajo que se deba realizar.

# NodeJS



# NodeJS



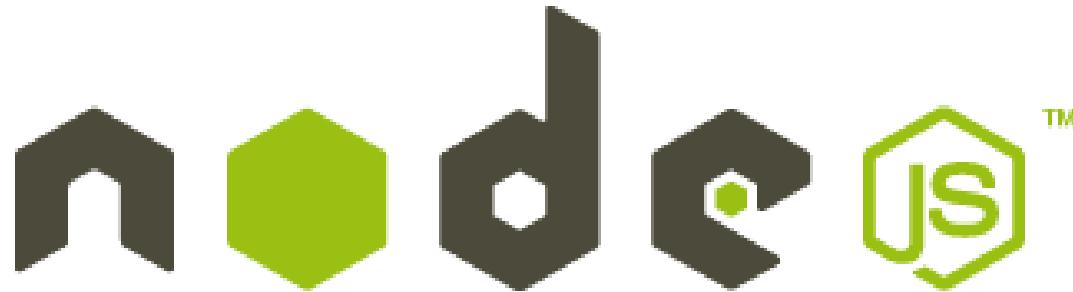
# NodeJS

```
cmd Select Node.js command prompt
C:\Users\javatpoint1>npm install pug --save
pug@2.0.0-alpha8 node_modules\pug
├── pug-runtime@2.0.1
├── pug-load@2.0.0 <pug-walk@0.0.3>
├── pug-strip-comments@0.0.1 <pug-error@0.0.0>
├── pug-linker@0.0.4 <pug-error@1.3.0, pug-walk@0.0.3>
├── pug-parser@2.0.0 <pug-error@1.3.0, token-stream@0.0.1>
└── pug-lexer@2.0.0 <pug-error@1.3.0, character-parser@2.2.0, is-expression@2.0.0>
    ├── pug-code-gen@0.0.7 <pug-error@1.3.0, doctypes@1.0.0, js-stringify@1.0.2, pug-attrs@2.0.1, void-elements@2.0.1, constantinople@3.0.2, with@5.0.0>
    └── pug-filters@1.2.1 <pug-error@1.3.0, pug-walk@0.0.3, resolve@1.1.7, constantinople@3.0.2, clean-css@3.4.13, jstransformer@0.0.3, uglify-js@2.6.2>
C:\Users\javatpoint1>
```

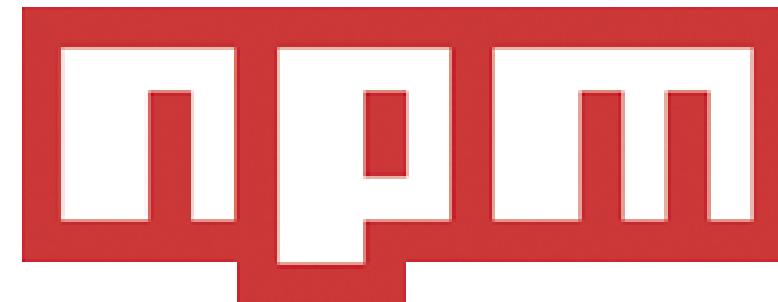
# Ventajas NodeJS

- Permite utilizar el mismo lenguaje (JavaScript), tanto en el cliente como en el servidor.
- Ofrece muy buena gestión de paquetes gracias a NPM (si quieres hacer algo, probablemente exista una librería/paquete que ya lo ofrezca).
- Detrás de Node hay una gran comunidad documentando, haciendo tutoriales y creando nuevos módulos.
- Está basado en eventos, así que toda la filosofía asíncrona que ya utilizas con AJAX en tu cliente, la puedes pasar al servidor.

# NPM



+



npm es el manejador de paquetes por defecto para Node.js

# NPM

- Es instalado automáticamente con el entorno. npm se ejecuta desde la linea de comandos y maneja las dependencias para una aplicación.
- Además, permite a los usuarios instalar aplicaciones Node.js que se encuentran en el repositorio.

# NPM

```
(jessie)ag_dubs@localhost:~/Projects/npm-sandbox/npm3/example3$ npm install
example3@1.0.0 /home/ag_dubs/Projects/npm-sandbox/npm3/example3
+-- mod-a@1.0.0
|   +-- mod-b@1.0.0
+-- mod-c@1.0.0
|   +-- mod-b@2.0.0
+-- mod-d@1.0.0
|   +-- mod-b@2.0.0
+-- mod-e@1.0.0

npm WARN example3@1.0.0 No description
npm WARN example3@1.0.0 No repository field.
(jessie)ag_dubs@localhost:~/Projects/npm-sandbox/npm3/example3$ tree -d node_modules/
node_modules/
├── mod-a
├── mod-b
├── mod-c
│   └── node_modules
│       └── mod-b
└── mod-d
    └── node_modules
        └── mod-b

9 directories
```

# NPM

- npm está escrito enteramente en JavaScript y fue desarrollado por Isaac Z. Schlueter a raíz de la frustración que experimentó mientras trabajando con CommonJS y considerando las deficiencias de otros proyectos similares como PHP (PEAR) y Perl (CPAN)

# Express JS



express 

The background image shows a laptop on a desk with various office supplies like a ruler, pens, and a calculator. A green hexagonal logo for Express.js is overlaid on the laptop screen.

