UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Angular Bevezető

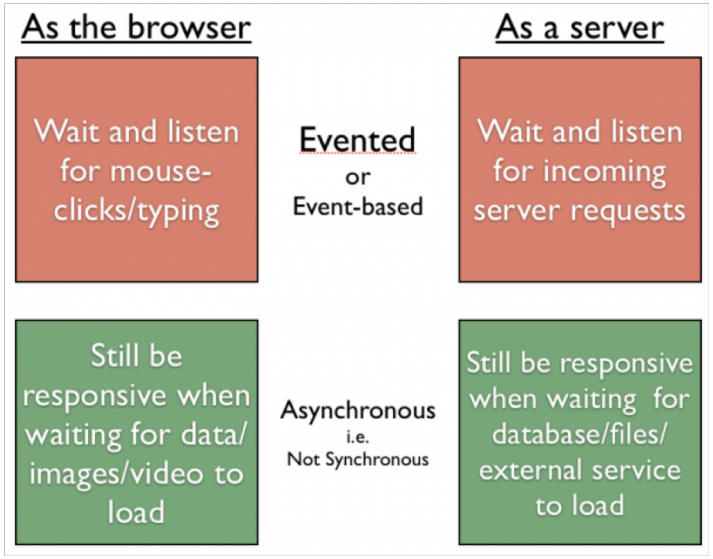
Dr. Bilicki Vilmos
Szoftverfejlesztés Tanszék

Összefoglaló

- JS
- AJAX
- DOM
- Angular architektúra áttekintés
  - Modulok
  - Komponensek
  - Szolgálttások és DI

JavaScript böngészőben

- A JavaScript tradícionális helye
- Örökzöld böngészők
- ES6 nem garantált
  - Babel
- Főbb területek
  - DOM manipuláció
  - AJAX



AJAX

- Asynchronous JavaScript and XML
- Aszinkron kommunikáció a szerver oldallal
- 2000: Web 2.0 – XMLHttpRequest
- A böngésző oldali JavaScript HTTP kéréseket küld a szervernek amley JSON objektumokkal válaszol
- Az oldal újratöltése nélkül teszi ezt

Példa

```
const http = require('http');

const server = http.createServer(function(req, res) {
    res.setHeader('Content-Type', 'application/json');
    res.setHeader('Access-Control-Allow-Origin', '*');
    res.end(JSON.stringify({
        platform: process.platform,
        nodeVersion: process.version,
        uptime: Math.round(process.uptime()),
    }));
});

const port = 7070;
server.listen(port, function() {
    console.log(`Ajax server started on port ${port}`);
});
```

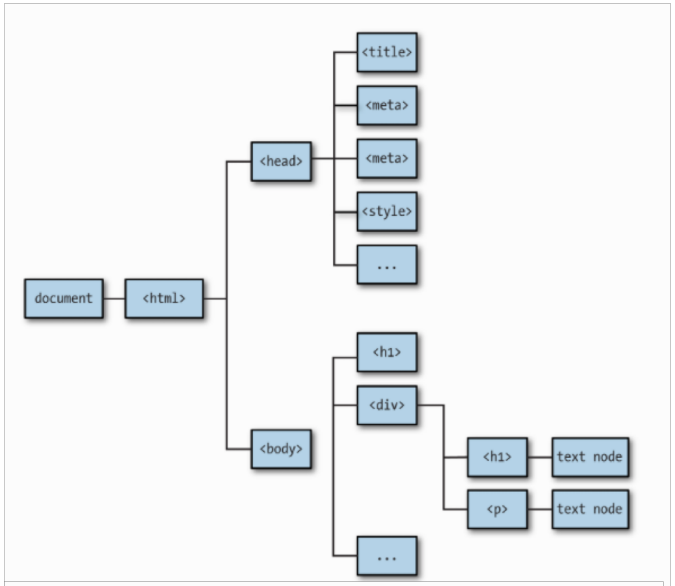
```
<script type="application/javascript;version=1.8">
    function refreshServerInfo() {
        const req = new XMLHttpRequest();
        req.addEventListener('load', function() {
            // TODO: put these values into HTML
            console.log(this.responseText);
        });
        req.open('GET', 'http://localhost:7070', true);
        req.send();
    }
    refreshServerInfo();
</script>
```

```
req.addEventListener('load', function() {
    // this.responseText is a string containing JSON; we use
    // JSON.parse to convert it to an object
    const data = JSON.parse(this.responseText);

    // In this example, we only want to replace text within the <div>
    // that has class "serverInfo"
    const serverInfo = document.querySelector('.serverInfo');

    // Iterate over the keys in the object returned from the server
    // ("platform", "nodeVersion", and "uptime"):
    Object.keys(data).forEach(p => {
        // Find elements to replace for this property (if any)
        const replacements =
            serverInfo.querySelectorAll(`[data-replace="${p}"]`);
        // replace all elements with the value returned from the server
        for(let r of replacements) {
            r.textContent = data[p];
        }
    });
});
```

```
setInterval(refreshServerInfo, 200);
```

DOM

- Document Object Model
  - Node osztály csomópontok
  - Parent/Child tulajdonságok




```
function printDOM(node, prefix) {
    console.log(prefix + node.nodeName);
    for(let i=0; i<node.childNodes.length; i++) {
        printDOM(node.childNodes[i], prefix + '\t');
    }
}
printDOM(document, '');
```

