BrainStationAP
Brainstation123


/BrainStationCR


/BrainStationCR

Angular 2: de framework a plataforma

Carlos Vega Rodriguez

Agenda

- Tooling (configurando TS, Webpack et al)
- Fundamentos de ES6/TypeScript
- Arquitectura basada en componentes
- Streams (RxJS)
  - Programación funcional reactiva
  - Flujo de datos unidireccional
  - Detección de cambios
- Inyección de dependencias
- Componentes y Directivas
- Sintaxis en las plantillas
- Formularios
- Navegación y rutas
- Buenas prácticas: estructura y estilo

Herramientas y entorno de desarrollo

Fundamentos de ES6 con TypeScript

Función de flecha (Arrow functions)

La **expresión de función flecha** (también conocida como **función flecha gruesa**) dispone de una sintaxis más corta comparada con la expresión de función convencional y vincula contextualmente el valor de this. Las Funciones Flecha siempre son anónimas.

(https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Arrow_functions)

Clases (Classes)

Las clases de JavaScript fueron introducidas en ECMAScript 6 y son azúcar sintáctica sobre la existente herencia basada en prototipos de JavaScript. La sintaxis de las clases **no** introduce un nuevo modelo de herencia orientada a objetos dentro del lenguaje.

(https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Classes)

Plantillas de cadena de texto (template strings)

Las plantillas de cadena de texto (template strings) son literales de texto que habilitan el uso de expresiones incrustadas. Es posible utilizar cadenas de texto de más de una línea, y funcionalidades de interpolación de cadenas de texto con ellas.

(https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/template_strings)

Parametros por defecto (Default Arguments)

Los **parámetros por defecto de una función** permiten que los parámetros formales de la función sean inicializados con valores por defecto si no se pasan valores o los valores pasados son undefined.

(https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Parametros_por_defecto)

Asignación por destructuring

La sintaxis de **destructuring assignment** es una expresión de JavaScript que hace posible la extracción de datos de arreglos u objetos usando una sintaxis que equivale a la construcción de arreglos y objetos literales.

(https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Destructuring_assignment)

TypeScript

Una de las quejas más comunes dentro de los detractores de JavaScript es la carencia de un tipado fuerte. Esto significa que es posible asignar valores de tipos inesperados a variables que se presuponen de un tipo diferente.

Dicho de otra forma, no es posible controlar ni detectar errores hasta que suceden en tiempo de ejecución.

TypeScript (cont).

TypeScript es un superset de JavaScript que permite agregar tipado fuerte y otros beneficios al lenguaje.

En general, TypeScript hace la lectura del código mucho más sencilla y permite organizar los módulos de una mejor forma.

Además, TypeScript puede ser implementado de manera incremental ya que el resultado de su «compilación» (o *transpilation*) es JavaScript puro.

TypeScript (cont).

Otro beneficio de utilizar TypeScript es que le permite (a los IDEs y editores que lo soporten) tener IntelliSense, auto compleción y refactoring básico de código.

Decoradores (Decorators)

Los decoradores hacen posible la anotación y modificación de clases y propiedades en tiempo de diseño.

(https://github.com/wycats/javascript-decorators)

Interfaces

Las interfaces están relacionadas directamente con TypeScript y NO forman parte del estandar. Se utilizan, primordialmente, para hacer el chequeo de tipo más sencillo y establecer contratos a los que los objetos y el código externo pueden adherirse.

(https://www.typescriptlang.org/docs/handbook/interfaces.html)