# Express JS

## Aplicaciones web

- Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.
- Con miles de métodos de programa de utilidad HTTP y middleware a su disposición, la creación de una API sólida es rápida y sencilla.
- Rendimiento
- Express proporciona una delgada capa de características de aplicación web básicas, que no ocultan las características de Node.js

# Express JS

The screenshot shows a code editor interface with the following details:

- Title Bar:** The title bar displays "app.js" and "home.html".
- Toolbar:** The toolbar includes buttons for "Code", "Split", "Design", "Title" (with a text input field), and various file operations like "Save", "Open", "Close", and "Print".
- Left Sidebar:** A vertical sidebar on the left contains icons for file operations such as New, Open, Save, Find, Replace, and others.
- Code Editor:** The main area contains the following code:

```
1 var express = require("express");
2 var cons = require("consolidate");
3
4 var app = express();
5 app.engine('html', cons.swig);
6 app.set('view engine', 'html');
7 app.set('views', __dirname + '/views');
8
9 app.get('/', function(req, res) {
10   res.render('home', {'title':'First Swig Template'});
11 });
12 app.get('*', function(req, res) {
13   res.send(404, 'Sorry the page you are requesting does not exist.');
14 });
15
16 var port = Number(process.env.PORT || 3000);
17 app.listen(port, function() {
18   console.log("Listening on " + port + "...");
19 });
```

The code uses ES6 syntax and imports the Express and Consolidate modules. It sets up a Swig engine for HTML rendering and defines routes for the root and all other paths, sending a 404 response for non-existent pages. It also listens on port 3000 or the environment variable PORT if specified.

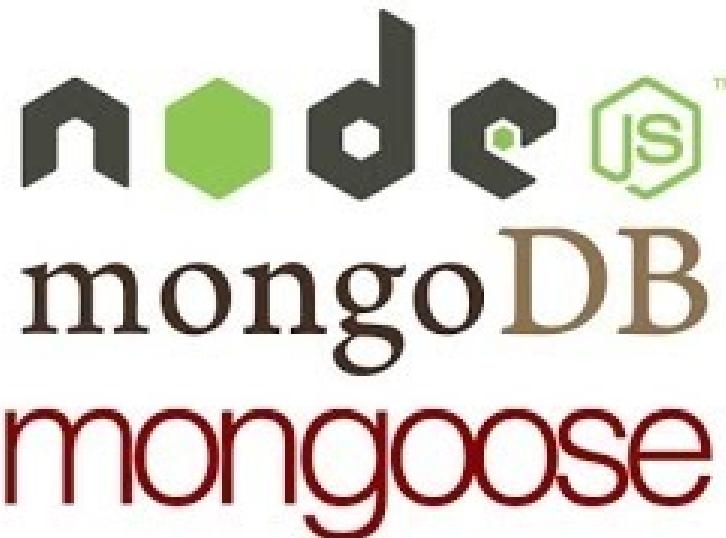
# Express JS

FOLDERS

- ▼ express2
  - ▼ node\_modules
    - .bin
    - express
    - jade
    - public
    - routes
    - ▼ views
      - index.jade
      - layout.jade
  - app.js
  - package.json

```
1  /**
2   * Module dependencies.
3   */
4
5
6  var express = require('express');
7  var routes = require('./routes');
8  var user = require('./routes/user');
9  var http = require('http');
10 var path = require('path');
11
12 var app = express();
13
14 // all environments
15 app.set('port', process.env.PORT || 3000);
16 app.set('views', path.join(__dirname, 'views'));
17 app.set('view engine', 'jade');
18 app.use(express.favicon());
19 app.use(express.logger('dev'));
20 app.use(express.json());
21 app.use(express.urlencoded());
22 app.use(express.methodOverride());
23 app.use(app.router);
24 app.use(express.static(path.join(__dirname, 'public')));
25
26 // development only
27 if ('development' == app.get('env')) {
28   app.use(express.errorHandler());
29 }
30
31 app.get('/', routes.index);
32 app.get('/users', user.list);
33
34 http.createServer(app).listen(app.get('port'), function(){
35   console.log('Express server listening on port ' + app.get('port'));
36 });
37
```

# Mongoose



# Mongoose

- Podemos usar el modulo Mongoose para conectarnos a la base de datos

```
var mongoose = require('mongoose');
var db = mongoose.createConnection( 'mongodb://localhost:27017/prueba' );
```

# Mongoose

- Construye modelos de información, para que todo sea mas asequible, y mas fácil de encontrar

```
var postSchema = mongoose.Schema({  
    titulo: { type : String, trim : true, index : true },  
    post : string,  
    slug : string,  
    autor : { type : Schema.Types.ObjectId, ref : 'autores' }  });  
  
var userSchema = mongoose.Schema({  
    name : { type : String, trim : true },  
    nick : { type : String, trim : true, index : true },  
    email: { type : String, trim : true },  
});
```

# Mongoose

- Como vemos, construimos Schema (Esquemas).
- Estos esquemas, es tan dados en Json (Bastante cercano a MongoDB por que utilizan BSON que es el mismo JSON pero en Binario ).
- Cada propiedad debe tener una definición, al igual se pueden colocar múltiples validaciones de esa propiedad.