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Simple HTML</title>
        <style>
            .callout {
                border: solid 1px #ff0080;
                margin: 2px 4px;
                padding: 2px 6px;
            }
            .code {
                background: #ccc;
                margin: 1px 2px;
                padding: 1px 4px;
                font-family: monospace;
            }
        </style>
    </head>
    <body>
        <header>
            <h1>Simple HTML</h1>
        </header>
        <div id="content">
            <p>This is a <i>simple</i> HTML file.</p>
            <div class="callout">
                <p>This is as fancy as we'll get!</p>
            </div>
            <p>IDs (such as <span class="code">#content</span>)
                are unique (there can only be one per page).</p>
            <p>Classes (such as <span class="code">.callout</span>)
                can be used on many elements.</p>
            <div id="callout2" class="callout fancy">
                <p>A single HTML element can have multiple classes.</p>
            </div>
        </div>
    </body>
</html>
```

DOM metódusok

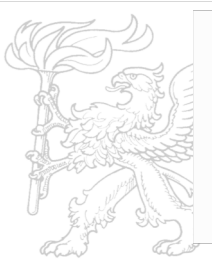
- get metódusok
- DOM lekérdezések
- DOM manipuláció

```
document.getElementById('content');      // <div id="content">...</div>
```

```
const callouts = document.getElementsByClassName('callout');
```

```
const paragraphs = document.getElementsByTagName('p');
```

```
document.querySelectorAll('.callout');
```



```
const para1 = document.getElementsByTagName('p')[0];
para1.textContent;     // "This is a simple HTML file."
para1.innerHTML;       // "This is a <i>simple</i> HTML file."
para1.textContent = "Modified HTML file";       // look for change in browser
para1.innerHTML = "<i>Modified</i> HTML file";  // look for change in browser
```

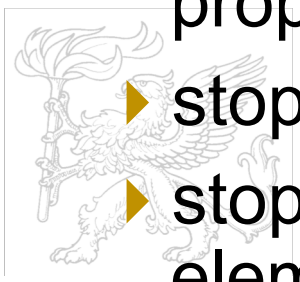
DOM események

- 200 szabvány esemény + böngésző által támogatott egyedi esemény
- addEventListener
- pereventDefault()

```
const highlightActions = document.querySelectorAll('[data-action="highlight"]');
for(let a of highlightActions) {
    a.addEventListener('click', evt => {
        evt.preventDefault();
        highlightParas(a.dataset.containing);
    });
}

const removeHighlightActions =
    document.querySelectorAll('[data-action="removeHighlights"]');
for(let a of removeHighlightActions) {
    a.addEventListener('click', evt => {
        evt.preventDefault();
        removeParaHighlights();
    });
}
```

Hierarchikus esemény kezelés

- A HTML hierarchikus
- Az eseményeket több szinten is kezelhetjük
  - “Milyen sorrendben kell az eseményeket lekezlenem?”
    - Capturing - Lektávolabbi őstől
    - Bubbling - A forrástól
- preventDefault – lemondja az eseményt (ettől propagálódik)
- stopPropagation – adott elem felett nem terjed
- stopImmediatePropagation – semmlyen további elem nem kapja meg (az aktuális sem)



Esemény típusok

- Fókusz események
- Űrlap események
- Beviteli eszköz események
- Média események
- Előrehaladás események
- Érintés események

jQuery

- Népszerű AJAX és DOM kezelő könyvtár
- Böngészőfüggetlenné teszi a kódunkat
- Egyszerűbb AJAX API
- Kompakt, kifejező eszköztár a DOM kezelésre

```
<script src="//code.jquery.com/jquery-2.1.4.min.js"></script>
```

```
$(document).ready(function() {
    // code here is run after all HTML has been
    // loaded and the DOM is constructed
});
```

```
$(function() {
    // code here is run after all HTML has been
    // loaded and the DOM is constructed
});
```

Jquery-be ágyazott DOM elemek

- Alapértelmezett megközelítésmód

```
$('p').text('ALL PARAGRAPHS REPLACED');
```

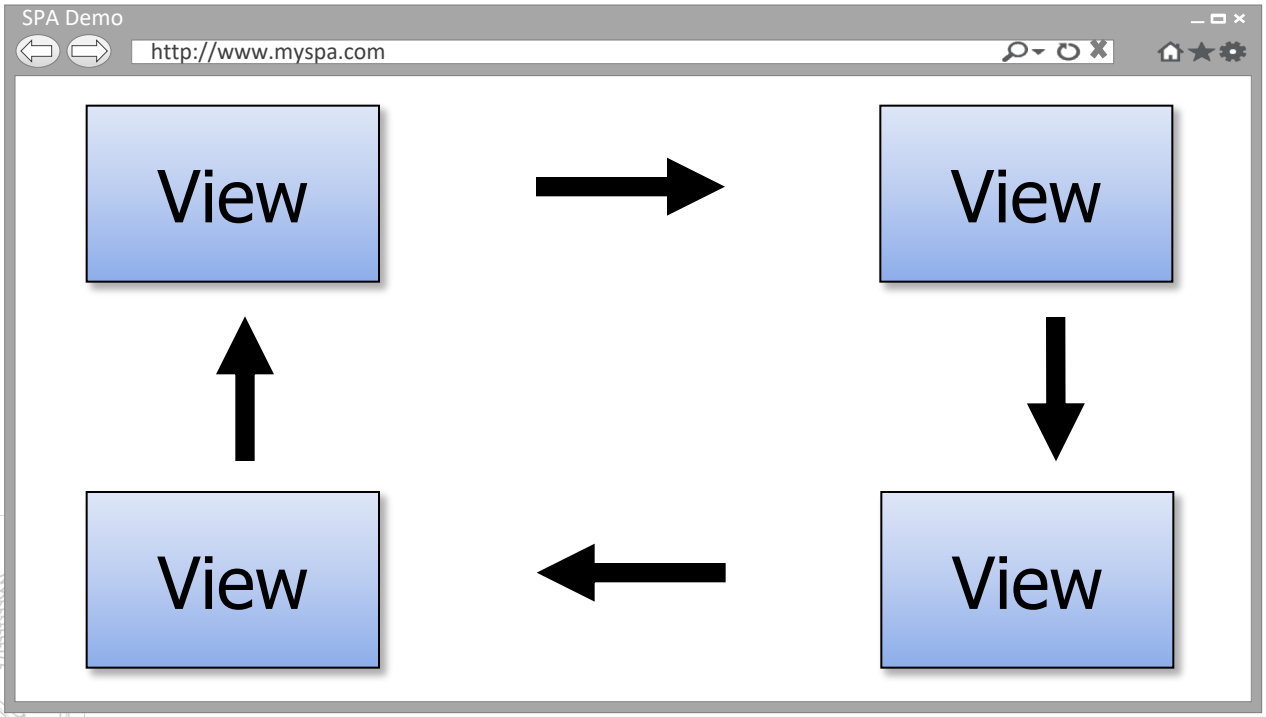
```
$('p')          // matches all paragraphs
    .eq(2)      // third paragraph (zero-based indices)
    .html('<i>THIRD</i> PARAGRAPH REPLACED');
```