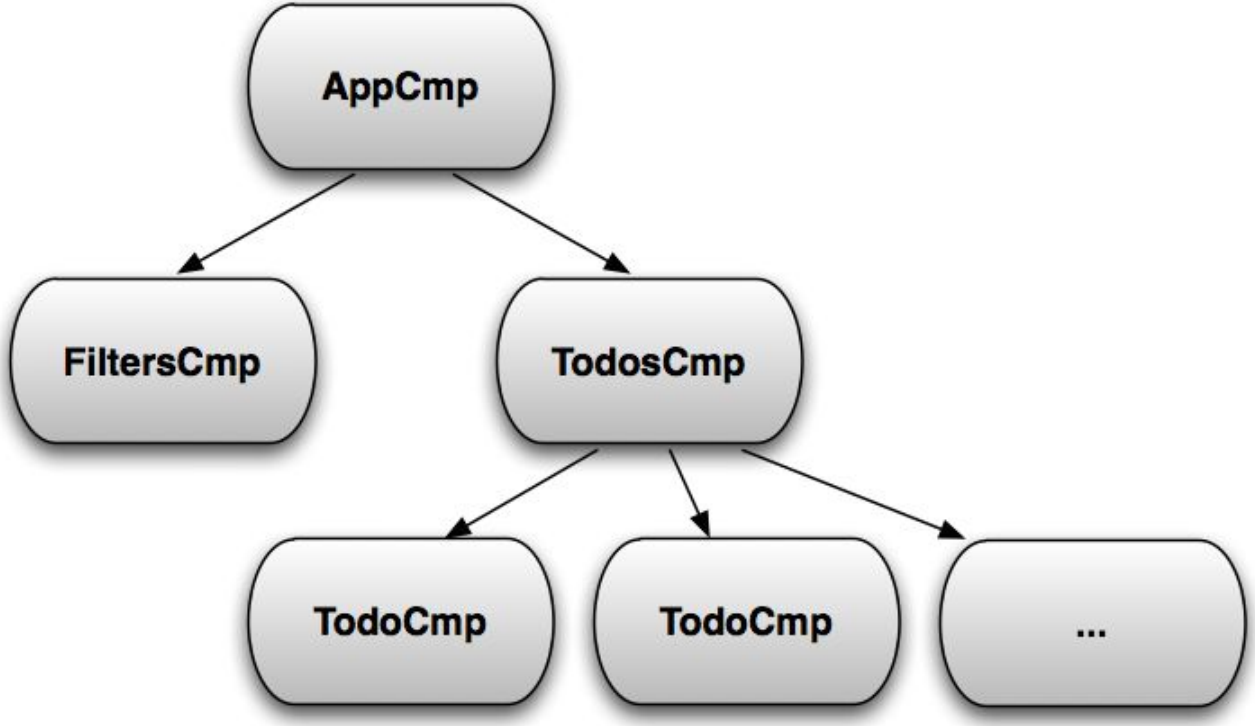
Actividad 1

https://github.com/angular-school/ts-workshop

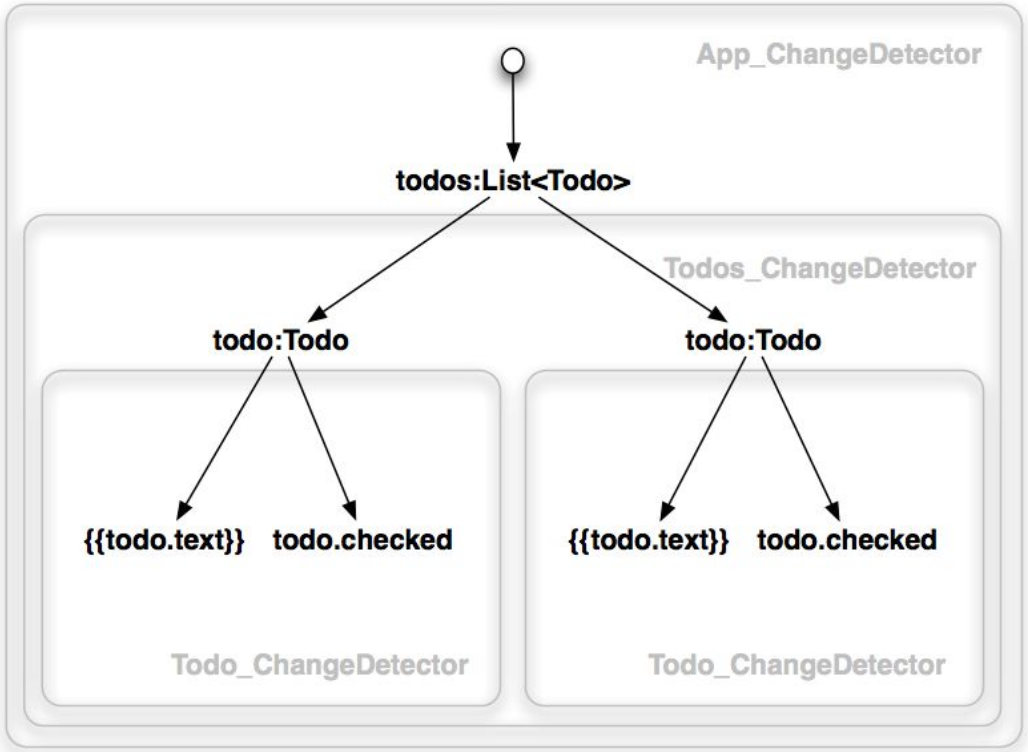
Arquitectura basada en componentes

Una aplicación de Angular 2 no es más que un árbol de componentes cuyos nodos son parte de un sistema reactivo.

Esto permite que la detección del cambio se maneje de una forma mucho más eficiente.



