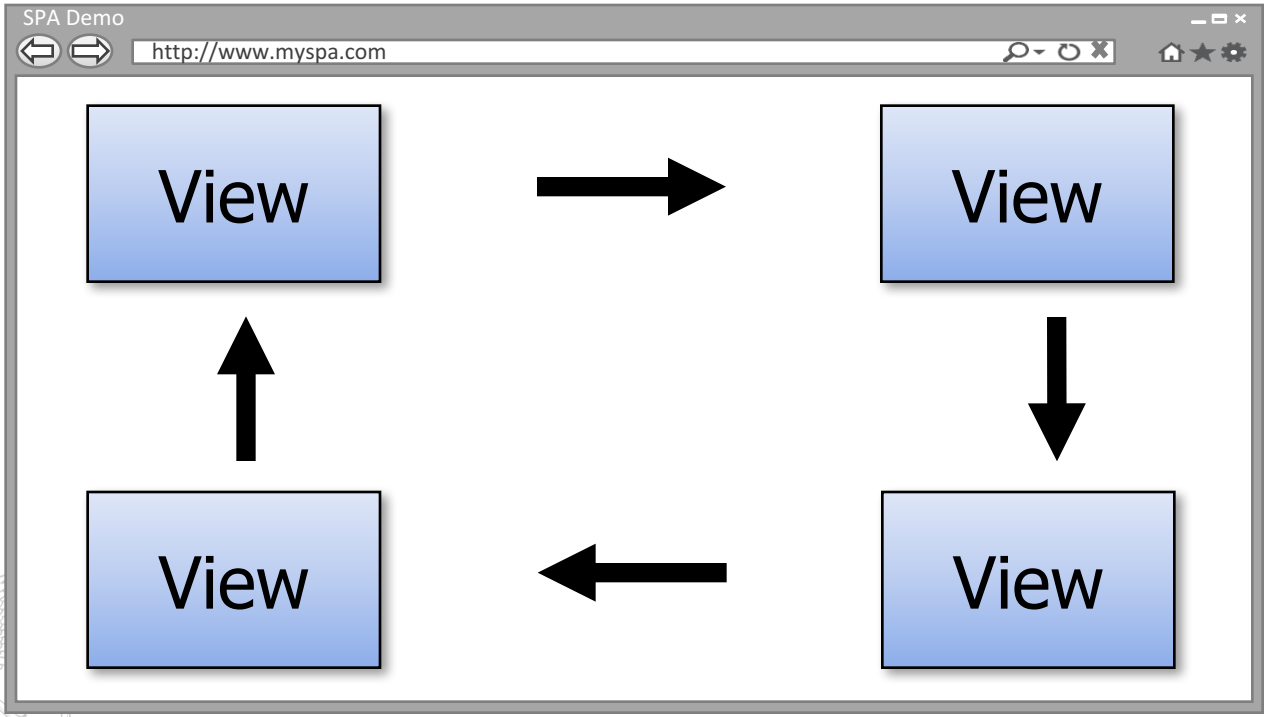
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

9. Angular + IONIC

Dr. Bilicki Vilmos
Szegedi Tudományegyetem
Informatikai Tanszékcsoport
Szoftverfejlesztés Tanszék

Egy oldalas alkalmazás (SPA)



SPA kihivások

DOM manipuláció

Történet

Modul betöltés

Routing

Gyorsítótárazás

Objektum modellezés

Adat Kötés

Ajax/Promises

Nézet betöltés

Data Binding | MVC | Routing | Testing

jqLite | Templates | History | Factories

ngular is a full-featured SPA framework

ViewModel | Controllers | Views | Directives

Controllers | Dependency Injection | Validation

Mi is az Angular?

- Egy mobil és webes alkalmazások fejlesztését lehetővé tévő keretrendszer.
- Az első verzió a GWT leváltására született 2009-ben (Google)
- A második verzió jelenleg béta
- Teljesen újra lett írva, át lett gondolva

A Web Konvergencia

- A Hibrid alkalamzás modell egyszerűbbé teszi a webes és a mobil alkalmazások konvergenciáját
- Minél mélyebben integráljuk a két világot annál bonyolultabb lesz a JS vagy TS kód
- Az Angular keretrendszer e problémák kezelésében segít

Angular 2

- Elvileg bármilyen JS-re fordítható nyelvet használhatunk

www.dartlang.org

www.typescriptlang.org

developer.mozilla.org/docs/web/javascript

Can be either ES5 or ES6

jQuery

- Lehetővé teszi a DOM manipulációt
- Nem ad struktúrát a kódunknak (explicit middleware)
- Nem támogatja a különböző adatfolyam kötéseket (kétirányú kötés)

Hello HTML

Hello World!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Hello Javascript

```
<p id="greeting1"></p>

<script>
var isIE = document.attachEvent;
var addListener = isIE
  ? function(e, t, fn) {
      e.attachEvent('on' + t, fn);}
  : function(e, t, fn) {
      e.addEventListener(t, fn, false);};
addListener(document, 'load', function(){
  var greeting = document.getElementById('greeting1');
  if (isIE) {
    greeting.innerText = 'Hello World!';
  } else {
    greeting.textContent = 'Hello World!';
  }
});
</script>
```

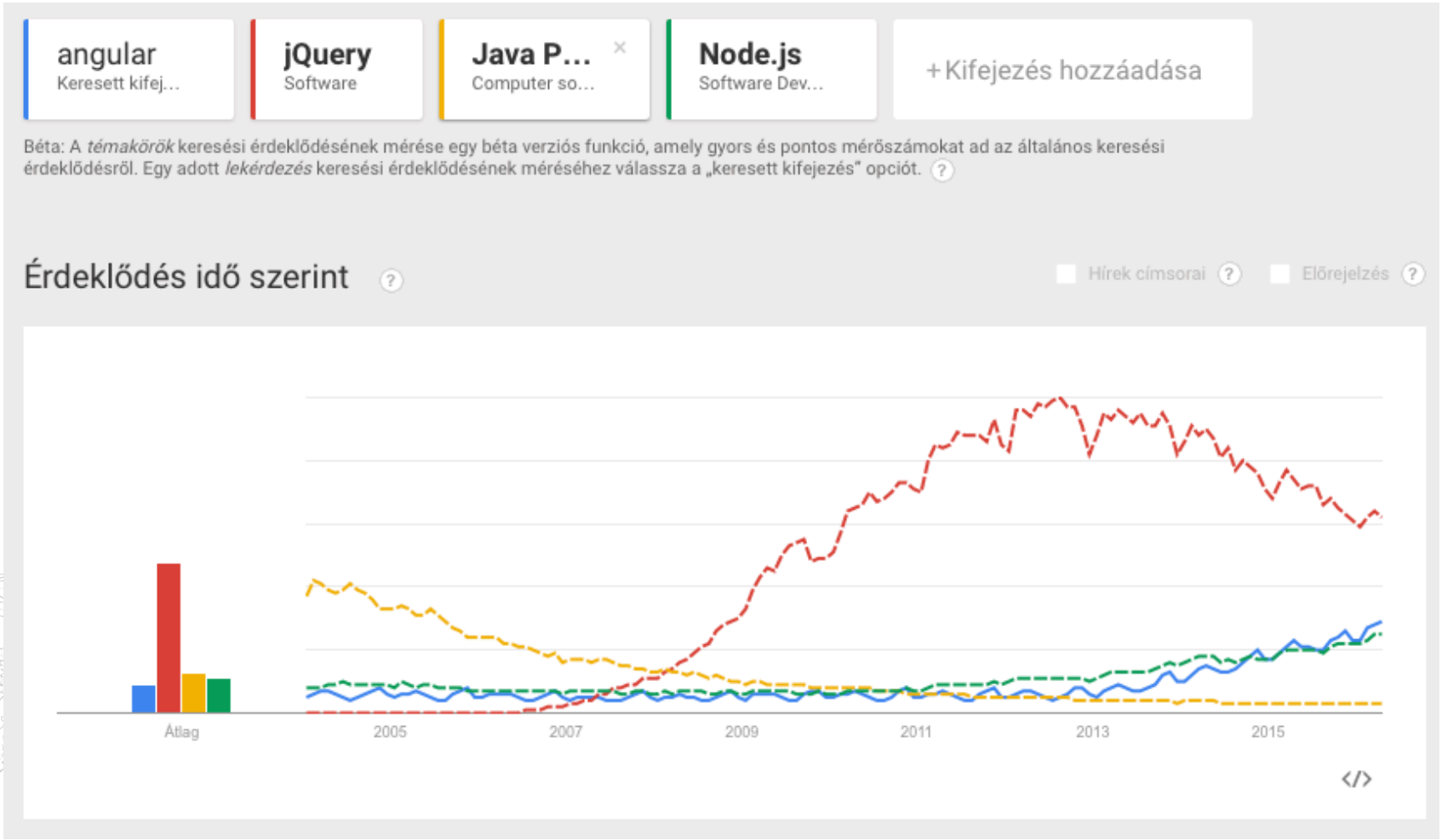
Hello JQuery

```
<p id="greeting2"></p>


<script>
$(function(){
  $('#greeting2').text('Hello World!');
});
</script>
```