# Mongoose

- Todos los valores van a ser validados, si existe algún error nos mostrara.

```
var Post = db.model('posts', postSchema);
var User = db.model('users', userSchema);
```

# Mongoose

- Ya con esto podemos subir información a la base de datos.
- En el momento de subida va a ser validada.

```
var PrimerUsuario = new User({
  name : 'Pepito Perez',
  nick : 'pepito',
  email: 'pepito@pepito.com'
});
PrimerUsuario.save(function(err, doc) {
  if(err)
    console.log(err);
  var PrimerPost = new Post({
    titulo: 'Este es mi primer Post!',
    post : 'Publicando mi primer post!! que felicidad!!',
    autor : doc._id,
  });
  PrimerPost.slug = slug( PrimerPost.titulo );
  PrimerPost.save( function(err, doc) {
    console.log(err);
    console.log(doc);
  })
});
```

# Mongoose

```
{  
  _id : 50903550a04313310c000001,  
  titulo : 'Este es mi primer Post!',  
  post : 'Publicando mi primer post!! que felicidad!!!',  
  slug : 'este-es-mi-primer-post',  
  autor : 2d2ac97cf59cee65f7a38e596c,  
}
```

# BB.DD. Relacionales VS No-SQL

1) **Not Only SQL**, las bases de datos NoSQL no utilizan el modelo relacional.

2) **Consistencia Eventual:**

1) No se implementan mecanismos rígidos de consistencia como los presentes en las bases de datos relacionales, donde la confirmación de un cambio implica una comunicación del mismo a todos los nodos que lo repliquen.

2) Esta flexibilidad hace que la consistencia se dé, eventualmente, cuando no se hayan modificado los datos durante un periodo de tiempo.

3) Esto se conoce también como BASE (Basically Available Soft-state Eventual Consistency, o coherencia eventual flexible básicamente disponible), en contraposición a ACID, su analogía en las bases de datos relacionales.

# BB.DD. Relacionales VS No-SQL

## Ventajas Base de Datos relacional

- 1) Está más adaptado su uso y los perfiles que los conocen son mayoritarios y más baratos.
- 2) Debido al largo tiempo que llevan en el mercado, estas herramientas tienen un mayor soporte y mejores suites de productos y add-ons para gestionar estas bases de datos.
- 3) La atomicidad de las operaciones en la base de datos. Esto es, que en estas bases de datos o se hace la operación entera o no se hace utilizando la famosa técnica del rollback.
- 4) Los datos deben cumplir requisitos de integridad tanto en tipo de dato como en compatibilidad.

# BB.DD. Relacionales VS No-SQL

## Desventajas Bases de Datos relacionales

- 1) La atomicidad de las operaciones juegan un papel crucial en el rendimiento de las bases de datos.
- 2) Escalabilidad, que aunque probada en muchos entornos productivos suele, por norma, ser inferior a las bases de datos NoSQL.

# BB.DD. Relacionales VS No-SQL

## Ventajas de una base de datos NoSQL

- 1) La escalabilidad y su carácter descentralizado.
- 2) Soportan estructuras distribuidas.
- 3) Suelen ser bases de datos mucho más abiertas y flexibles.
- 4) Permiten **adaptarse a necesidades** de proyectos mucho más fácilmente que los modelos de Entidad Relación.
- 5) Se pueden hacer cambios de los esquemas sin tener que parar bases de datos.
- 6) **Escalabilidad horizontal:** son capaces de crecer en número de máquinas, en lugar de tener que residir en grandes máquinas.
- 7) Se pueden ejecutar en máquinas con pocos recursos.
- 8) Optimización de consultas en base de datos para grandes cantidades de datos.

# BB.DD. Relacionales VS No-SQL

## Desventajas de una base de datos NoSQL

- 1) No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos:
  - 1) Soportan lo que se llama consistencia eventual.
- 2) Problemas de compatibilidad entre instrucciones SQL:
  - 1) Las nuevas bases de datos utilizan sus propias características en el lenguaje de consulta y no son 100% compatibles con el SQL de las bases de datos relacionales.
  - 2) El soporte a problemas con las queries de trabajo en una base de datos NoSQL es más complicado.
- 3) **Falta de estandarización.** Hay muchas bases de datos NoSQL y aún no hay un estándar como si lo hay en las bases de datos relacionales. Se presume un futuro incierto en estas bases de datos.
- 4) **Soporte multiplataforma.** Aún quedan muchas mejoras en algunos sistemas para que soporten sistemas operativos que no sean Linux.
- 5) Suelen tener herramientas de administración no muy usables o se accede por consola.