Árbol de componentes



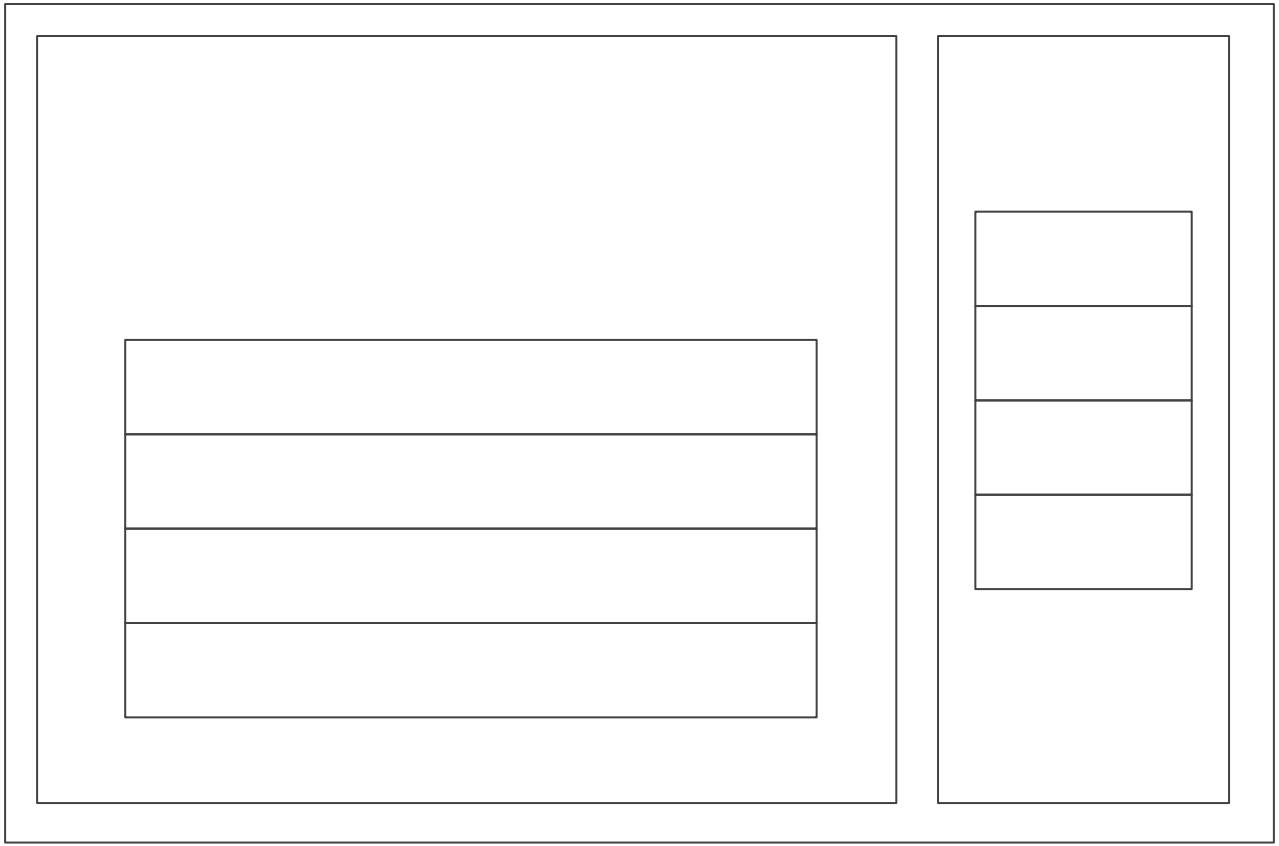
Sistema de detección de cambios