Hello AngularJS

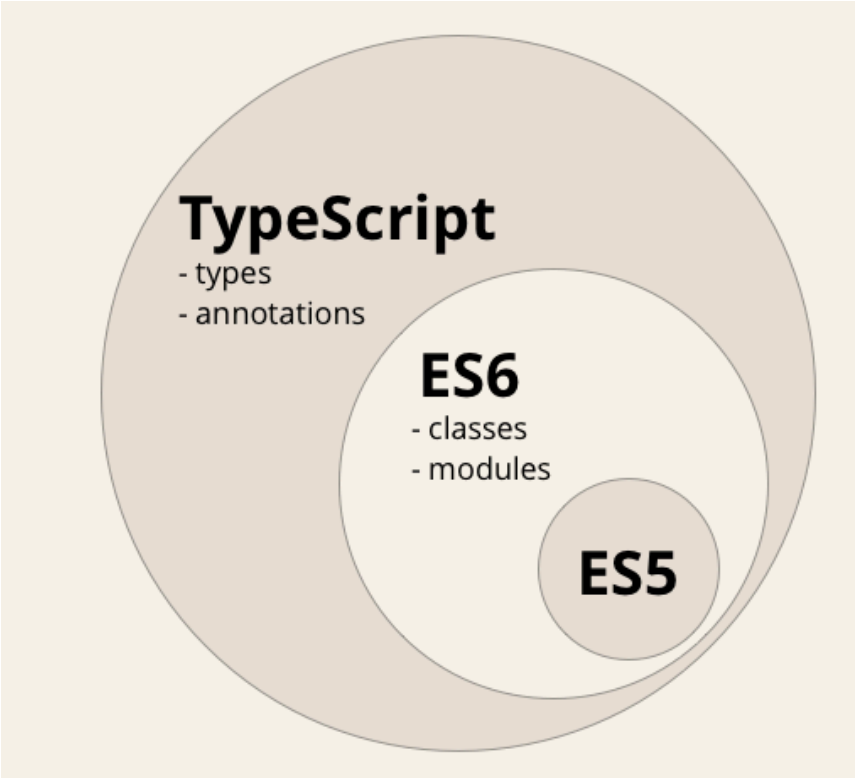
```
<p ng:init="greeting = 'Hello
   World!'">{{greeting}}</p>
```

Más keretrendszerek



TypeScript

- Google + MS
- Lefordítható JS-re
  - Transplier (babel)
- Képességek
  - Típusok
  - Osztályok
  - Annotációk
  - Csomag kezelés
  - Utility-k



Érdekesebb képességek

- Vastag nyíl funkciók (Fat Arrow Function)

```
// ES5-like example
var data = ['Alice Green', 'Paul Pfifer', 'Louis Blakenship'];
data.forEach(function(line) { console.log(line); });
```

```
// Typescript example
var data: string[] = ['Alice Green', 'Paul Pfifer', 'Louis Blakenship'];
data.forEach( (line) => console.log(line) );
```

- Sablon stringek (Template Strings)
  - Több soros
  - Változók használata

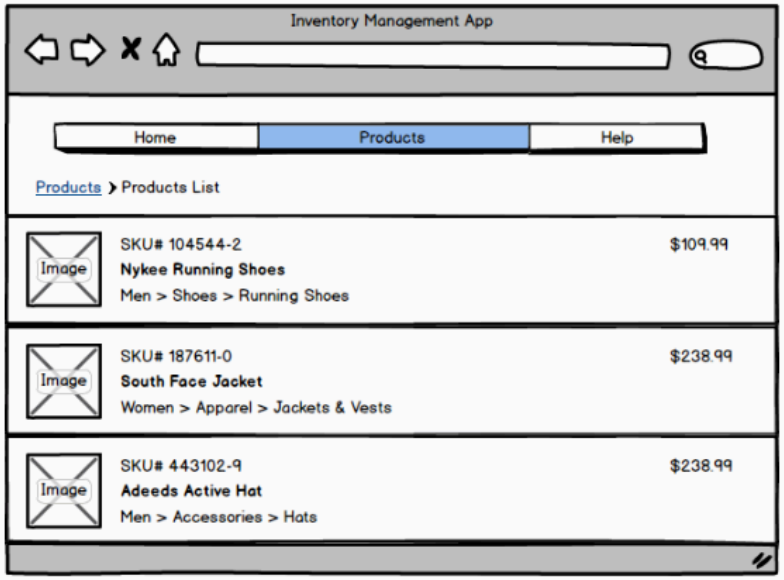
```
var firstName = "Nate";
var lastName = "Murray";

// interpolate a string
var greeting = `Hello ${firstName} ${lastName}`;

console.log(greeting);
```

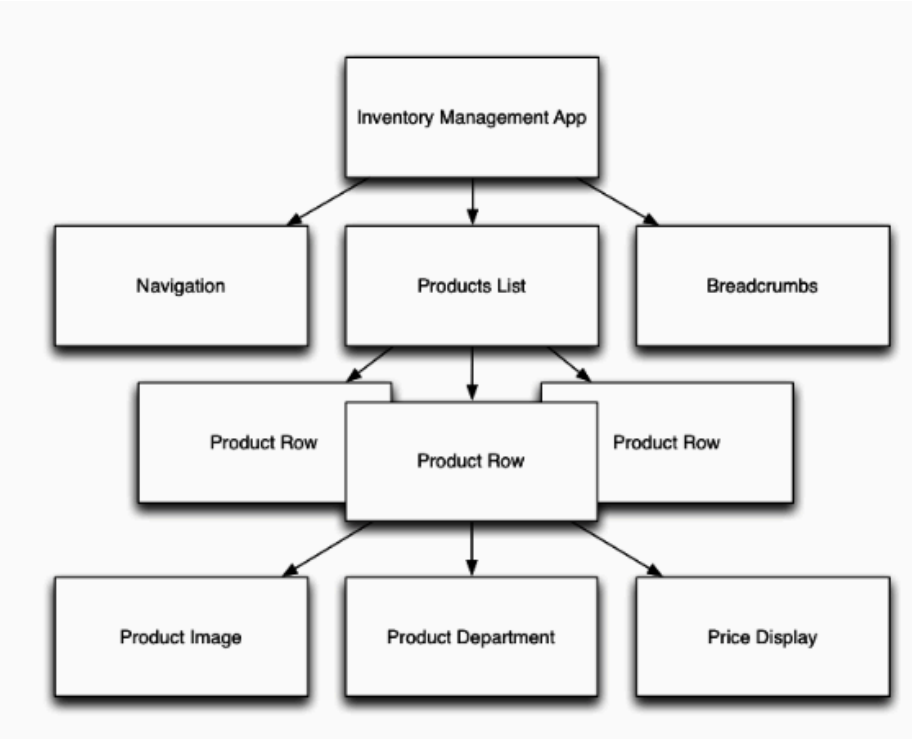
```
var template = `
<div>
  <h1>Hello</h1>
  <p>This is a great website</p>
</div>
`

// do something with `template`
```