```
$('p').remove();
```

```
$('p')
    .after('<hr>')
    .before('<hr>');
```

```
$('p')
    .after('<hr>')
    .append('<sup>*</sup>')
    .filter(':odd')
    .css('color', 'red');
```

Egy oldalas alkalmazás (SPA)



SPA kihivások

DOM manipuláció

Routing

Adat Kötés

Történet

Gyorsítótárazás

Ajax/Promises

Modul betöltés

Objektum modellezés

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

jQuery

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

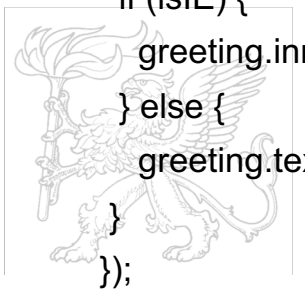
Hello HTML

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>
```



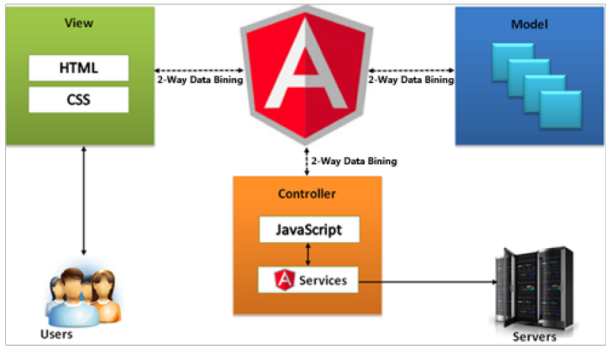
Hello JQuery

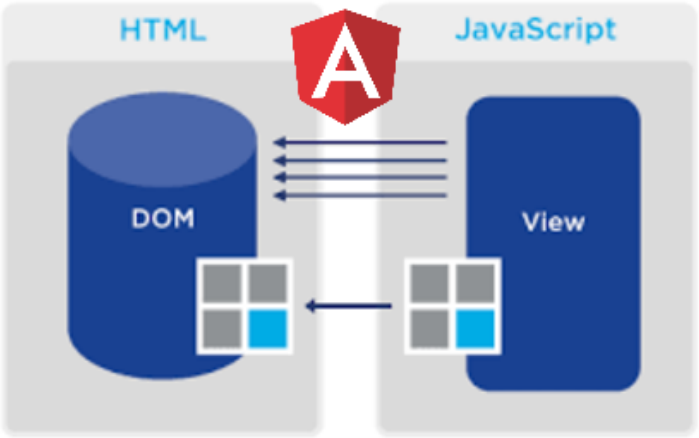
```
<p id="greeting2"></p>


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

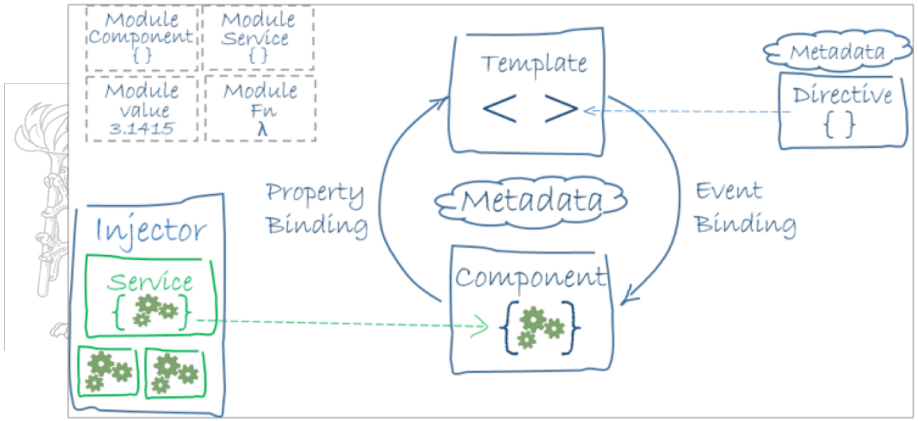
Hello AngularJS

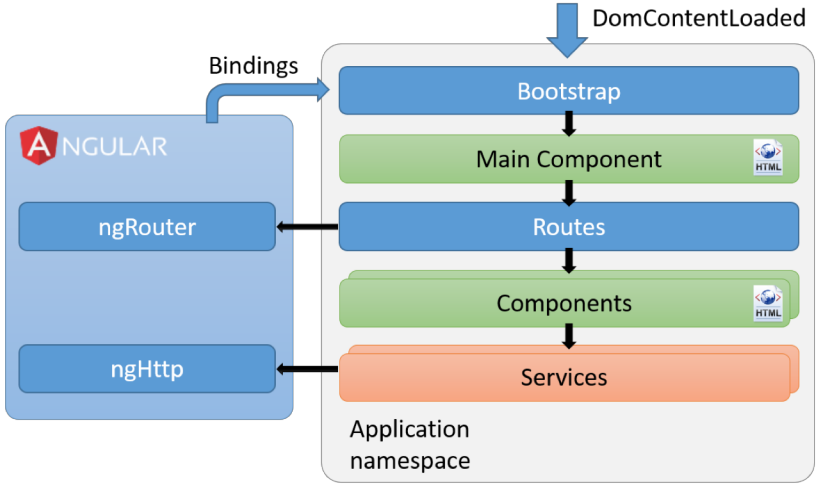
```
<p ng:init="greeting = 'Hello
   World!'">{{greeting}}</p>
```

Angular egy fólián

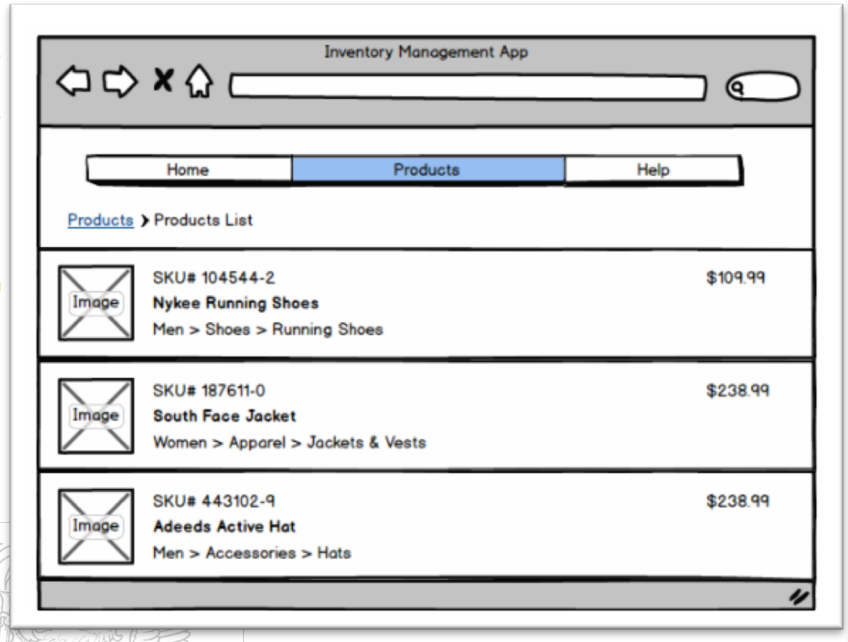


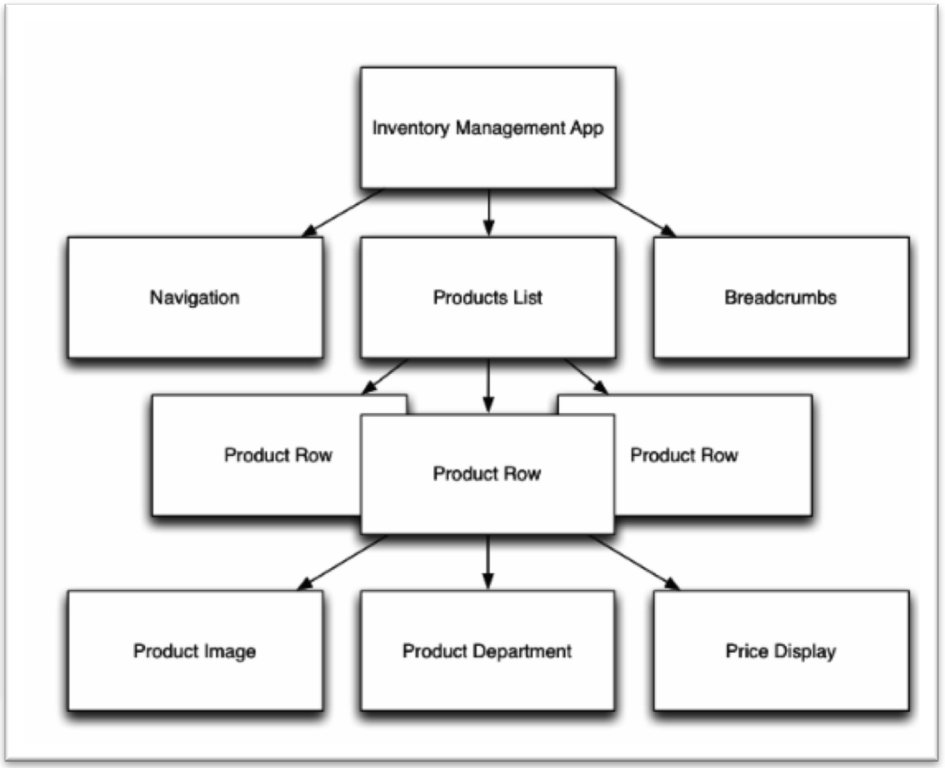






Angular koncepció





Kód ☺

Controller