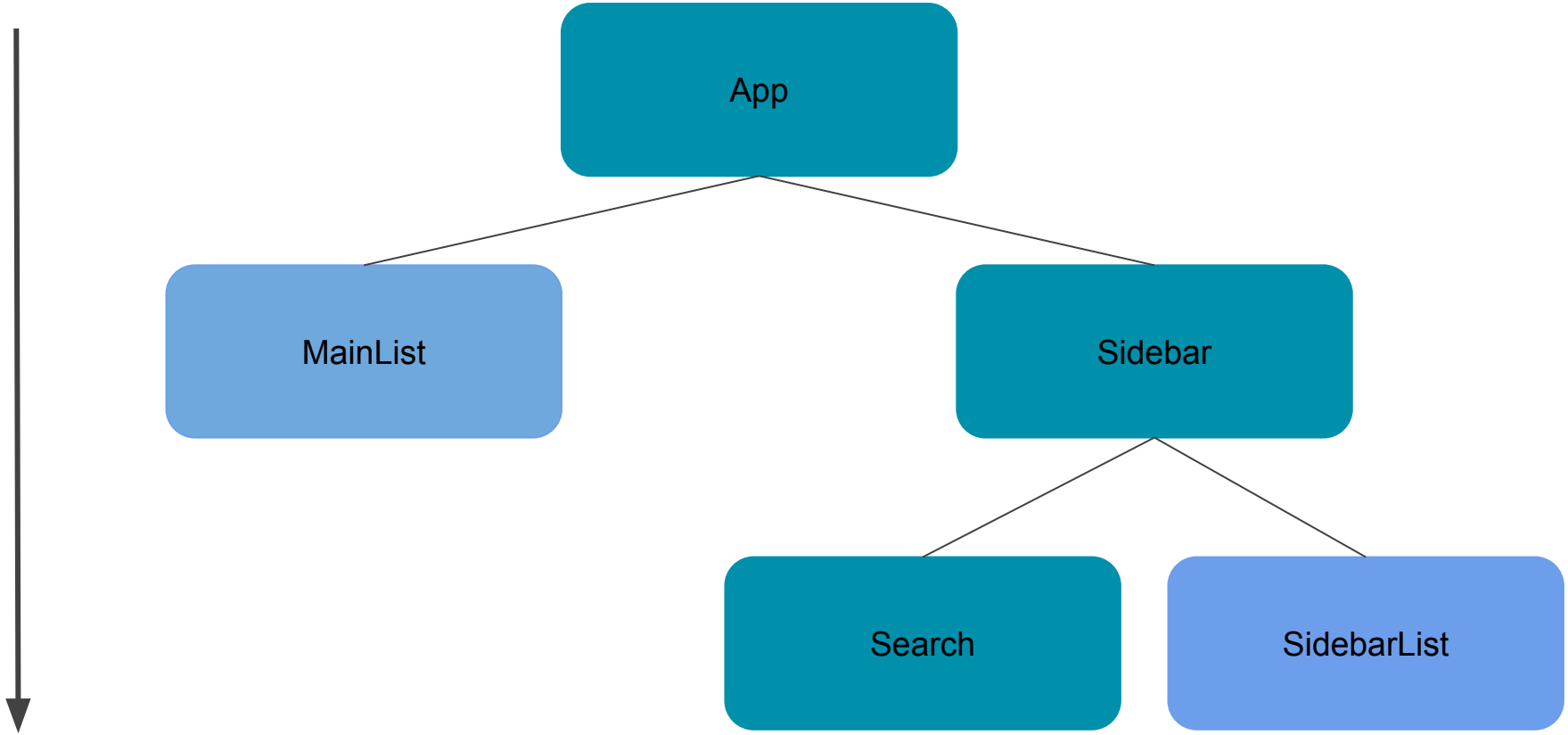
Un componente recibe entradas, y las procesa para producir un resultado.