Angular megközelítés

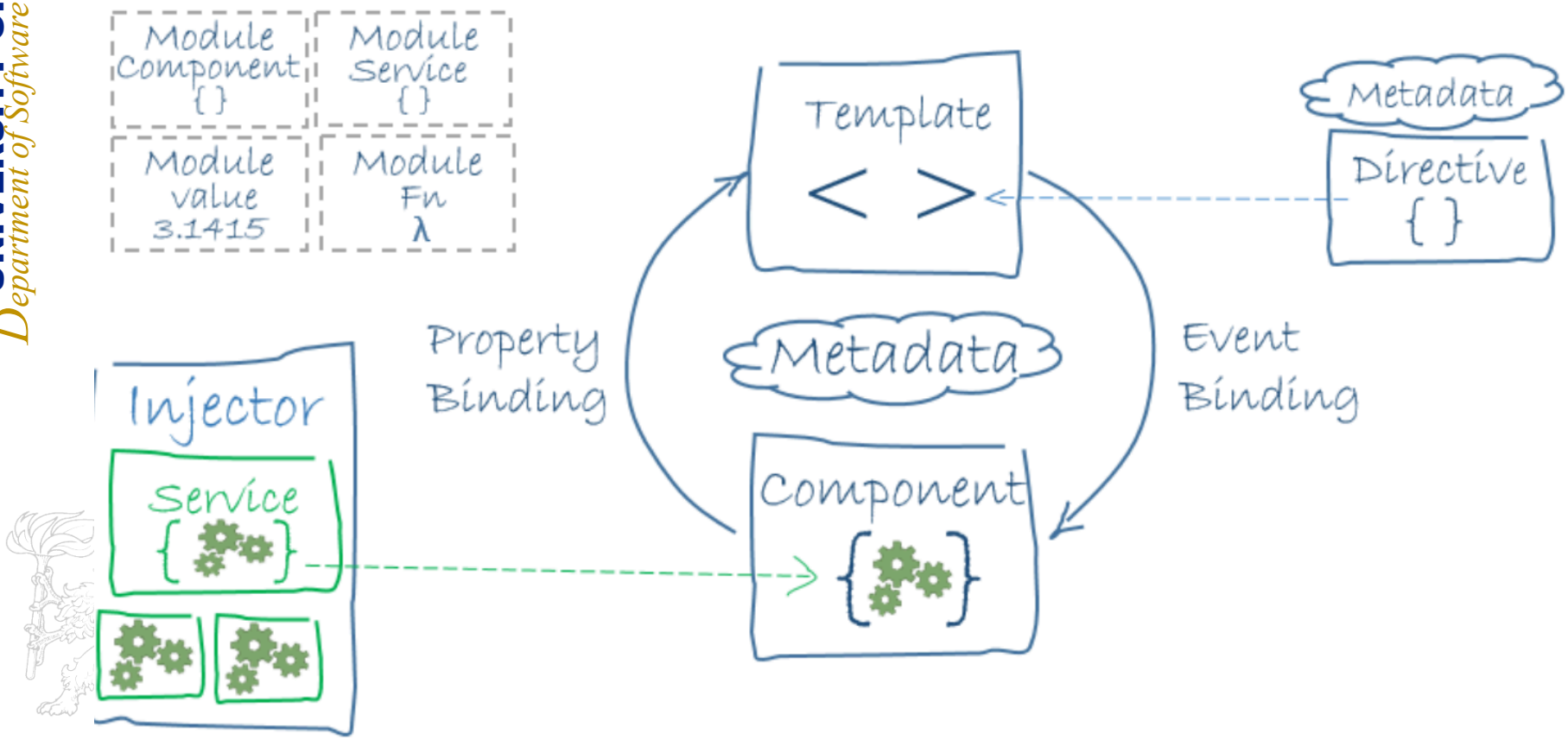
- Az alkalmazások az alábbi egységekből állnak:
  - **HTML *templates* (sablonok)** Angular jelölő elemekkel
  - ***component* classes (komponensek)** a sablonok kezelésére
  - ***services (szolgáltatások)*** az alkalmazás logika megvalósítására
  - Angular indító ***bootstrapper*** a gyökér komponens
- Az Angular átveszi a tartalom megjelenítést, a szerver kezelés és a logika megvalósítást (AJAX motor, implicit middleware)

Angular megközelítés





Angular architektúra



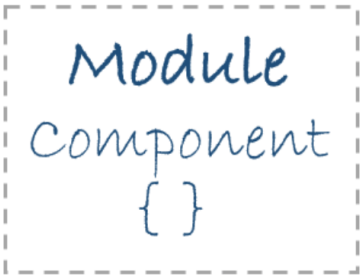
Hello World

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://code.angularjs.org/2.0.0-alpha.26/angular2.sfx.
    </script>
    <script src="main.js"></script>
</head>
    <body>
        <my-application></my-application>
    </body>
</html>
```

```
18      <script>
19        System.config({
20          packages: {
21            app: {
22              format: 'register',
23              defaultExtension: 'js'
24            }
25          }
26        });
27        System.import('app.js')
28              .then(null,
console.error.bind(console));
29      </script>
30
31      <hello-world></hello-world>
32
```

```
3 import { bootstrap } from
"angular2/platform/browser";
4 import { Component } from
"angular2/core";
5
6 @Component({
7   selector: 'hello-world',
8   template: `
9   <div>
10    Hello world
11  </div>
12  `
13 })
14 class HelloWorld {
15 }
16
17 bootstrap(HelloWorld);
```

Modulok



- Az Angular alkalmazások modulokból állnak
- Egyes modulok függvényeket, változókat, osztályokat exportálnak, ezeket máshol más modulok importálják azaz felhasználják
- Az alkalmazás ilyen modulok gyűjteménye. Lehetőség szerint minden modul egy dolgot csinál.
- A modul általában egy egybe tartozó funkciót képvisel. Pl.: egy osztály

Példa

- Az import utatsítás segítségével tudatjuk a rendszerrel, hogy az AppComponent elemet az app.component modulból tudja betölteni
- A **modul neve** (modul id) gyakran megegyezik a fájlnévvel

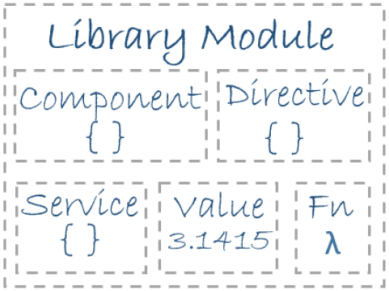
app/app.component.ts (excerpt)

```
export class AppComponent { }
```

app/main.ts (excerpt)

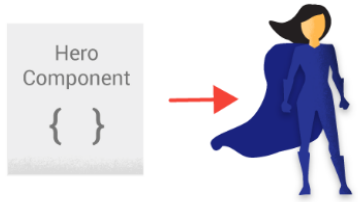
```
import {AppComponent} from './app.component';
```

Könyvtár Modulok



- Egyes modulok más modulok gyűjteményei, könyvtárai
- Az Angular ilyen modulok gyűjteményében valósítja meg az alap funkcionalitását..
- Az angular2/core könyvtár az elsődleges funkciókat megvalósító, több privát modult is elfedő modul
- Vannak más Angular könyvtárak is pl: angular2/common, angular2/router, és angular2/http.

A Komponens elem



- A Komponens (**Component)** a képernyő egy részét kezeli. Ezt a képernyő részt nézetnek view) nevezzük.
- A nézethez kötődő alkalmazás logikát is a komponens osztályon belül adjuk meg.
- Az osztály a nézetettel megfelelő API-kon keresztü kommunikál.
- A komponesek életútját az Angular kezeli a felhasználó események függvényében. Ezekre a változásokra fel tudunk íratkozni (Lifecycle Hooks).

Példa

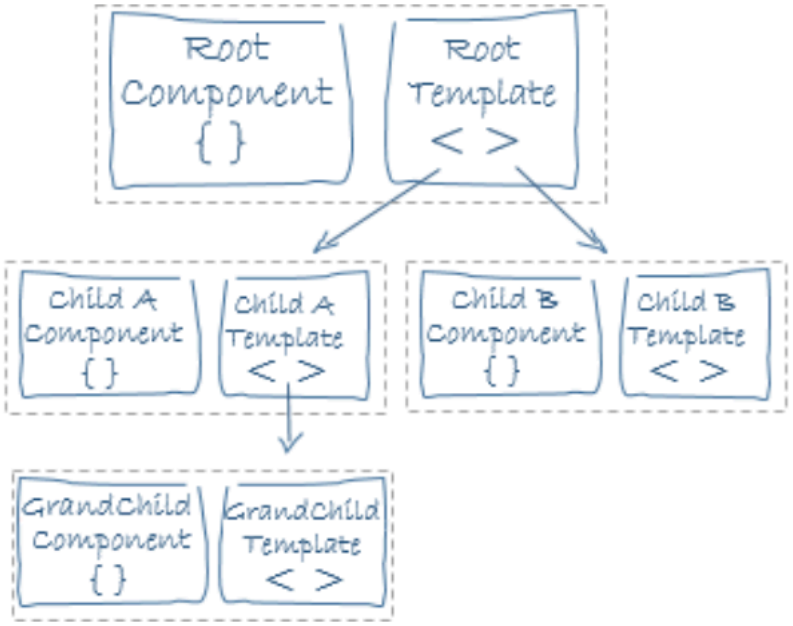
app/hero-list.component.ts

```
1.  export class HeroListComponent implements OnInit {
2.    constructor(private _service: HeroService){ }
3.
4.    heroes:Hero[];
5.    selectedHero: Hero;
6.
7.    ngOnInit(){
8.      this.heroes = this._service.getHeroes();
9.    }
10.
11.   selectHero(hero: Hero) { this.selectedHero = hero; }
12. }
```

A Sablon (Template)

- A komponens által képviselt nézetet (view) a sablon (template) segítségével definiálhatjuk.
- Ez a sablon egy HTML elem halmaz kiegészítve Angular elemekkel
- Tetszőlegesen keverhetjük a saját és az eredeti elemeket
- Ez a hierarchia adja az alkalmazás vázát (esemény kezelés, …)

Példa



app/hero-list.component.html