```
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };

  constructor() { }

  ngOnInit() {
  }

}
```

View

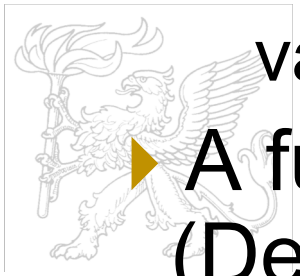
```
<h2>My Heroes</h2>

<ul class="heroes">
  <li *ngFor="let hero of heroes"
    [class.selected]="hero === selectedHero"
    (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Angular koncepció

- Platform és keretrendszer HTML és JS/TS kliens alkalmazások fejlesztésére
- Alap építőelem: NgModules
  - Fordító környezetet biztosít a komponenseknek
  - Funkcionális csoportokba rendezi a kódot
  - root modul – indítás

Dekorátorok

- Az angular elemek (service,component, pipe, ...) egyszerű TS osztályok megfelelő dekorációval
  - Komponensek: a view – nézetet specifikálják (sablonok segítségével, HTML + Angular jelölő nyelv)
  - Szolgáltatások: a modelt - vezérlőt valósíthatják meg
- A futaásidejű függőségeket DI (Depenedncy Injection) – Függőség Injektálás segítségével adják meg



Modulok

- NgModules – nem azonosak a TS/JS/ES modulokkal
- Hierarchikus funkcionális egységeket specifikálnak
  - Minden Angular alkalmazásnak van egy gyökér modulja (root module), ezt AppModule-nak szokták nevezni (app.module.ts fájl)
- Hasonlóan a JS/TS modulokhoz itt lehet exportálni és importálni
- Nagyobb alkalmazások több *feature* moduból állhatnak
- Egyedi képesség: lazy-loading csak akkor tölti be amikor szükség van rá

NgModule metaadatok

- Egyszerű @NgModule() dekorációval ellátott TS osztály
  - @NgModule() metaadatokat befogadó dekorátor függvény
    - declarations: az adott NgModule-hoz tartozó komponensek, csövek és direktívák
    - exports: Azon deklarációk halmaza amelyeknek láthatónak kell lennie msá NgModulok-ból és a komponens sablonból
    - imports: Más kiexportált modulok amelyekre ebben az NgModul-ban szükség van
    - providers: az igénybevett globális vagy lokális szolgáltatások listája (lokális a preferált)
    - bootstrap: az alkalmazás indító nézete (root NgModule)