# Arquitectura basada en componentes

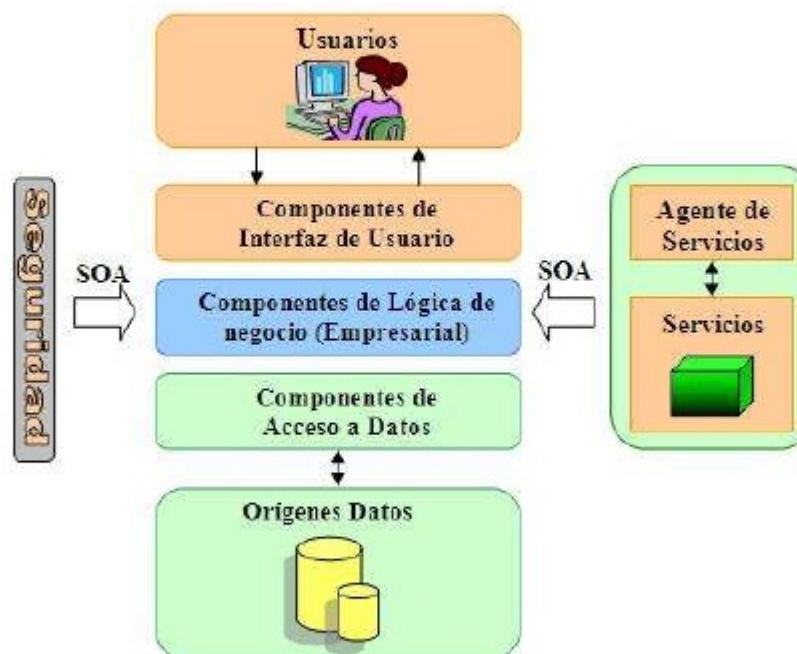


# Arquitectura basada en componentes

- Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema.
- Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas.

# Arquitectura basada en componentes

## TIPOS DE COMPONENTES



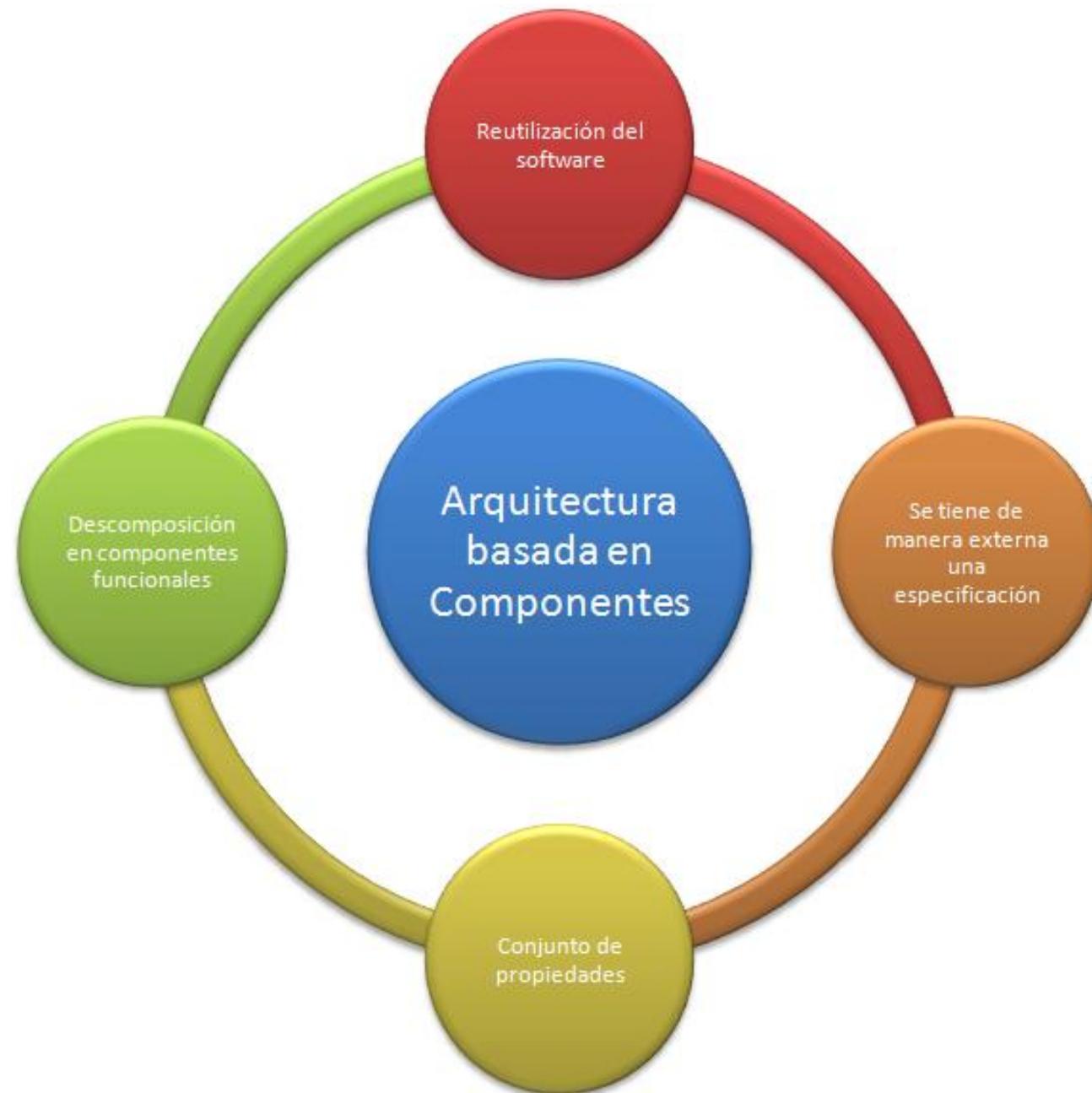
# Arquitectura basada en componentes

- Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado.