```
1. <h2>Hero List</h2>
2.
3. <p><i>Pick a hero from the list</i></p>
4. <div *ngFor="#hero of heroes" (click)="selectHero(hero)">
5.   {{hero.name}}
6. </div>
7.
8. <hero-detail *ngIf="selectedHero" [hero]="selectedHero"></hero-detail>
```

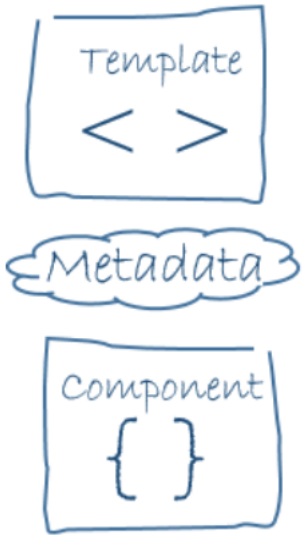
Angular metainformáció

- A metainformáció segítségével írjuk le egy-egy osztály kezelési módját.
- TypeScript esetén ezt az információt **dekorátor (decorator)** segítségével adhatjuk meg
- A dekorátor egy függvény amely gyakran konfigurációs paraméterekkel is rendelkezik.
- A @Component dekorátor a megfelelő paraméterekkel tudatja az Angular keretrendszerrel, hogy mit és hogyan cselekedjen a komponensünkkel és a nézetünkkel.

Példa

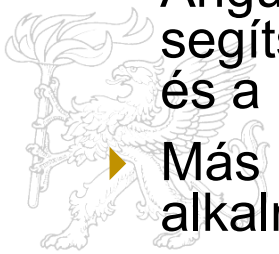
app/hero-list.component.ts (metadata)

```
1. @Component({
2.   selector:    'hero-list',
3.   templateUrl: 'app/hero-list.component.html',
4.   directives:  [HeroDetailComponent],
5.   providers:   [HeroService]
6. })
7. export class HeroesComponent { ... }
```

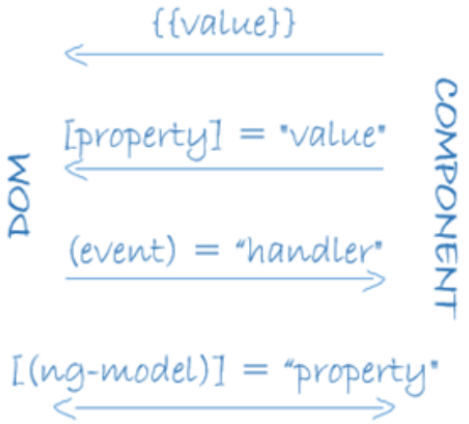



A @Component dekorátor

- selector - egy css kiválasztó amely tudatja az Angular keretrendszerrel, hogy itt le kell cserélnie a tartalmat a kompenes tartalmára. Pl.: <hero-list> csomópontot a view-ben leírt és kiértékelt tartalomra.
- templateUrl – a komponens sablonjának elérési útvonala
- directives – a komponens által felhasznált komponensek és direktívák listája
- providers – a komponens által felhasznált szolgáltatások listája.
- A @Component függvény a megadott kofigurációs objektumot metaadattá alakítja és a komponens osztályhoz csatolja. Az Angular futás időben olvassa ki ezt a metainformációt és e segítségével tudja, hogy mit kell tennie. A sablon a metainformáció és a komponens együttesen határozzák meg a nézetet.
- Más metadata meghatározó dekorátorokat is hasonlóan alkalmazunk pl.: @Injectable, @Input, @Output, @RouterConfig



Adatkötés



- E nélkül nekünk kellene az egyes komponensekbe beírni az adatokat, kilvasni azokat és a felhasználói eseményeket lekezelni.
- Ez nagyon munka igényes (jQuery)
- Az angular több fajta **adat kötést** is támogat. Címke segítségével összeköti a sablon elemeket és a komponens többi elemét.
- Négy kötés típus közül lehet választani

Adat kötés

- Interpolation: Ezzel a megoldással kapcsos zárójélbe tesszük a tulajdonság nevét. Egyirányú kötés: {{myHero}}.

- Tulajdonság kötés: Egy nézet elem adott tulajdonságát szeretnénk módosítani.

```
<img [src] = "heroImageUrl">
```

- Esemény kötés: A felhasználói DOM események kezelésére.Ezzel a frissített értékek is a komponensekbe kerülnek.

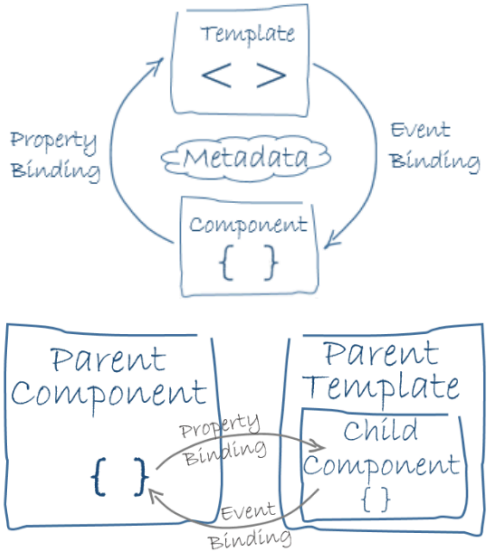
```
<button (click)="onClickMe()">Click me!</button>
```

- Kétirányú adat kötés : az előző kettő kombinációja az ngModel direktíva segítségével. Az eleme érétke megkapja a tulajdonság értékét és változás esetén a tulajdonság értéke az új elem érték lesz

```
<input [(ngModel)]="hero.name">
```

Adat kötés

- Az Angular keretrendszer minden JavaScript esemény ciklusban egyszer ellenőrzi az összes adat kötést a gyökértől kiindulva mélységében a gyerekekig



A Direktíva

- Az Angular sablonok dinamikusak. Amikor a kerendszer rendereli a DOM-ot akkor az egyes elemeket a direktíváknak megfelelően kezeli.
- A direktíva egy osztály metaadtattal. TypeScript esetén a @Directive dekorátor segítségével tudunk az osztályhoz metadatot csatolni.
- Két csoportra oszthatjuk a direktívákat:.
  - **Struktúrális** direktívák megváltoztatják a kinézetét DOM elemek hozzáadásával vagy eltávolításával.
  - **Atribútum** direktíva elemek a vislekedését változtatják meg egy elemenek..

A szolgáltatás

- A ”Szolgáltatás" egy széles kategória bármi lehet benne ami az alkalmazásnak szükséges.
- Minden lehet szolgáltatás
- A szolgáltatás általában egy jól meghatározott szűk funkció listával bíró osztály.
- Az Angular nem definiál semmilyen kötöttséget sem. Nincs *ServiceBase* osztály.
- A szolgáltatások minden Angular alkalmazás alapvető elemei.

Komponens vs Szolgáltatás

- A komonenseknek szolgáltatásokra kellene épülnie:.
  - A szerverrel történő kommunikáció (pl.: adat lekérés)
  - Felhasználó adat validáció
  - Naplózás
  - Helyi adattár
- A komponens dolga a felhasználói élmény (események, nézet) biztosítása
- A feladata a nézet és az alkalmazás logika közötti közvetítés megvalósítása
- A megfelelő komponens az adat kötéshez és esemény kezeléshez tartozó metódusokat tartalmazza
- Minden mást a szolgáltatásokhoz delegál

Példa

app/logger.service.ts (class only)

```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

Függőség Injektálás

- A ”Függőség Injektálás" az objektumok kiszolgálásának egy divatos módja. Amire szüksége van megkapja.
- A legtöbb függőség az szolgáltatás

Függőség Injektálás

- Amikor az Angular egy komponenst készít akkor először egy **Injektor** –tort igényel.
- Az Injektor az előre létrehozott szolgáltatás példányokat tartalmazza
- Amennyiben az adott szolgáltatás példány még nem létezik akkor létrehoz egyet ée hozzáadja a csomaghoz
- Amikor minden igényelt szolgáltatást beinjektált akkor hívja meg az adott komponens konstruktorát.