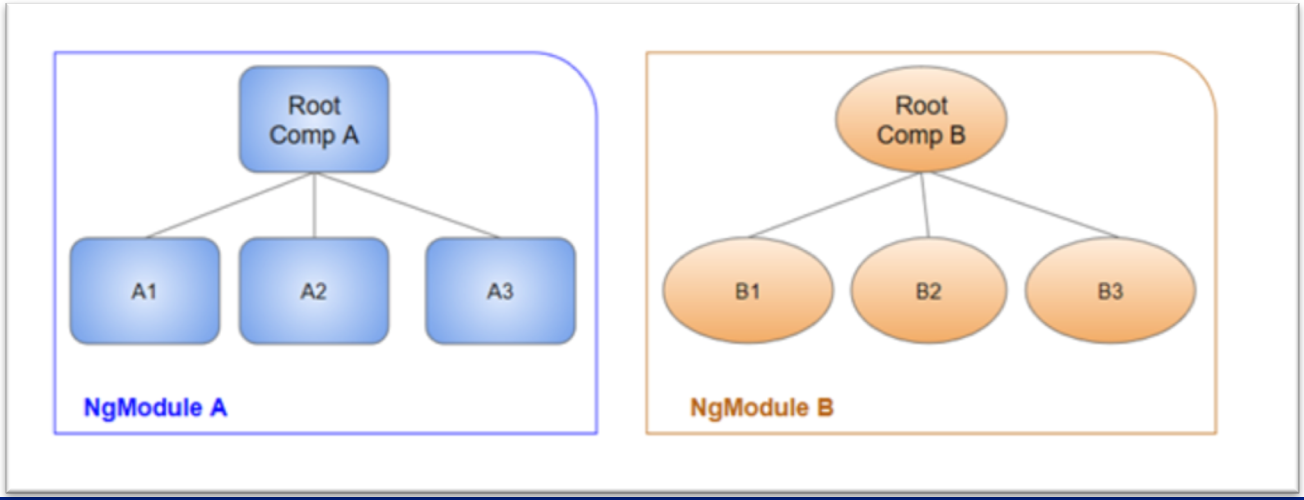
NgModule

src/app/app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

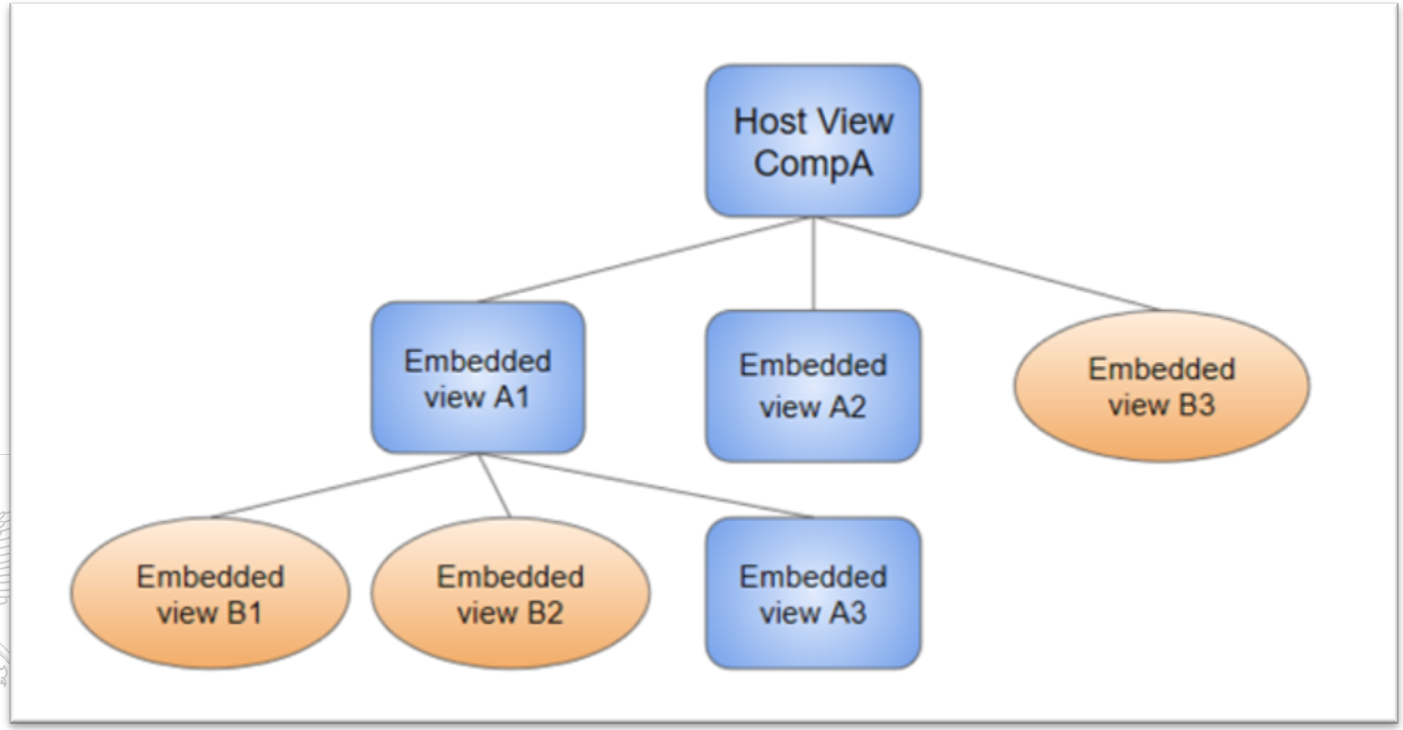
NgModule

- A saját komponenseinek *fordítási kontextust* biztosít (egyedi tsconfig.json)
- A gyökér modul minden alapja
- Az adott NgModul-ban lévő komponensek osztoznak a fordítási környezeten (ez tranzitív)



NgModule vs. View hierarchia

- host view



NgModule vs. JS modulok

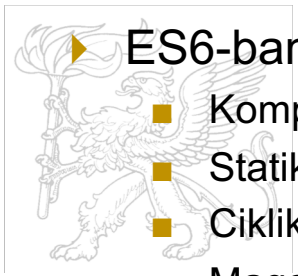
- Az alap a JS/ES/S modul rendszer
- Ezt egészíti ki az NgModul rendszer

```
import { NgModule }      from '@angular/core';
import { AppComponent } from './app.component';
```

```
export class AppModule { }
```

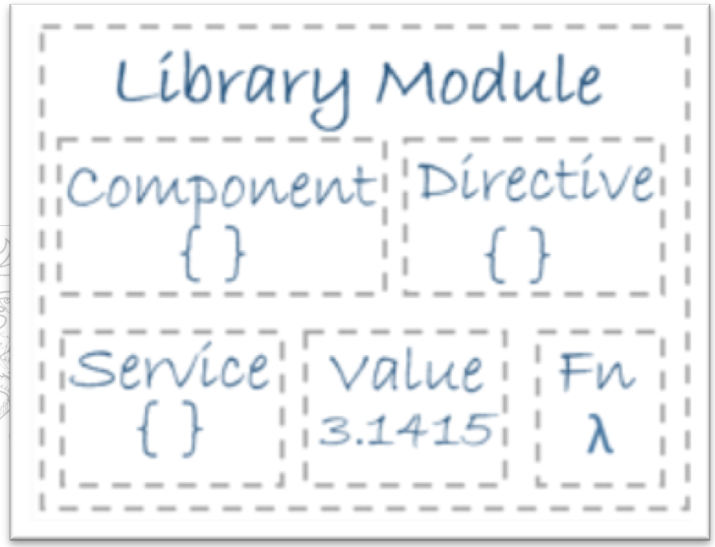
JS modul rendszer

- Hagyományosan nem nyelvi szinten hanem könyvtárak segítségével valósították meg pl.:
  - CommonJS Modules: szinkron betöltés
  - Asynchronous Module Definition (AMD): aszinkron betöltés
- Konvenció
  - A modul egy a betöltés után végrehajtható kódrész
  - A nem exportált változói, deklarációi, ... Lokális szkópban maradnak, kívülről nem elérhetőek
  - A modulok singelton-ok. Ha többször importáljuk is be csak egy példánya van.
  - Lehetséges hivatkozások
    - Relatív ('../model/user')
    - Abszolút ('/lib/js/helpers')
    - Nevesített ('util')
- ES6-ban került nyelvi szintre
  - Kompakt nyelvi megoldás: CommonJS
  - Statikusan elemezhető
  - Ciklikus függőségek kezelése
  - Maga a betöltő (Loader) nem része a szabványnak