# Arquitectura basada en componentes

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.

# Arquitectura basada en componentes



# Arquitectura basada en componentes

## Principios Fundamentales

- Un componente es un objeto de software específicamente diseñado para cumplir con cierto propósito.
- Los principios fundamentales cuando se diseña un componente es que estos deben ser:

# Arquitectura basada en componentes

## Principios Fundamentales

- **Reusable**

- Los componentes son usualmente diseñados para ser utilizados en escenarios diferentes por diferentes aplicaciones, sin embargo, algunos componentes pueden ser diseñados para tareas específicas.

# Arquitectura basada en componentes

## Principios Fundamentales

- **Sin contexto específico**

- Los componentes son diseñados para operar en diferentes ambientes y contextos.
- Información específica como el estado de los datos deben ser pasadas al componente en vez de incluirlos o permitir al componente acceder a ellos.

# Arquitectura basada en componentes

## Principios Fundamentales

- **Extensible**
  - Un componente puede ser extendido desde un componente existente para crear un nuevo comportamiento.

# Arquitectura basada en componentes

## Principios Fundamentales

- **Encapsulado**

- Los componentes exponen interfaces que permiten al programa usar su funcionalidad.
- Sin revelar detalles internos, detalles del proceso o estado.

# Arquitectura basada en componentes

## Principios Fundamentales

- **Independiente**

- Los Componentes están diseñados para tener una dependencia mínima de otros componentes.
- Por lo tanto los componentes pueden ser instalados en el ambiente adecuado sin afectar otros componentes o sistemas.

# Arquitectura basada en componentes

## Beneficios

- **Facilidad de Instalación**

- Cuando una nueva versión esté disponible, usted podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.

# Arquitectura basada en componentes

## Beneficios

- **Costos reducidos**

- El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.

# Arquitectura basada en componentes

## Beneficios

- **Reusable**
  - El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.

# Arquitectura basada en componentes

## Beneficios

- **Facilidad de desarrollo**

- Los componentes implementan un interface bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.

# Arquitectura basada en componentes

## Beneficios

- **Mitigación de complejidad técnica**
  - Los componentes mitigan la complejidad por medio del uso de contenedores de componentes y sus servicios.
  - Ejemplos de servicios de componentes incluyen activación de componentes, gestión de la vida de los componentes, gestión de colas de mensajes para métodos del componente y transacciones.

# Arquitectura basada en componentes

## Ejemplos

- Componentes de interfaz de usuario, como grillas, botones, etc., generalmente conocidos como “controles”.
- Componentes de ayuda que exponen un conjunto específico de funciones usados por otros componentes.

# Arquitectura basada en componentes

## Ejemplos

- Componentes que se no se usan con mucha frecuencia o son intensivos en recursos y deben ser actividades usando una aproximación de solo en el momento justo (Just in Time (JIT)).
  - Estos son comunes en escenarios de componentes distribuidos o en componentes remotos.

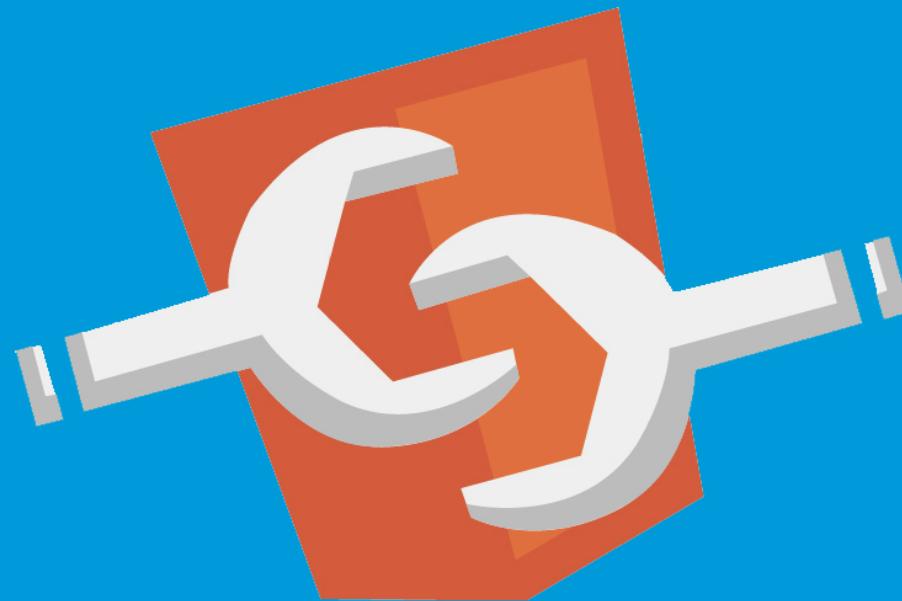
# Arquitectura basada en componentes

## Ejemplos

- Componentes encolados, aquellos cuyos métodos pueden ser ejecutados de forma asíncrona usando colas de mensajes del tipo almacenamiento, entrega.