Példa

app/hero-list.component (constructor)

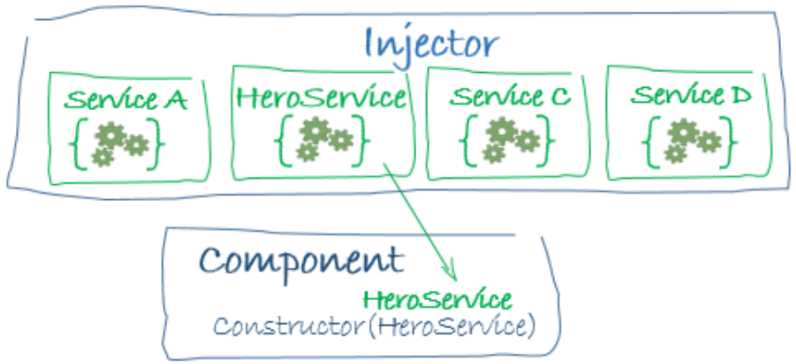
```
constructor(private _service: HeroService){ }
```

app/main.ts (excerpt)

```
bootstrap(AppComponent, [BackendService, HeroService, Logger]);
```

app/hero-list.component.ts (excerpt)

```
@Component({
  providers:   [HeroService]
})
export class HeroesComponent { ... }
```



Alap direktíva készlet

- NgIf
- NgSwitch
- NgStyle
- NgClass
- NgFor


```
27      <div>
28        <span [ngStyle]="{color: 'red'}"
[style.font-size.px]="fontSize">
29          red text
30        </span>
31      </div>
```

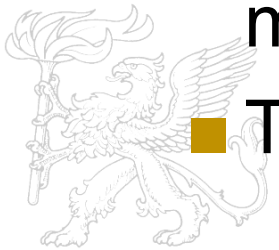
```
14 .bordered {
15   border: 1px dashed black;
16   background-color: #eee;
17 }
```

```
7   <div [ngClass]="{bordered:
false}">This is never bordered</div>
8   <div [ngClass]="{bordered: true}">This
is always bordered</div>
```

```
<table class="ui celled table">
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tr *ngFor="#p of item.people">
    <td>{{ p.name }}</td>
    <td>{{ p.age }}</td>
  </tr>
</table>
```

Űrlap kezelés

- Problémák:
  - A beviteli mezők gyakran az oldalon és a szerveren is változásokat indukálnak
  - Az oldalon belüli változások gyakran komplex összefüggésekkel írhatóak le
  - A felhasználói bevitelt validálni kell és megfelelően jelezni kell a hibás bevitelt
  - Tesztelhető kódot szeretnénk



Űrlapok

- A következő eszközökre épít:
  - Vezérlők (Controls (ControlGroups)) A beviteli komponenseket intelligens elemekbe burkolja
  - Validátorok (Validators) a bemenetet validálja
  - Megfigyelők (Observers) a változások kezelésére ad eszközöket

Vezérlő

- Az Angular űrlap legkisebb eleme
  - Változott-e (dirty)
  - Hibás-e

```
// create a new Control with the value "Nate"
let nameControl = new Control("Nate");

let name = nameControl.value; // -> Nate

// now we can query this control for certain values:
nameControl.errors // -> StringMap<string, any> of errors
nameControl.dirty  // -> false
nameControl.valid  // -> true
// etc.
```

```
<!-- part of some bigger form -->
<input type="text" ngControl="name" />
```

Vezérlő csoportok

- Együtt kezeli a befoglalt vezérlőket

```
let personInfo = new ControlGroup({
    firstName: new Control("Nate"),
    lastName: new Control("Murray"),
    zip: new Control("90210")
})
```

```
personInfo.value; // -> {
//   firstName: "Nate",
//   lastName: "Murray",
//   zip: "90210"
// }

// now we can query this control group
for certain values, which have sensible
// values depending on the children
Control's values:
personInfo.errors // ->
StringMap<string, any> of errors
personInfo.dirty  // -> false
personInfo.valid  // -> true
// etc.
```

Példa


```
3
4  @Component({
5    selector: 'demo-form-sku',
6    directives: [FORM_DIRECTIVES],
7    template: `
8    <div class="ui raised segment">
9      <h2 class="ui header">Demo Form:
Sku</h2>
10     <form #f="ngForm"
11           (ngSubmit)="onSubmit(f.value)"
12           class="ui form">
13
14       <div class="field">
15         <label
for="skuInput">SKU</label>
16         <input type="text"
17                id="skuInput"
18                placeholder="SKU"
19                ngControl="sku">
20       </div>
21
22       <button type="submit" class="ui
button">Submit</button>
23     </form>
24   </div>
25   `
26 })
27 export class DemoFormSku {
28   onSubmit(form: any): void {
29     console.log('you submitted value:',
form);
30   }
31 }
```

ngForm - NgForm

- ngForm kontroll csoport
- ngSubmit kimenet
- Importálás után automatikusan minden form elemhez hozzárendelődik (NgForm)
- ngControl – ez hozza lérte az intelligens vezérlőt

FormBuilder

- A háttér struktúra egyszerű létrehozása


```
import { Component } from 'angular2/core';
import {
  FORM_DIRECTIVES,
  FormBuilder,
  ControlGroup
} from 'angular2/common';

@Component({
  selector: 'demo-form-sku-builder',
  directives: [FORM_DIRECTIVES],
  template: `
  <div class="ui raised segment">
    <h2 class="ui header">Demo Form: Sku with Builder</h2>
    <form [ngFormModel]="myForm"
          (ngSubmit)="onSubmit(myForm.value)"
          class="ui form">

      <div class="field">
        <label for="skuInput">SKU</label>
        <input type="text"
               id="skuInput"
               placeholder="SKU"
               [ngFormControl]="myForm.controls['sku']">
      </div>

    <button type="submit" class="ui button">Submit</button>
    </form>
  </div>
  `
})
export class DemoFormSkuBuilder {
  myForm: ControlGroup;

  constructor(fb: FormBuilder) {
    this.myForm = fb.group({
      'sku': ['ABC123']
    });
  }

  onSubmit(value: string): void {
    console.log('you submitted value: ', value);
  }
}
```

Validátorok

- Validators modul

```
21
22        <div class="field"
23            [class.error]="!sku.valid && sku.touched">
24          <label for="skuInput">SKU</label>
25          <input type="text"
26                 id="skuInput"
27                 placeholder="SKU"
28                 [ngFormControl]="sku">
29           <div *ngIf="!sku.valid"
30             class="ui error message">SKU is
invalid</div>
31           <div *ngIf="sku.hasError('required')"
32             class="ui error message">SKU is
required</div>
33        </div>
34
35        <div *ngIf="!myForm.valid"
36          class="ui error message">Form is invalid</div>
37
```

```
43 export class DemoFormWithValidationsExplicit {
44   myForm: ControlGroup;
45   sku: AbstractControl;
46
47   constructor(fb: FormBuilder) {
48     this.myForm = fb.group({
49       'sku':  ['', Validators.required]
50     });
51
52     this.sku = this.myForm.controls['sku'];
53   }
54
55   onSubmit(value: string): void {
56     console.log('you submitted value: ', value);
57   }
58 }
```

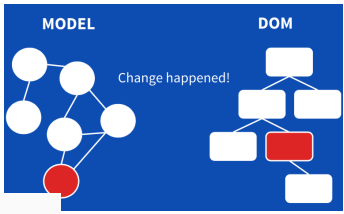
```
1 export class Validators {
2   static required(c: Control): StringMap<string,
boolean> {
3     return isBlank(c.value) || c.value == "" ?
{"required": true} : null;
4   }
```

Változás kezelés

```
@Component({
  template: `
    <h1>{{firstname}} {{lastname}}</h1>
    <button (click)="changeName()">Change name</button>
  `
})
class MyApp {

  firstname:string = 'Pascal';
  lastname:string = 'Precht';

  changeName() {
    this.firstname = 'Brad';
    this.lastname = 'Green';
  }
}
```

```
@Component()
class ContactsApp implements OnInit{

  contacts:Contact[] = [];

  constructor(private http: Http) {}

  ngOnInit() {
    this.http.get('/contacts')
      .map(res => res.json())
      .subscribe(contacts => this.contacts = contacts);
  }
}
```

Változás detektálás



- Hogyan veszük észre, hogy valami változott az utolsó képernyő frissítés óta?
  - DOM különbség -> Virtuális DOM
- A változásnak számos oka lehet:
  - **Események** - kattintás, felküldés, …
  - **XHR** – adat letöltés szerverről
  - **Időzítők**- setTimeout(), setInterval()
- Ezek mind aszinkronok