Tiene **límites** definidos.

Esto presenta dos problemas básicos:

- ¿Cómo compartir funcionalidad?
- ¿Cómo darles a los componentes el contexto necesario?




App
MainList
Sidebar
Search
SidebarList

Necesitamos

- Bloques para construir UI (que puedan ser reutilizados)
- Mantener una separación clara entre la lógica de negocio y estos bloques.

Necesitamos dos tipos de componentes.

Componente «tonto»

- No tiene dependencias con el resto de la aplicación.
- Recibe los datos y el contexto de otro componente.
- Tiene CSS asociado.
- Muy pocas veces contiene su propio estado.
- Puede utilizar o contener otros componentes tontos.
- Se pueden utilizar para hacer layout.

Componente «inteligente»

- Encapsula un conjunto de otros componentes (tontos e inteligentes).
- Mantienen el estado propio y el de sus hijos.
- Proveen contexto a sus hijos y ejecutan acciones.
- Muy pocas veces tienen estilos o html complejo.

Pero, ¿cómo los conectamos?

Streams

- Programación funcional reactiva (FRP)
- Flujo de datos unidireccional
- Detección de cambios

Programación funcional reactiva

Es posible modelar cualquier entrada/salida que suceda en un navegador utilizando un stream (o un stream de streams).

- Eventos
- Variables
- Estructuras de datos

El concepto de stream parece ser ideal para modelar nuestro problema.

¿Es posible construir aplicaciones únicamente utilizando streams?

La forma mas sencilla de describir un stream es compararlo con un arreglo (Array) asincrónico.

```
const arr = [1, 2, 3, 4]
```

Pese a esto, los streams por sí mismos no proveen una manera de crearlos, operar sobre y suscribirse a ellos. Es necesaria una capa extra de abstracción.

Observables

Lo primero que debemos entender es que **un observable no es un stream**. De hecho, la relación stream/observable es mucho mas sencilla de comprender si se piensa en el Observable con un API para interactuar con y operar sobre un stream.

https://jsbin.com/himarad/edit?js,console
https://jsbin.com/pilobuq/edit?js,console