<http://geeks.ms/jkpelaez/2009/04/18/arquitectura-basada-en-componentes/>

# WebComponents



[https://bbvaopen4u.com/es/actualidad/web-components  
-presente-y-futuro-en-el-desarrollo-web](https://bbvaopen4u.com/es/actualidad/web-components-presente-y-futuro-en-el-desarrollo-web)

# WebComponents

## Origen

- Los web components surgieron como propuesta por parte de Google a la W3C, prácticamente a la par que el framework MVC de JavaScript AngularJS, también creado por la compañía del buscador

# WebComponents

## Origen

- AngularJS nos proporciona controladores, enrulado, filtros, servicios, etc, y las directivas estaban ahí, aunque en un principio no eran tan utilizadas.
- Con el tiempo empezaron a extenderse debido a terceros desarrolladores que compartían, a modo de proyectos Open Source, sus implementaciones.

# WebComponents

## Origen

- De esta manera, en nuestras aplicaciones podíamos importar estos proyectos y utilizarlos en nuestros desarrollos.
- Paralelamente, la W3C trabajaba en el concepto de los web components que bebían mucho de este concepto iniciado en AngularJS.

# WebComponents

## Elementos de un Web Component



### WEB COMPONENTS

TEMPLATES

```
<template id="">  
/</template>
```

SHADOW  
DOM

```
div  
#document-fragment  
span
```

HTML  
IMPORTS

```
<link rel="import"  
href="part.html">
```

CUSTOM  
ELEMENTS

```
<my-elem>  
</my-elem>
```

# WebComponents

## Templates

- Los Templates son un elemento de HTML5 que nos permiten crear plantillas para presentar los datos, y son nativos del navegador.

```
<template id="profileTemplate">
```

```
  <div class="profile">
```

```
    <img src="" class="profile_img">
```

```
    <div class="profile_name"></div>
```

```
    <div class="profile_social"></div>
```

```
  </div>
```

```
</template>
```

# WebComponents

## Templates

- Esto representaría el marcado para un perfil de usuario en una aplicación web.
- Nada del código englobado dentro de <template> será visible en el navegador hasta que sea activado, aunque esté dentro del código.

# WebComponents

## Templates

- Para poder activarlo e injectarle datos, debemos hacerlo a través de JavaScript con un código similar al siguiente:

```
var template = document.querySelector('#profileTemplate');

template.querySelector('.profile_img').src = avatar.jpg';

template.querySelector('.profile_name').textContent = Mi Nombre';

template.querySelector('.profile_social').textContent = Sígueme en Twitter';

document.body.appendChild(template);
```

# WebComponents

## Custom Elements

- Los Custom Elements o elementos personalizados nos permiten definir nuestro propio elemento HTML (similar a las directivas de AngularJS).
- Según el estándar, la forma correcta de crear un Custom Element es con la función Object.create de JavaScript y el elemento HTMLElement.prototype.

# WebComponents

## Custom Elements

```
<template id="profileTemplate">
  <div class="profile">
    <img src="" class="profile_img">
    <div class="profile_name"></div>
    <div class="profile_social"></div>
  </div>
</template>
<script>
```

```
  <script>
    var MyElementProto = Object.create(HTMLElement.prototype);
    window.MyElement = document.registerElement('user-profile', {
      prototype: MyElementProto
      // other props
    });
  </script>
```



# WebComponents

## Shadow DOM

- El Shadow DOM es el DOM que encapsula nuestro componente.
- No pertenece al DOM original del documento o página web, si no que se encuentra dentro del componente que hemos creado.

# WebComponents

## Shadow DOM

- Tiene su propio ámbito de ejecución y su propio estilo CSS.
- Piensa en ello como un DOM anidado al DOM primario de la página.

# WebComponents

## Shadow DOM

- Una manera de ver el shadow DOM es en el inspector web del navegador Chrome, clicando en nuestro elemento recién creado, y así podremos ver el DOM que encapsula el componente:

# WebComponents

## Shadow DOM

▼ *<user-profile>*

  ▼ *#shadow-root (user-agent)*

```
<div class="profile">
```

```
  <img src="" class="profile_img">
```

```
  <div class="profile_name"></div>
```

```
  <div class="profile_social"></div>
```

```
</div>
```

```
</user-profile>
```

# WebComponents

## HTML Imports

- Al igual que en CSS podemos importar archivos externos con @import y en JavaScript usando 'require' (utilizando Node.js o Browserify) o 'import' {module} from 'libreria.js', con HTML Imports podemos importar ficheros HTML que contengan los web components.