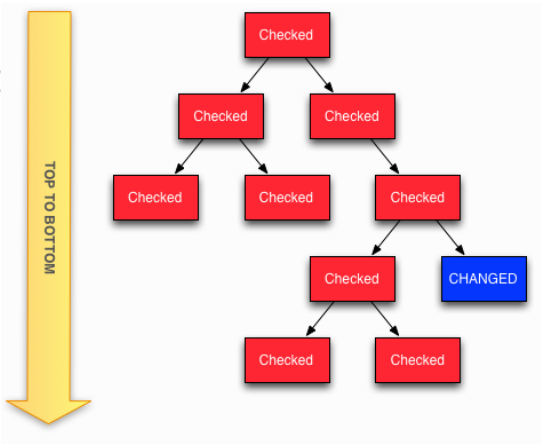
```
@Component({
  selector: 'my-component',
  template: `
  Name: {{name}}
  <button (click)="changeName()">Change!
</button>
  `
})
class MyComponent {
  name: string;
  constructor() {
    this.name = 'Felipe';
  }

  changeName() {
    this.name = 'Nate';
  }
}
```

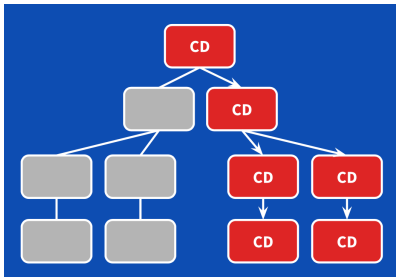
```
@Component({
  selector: 'my-component',
  template: `
  Name: {{name}}
  `
})
class MyComponent {
  name: string;
  constructor() {
    setTimeout(() => this.name = 'Felipe', 2000);
  }
}
```

```
@Component({
  selector: 'my-component',
  template: `
  Name: {{name}}
  `
})
class MyComponent {
  name: string;
  constructor(private http: Http) {
    this.http.get('/names/1')
      .map(res => res.json())
      .subscribe(data => this.name =
data.name);
  }
}
```

Változás detektálás

- Egy NgZones objektumban van összefogva és ott minden JS körben az onTurnDone esemény hatására elindul a változás detektálás
- Minden komponensenek saját változás detektora van (ez az egész fára egy változás detektálást indít el)
- Egy változás detektor fa épül. Itt a gyökértől megy lefelé az ellenőrzés
- Optimalizálás lehet (OnPush):
  - Megváltoztathatalan objektumokra
  - Observalbe-ra – markForCheck()



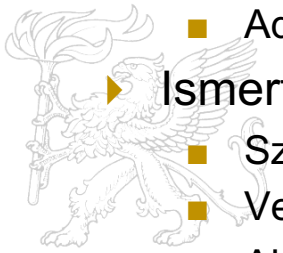



```
constructor(private cd: ChangeDetectorRef) {}
```

```
  ngOnInit() {
    this.addItemStream.subscribe(() => {
      this.counter++; // application state changed
      this.cd.markForCheck(); // marks path
    })
  }
}
```

Forgalimránányítás - Routing

- Kliens oldalon vagyunk:
  - Egy URL is elegendő lenne
  - De
    - Nem tudunk úgy firssíteni, hogy megmaradjon az állapotunk
    - Nem tudjuk lemeneteni az elérési címet
    - Nem tudjuk megosztani az URL-t
- Az alkalmazást különböző területekre bontjuk
- Ezen területeket külön URL érjük el
- Segítségével:
  - Elkölönítjük az alkalmazás területeit
  - Az alkalmazás állapotát megfelelő eszköztárral kezeljük
  - Adott területeket adott szinten védhetünk
- Ismert minta alapján
  - Szerver (server)
  - Vezérlő (controller)
  - Akció (action)



Forgalomirányítás

- Első megoldások belső hivatkozások:

```
1 <!-- ... lots of page content here ... -->
2 <a name="about"><h1>About</h1></a>
```

```
http://something/#about
```

- # alapú forgalomirányítás (Hash based routing)

```
http://something/#/about
```

- HTML5 - a böngésző módosíthatja a böngészés múltat szerver oldali lekérdezés nélkül (ezt a böngészőnek és a szevernek is támogatnia kell)
  - pushState

Forgalomirányítás

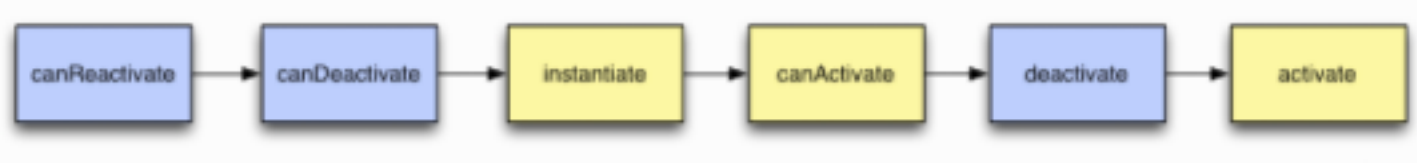
- Komponenshez kötődő utak
- Elemek
  - RouteConfig
    - path, name, component/redirectTo
  - RouterOutlet
    - Hova szeretnénk az oldalt kihelyezni
  - RouterLink
    - Hova menjen (oldal betöltés nélkül)


```
@RouteConfig([
  { path: '/', name: 'root', redirectTo: ['/Home'] },
  { path: '/home', name: 'Home', component: HomeComponent },
  { path: '/about', name: 'About', component: AboutComponent },
  { path: '/contact', name: 'Contact', component: ContactComponent },
  { path: '/contactus', name: 'ContactUs', redirectTo: ['/Contact'] },
])
```

```
@Component({
  selector: 'router-app',
  directives: [ROUTER_DIRECTIVES],
  template: `
  <div>
    <nav>
      <a>Navigation:</a>
      <ul>
        <li><a [routerLink]="['/Home']">Home</a></li>
        <li><a [routerLink]="['/About']">About</a></li>
        <li><a [routerLink]="['/Contact']">Contact us</a></li>
      </ul>
    </nav>

    <router-outlet></router-outlet>
  </div>
```

Forgalomirányítás

- Események/Életciklus



- Védett oldalak kezelése (UI élmény szempontjából, nem biztonság szempontjából)

```
16 @CanActivate(
17   (nextInstr: any, currInstr: any) => {
18     let injector: any =
Injector.resolveAndCreate([AuthService]);
19     let authService: AuthService =
injector.get(AuthService);
20     return authService.isLogged();
21   }
22 )
```

HTTP

- Aszinkron HTTP könyvtár
  - Callback
  - Promise
  - Observable alapú

```
4  import {Component} from 'angular2/core';
5  import {Http, Response} from 'angular2/http';
6
7  @Component({
8    selector: 'simple-http',
9    template: `
10   <h2>Basic Request</h2>
11   <button type="button"
(click)="makeRequest()">Make Request</button>
12   <div *ngIf="loading">loading...</div>
13   <pre>{{data | json}}</pre>
14 `
15 })
16 export class SimpleHTTPComponent {
17   data: Object;
18   loading: boolean;
19
20   constructor(public http: Http) {
21   }
22
23   makeRequest(): void {
24     this.loading = true;
25
this.http.request('http://jsonplaceholder.typicode.com/posts/1

26       .subscribe((res: Response) => {
27         this.data = res.json();
28         this.loading = false;
29       });
30   }
31 }
```

AJAX - Aszinkron

- Szinkron kód átlátható, kezelhető.
- Átlátható, érthető aszinkron kód?
- Események fúziója?

```
try {
  for (var obj in objs) {
    doSomething(obj);
  }
} catch (e) {
  handleError(e);
} finally {
  doCleanup();
}
```

```
interface Observable<T> {
  onNext(value: T) : void
  onError(error: Error) : void
  onCompleted() : void
}
```

Példa probléma

- Javaslat gépelés közben
  - Ne kérdezzük le minden billenytyű lenyomáskor
  - Ne küldjük el többször ugyanazt a lekérdezést
  - Kezeljük a soron kívüli válaszokat

Aszinkron programozás

- Megoldásmódok
  - Callback
    - Egyszerű a szinkron kódhoz közelálló megoldás:

```
function isUserTooYoung(id, callback) {
    openDatabase(function(db) {
        getCollection(db, 'users', function(col) {
            find(col, {'id': id},function(result) {
                result.filter(function(user) {
                    callback(user.age < cutoffAge)
                })
            })
        })
    })
}
```