RxJS

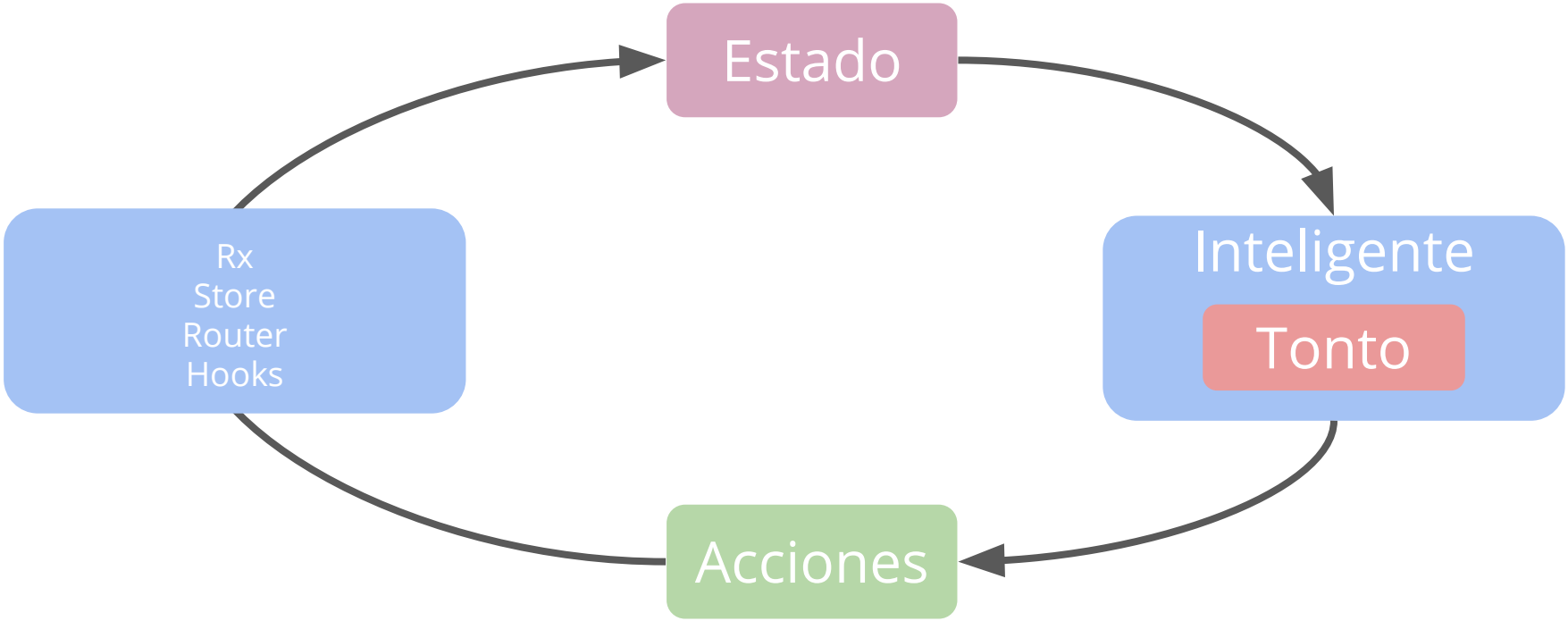
RxJS es una librería que permite crear programas basados en flujos de eventos observables.

La librería permite hacer uso de los flujos (streams) de eventos de una manera similar a como se opera sobre colecciones de objetos (arrays y similares).

Actividad 2

http://reactivex.io/learnrx/

Flujo de datos unidireccional


Estado
Inteligente
Tonto
Acciones
Rx
Store
Router
Hooks

- **Estado:** mantiene el estado de la aplicación, ejecuta las notificaciones de cambio.
- **Componente inteligente:** provee contexto mientras sirve como puente entre la lógica de negocio y las vistas
- **Componente tonto:** UI, reutilizable, entradas/salidas
- **Acciones:** se encarga de ejecutar cambios en el estado y de los efectos secundarios.

La anatomía de un store

---

https://plnkr.co/edit/heYVpj9q2RDe5YbqTaY9?p=info

Detección de cambios

Los cambios pueden suceder a raíz de:

- Eventos
- Timers
- Requests

Detección de cambios

Como hemos dicho antes, una aplicación de Angular 2 es, en realidad, un sistema reactivo compuesto por cada uno de los nodos (componentes) del árbol. Este sistema propaga los bindings desde la raíz del árbol hacia las hojas lo que lo convierte en un grafo dirigido y permite deshacerse del ciclo de digestión y los molestos límites recursivos.

Detección de cambios (cont).

Por defecto, este algoritmo recorre todo el árbol (pero lo hace de una manera muy eficiente que ha probado, en varios benchmarks, ser más rápido que sus competidores actuales) pero si se hace uso de objetos inmutables u observables puede mejorarse aún más.

---

https://plnkr.co/edit/3Au7w4Eu5ztLGHV65wUM

Inyección de dependencias

https://plnkr.co/edit/euudOi6TPA6zyrv6Vl36?p=preview

Componentes y Directivas

Existen tres tipos de directivas en Angular 2:

- Estructurales
- De atributo
- Componentes

Componentes

Los componentes en Angular 2 son directivas que poseen una plantilla.