# WebComponents

## Compatibilidad en los diferentes navegadores

- Como todo estándar web, cada navegador se toma su tiempo en implementarlo y que esté 100% utilizable.
- Uno de los mejores sitios para comprobar si cierto elemento, propiedad CSS o API JavaScript, está disponible en los navegadores es la web caniuse.com

# WebComponents

## Compatibilidad en los diferentes navegadores

- En concreto para el tema de web components, tenemos una adaptada a los elementos principales del estándar, y es:
  - <http://jonrimmer.github.io/are-we-componentized-yet/>

# WebComponents

## Compatibilidad en los diferentes navegadores

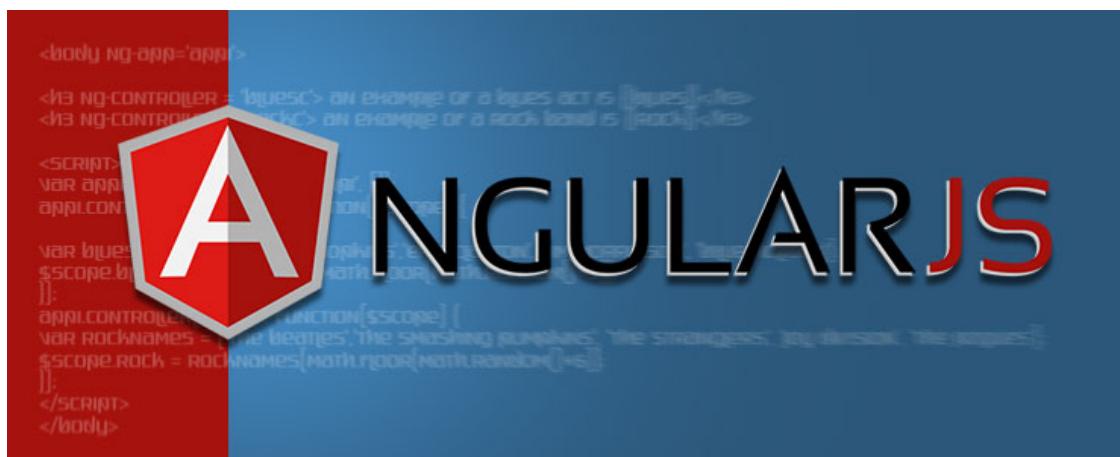
	Specged	Implementation				
		Polyfill	Chrome / Opera	Firefox	Safari	Edge
Templates			Stable	Stable	8	13
HTML Imports			Stable	On Hold	No Active Development	Vote
Custom Elements			Stable	Flag	Prototype	Vote
Shadow DOM			Stable	Flag	WebKit Nightly	Vote

# WebComponents

## **La Web orientada a Componentes**

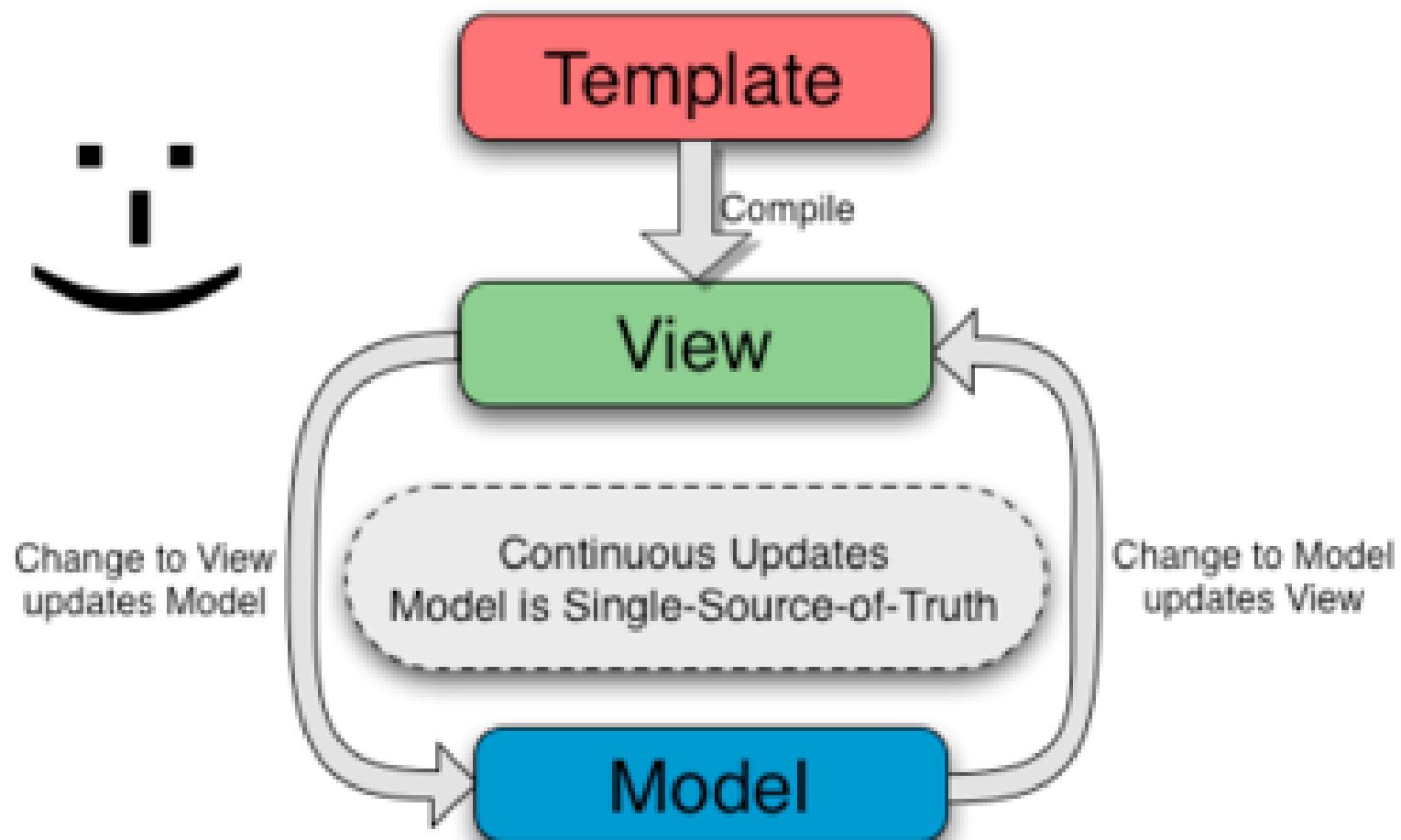
- En el pasado año vimos como el desarrollo de aplicaciones web se estaba “componetizando”.
- Con el impulso de React, Polymer y Angular, esta nueva forma de desarrollar se está popularizando.