  - Promise, Future
    - Egy proxy THEN metódussal

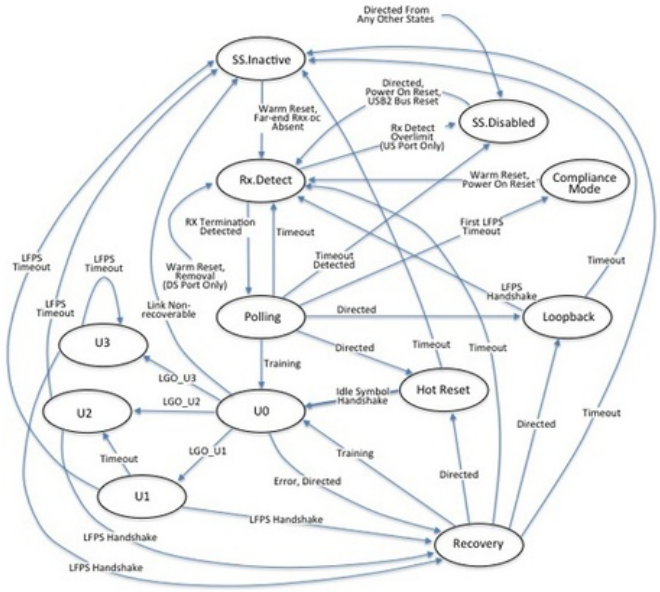
```
function isUserTooYoung(id) {
    return openDatabase(db)
        .then(getCollection)
        .then(find.bind(null, {'id': id}))
        .then(function(user) {
            return user.age < cutoffAge;
        });
}
```

  - Observer
    - Rekatív programozás

Callback hell

- Pokol piramisa
- A program állapota a komplexitás miatt átláthatatlan lesz
- A programozó a sorok sorrendjét a végrehajtás sorrendjének veszi

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```



Promise alapokon

```
import {Injectable} from 'angular2/core';
import {URLSearchParams, Jsonp} from 'angular2/http';

@Injectable()
export class WikipediaService {
  constructor(private jsonp: Jsonp) {}

  search (term: string) {
    var search = new URLSearchParams()
    search.set('action', 'opensearch');
    search.set('search', term);
    search.set('format', 'json');
    return this.jsonp
                .get('http://en.wikipedia.org/w/api.php?callback=JSONP_CALLBACK', {
                .toPromise()
                .then((response) => response.json()[1]);
  }
}
```



```
// check the plnkr for the full list of imports
import {...} from '...';

@Component({
  selector: 'my-app',
  template: `
    <div>
      <h2>Wikipedia Search</h2>
      <input #term type="text" (keyup)="search(term.value)">
      <ul>
        <li *ngFor="#item of items">{{item}}</li>
      </ul>
    </div>
  `
})
export class App {
  items: Array<string>;
  constructor(private wikipediaService: WikipediaService) {
  }

  search (term) {
    this.wikipediaService.search(term)
                         .then(items => this.items = items);
  }
}

bootstrap(App, [WikipediaService, JSONP_PROVIDERS])
    .catch(err => console.error(err));
```

Observable alapokon

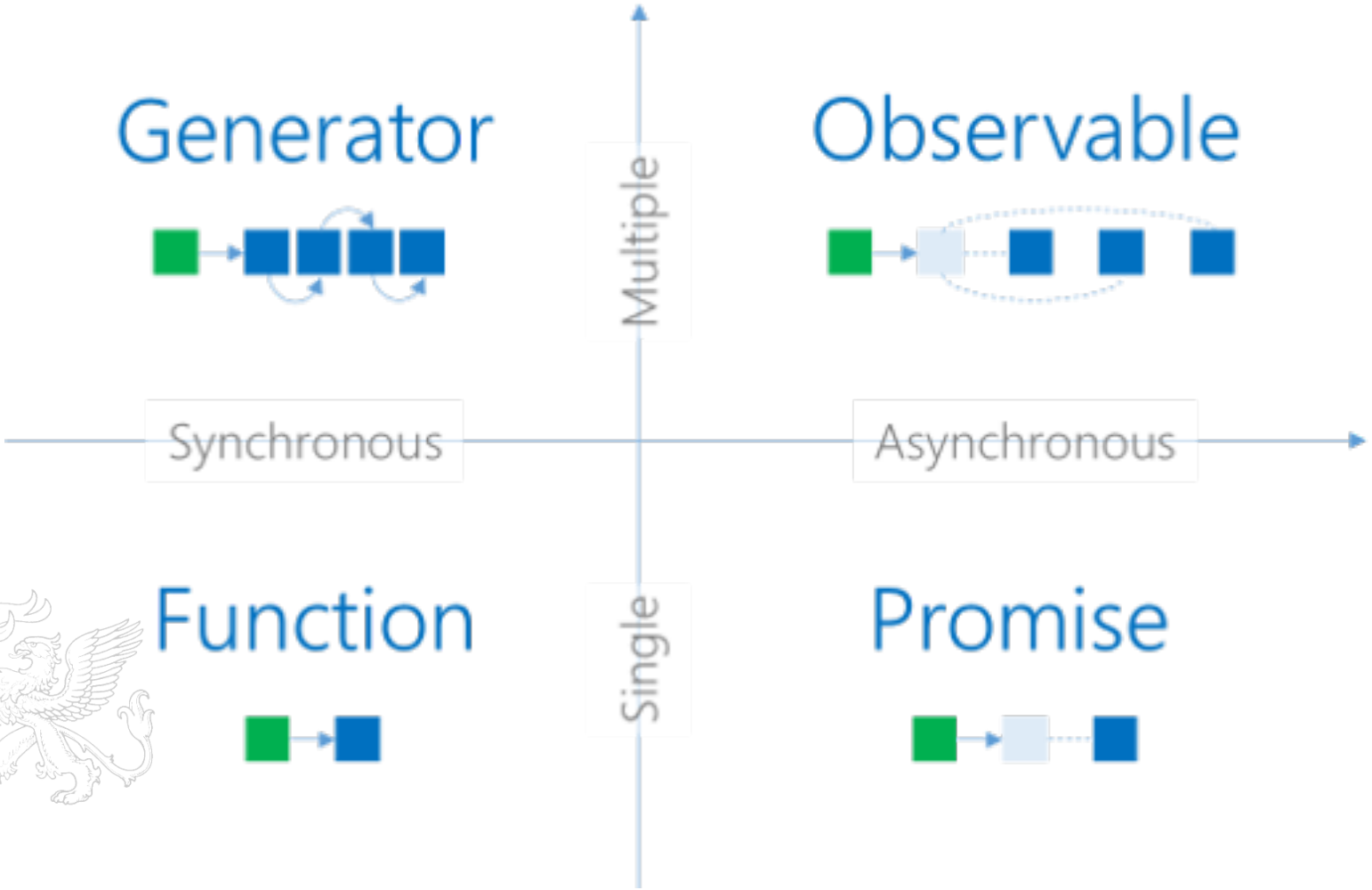
```
<input type="text" [ngFormControl]="term"/>
```

```
export class App {
  items: Array<string>;
  term = new Control();
  constructor(private wikipediaService: WikipediaService) {
    this.term.valueChanges
             .debounceTime(400)
             .subscribe(term => this.wikipediaService.search(term).then(items =>
  }
}
```



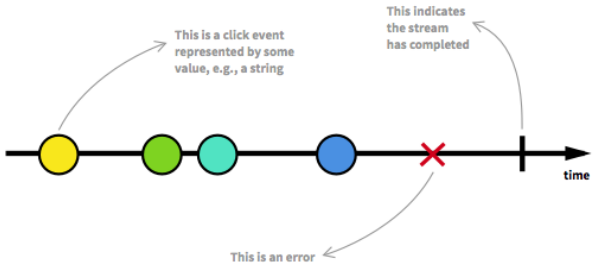
```
this.term.valueChanges
         .debounceTime(400)
         .distinctUntilChanged()
         .flatMap(term => this.wikipediaService.search(term))
         .subscribe(items => this.items = items);
```