Angular könytárak

- Angular mint TS modulok gyűjteménye
- @angular prefix
- Jellemzően npm-el kezeltek




```
import { Component } from '@angular/core';
```

```
import { BrowserModule } from '@angular/platform-browser';
```

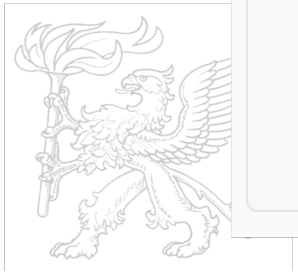
Komponensek

- A képernyő adott részéért felelős
- Az alkalmazás logika az osztályon belül kerül definiálásra
- Az osztály API-ban megadott tulajdonságok és metódusok segítségével kommunikál a view-vel
- A komponensek életciklusa (létrehozás, frissítés, megsemmisítés) az Angular keretrendszer feladata: életciklus horgonyok pl.: ngOnInit()

Példa

src/app/hero-list.component.ts (class)

```
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```



Komponens metaadatok

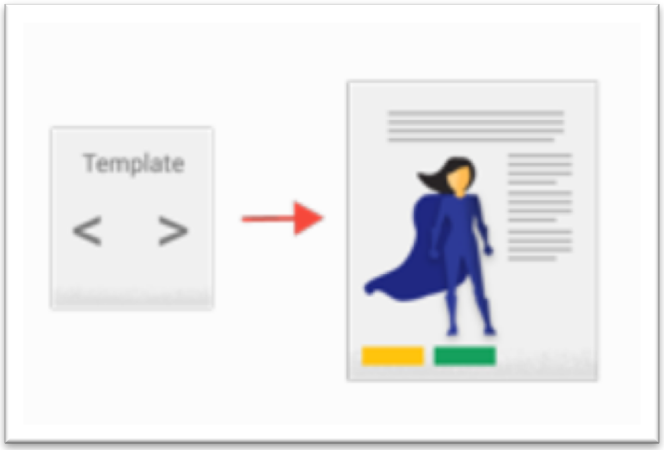
- @Component() dekorátor függvény
  - selector: a sablon eleme neve
  - templateUrl/template: a komponens view részét leíró HTML + Angular sablon – host view
  - Providers: a felhasznált szolgáltatások listája

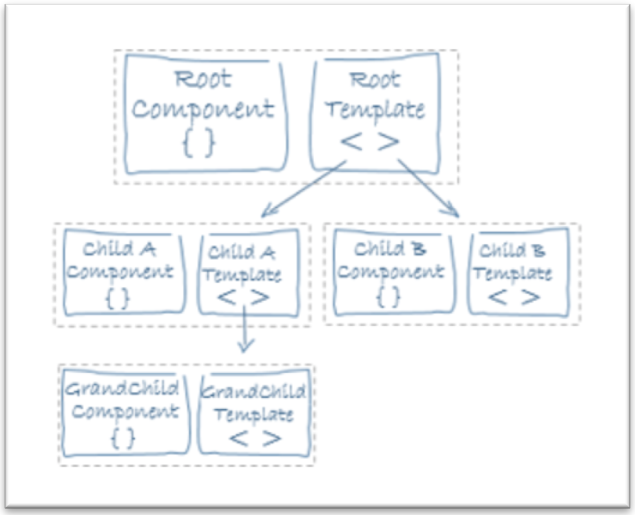
src/app/hero-list.component.ts (metadata)

```
@Component({
  selector:    'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
/* . . . */
}
```

Sablonok és nézetek

- A view-et a sablon segítségével adjuk meg
- Ez egy HTML részlet amely elmondja a tartalom renderelését
- Jellemzően hierarchiába rendezettek
- host view – view hierarchy, embedded views





Sablon szintaxis

- Reguláris HTML + Angular sablon szintakszis
- DOM – manipuláció
- Adatkötés: DOM – alkalmazás összekötés
- Csövek: megjelenítés előtti adatranszformáció
- Direktívák: a megjelenítés alkalmazás logikája

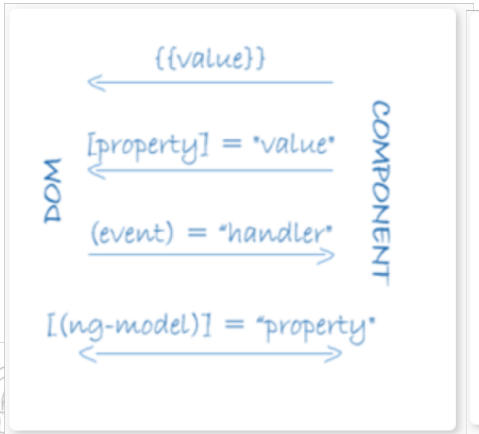
Példa

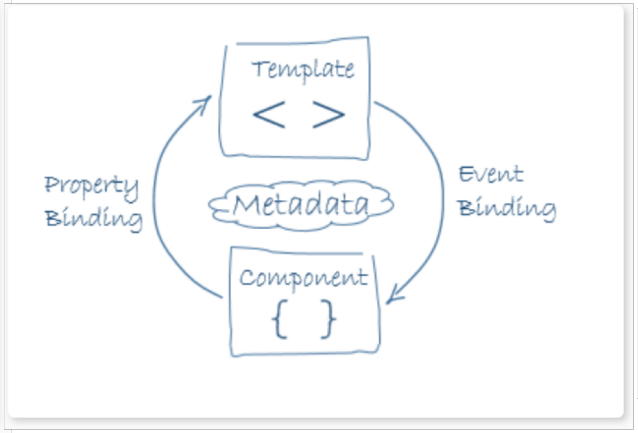
src/app/hero-list.component.html

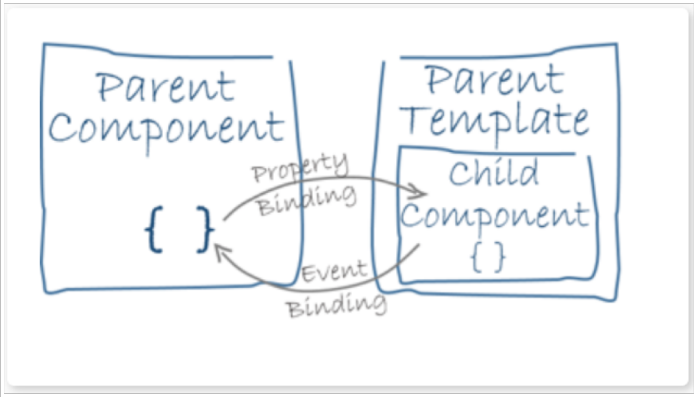
```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