# Angular

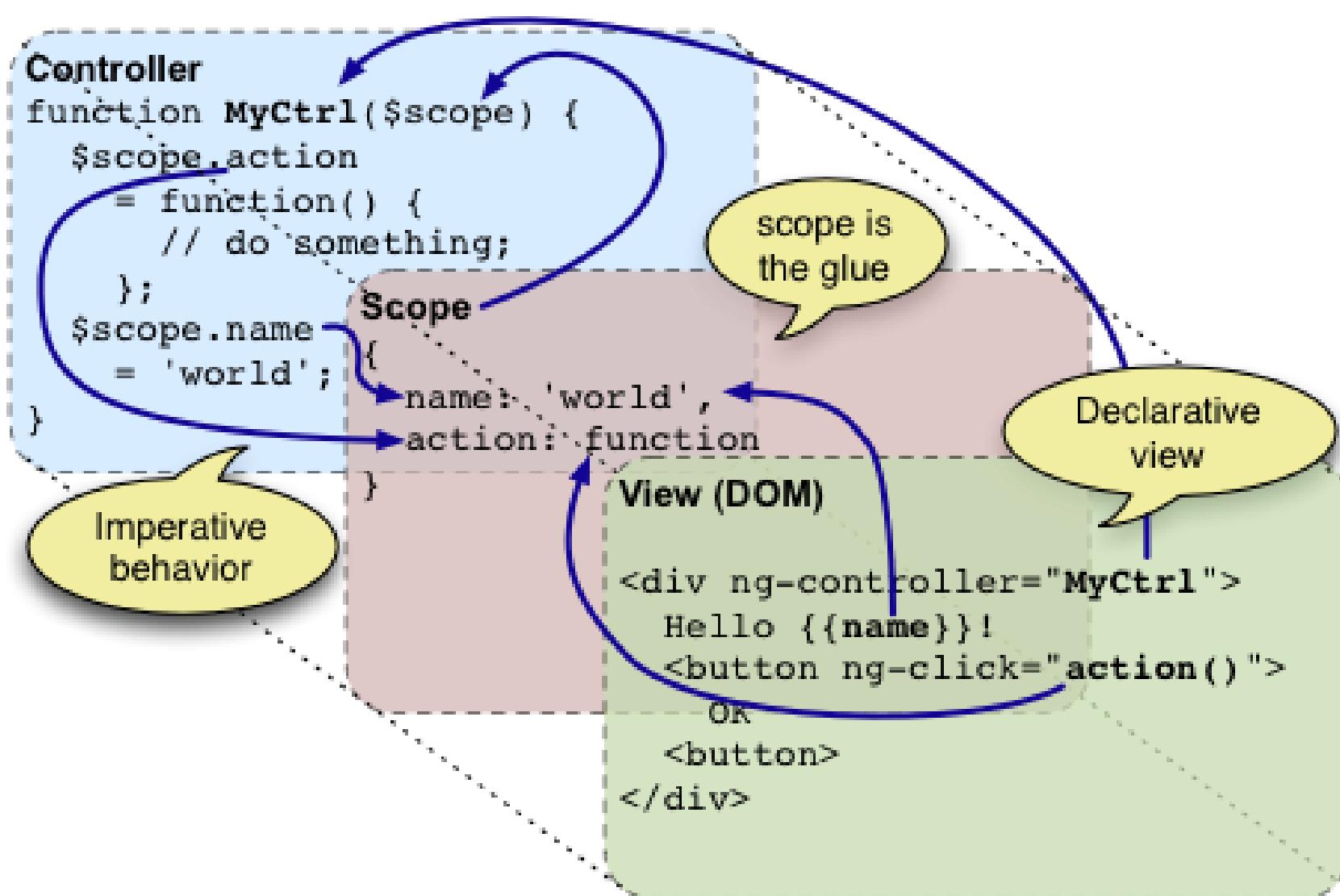


# Angular

## Two-Way Data Binding



# Angular



# Angular

Data Binding

MVC

Routing

Testing

jqLite

Templates

History

Factories



ngularJS is a full-featured  
SPA framework

ViewModel

Controllers

Views

Directives

Services

Dependency Injection

Validation

# Angular