Összefoglalva



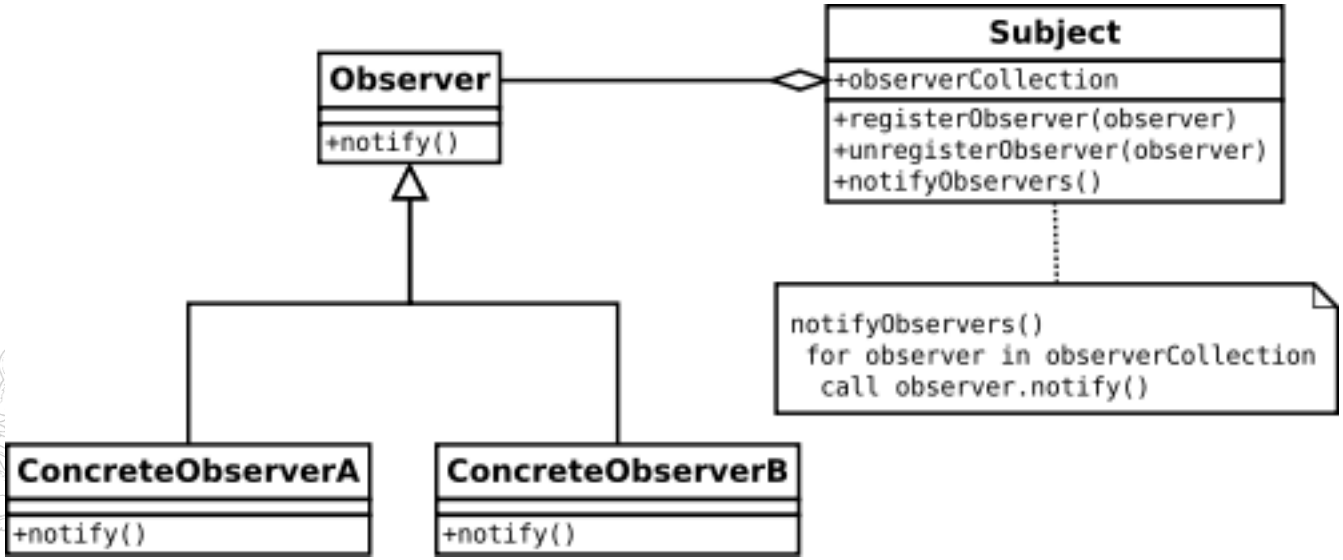
Reaktív programozás - RxJS

- Aszinkron adatfolyam kezelés
  - Esemény buszok
  - Esmény kezelők
- Bármiből gyárthatunk adatfolyamot
- Folyamon alapuló logika
  - Egyik folyam lehet egy másik folyam forrása
  - Összeolvasztahtunk több folyamot, …
- Folyam: a bekövetkezett események időben rendezett folyamatos listája
- Az érték halmaza:
  - Normál értékek
  - Hibák
  - Folyam vége
- Ezen eseményeket aszinkron módon kezeljük:
  - Függvény a normál értékekre
  - Függvény a hibákra
  - Függvény a végére



Reaktív programozás - RxJS

- Folyamokra fel lehet íratkozni
- A folyamokat “megfigyeljük”
- Megfigyelő tervezési minta



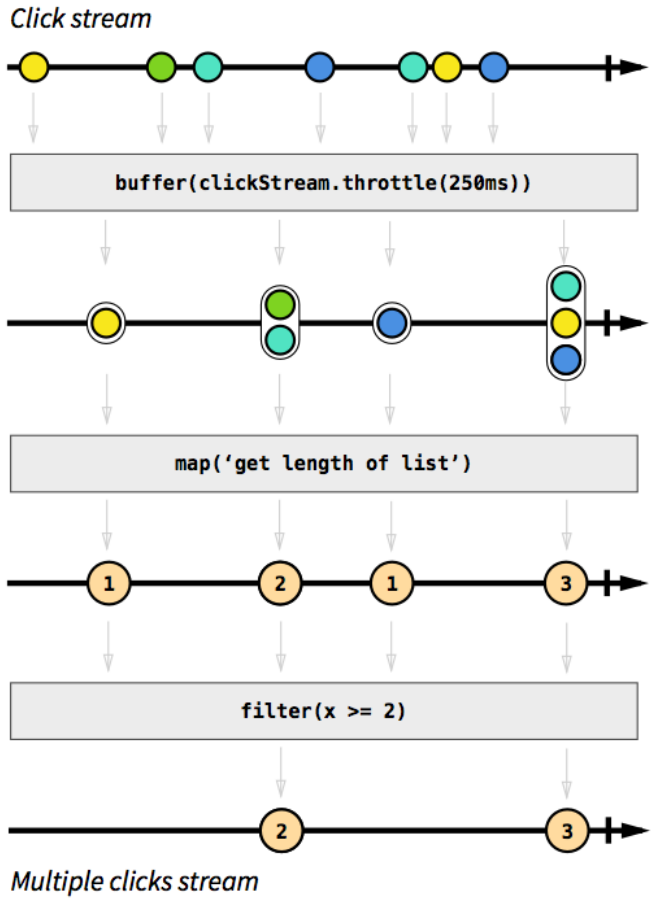
Reaktív programozás - RxJS

- A folyamok általában megváltoztathatatlanok (immutable)
- Új folyamok előállítása:
  - map – új folyamot állít elő a megfelelő F leképezéssel
  - filter – kiszűr adott eseményeket
  - scan - aggregáló függvénnyel minden beérkezett értéket aggregál
  - …


```
  clickStream: ---c----c--c----c------c-->
               vvvvv map(c becomes 1) vvvv
               ---1----1--1----1------1-->
               vvvvvvvvv scan(+) vvvvvvvvv
counterStream: ---1----2--3----4------5-->
```

Reaktív programozás - RxJS

- Dupla kattintás:
  - 250 ms – esemény csend alapján csoportosítjuk
  - Összeszámoljuk
  - Kiszűrjük



RxJS

- Obervables
  - Push alapú lista
  - Adat forrás -> adat nyelő (felíratkozott lista tagok)
- LINQ
- Schedulers

Observables

```
// Creates an observable sequence of 5 integers, starting from 1
var source = Rx.Observable.range(1, 5);

// Prints out each item
var subscription = source.subscribe(
    x => console.log('onNext: ' + x),
    e => console.log('onError: ' + e.message),
    () => console.log('onCompleted'));

// => onNext: 1
// => onNext: 2
// => onNext: 3
// => onNext: 4
// => onNext: 5
// => onCompleted
```

Observables

- Forró vs. Hideg (Hot vs Cold)
  - Forró
    - Amennyiben senkit sem érdekel akkor is fut a háttérben
    - Amikor valaki fleíratkozik a legutolsó értéket kapja meg
    - Megvan osztva a fleíratkozottak között
  - Hideg
    - Csak akkor kezdi el az értékek kiadását, ha valaki felíratkozott rá
    - Nincs megosztva a fleíratkozottak között

RxJS operátorok

Usage	Operators
Creating an observable sequence	1. create 2. defer 3. generate 4. generateWithAbsoluteTime 5. generateWithRelativeTime 6. range 7. using
Converting events or asynchronous patterns to observable sequences, or between Arrays and observable sequences.	1. from 2. fromArray 3. fromCallback 4. fromNodeCallback 5. fromEvent 6. fromEventPattern 7. fromPromise 8. of 9. toArray 10. toMap 11. toPromise 12. toSet
Combining multiple observable sequences into a single sequence.	1. amb 2. prototype.amb 3. combineLatest 4. proto.combineLatest 5. concat 6. prototype.concat 7. startWith 8. merge 9. prototype.merge 10. mergeAll 11. repeat 12. prototype.repeat 13. withLatestFrom 14. zip 15. prototype.zip
Functional - Sharing Side Effects	1. let 2. publish 3. publishLast 4. publishValue 5. replay 6. share 7. shareLast 8. shareReplay 9. shareValue
Mathematical operators on sequences	1. aggregate 2. average 3. count 4. max 5. maxBy 6. min 7. minBy 8. reduce 9. sum
Time-based operations	1. debounce 2. debounceWithSelector 3. delay 4. interval 5. sample 6. timeInterval 7. timer 8. timeout 9. timeoutWithSelector 10. timestamp
Handling Exceptions	1. catch 2. prototype.catch 3. finally 4. onErrorResumenext 5. prototype.onErrorResumeNext 6. retry
Filtering and selecting values in a sequence	1. concatMap 2. concatMapObserver 3. elementAt 4. elementAtOrDefault 5. filter 6. flatMap 7. flatMapLatest 8. flatMapObserver 9. find 10. findIndex 11. first 12. firstOrDefault 13. includes 14. last 15. lastOrDefault 16. map 17. pluck 18. select 19. selectConcat 20. selectMany 21. selectManyObserver 22. selectSwitch 23. single 24. singleOrDefault 25. skip 26. skipLast 27. skipLastWithTime 28. skipUntil 29. skipWhile 30. take 31. takeLast 32. takeLastBuffer 33. takeLastBufferWithTime 34. takeLastWithTime 35. takeWhile 36. where

RxJS operátorok

Grouping and Windowing	1. buffer 2. bufferWithCount 3. bufferWithTimeOrCount 4. groupBy 5. groupByUntil 6. groupJoin 7. join 8. window 9. windowWithCount 10. windowWithTime 11. windowWithTimeOrCount
Imperative Operators	1. case 2. do 3. doOnNext 4. doOnError 5. doOnCompleted 6. doWhile 7. for 8. if 9. tap 10. tapOnNext 11. tapOnError 12. tapOnCompleted 13. while
Primitives	1. empty 2. never 3. return 4. throw

Időzítők

- A futtatandó feladatok mendzselése
  - Adat struktúra
  - Várakozási sor
  - Futási kontextus
  - Időzítés
- Virtuális idő
- Időzítő típusok
  - Standard
    - Adott időpontban
    - Adott időpontig
  - Rekurzív
  - Periodikus