Adatkötés

- Keretrendszer nélkül manuálisan kellene a HTML vezérlők adatait frissíteni és a felhasználói interakciókat akciókként kezelni
- Saját push – pull logika → jQuery
- Kétirányú adatkötés: a sablon egy része és a kód egyrész közötti kapcsolat
- Minden JS eseménykezelési ciklusban frissíti a gyökértől lefelé a gyerekek felé haladva







src/app/hero-list.component.html (binding)

```
<li>{{hero.name}}</li>
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
<li (click)="selectHero(hero)"></li>
```

src/app/hero-detail.component.html (ngModel)

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

Csövek

- MVVM megvalósítását segítik
- A megjelenítési adatok transzformációját végzik
- @Pipe dekoráció, cső operátor: |
- {{interpolated_value | pipe_name}}
- Beépített csövek

```
<!-- Default format: output 'Jun 15, 2015'-->
 <p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15, 2015'-->
<p>The date is {{today | date:'fullDate'}}</p>

 <!-- shortTime format: output '9:43 AM'-->
 <p>The time is {{today | date:'shortTime'}}</p>
```

Direktívák

- Az Angular sablonok dinamikusak
- Amikor az Angular rendereli a tartalmat akkor a DOM-at a *direktívák* alapján alakítja át
- @Directive() dekoráció
- A komponensek a direktívákat egészítik ki sablon orientált képességekkel
- Két típusa van
  - struktúrális: DOM elemek hozzáadásával, eltávolításával, kicserélésével változtaja meg a kinézetet
  - attribútum: a viselkedést és a megjelenítést befolyásolják

src/app/hero-list.component.html (structural)

```
<li *ngFor="let hero of heroes"></li>
<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

src/app/hero-detail.component.html (ngModel)

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

Szolgáltatások

- MWV (Model Whatever you want View)
- Komponens: felhasználói élmény
  - A view közvetlen kiszolgálása
  - Adott feladatokat kiszervezhet: adatkezelés a szerver felé, input validáció, naplózás, ...
- Modularitás, újrafelhasználhatóság
- Szolgáltatás: bármely érték, függvény, képesség amire egy alkalmazásnak szüksége van
- Tipikusan: egy csak adott feladatra koncentráló osztály
- Injektálható szolgáltatás osztály: bármely komponensnek elérhető (különböző megvalósítások)

Példa

src/app/logger.service.ts (class)

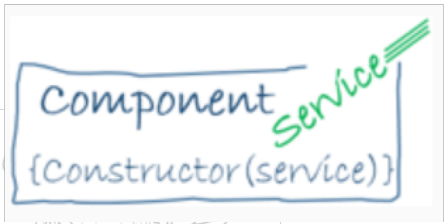
```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

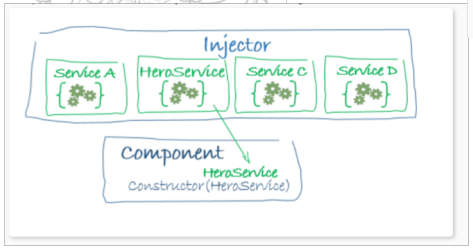
src/app/hero.service.ts (class)

```
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

Függőség Injektálás - DI

- Az Angular keretrendszerben széleskörben alkalmazott
- Kontextus függő laza csatolás
- @Injectable() dekorátor
  - Automatikusan alkalmazás szkópú injektálható szolgáltatás
- @NgModule, @Component
  - Komponens, modul szkópú szolgáltatás
- Szolgáltató (Provider) adja meg az injektálható szolgáltatás függőségét, létrehozásának módját (DI Token)






```
@NgModule({
  providers: [
  BackendService,
  Logger
 ],
 ...
})
```

src/app/hero-list.component.ts (component providers)

```
@Component({
  selector:    'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers:  [ HeroService ]
})
```

Összefoglaló

- JS
- AJAX
- DOM
- Angular architektúra áttekintés
  - Modulok
  - Komponensek
  - Szolgálttások és DI