Para crearlos basta con anotar una clase ES6 con el decorador @Component (o, en su defecto, utilizar las funciones que angular provee para trabajar con ES5).

Directivas Estructurales

Cambian el DOM removiendo/agregando elementos. En Angular 2, el módulo Common contiene algunas de ellas como lo son: NgFor, NgIf y NgSwitch.

Actividad 3

Visite la seccion “Make a structural directive” y cree una directiva estructural siguiendo las instrucciones.

https://angular.io/docs/ts/latest/guide/structural-directives.html

Directivas de atributo

Cambian la apariencia o el comportamiento de un elemento en el DOM pero no agregan o remueven elementos adicionales.

NgClass y NgStyle son ejemplos de directivas de este tipo incluidas en el módulo Common de Angular.

Actividad 4

Visite la seccion “Build a simple attribute directive” y cree una directiva de atributo siguiendo las instrucciones.

https://angular.io/docs/ts/latest/guide/attribute-directives.html

Plantillas y Formularios

https://plnkr.co/edit/aKWTSGw8rb77HmYt5FUj

Servicios

https://plnkr.co/edit/72Zm2PwxAeAXNtrsTnPb?p=preview

Actividad 5

Utilizando los ejemplos vistos anteriormente desarrolle un clon basico de Trello (trello.com). Por el momento, cree los componentes y servicios. La navegación y rutas pueden esperar.

NgModule, navegación y rutas

Actividad 6

Utilizando el ejemplo y los conocimientos adquiridos sobre navegación y rutas, separe los componentes en módulos y agregue el enrutador.

Buenas practicas

- Codelyzer
- TSLint
- Guía de estilo

En resumen

- Detección de cambios más eficiente.
- Desarrollo de aplicaciones más sencillo.
- No limita las aplicaciones al DOM.
- Permite componer interfaces declarativas por medio de bloques.
- Es flexible en el uso de ES5, ES6, TypeScript.
- Permite ejecutar código de angular 1 y 2 al mismo tiempo (lo que hace la migración más sencilla)

Carlos Vega

carlosve.ga

clmvega@gmail.com

- /__el_Negro (Twitter)
- /alterx (GitHub)
- /@_carlosvega (Medium)

Referencias

---

https://angular.io/docs/ts/latest/

http://victorsavkin.com/post/118372404541/the-core-concepts-of-angular-2

http://victorsavkin.com/post/102965317996/angular-2-bits-unified-dependency-injection

http://victorsavkin.com/post/119943127151/angular-2-template-syntax

Referencias (cont).

---

http://victorsavkin.com/post/114168430846/two-phases-of-angular-2-applications

http://victorsavkin.com/post/110170125256/change-detection-in-angular-2

https://www.youtube.com/channel/UCzrskTiT_ObAk3xBkVxMz5g

http://larseidnes.com/2014/11/05/angularjs-the-bad-parts/

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Referencias (cont).

---

http://victorsavkin.com/post/123555572351/writing-angular-2-in-typescript

https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

http://xgrommx.github.io/rx-book/index.html

http://reactivex.io/tutorials.html

https://docs.google.com/document/d/1q6g9UlmEZDXgrkY88AJZ6MUrUxcnwhBGS0EXbVlYicY/edit

Referencias (cont).

---

https://github.com/angular/universal

https://github.com/angular/universal-starter

https://medium.com/@mjackson/universal-javascript-4761051b7ae9

http://www.typescriptlang.org/

http://reactivex.io/

Referencias (cont).

---

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/

http://blog.thoughtram.io/angular/2015/05/18/dependency-injection-in-angular-2.html

http://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html

https://vsavkin.com/angular-router-preloading-modules-ba3c75e424cb#.s1zx6pynl

Referencias (cont).

---

https://vsavkin.com/angular-router-understanding-redirects-2826177761fc#.agm0cvep9

https://vsavkin.com/the-powerful-url-matching-engine-of-angular-router-775dad593b03#.d6rpvqwxk